

TOTALLY R.A.D WITH SAS® SCREEN CONTROL LANGUAGE

Robert W. Simmons, The Upjohn Company

ABSTRACT

This paper discusses Rapid Application Development with SAS® Screen Control Language within IBM's Virtual Machine/Enterprise Systems Architecture (VM/ESA). Specifically, it includes examples and source code for legends, menus, pop-up windows, single-row data entry screens, and multi-row data entry screens.

INTRODUCTION

With the introduction of SAS® 6.06 and 6.07, the SAS Institute provides numerous changes and enhancements to SAS/AF® software. Among the most significant, SAS® Screen Control Language (SCL) gives the analyst the power and flexibility to develop complex, customized applications with relative ease and speed. Coupled with other SAS Institute products such as SAS/SHARE® software for concurrent update access to SAS data libraries, SAS/ACCESS® software for direct access to native databases, and the SAS® SQL Procedure for manipulating tables and views; SAS/AF® and SCL allows the analyst to become totally R.A.D.

INVOKING THE APPLICATION

In the VM/ESA environment, users invoke most applications by entering a single command. At The Upjohn Company, we use REXX execs to invoke SAS® applications. Below, I've included the source code from a REXX exec called SAMPLE1 EXEC A1:

```
SAMPLE1 EXEC A1:
01 /* REXX EXEC that invokes a SAS application. */
02 'vmfclear'
03 'saslink 6.07'
04 'sas (autoexec=SAMPLE1)'
05 'vmfclear'
06 exit
```

Description of important lines:

- 03 Link to current release of SAS.
- 04 Invoke SAMPLE1 SAS A1.

You can modify this EXEC to suit your needs, but line 4 is the key line. It invokes SAS® and immediately executes the code contained in SAMPLE1 SAS A1, shown below:

```
SAMPLE1 SAS A1:
01 libname SAMPDRV server=SAMPLE;
02 libname SAMPRO 'SAMPRO *';
03 dm 'af=SAMPLE.CAT.MAIN.PROGRAM';
04 dm 'bye';
```

Description of important lines:

- 01 Define server data library.
- 02 Define read/only data and catalog library.

- 03 Invoke MAIN SCL program.
- 04 Exit SAS Display Manager.

SAMPLE1 SAS A1 shows how to invoke a SAS/AF® application from within the SAS® Display Manager. This is necessary if the application submits Base SAS® code. If your application does not submit Base SAS® code, the following REXX exec and SAS® program are more efficient by bypassing the SAS® Display Manager:

```
SAMPLE2 EXEC A1:
01 /* Invoke without the SAS Display Manager. */
02 'vmfclear'
03 'saslink 6.07'
04 'sas SAMPLE2'
05 'vmfclear'
06 exit
```

Description of important lines:

- 04 Invoke SAMPLE2 SAS A1.

```
SAMPLE2 SAS A1:
01 libname SAMPDRV server=SAMPLE;
02 libname SAMPRO 'SAMPRO *';
03 proc display c=SAMPLE.CAT.MAIN.PROGRAM;
```

Description of important lines:

- 03 Invoke MAIN SCL program using the DISPLAY procedure.

THE LEGEND

Most applications use program function keys to perform standard tasks such as scrolling or exiting a screen. In addition, applications commonly display directions or messages somewhere on the screen. I chose to standardize these common attributes through the use of a SCL legend. I defined the legend to always appear at the top of the screen containing the PF key definitions, directions, and messages for the particular task at hand. Customized macros allow the analyst to easily change the information contained in the legend at any time. An example legend appears below:

PF Keys: PF3=End PF7=Up PF8=Down
Directions: Fill in data fields and press ENTER.
Messages: Name and Date required.

SCREEN

The SCL commands that define the above legend follow:

```
01 INIT:
02 call wregion(1,1,5,80,'');
03 call legend('','','pink');
04
05 %keys('PF3=End PF7=Up PF8=Down');
06 %dir('Fill in data fields and press ENTER.');
```

```
09 return;
```

Description of important lines:

- 02 Define size and location of legend.
- 03 Display legend box.
- 05 Display keys.
- 06 Display directions.
- 07 Display message.
- 08 Place new contents of legend over old.

