Advanced Features in SAS/AF®

Mary Beth Naim, The Lubrizol Corporation

ABSTRACT

SAS/AF Software is used to create user-friendly front ends to application system. It is easy to get started creating and linking together simple screens, complete with validation of user entered responses. The true strenght of SAS/AF, however, lies in its not so obvious advanced features. Through the intricate application of numerous advanced features, entire information systems can be driven from their AF front end. This paper will discuss a small sample of these features such as program code, design approaches, utilities and problems that can be encountered.

INTRODUCTION

It doesn't take a lot of exposure to SAS/AF before you wonder if you can do more than just create great looking screens and set attributes. Both the AF manual and the AF course give just an inkling of AF's capabilities. These capabilities extend far beyond the simple entry and validation of data. After completing a year-long project that centered around AF; our team has been able to assemble a portfolio of tips and techniques to follow when using this product. The areas addressed in this paper are:

Programming Techniques - Examples of code that is under the program screen.
Development Considerations - Different ways to develop systems when using SAS/AF.
Utility Procedures - Examples of the more involved procedures such as printing screens.
Problems To Watch Out For - Bad memories revisited.

PROGRAMMING TECHNIQUES

Running under TSO/MVS our customers are used to using the 'end' key to back out of ISPF panels. In AF the 'enter' key is used to submit a menu screen and the 'end' key is used to submit a program screen. When reviewing the prototype of our projected system, the customers were confused by this. To eliminate confusion, they requested that the 'enter' key be used throughout the system, to continue to the next screen. The following logic was put under each program screen to accomplish this:

```
            ENTER.PROGRAM
Command ===>              Text Editor

   Example of Code that Allows the Screen

   to be Submitted Using the Enter Key

 &
-------------------------------------------------------------------
```

```
### enter
%macro enter;
%if &_dcall NE INITIAL %then
 %do;
  %if (&_derron = %str() or &_derron = %str( ))
    AND &_dkey = 0          %then
     %let _dcmnd = END;
 %end;
%mend enter;
# # #
```

As an added feature, we allow the customer to branch from the current screen to any previously displayed screen. The branching options are listed on the bottom of the screen as a number of action fields. The code reviews these fields from left to right and branches to the first screen selected, without any further validation. This eliminates the need to ensure that only one option is selected. If branching from a screen is selected, all of the program logic under the screen must be bypassed. The code to branch to other screens is displayed below:

```
            BRANCH.PROGRAM
Command ===>                  Text Editor

        Example of Screen Branching Logic

        Please Enter your name: &name___

   & Return to Main Menu      & Return to Screen 1

    & &
   ---------------------------------------------------------------------
```

```
### validate
%macro validate;
%if &_dcall = INITIAL %then
 %do;
  %let RETURNMM = %str();
  %let RETURNS1 = %str();
 %end;
%else
 %do;
  %if (&returnmm = %str( ) or &returnmm = %str()) AND
     (&returns1 = %str( ) or &returns1 = %str()) AND
     (&_dkeydef NE HELP)                %then
  %do;
    %put  The rest of the screen logic goes here;
  %end;
 %else
  %do;
    %if &returnmm = X  %then
     %let branch1 = X;
     %else
     %if &returns1 = X  %then
       %let branch2 = X;
  %end;
 %end;
%mend validate;
# # #

##field4
  %let RETURNMM  = %str();
  %let BRANCH1   = %str();
  PROC DISPLAY C = SASDATA.SUGI.MAIN.MENU;
  RUN;
```

Attribute settings fore each screen variable, going from top to bottom and left to right.

Field1 - Associated SAS Macro Variable =
NAME
Field2 - Action Type
Associated SAS Macro Variable =
RETURNMM
Field3 - Action Type
Associated SAS Macro Variable =
RETURNS1
Field4 - Action Type
Protected
Non-display
Associated SAS Macro Variable =
BRANCH1
Field5 - Action Type
Protected
Non-display
Associated SAS Macro Variable =
BRANCH12

Note the extra fields named branch1 and branch2. These action fields are used to branch out of the ### logic so that the PROC DISPLAY which initiates branching occurs outside of the ### logic. If the PROC DISPLAYs are put inside of the ### logic, the next time the original screen is executed the following message is generated:

```
ERROR 1105: AN EXECUTING MACRO MAY NOT BE REDEFINED, DUMMY
MACRO WILL COMPILE.
```

This happens because the end of the ### macro is never encountered. The next time the screen is executed SAS thinks that the macro is being redefined. This results in an error.