Only the MAIN SCL program must execute the WREGION and LEGEND commands. All subsequent programs need only execute lines 5-8. The %KEYS, %DIR, and %MSG macros follow. They use the PUTLEGEND command to display the information contained in the TEXT parameter.

```
01 %macro KEYS(TEXT,REFRESH);
02 do;
03   call putlegend(1,' PF Keys: '
04   ||&TEXT,'cyan');
05   %if &REFRESH=R %then
06     %str(call legend('','','pink'));
07   end;
08 %mend;
09
10 %macro DIR(TEXT,REFRESH);
11 do;
12   call putlegend(2,
13   'Directions: '||&TEXT,'yellow');
14   %if &REFRESH=R %then %str(call
15   legend('','','pink'));
16   end;
17 %mend;
18
19 %macro MSG(TEXT,REFRESH);
20 do;
21   call putlegend(3,
22   ' Messages: '||&TEXT,'red');
23   %if &REFRESH=R %then %str(call
24   legend('','','pink'));
25   MSGFLAG=1;
26   end;
27 %mend;
```

All three macros have an optional REFRESH parameter which allows you to change the contents of the legend while the SCL program is processing. For example, you might want to display a message while the SCL program submits and processes Base SAS® code to create a report:

```
01 MAIN:
02 %msg('Generating report, please wait...',R);
03 submit continue status;
04 data step...
05 proc sort...
06 proc print...
07 run;
08 end submit;
09 %msg('Report finished.');
```

Note in the INIT section, the PUSHLEGEND command replaces the contents of the legend (set by the calling program), with the new contents. Before returning to the calling program, use the POPLEGEND command to restore the previous contents:

```
01 TERM:
02 call poplegend();
03 return;
```

MENUS

At Upjohn, we experimented with many types of menus. We tested MENU entries using PF keys, PROGRAM entries using the push button attribute, and PROGRAM

entries using choice groups. None of these gave us the flexibility we needed in a fully functional menu. After further investigation, we discovered that the INITIAL attribute and MODIFIED statement were exactly what we were looking for. The following screen definition contains three menu functions:

```
Build: DISPLAY MAIN.PROGRAM (E)
Command --->
01 %FUNC1
02
03 %FUNC2
04
05 %FUNC3
06
.
```

Use the attribute screen to define each function. For example, you would define FUNC1 as follows:

```
Build: ATTR MAIN.PROGRAM (E)
Command --->
Use the scroll commands or functions keys to review...
Field name: FUNC1 Frame: 1 Row: 1 Col: 3 Length: 50
Alias: FUNC1 Choice group: Pad:
Type: CHAR Protect: YES NO INITIAL
Format: Just: LEFT RIGHT CENTER NONE
Inform:
Error color: RE attr: REVERSE Help: .....
List:
Initial: View Data
Replace:
Options: CAPS CURSOR REQUIRED AUTOSKIP NOPROMPT
```

The source code for this sample menu follows:

```
01 INIT:
02 call wdef(6,1,32,80);
03 call pushlegend();
04 %keys('PF3=End');
05 %dir('Tab cursor to function and press ENTER.');
```

```
06 %msg('');
07 call setcr('STAY','RETURN','MODIFY');
08 return;
09
10 MAIN:
11 if _STATUS_='E' then return;
12 %msg('');
13
14 if modified(FUNC1) then
15   call display('SAMPLE.CAT.VIEW');
16 if modified(FUNC2) then
17   call display('SAMPLE.CAT.EDIT');
18 if modified(FUNC3) then
19   call display('SAMPLE.CAT.PRINT');
20 return;
21
22 TERM:
23 call poplegend();
24 return;
```

Description of important lines:

- 01 The INIT section executes only once when the program is invoked.
- 02 The WDEF command re-sizes the menu to appear just below the legend (defined earlier).
- 03 Replace previous legend with new legend.
- 04 Initialize the PF key definitions in the new legend.
- 05 Initialize the directions in the new legend.
- 06 Initialize the message in the new legend.
- 07 The SETCR command defines the cursor attributes. STAY indicates that the cursor should not move after pressing the ENTER key. RETURN means that a carriage return will pass control to the application whether or not the field was modified. MODIFY means

that a carriage return on a field is considered a modification of the field.

- 10 The MAIN section, in this case, executes every time the user presses the ENTER key or modifies a field.
- 11 Avoid executing the MAIN section if the user is exiting the program.
- 12 Clear the message.
- 14 If the user places the cursor on FUNC1 and presses ENTER, then call the VIEW program.
- 16 If the user places the cursor on FUNC2 and presses ENTER, then call the EDIT program.
- 18 If the user places the cursor on FUNC3 and presses ENTER, then call the PRINT program.

- 22 The TERM section only executes when the program ends.
- 23 Display the previous contents of the legend.