Another side affect of the branching described above, is a problem with required fields. If there are fields on the screen that are required by the attribute setting, SAS will force the user to fill in these fields. If the user wants to branch out of the screen, required fields should be ignored. To get around this problem we used ### logic under the screen instead of attribute setting to enforce required fields.

```
                REQUIRED.PROGRM
Command ===>                        Text Editor

        Example of Screen Branching Logic

          With Required Fields

        Please Enter your name: &name___

  & Return to Main Menu      & Return to Screen 1
    & &
  ----------------------------------------------------------------------
### validate
%macro validate;
%if &_dcall = INITIAL %then
 %do;
  %let RETURNMM = %str();
  %let RETURNS1 = %str();
 %end;
%else
 %do;
  %if (&returnmm = %str( ) or &returnmm = %str()) AND
    (&returns1 = %str( ) or &returns1 = %str()) AND
    (&_dkeydef NE HELP)                    %then
  %do;
    %if &name = %str() or &name = %str( ) %then
     %do;
      %let _dmsg   = You must enter your Name! ;
      %let _derron  = name;
      %let _dcursor = name;
     %end;
  %end;
 %else
  %do;
    %if &returnmm = X  %then
     %let branch1 = X;
    %else
     %if &returns1 = X  %then
      %let branch2 = X;
  %end;
 %end;
%mend validate;
# # #

##field4
 %let RETURNMM  = %str();
 %let BRANCH1   = %str();
 PROC DISPLAY C = SASDATA.SUGI.MAIN.MENU;
 RUN;

##field5
 %let RETURNS1  = %str();
 %let BRANCH2   = %str();
 PROC DISPLAY C =
   SASDATA.SUSGI.SCREEN1.PROGRAM;
 RUN;

# #
 PROC DISPLAY C=SASDATA.SUGI.NEXT.PROGRAM;
 RUN;
```

An extra benefit of using ### logic other than attributes to enforce required fields is the customized error messages which can be coded into the logic. These custom messages can be much more explicit than the default AF message:

Default Error Message

```
                ERRORMSG.PROGRM
Command ===>
ERROR: Required fields are blank or in error.
          Example of Screen Branching Logic

            With Required Fields

          Please Enter your name: _____
```

Customized Error Message

```
                ERRORMSG.PROGRM
Command ===>
You must enter your name.
          Example of Screen Branching Logic
            With Required Fields

          Please Enter your name: _____
```

On a number of occasions we were asked to design screens with exclusive selections. If there are only two fields involved, a nested 'IF' structure is all that is needed to validate the entries. If there are more than two fields involved in the selection and only a certain number of valid combinations are allowed, the 'IF' structure can become quite involved. In these cases we found that it is more compact and economical to use a loop that counts the number of fields selected and then validates the entries based on this count.

```
                  LOOP.PROGRAM
Command ===>                        Text Editor

          Select One Of The Following Options

                  & Option 1
                  & Option 2
                  & Option 3

          --------------------------------------------------------------------
### select
%macro select;
%if &_dcall=INITIAL %then
 %do;
  %let option1 = %str();
  %let option2 = %str();   /* initialize all selections */
  %let option3 = %str();
 %end;
%else
 %do;
  %let validsel = 0;      /* initialize selection counter */
  %do x = 1 %to 3;
   %if (&&option&x NE %str( ) AND
       &&option&x NE %str()) %then
    %do;
     %let validsel = %EVAL(&validsel + 1);
    %end;
  %end;       /* if entry is found increment validsel */
            /* no options selected flag as an error */
  %if &validsel < 1 %then
   %do;
    %let _dmsg = You must select one of the options;
    %let _dcursor = option1;
    %let _derron  = option1;
   %end;
 %else
                /* more than 1 option selected error */
    %if &validsel > 1 %then
     %do;
      %let _dmsg = You can only select one of the options;
      %let _dcursor = option1;
      %let _derron  = option1;
     %end;
 %end;
%mend select;
 ###
```

DEVELOPMENT CONSIDERATIONS

When developing an AF system there are many techniques that can be employed to save time, increase efficiency and assist in debugging.