At run time, the menu should look like this:

PF Keys: PF3=End Directions: Tab cursor to function and press ENTER. Messages:
View Data Edit Data Print Data

POP-UP WINDOWS

To provide the look and feel of a PC-based environment, you can use PROGRAM entries to design pop-up windows. You can define the windows to be any size and to appear anywhere on the screen. Pop-up windows can contain a fixed number of fields or can make use of extended tables to have a scroll area. As a standard, I chose to always display pop-up windows in the upper-right corner of the current menu or data entry screen. The example below shows a pop-up window, with a fixed number of fields, that displays after selecting the 'Edit Data' option from the main menu. In this case, I defined the fields just like the menu fields using the INITIAL option. All the user has to do is place the cursor on an option and press ENTER. Notice that the legend has changed to reflect what the user must do.

PF Keys: PF3=End Directions: Tab cursor to edit function and press ENTER. Messages:	Edit Employee Data Edit Inventory Data Edit Payroll Data
View Data Edit Data Print Data	

After selecting 'Edit Employee Data', the user sees the next pop-up window:

PF Keys: PF3=End Directions: Tab cursor to edit function and press ENTER. Messages:

View Data Edit Data Print Data	Update Employee Data Add New Employee
--------------------------------------	--

Let's say that the user now selects the 'Update Employee Data' option. This pop-up window could call another pop-up window to prompt for the employee's name:

PF Keys: PF3=End Directions: Enter any part of name and press ENTER. Messages:	Name: simmons
View Data Edit Data Print Data	

After entering part of the employee's name, another pop-up window would display a list of matches. This type of list, displayed below, uses an extended table and allows the user to scroll through the names.

PF Keys: PF3=End PF7=Up PF8=Down Directions: Tab cursor to employee and press ENTER. Messages:	Name: FITZSIMMONS, CHARLES E. FITZSIMMONS, MICHAEL T. SIMMONS, ALAN J. SIMMONS, DOROTHY L. SIMMONS, BENNET M. SIMMONS, CURTIS T. SIMMONS, DAYTON M.
View Data Edit Data Print Data	

The screen definition, general attributes, and source code for the extended table pop-up window follows:

BUILD: DISPLAY NAME.PROGRAM (E) Command ----> 01 Name: &NAME 02 .

BUILD: GATTR NAME.PROGRAM (E) Command ----> Window Name: Start row: 6 col: 41 Number of rows: 5 cols: 40 Banner: COMMAND SELECT NONE General Help: Keys: Lookup data set: Command menu: Prompt char: ? System Options: NO EXIT RESIDENT EXTENDED TABLE Parent Name: Libref: " Type: Catalog: " Window Type: STANDARD DIALOG DIALOG BOX LIST HELP Scroll Bars: HORIZONTAL VERTICAL

```

01 INIT:
02 call pushlegend();
03 %keys('PF3=Cancel');
04 %dir("Enter any part of name and press ENTER.");
05 %msg('');
06 NAME='';
07 call setcr('HOME','RETURN','MODIFY');
08 return;
09
10 MAIN:

```

```

11 if _STATUS_ = 'E' then return;
12 %msg('');
13 if NAME = '' then
14   call display('NAMELIST.PROGRAM', NAME);
15 else %msg('Name required.');
```

Description of important lines:

06 Initialize the NAME field.

07 Define the cursor attributes.

13 If the user entered a name then call the NAMELIST program passing the value of name.

15 Otherwise, display a message telling the user that NAME is required.

The screen definition, general attributes, and source code for the pop-up window that lists employee names follow:

```

-BUILD: DISPLAY NAMELIST.PROGRAM (E)
Command ---->

01 ***
02 %NAME
.
```

```

-BUILD: GATTR NAMELIST.PROGRAM (E)
Command ---->

Window Name:
Start row: 6      col: 56
Number of rows: 19 cols: 32 Banner: COMMAND SELECT NONE

General Help: .....
Keys: .....
Lookup data set: .....
Command menu: .....
Prompt char: ?
System Options: NO EXIT RESIDENT EXTENDED TABLE

Parent Name: ..... Libref: *
Type: ..... Catalog: *

Window Type: STANDARD DIALOG DIALOG BOX LIST HELP
Scroll Bars: HORIZONTAL VERTICAL
```

```

01 entry FINDNAME $ 30;
02
03 INIT:
04 %msg("Retrieving employees...", R);
05 %msg('');
06 call pushlegend();
07 %keys('PF3=Cancel PF7=Up PF8=Down');
08 %dir("Tab cursor to employee and press ENTER.");
09 %msg('');
10 SSN='';
12 E=open("SAMPLE.EMPLOYEE(where=||
13   "(NAME ? '||FINDNAME||')", 'I');
14 call set(E);
15 call setcr('STAY', 'RETURN', 'MODIFY');
16 call setrow(0, 1, 'N', 'Y');
17 return;
18
19 GETROW:
20 if fetchobs(E, _CURROW_) = -1 then call endtable();
21 return;
22
23 PUTROW:
24 RC=unselect(_CURROW_);
25 RC=fetchobs(E, _CURROW_);
26 call display('EDITEMP.PROGRAM', NAME, SSN);
27 return;
28
29 TERM:
30 call poplegend();
31 call close(E);
32 return;
```