It is helpful to include a PROC BUILD as one of the options on the main menu screen. When this option is selected it will call a program screen that has a PROC BUILD statement under the dashed line. The PROC BUILD will execute, changing control from the display mode to the edit mode. This causes the SAS catalog containing the screens to be displayed. From there you can simply back out of the SAS catalog holding the screen, causing the PROC DISPLAY to become active once again. When the system is ready to be moved into production, the capability of executing the PROC BUILD can be maintained. Although there shouldn't be any text on the menu to indicate that any maintenance option exists, a section of the system's maintenance manual can instruct the support personnel to select option 'PB' or PROC BUILD on the menu. This simplifies the incorporation of any modifications or updates to the system. We also recommend that write access to the screen datasets be restricted to the support personnel for added safety.

Example of a main menu with the Proc Build option:

```
                 MAIN.MENU
Command ===>                       Text Editor

     This is the Main Menu of the Application System

                   1 ... Option 1
                   2 ... Option 2
                   3 ... Option 3
                   4 ... Proc Build
                   X ... Exit the System
```

Attribute settings for the main menu:

```
            Attributes for MENU & CBT screens
Command ===>

Parent Screen    Name            Libref          *
                       Type             Catalog  *

Option  Key     Name      Type    Libref  Catalog

   1    __      SCREEN1     PROGRAM       *      *
   2    __      SCREEN2     PROGRAM       *      *
   3    __      BRANCH      PROGRAM       *      *
   4    __      BUILD       PROGRAM       *      *
   X    __      EXIT        PROGRAM       *      *
   6    __      _______     _______       *      *
   7    __      _______     _______       *      *
   8    __      _______     _______       *      *
   9    __      _______     _______       *      *
  10    __      _______     _______       *      *
```

Accompanying PROC BUILD program screen:

```
                 BUILD.PROGRAM
Command ===>                       Text Editor

-----------------------------------------------------
PROC BUILD C=SASDATA.SUGI;
RUN;
PROC DISPLAY;
RUN;
```

Our system is executed automatically from a CLIST called through an ISPF panel. The CLIST handles the allocation of files and the actual call to SAS with option settings. Running under this environment, our customers can use the system without knowing anything about SAS.

One problem encountered when executing from a CLIST is that the log is deleted when SAS is terminated automatically by the CLIST. In order to save the data in the log, the output must be directed to a sequential file, which is especially important during testing. This technique can also be helpful when developing code in the display manager, especially in those cases where a code error is severe enough to terminate the SAS session. The log file can help you trace where you were in the code when the session ended. An example of a driver CLIST with a log file specified is given here:

```
EDIT -- ISU0570.CLIST(SUGI) - 01.12 ------ COLUMNS 001 072
COMMAND ===>                          SCROLL ===> DATA
********************** TOP OF DATA ******************************
000001   PROC 0
000002   CONTROL  LIST MSG  SYMLIST  CONLIST
000003   FREE DD(SASEXEC SCRLIB)
000004    ALLOC DD(SASEXEC) DA('PDS5.ISU0570.LIB(SECOND)')
000005      SHR
000006    ALLOC DD(SCRLIB)  DA('ISU0570.SAS.DATA') SHR
000006   %SAS +
000007     LOG(""ISU0570.SASLOG.DATA"") +
000008    OPTIONS('MACRO,NOTES,MAUTOSOURCE, +
000009      SYSIN=SASEXEC,MCOMPILE, NOGRAPHICS,+
000010     NOIMS, DMS,DQUOTE,MACROGEN,MPRINT, +
000010     SYMBOLGEN,SOURCE,MLOGIC') +
000011       AUTOS(""PDS5.ISU0570.LIB"")
000012   FREE DD(BVPDD INP)
000013   EXIT
 ****************** BOTTOM OF DATA ****************************
```

```
EDIT -- PDS5.ISU0570.LIB(SECOND) 1.03 -- COLUMNS 001 072
COMMAND ===>                          SCROLL ===> DATA
********************** TOP OF DATA **********************
000001
000002  PROC DISPLAY C=SCRLIB.SUGI.MAIN.MENU;RUN;
000003
******************** BOTTOM OF DATA *****************
```

Once the system is pieced together and ready to be tested, the CLIST is created to drive the system. Macros within the system are moved to a PDS (Partitioned Data Set) library and the autocalls option is turned on. One of the major benefits of using autocall is that conditionally executed macros are only compiled if they are actually called.

UTILITY PROCEDURES

If you have been in SAS/AF and have tried to get a hard copy of your screen, you quickly find that it isn't a straightforward process. There are a number of steps which must be followed in order to print an AF screen. The first step is to create a FORM type screen. This is done by typing 'Edit Screenname.FORM' on the command line of the AF catalog. The following three screens will be displayed automatically, prompting you to enter information about the destination printer.

```
                    Printer selection list
Command ===>
                                  _ IBM 6670
                                  _ IBM 3800
                                  _ Xerox 2700
                                  _ Xerox 5700
                                  _ Xerox 9700
                                  _ Line Printer
```

```
              Text Body and Margin Information
Command ===>
                    Text Body Information

      Characters per Line: 72      Lines on First Page: 54
         Lines on Following: 54

First Page Margins          Left: __  Top: __  Bottom: __

Following Pages  Left: __  Top: __  Bottom: __
```

```
                    TSO Print File Parameters
Command ===>

Destination: _____      Class: A          Program: _____
Forms:       ____       Ucs: ____         Outlim: _____
Copies:      1          Fcb: ____         Hold: _

Carriage Control Information  No CC: _

Signal Page Skips Before   First Print Control Language:
                                First Text  Page:                X
                                Following Print Control Language:
                                Following Text  Pages:           X

Additional Information needed only for IBM 3800 printers

Character Tables: _ _ _ _ _ _   Burst: _   Optcode=J: _
Flash Name: _  Flash Count: _  Modify Name: _  Modify Trc: _
```

This form screen can then be used by a batch job which prints the screen. The batch job actually executes a PROC BUILD of the FORM screen with the screen to be printed in the select statement. In the following example the FORM screen is called PRINT.FORM in our test catalog.

```
EDIT - PDS5.ISU0570.LIB(SCREENPT) 1.32 -- COLUMNS 1 072
COMMAND ===>                          SCROLL ===> DATA
****************************** TOP OF DATA ***********************
000100 // ISU0570D JOB MBN,'MARY BETH NAIM',
000200 // CLASS=Q, MSGCLASS=X,REGION=2048K,
000300 // NOTIFY=ISU0570,MSGLEVEL=(0,0)
000500 // STEP1   EXEC SAS,OPTIONS='GEN=0                          X
000600          NOSOURCE NOSOURCE2 NOMACROGEN'
001200 // SASDATA  DD DSN=ISU0570.SAS.DATA,
001300 //  DISP=SHR
001400 // SYSOUT DD SYSOUT=2
001500 // SYSPRINT   DD *
002300 // SYSIN      DD *
002410        FORM=PRINT;
002420        SELECT REQUIRED.PROGRAM;
002500 /*
002600 //
************************** BOTTOM OF DATA ********************
```

A few additional notes pertaining to the FORM screen:

- It is possible to list multiple screens on the select statement.
- The PROC DISPLAY of the FORM screen must be executed in a batch or non- interactive mode (can not be done from the display manager) if running a version under 5.16.
- The form file must be in the same catalog as the screen that is being printed.

Another utility feature available in AF is the triple equal signs. This function is used to copy the code under an AF program screen into an external file. This can have several applications. It can be used as another means of printing the program code, or down-loading data to a file for entry into a non-SAS program. The following example shows how to copy the code into a partitioned data set (PDS) file under TSO. The first step in this process is to allocate the PDS member.

```
Command ===>                                SAS(r) Log  15:30

 2 tso alloc fi(port) da('pds5.isu0570.lib(import)') old;
 2                                  run;
```

One limitation to this copy is that it will only copy the code under the screen and not the display portion of the screen (anything above the dashed line). Next, edit the screen that is to be copied, and place three = signs followed by the ddname before the fist line of code. After the last line of code enter three more equal signs.

```
                    MOVECODE.PROGRAM
Command ===>                                  Text Editor

       Example of Code Being Copied Into an External File
 &
   ------------------------------------------------------------------------
=== port
 ### enter
%macro enter;
%if &_dcall NE INITIAL %then
 %do;
   %if (&_derron = %str() or &_derron = %str( ))
     AND &_dkey = 0              %then
       %let _dcmnd = END;
 %end;
%mend enter;
 ###
===
```

The copy is actually executed when a PROC DISPLAY of the screen is submitted. The end result of this process is a member called PDS5.ISU0570.LIB(import) that contains the code under the screen.

```
Command ===>                          SAS(r) Log  15:34

 4 proc display c=sasdata.sugi.movecode.program;
NOTE: PROFILE LIBRARY HAS NOT BEEN SPECIFIED.
NOTE: TEMPORARY PROFILE HAS BEEN OPENED.

 4                          run;
```

```
EDIT - PDS5.ISU0570.LIB(IMPORT) - 1.00 -- COLUMNS 01 072
COMMAND ===>                        SCROLL ===> DATA
 ********************************* TOP OF DATA **********************************
 000001 ### enter
000002 %macro enter;
000003 %if &_dcall NE INITIAL %then
000004  %do;
000005     %if (&_derron = %str() or &_derron = %str( ))
000006       AND &_dkey = 0              %then
000007         %let _dcmnd = END;
000008  %end;
000009 %mend enter;
 000010 ###
 ******************************** BOTTOM OF DATA ********************************
```

PROBLEMS TO WATCH OUT FOR

During the development of our system we managed to run into a few errors which took a considerable amount of time and patience to locate and rectify. In hopes of preventing anyone else from making these same mistakes I have included them in this paper.