Description of important lines:

01 The ENTRY statement defines parameters that are passed from and to the calling program.

04 Immediately display a message in the legend telling the user that the program is fetching data.

05 Clear the message after fetching the data.

12 Open and subset the SAMPLE.EMPLOYEE dataset using the WHERE option.

14 Link the dataset variable NAME to the screen variable NAME.

16 Define the extended table to have a dynamic number of rows (0), allow only one selection (1), and do not move selected item to top of list (N).

19 The GETROW section executes every time the screen needs to be refreshed. It reads information from the dataset into the extended table.

20 The FETCHOBS function reads and displays one row at a time until the extended table fills up the screen or until the last row is read. CALL ENDTABLE closes and dynamic extended table.

23 The PUTROW section executes once for each row in the extended table that is modified or selected.

24 The UNSELECT function unselects the current selected row so that PUTROW won't continue to execute for that row.

25 Fetch the value of the SSN field for the current row.

26 In this example, call the EDITEMP program and pass the values contained in the current row's NAME and SSN fields.

31 Close the employee file.

Single-Row Data Entry Screens

Historically, most technologies and even SAS® software only supported single-row data entry screens. With SAS® 5.18 you could use the FSEDIT Procedure to add, delete, or update rows in a SAS dataset. You could even customize the screen and implement a range of field validation attributes. However, you could not perform cross-field or cross-table validation. SCL has changed all that. Now you can supplement the FSEDIT data entry screen with SCL code. This gives you the power and flexibility to perform cross-field and cross-table validation, implement table look-ups, execute calculations, and much more. Better yet, if you still don't like some of the limitations of FSEDIT, you can customize your own single-row data entry screen from scratch. In the previous section, I used pop-up windows to search for and select an employee. Continuing with this example, the following single-row data entry screen displays after selecting a particular employee. The social security number is the unique key for the file.

```

PF Keys: PF3=End PF5=Delete
Directions: Enter employee information in fields below.
Messages:
```

```

Name: _____ SSN: _____
Title: _____ Hired: _____
Terminated: _____

Unit: _____ Building: _____ Floor: _____ Mailstop: _____
Ext.: _____ Status: _____ (A=Act. I=Inac. T=Term. D=Dead)

Birthdate: ____/____/____ Sex: ____ Race: ____
```

The following source code shows how to update information from the employee file. For the purpose of brevity, it only

contains one example each of a single-field, a cross-field, and a cross table validation; and a selection list.

```

01 entry NAME $ 30 SSN $ 11;
02
03 INIT:
04 call pushlegend();
05 %keys('PF3=End');
06 %dir('Enter employee information in fields' ||
07   below.'');
08 %msg('');
09 E=open("SAMPLE.EMPLOYEE{where=(SSN='*'||
10   SSN||'')}","U");
11 call set(E);
12 RC=fetch(E);
13 control always;
14 return;
15
16 MAIN:
17 if _STATUS_='E' then return;
18 %msg('');
19
20 /* Example of cross-table validation. */
21 U=open("SAMPLE.UNITS{where=(UNIT='*'||UNIT||'')}");
22 RC=fetch(U);
23 call close(U);
24 if RC=0 then do;
25   %msg('Invalid Unit number. ');
26   cursor UNIT;
27   return;
28 end;
29
30 /* Example of single-field validation. */
31 if STATUS not in('A','I','T','D') then do;
32   %msg('Invalid Status code. ');
33   cursor STATUS;
34   return;
35 end;
36
37 /* Example of cross-field validation. */
38 if STATUS in('A','I') and TERMDATE=_BLANK_
39 then do;
40   %msg('No termination date allowed' ||
41     ' for Status of A or I. ');
42   TERMDATE=_BLANK_;
43   cursor STATUS;
44   return;
45 end;
46 if STATUS in('T','D') and TERMDATE=_BLANK_
47 then do;
48   %msg('Termination date required. ');
49   cursor TERMDATE;
50   return;
51 end;
52
53 /* Example of selection list. */
54 if RACE not in('CC','AA','HS','AS','OT')
55 then do;
56   call wregion(6,57,7,24,'');
57   LISTITEM=showlist('CC=Caucasian',
58     'AA=African American',
59     'HS=Hispanic',
60     'AS=Asian',
61     'OT=Other');
62   RACE=substr(LISTITEM,1,2);
63 end;
64
65 if update(E)=0 then
66   %msg('Employee information updated. ');
67 else %msg('Update error, contact programmer. ');
68 return;
69
70 TERM:
71 call poplegend();
72 call close(E);
73 return;

```

Description of important lines:

- 01 Pass the NAME and SSN values to and from the program.
- 09 Open the employee file for update and perform a where clause by the SSN.
- 11 Link all the screen variables to the variables in the dataset.