SAS does not allow a data step within the ### logic of a program screen. In addition, any macros called from within the ### logic can not contain a data step.

```
                    DATASTP.PROGRAM
Command ===>                                  Text Editor

          Example of a Program Attempting to Use a

          Data Step Inside of the Triple Pound Logic

            & Select Here to Create a New Data Set

   ------------------------------------------------------------------------
 ### tryds
%macro tryds;
 %if &_dcall NE INITIAL %then
  %do;
    data newtest;
      set sasdata.test;
    run;
  %end;
%mend tryds;
  ###
```

```
Command ===>                          SAS(r) Log  15:40

 3 proc display c=sasdata.sugi.datastp.program;
NOTE: PROFILE LIBRARY HAS NOT BEEN SPECIFIED.
NOTE: TEMPORARY PROFILE HAS BEEN OPENED.
WARNING: Unresolved text from macro tryds has been ignored:
data newtest;
        set sasdata.test;
           run;

 3                          run;
```

Use the dquote option with double quotes around the character variables to avoid having SAS mistakenly interpret entered data as HEX.

```
                    HEX.PROGRAM
Command ===>                                  Text Editor

          Entering Character Data That Could

          Be Mistaken For Hexadecimal:  &hex____

   ------------------------------------------------------------------------
##hex
%macro checkhex;
%IF "&hex" NE " " %THEN
  %put Hex Found;
%mend checkhex;
%checkhex
run;
```

```
Command ===>                          SAS(r) Log  15:44

 9+ %macro checkhex;
     1-<DISPLAY>
10+ %IF "65A" NE " " %THEN
     1-<DISPLAY>
11+   %put Hex Found;
     1-<DISPLAY>
12+ %mend checkhex;
     1-<DISPLAY>
13+ %checkhex
     1-<DISPLAY>
405:  COLUMN 2
ERROR 405: INVALID NUMERIC HEX LITERAL OR NUMBER
        FOLLOWED BY ONE OF THE LETTERS "A"-"F".
        USE AN "X" AFTER THE NUMERIC HEX LITERAL
        OR USE A BLANK BETWEEN THE NUMBER(S) AND
        THE LETTER(S).
```

Same code with the dquote option turned on:

```
Command ===>                    SAS(r) Log  15:48

20+
    1-<DISPLAY>
21+
    1-<DISPLAY>
21+ %macro checkhex;
    1-<DISPLAY>
22+  %IF "65A"  NE " " %THEN
    1-<DISPLAY>
23+   %put Hex Found;
    1-<DISPLAY>
24+ %mend checkhex;
    1-<DISPLAY>
Hex Found
25+ %checkhex
    1-<DISPLAY>
28+run;
    1-<DISPLAY>
19                      run;
   ----------------------------------------------------------------------
--------
Command ===>                    Program Editor
```

CONCLUSION

As I attempted to show here, there are many things that can be done with the advanced features of SAS/AF. The in-depth validation capabilities on user entry fields supplied by the attribute settings, alleviate the majority of coding needed under the screen. Of course there can always be some non-standard requirements for a screen that necessitates additional customized validation. A program screen is capable of interpreting any of the # logic specific to AF or standard SAS macro code. Using a combination of these two techniques, it is possible to code just about anything on a program screen.

After working in AF over a period of time, we have found that programmers generally want the flexibility to, in a given situation, execute interactively or in batch, with or without a log file, and using a number of different options. SAS has proven flexible enough to adapt to each programmer rather than having the programmer adapt to SAS.

Utility procedures such as maintaining libraries and copying screens and catalogs are all straightforward and easy to follow. The only slight difficulties might be encountered when first creating the FORM screen and the batch job to print a screen. After that, it can simply be modified and re-used

The documentation for advanced features in AF is somewhat sparse, but the SAS Views which accompanies the SAS/AF course and "Technical Report: P-149" are very helpful resource documents.