- 12 Fetch the requested employee information.
- 13 Always execute the MAIN section when the ENTER key is pressed.

- 20 /* Example of cross-table validation. */
- 21 Open the UNITS file for input and perform a where clause by the given Unit number.
- 22 Try to fetch the row for the given unit number.
- 23 Close the UNITS file.
- 24 If the given unit does not exist, then display an error message, place the cursor on the UNIT field, and return control to the screen.

- 30 /* Example of single-field validation. */
- 31 If STATUS is not one of A, I, T, or T then display an error message, place the cursor on the STATUS field, and return control to the screen.

- 37 /* Example of cross-field validation. */
- 38 If STATUS is Active or Inactive and the termination date is not blank, then display an error message and return control to the screen.

- 46 If STATUS is Terminated or Dead and the Termination date is blank, then display an error message and return control to the screen.

- 53 /* Example of selection list. */
- 54 If RACE is not in the list of valid values, then display a selection list.
- 56 Set the size of the selection list before displaying.
- 57 Display the list and set LISTITEM to the value that the user selects.
- 62 Set RACE to the first two characters of the LISTITEM.

- 65 After validating all variables on the screen, attempt to update the employee information. If the update is successful, then display an appropriate message. If not, display an appropriate error message. You can also check for a variety of return codes to find out exactly why the update did not work.

Multiple-Row Data Entry Screens

In some cases, a multiple-row data entry screen is more desirable. Take for example, an inventory where you want to keep a log of all credit and debit transactions for each inventory item. After searching for the particular inventory item of interest, you would like to view and scroll through multiple transactions at a time. In addition, you would also like to insert, update, and delete data. This type of data entry screen is one of the most complex SCL program that you will encounter. But I have made it simple through the use of reusable code. Below is a multiple-row data entry screen. The dashed line separates the top 'static' portion of the screen from the bottom 'scrollable' portion of the screen.

FF Keys: PF1-Top PF2-Bottom PF3-End PF7=Up PF8=Down
 Directions: Tab cursor to edit function and press ENTER.
 Messages:

```

Edit Function> INSERT UPDATE DELETE
Selected Inventory Item...
Stock#: 294-100M Catalog#: P404 Lot#: L41878
Supplier: BEROL MIRADO, INC.
Description: LEAD PENCIL (2.5) - YELLOW
Current Qty: 8.432 Reorder Qty: 2.000

Credit Debit Acct Date Unit Name
-----
10.000      0900 04/23/1992 5058 MANN, CHARLES E.
          20 0400 04/27/1992 9922 SIMMONS, ROBERT W.
           5 0400 04/28/1992 7240 RINDLER, SUSAN B.
          10 0400 04/30/1992 3821 KALMAN, ERICA K.
          100 0650 05/13/1992 2938 SMITH, ADAM L.
          250 0650 05/20/1992 2938 SMITH, ADAM L.
  
```

In order for the screen to scroll, you must select the **EXTENDED TABLE** option from the GATR screen. The actual screen definition for the above extended table follows:

```

Build: DISPLAY EDITINV.PROGRAM (E)
Edit Function> INSERT UPDATE DELETE
Selected Inventory Item...
Stock#: &STOCKNUM Catalog#: &CATNUM Lot#: &LOTNUM
Supplier: &SUPPLIER
Description: &DESCRIPT
Current Qty: &CURQTY Reorder Qty: &REQQTY

Credit Debit Acct Date Unit Name
-----
&CREDIT &DEBIT &ACCT &DATE &UNI &NAME
  
```

Here is the source code followed by the reusable code:

```

01 entry STOCKNUM $ 10 CATNUM $ 15 LOTNUM $ 15;
02
03 INIT:
04 $msg('Retrieving Inventory Item...',R);
05 $let SCROLL=CREDIT DEBIT ACCT DATE UNIT SSN;
06 $let CURSOR=CREDIT;
07
08 E=open('SAMPLE.EMPLOYEE','I');
09 call set(E);
10 I=open('SAMPLE.INVENTOR(wher=(STOCKNUM='''
11 ||STOCKNUM|''' and CATNUM='''
12 CATNUM|''' and LOTNUM='''||LOTNUM|''')''','U');
13 call set(I);
14 $inc INIT;
15 return;
16
17 VARINIT:
18 CREDIT=-;
19 DEBIT=-;
20 ACCT=-;
21 DATE=-;
22 UNIT=-;
23 SSN='';
24 NAME='';
25 return;
26
27 REFRESH:
28 $inc REFRESH;
29 link CALC;
30 return;
31
32 CALC:
33 submit continue SQL;
34 select sum(CREDIT)-sub(DEBIT) as CURQTY
35 from SAMPLE.INVENTOR;
36 endsbmit;
37 return;
38
39 INSERT:
40 $inc INSERT;
41 return;
42
43 UPDATE:
44 $inc UPDATE;
45 return;
46
47 DELETE:
48 $inc DELETE;
49 return;
50
51 GETROW:
  
```

```

52 $inc GETROW;
53 if _CURROW-_INS^=0 then call setkey(E);
54 return;
55
56 PUTROW:
57 $inc PUTROW;
58 return;
59
60 UPDCHK:
61 /* update and insert validation checks */
62 UPDOK=1;
63 return;
64
65 DELCHK:
66 /* delete validation checks */
67 DELOK=1;
68 return;
69
70 TERM:
71 $inc TERM;
72 call close(I);
73 call close(E);
74 return;
  
```

Description of important lines:

- 01 The calling program passes the stock number, catalog number, and lot number for the inventory item to be edited.
- 05 The macro variable SCROLL will be referenced in other parts of the program. It should contain only the input variables for the dataset you are editing.
- 06 The macro variable CURSOR contains the name of the variable that you want the cursor to appear on when the user inserts, updates, or deletes.
- 08 Open the employee dataset for input.
- 09 Associate the employee variables with the SCL variables.
- 10 Open the inventory dataset for update. Only select those observations for the given stock number, catalog number, and lot number.
- 13 Associate the inventory variables with the SCL variables.
- 14 Include reusable code contained in INIT (see INIT reusable code).
- 17 The VARINIT section initializes input variables.
- 27 The REFRESH section reinitializes the scroll area when the user is canceling from an insert, update, or delete.
- 28 Include reusable code contained in REFRESH (see REFRESH reusable code).
- 29 Perform the calculation statements in the CALC section.
- 32 The CALC section contains, if necessary, any calculations that must be performed.
- 33 In this example, an SQL submit statement is used to calculate the current quantity of the inventory item.
- 39 The INSERT section executes when the user selects the INSERT function.
- 40 Include reusable code contained in INSERT (see INSERT reusable code).
- 43 The UPDATE section executes when the user selects the UPDATE function.
- 44 Include reusable code contained in UPDATE (see UPDATE reusable code).

- 47 The DELETE section executes when the user selects the DELETE function.
- 48 Include reusable code contained in DELETE (see DELETE reusable code).
- 51 The GETROW section reads and displays observations from the inventory dataset.
- 52 Include reusable code contained in GETROW (see GETROW reusable code).
- 53 If not end-of-data or end-of-screen then get name from employee dataset using SSN as the key.
- 56 The PUTROW section gets screen input and updates the inventory data.
- 57 Include reusable code contained in PUTROW (see PUTROW reusable code).
- 60 The UPDCHK section checks all insert and update information.
- 61 You can add single-field, cross-field, or cross-table validation. You can also display selection lists. In this example, you would want to display a selection list of names that satisfy the users input from the NAME field.
- 62 Setting UPDOK to 1, signifies that everything validated OK.
- 65 The DELCHK section checks deletion validation information.
- 66 You can add validation checks to make sure the user isn't improperly deleting something.
- 67 Setting DELOK to 1, signifies that it is OK to delete.
- 70 The TERM section executes when the user ends the program or when the user cancels out of insert, update, or delete mode.
- 71 Include reusable code contained in TERM (see TERM reusable code).
- 72 Close the inventory file.
- 73 Close the employee file.

Listed below are the source code and description for all the reusable code sections.

(INIT reusable code)

```
01 call wdef(5,1,32,80);
02 msg(' ');
03 call pushlegend();
04 keys('PF1=Top PF2=Bottom PF3=End PF7=Up' ||
05 'PF8=Down');
06 dir('Tab to edit function and press ENTER. ');
07 msg(' ');
08 call setcr('STAY','RETURN','MODIFY');
09 control label;
10 call setrow(0,99,'N','Y');
11 INS=0;
12 UPD=0;
13 DEL=0;
14 link REFRESH;
```

Description of important lines:

- 08 The SETCR function sets the attributes of the cursor such that when the cursor is placed on a field and the ENTER key is pressed, the program considers the field modified.
- 09 The LABEL option of the CONTROL statement tells the program to execute a labeled section that has the same name as the field that is modified.

- 10 The SETROW function defines the attributes of the extended table.
- 11 Initialize the insert, update, and delete flags.
- 14 Link to the REFRESH section and initialize the attributes of the screen.

(REFRESH reusable code)

```
01 RC=field('unprotect','INSERT UPDATE DELETE');
02 RC=field('color green','INSERT UPDATE DELETE');
03 if UPD then cursor UPDATE;
04 else if DEL then cursor DELETE;
05 else cursor INSERT;
06 keys('PF1=Top PF2=Bottom PF3=End PF7=Up' ||
07 'PF8=Down');
08 call setfkey(fkeyname(1),'top');
09 call setfkey(fkeyname(2),'bottom');
10 call setfkey(fkeyname(3),'end');
11 call setfkey(fkeyname(7),'up');
12 call setfkey(fkeyname(8),'down');
13 INS=0;
14 UPD=0;
15 DEL=0;
16 INSON=0;
17 UPDOK=0;
18 DELOK=0;
```

Description of important lines:

- 01 Unprotect the INSERT, UPDATE, and DELETE fields so the user can tab to them.
- 02 Set the color of the INSERT, UPDATE, and DELETE fields to green.
- 03 Place the cursor on the last chosen option.
- 08 Reset the PFKey settings.
- 13 Reset the insert, update, and delete flags.

(INSERT reusable code)

```
01 msg('Enter data and press ENTER (or press' ||
02 'PF3 to cancel). ');
03 RC=field('color red','INSERT');
04 RC=field('protect','INSERT UPDATE DELETE');
05 keys('PF3=Cancel');
06 call setfkey(fkeyname(3),'cancel');
07 call setfkey(fkeyname(1),' ');
08 call setfkey(fkeyname(2),' ');
09 call setfkey(fkeyname(7),' ');
10 call setfkey(fkeyname(8),' ');
11 INS=1;
12 call execcmd('top');
13 cursor &CURSOR;
```

Description of important lines:

- 03 Turn INSERT field red.
- 04 Protect INSERT, UPDATE, and DELETE fields.
- 06 Set PF3 to CANCEL.
- 07 Turn other PFkeys off.
- 11 Set insert mode flag on.
- 12 Scroll to top of data.
- 13 Place cursor on first data field.

(UPDATE reusable code)

```
01 RC=rewind(1);
02 if fetch(1)^=0 then do;
03 msg('No data to update. ');
04 return;
05 end;
06 msg('Update data and press ENTER (or press' ||
07 'PF3 to cancel). ');
08 RC=field('color red','UPDATE');
09 RC=field('protect','INSERT UPDATE DELETE');
10 keys('PF3=Cancel');
11 call setfkey(fkeyname(3),'cancel');
12 UPD=1;
13 cursor &CURSOR;
```

Description of important lines:

- 01 Position dataset pointer at beginning of data in where clause.
- 02 If no data found, then display 'no data to update' message.
- 08 Turn UPDATE field red.
- 09 Protect the INSERT, UPDATE, and DELETE fields.
- 11 Set PF3 to CANCEL.
- 12 Turn the UPDATE flag on.
- 13 Place the cursor on the first data field.

(DELETE reusable code)

```
01 RC=rewind(I);
02 if fetch(I)^=0 then do;
03   %msg('No data to delete.');
```

04 return;

```
05 end;
06 %msg('Select data to delete (or press PF3 to cancel).');
```

07 to cancel.);

```
08 RC=field('color red', 'DELETE');
```

09 RC=field('protect', 'INSERT UPDATE DELETE');

```
10 %keys('PF3=Cancel');
```

11 call setfkey(fkeyname(3), 'cancel');

```
12 DEL=1;
```

13 cursor <CURSOR;

Description of important lines:

- 01 Position dataset pointer at beginning of data in where clause.
- 02 If no data found, then display 'no data to delete' message.
- 08 Turn DELETE field red.
- 09 Protect the INSERT, UPDATE, and DELETE fields.
- 11 Set PF3 to CANCEL.
- 12 Turn the DELETE flag on.
- 13 Place the cursor on the first data field.

(GETROW reusable code)

```
01 If INS & CURROW=1 then do;
02   RC=field('color green rev', "&SCROLL");
03   RC=field('unprotect', "&SCROLL");
04   if ^INSON then link VARINIT;
05   INSON=1;
06 end;
07 else if UPD then do;
08   RC=field('color green rev', "&SCROLL");
09   RC=field('unprotect', "&SCROLL");
10 end;
11 else if DEL then do;
12   RC=field('color green', "&CURSOR");
13   RC=field('unprotect', "&CURSOR");
14 end;
15 else do;
16   RC=field('color pink', "&SCROLL");
17   RC=field('protect', "&SCROLL");
18 end;
19 if fetchobs(I, CURROW-INS)--1 then
20   call endtable();
```

Description of important lines:

- 01 If insert mode is on and GETROW is processing the first row then...
- 02 ...turn all input fields to green, reverse video.
- 03 ...unprotect all input fields.
- 04 ...if first input, then initialize input variables.
- 05 ...set first input flag.
- 07 If update mode is on then...
- 08 ...turn all input fields to green, reverse video.
- 09 ...unprotect all input fields.
- 11 If delete mode is on then...
- 12 ...only turn the first input field to green.
- 13 ...only unprotect the first input field.
- 15 If not in insert, update, or delete mode...

- 16 ...turn all input fields to pink.
- 17 ...protect all input fields.
- 19 If at end of data or end of screen, then closed the dynamic table.

(PUTROW reusable code)

```
01 RC=unselect ( CURROW_);
02 if INS then do;
03   link UPDCHK;
```

04 if UPDOK then do;

```
05   RC=append(I);
06   if RC=0 then do;
07     link INSERT;
08     link CALC;
```

09 %msg('Data inserted. Continue '||

10 ' inserting (or press PF3 to cancel).');

```
11 end;
12 else if RC=630058 then do;
13   %msg('Data not inserted. Data already '||
14   ' exists.');
```

15 end;

```
16 else %msg('Data not inserted, call '||
17   ' programmer.');
```

18 UPDOK=0;

```
19 INSON=0;
```

20 end;

```
21 end;
22 else RC=fetchobs(I, CURROW_);
23 if UPD then do;
24   link UPDCHK;
```

25 if UPDOK then do;

```
26   RC=update(I);
27   if RC=0 then %msg('Data updated. Continue '||
28   ' updating (or press PF3 to cancel).');
```

29 else if RC=630058 then %msg('Data not '||

30 ' updated. Data already exists.');

31 else %msg('Data not updated, call '||

32 ' programmer.');

```
33 UPDOK=0;
```

34 end;

```
35 end;
36 else if DEL then do;
37   link DELCHK;
```

38 if DELOK then do;

```
39   if delobs(I)=0 then do;
40     %msg('Data deleted. Continue '||
41     ' deleting (or press PF3 to cancel).');
```

42 RC=rewind(I);

```
43 if fetch(I)^=0 then call
44   execcmd('cancel');
```

45 end;

```
46 else %msg('Data not deleted, call '||
47   ' programmer.');
```

48 DELOK=0;

```
49 end;
```

50 end;

Description of important lines:

- 01 Unselect the selected row.
- 02 If insert mode is on then...
- 03 ...link to UPDCHK to validate the input fields.
- 04 If the validation is OK then...
- 05 ...add new record to dataset.
- 06 If the insert was OK then...
- 07 ...prepare for inserting the next record.
- 08 ...perform calculations to reflect new data.
- 12 If the data already exists then display message to user.
- 16 If the insert was not OK then display message to user.
- 18 Reset insert flags.
- 22 If not in insert mode, then set the dataset pointer to the appropriate record.
- 23 If update mode is on then...
- 24 ...link to UPDCHK to validate the updated fields.
- 25 If the validation is OK then...

```

26 ...update the data.
27 If the update was OK then display message.
29 If the data already exists, then display message.
31 If the update was not OK then display message.
32 Reset update flag.

36 If delete mode is on then...
37 ...linkto DELCHK to see if the data is OK to delete.
38 If OK to delete then...
39 ...delete the record.
42 If delete was OK then check to see if there is any
data left.
43 If no data left, then cancel delete mode.
45 If delete was not OK, then display message.
47 Reset delete flag.

```

(TERM reusable code)

```

01 if STATUS = 'C' then do;
02   STATUS = 'R';
03   If INS then %msg('Insert mode canceled.');
```

Description of important lines:

```

01 If user pressed PF3 to cancel insert, update, or delete
mode, then...
02 ...set _STATUS_ to return to program.
03 Display appropriate cancel message.
08 Initialize screen.
09 Return to program.
11 If user is exiting program, then remove the last legend.

```

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.