Dataset Manipulation with Screen Control Language (SCL)

Marge Scerbo, University of Maryland at Baltimore

Introduction

SAS-AF Screen Control Language (SCL) can serve a multitude of purposes from simple menu creation to complex data analysis. SCL is not only powerful; it is large and complex and at times quite difficult to use. Although some syntax crosses between base SAS and SCL, it is important to remember that SCL is a separate entity and that in some areas, its actions are different from base SAS code. Although datasets can be edited using SAS/FSP, this paper will discuss the area of dataset manipulation using SAS/AF Screen Control Language. A basic understanding of Version 6.06 AF screens and SCL is assumed.

Sample Dataset

The dataset Paper.Sample, used in the examples in this paper, will contain information about universities offering Informatics programs. The variables names and formats in the dataset are:

UNIV	$	50
SCHOOL	$	50
PROGRAM	$	50
ADDRESS	$	50
CITY	$	15
STATE	$	2
COUNTRY	$	15
ZIP	$	10
CONTACT	$	25
AFFIL	$	10

Contents

Note that all the variables in this dataset are character. There are certain SCL functions that are either numeric or character type specific. Many of these type-specific functions will have corresponding functions for the opposite type. Because character values are more complex to deal with in SCL, only the character-based functions will be noted.

Sample Screens

The basic data entry screen used for adding or updating observations in the sample dataset would be:

```
University:  &univ_____________________
School:      &school___________________
Program:     &program__________________
Address:     &address__________________
City:        &city_________
State:       &S            Zip:  &zip_____
Country:     &country______
Contact:     &contact______________
Affiliation: &affil____
```

Screen #1

The variables on this screen are self-descriptive except for AFFIL which is the affiliation of the program. This is a descriptive term with the list of 10 options, including Medical, Nursing, Graduate, stored in List format. This variable's list attribute will contain =AFFILIA, requiring a choice from that list. Required fields include UNIV, SCHOOL, PROGRAM, CITY, and AFFIL. The screen variable &S has been given an alias of STATE, matching the variable in the dataset.

The menu program attached to Screen #1 will allow for three editing options: Add, Change and Delete. On this first menu the University name will be required for Choices 1, 2 or 3:

```
      Main Edit Menu

         1 -- Add
         2 -- Change
         3 -- Delete
         9 -- Exit

      Enter Your Choice:   &

      University:  &univ_______________
```

Screen #2

On this screen the single & field attribute alias is set to CHOICE and is a required field. CHOICE has been set to allow input of 1, 2, 3 or 9 in the attribute list. These screens (#1 and #2) will serve as the two entry screens for the system described in the paper.

Background Notes

When first encountering AF SCL in an attempt to manipulate the data stored in a SAS dataset, the initial inclination is to use SUBMIT blocks to perform many of the tasks. But since most systems in the end are user-based systems where speed and efficiency play a key role, SUBMIT blocks might not prove satisfactory. Within a SUBMIT block, basic SAS code is stored and executed. This requires additional compilation and memory use. Rather, SCL provides a group of functions for manipulation of SAS datasets. These functions can accomplish many of the tasks undertaken in a DATA step. SCL code can be compiled and then tested using TESTAF within PROC BUILD screens; SUBMIT blocks require use of the AF command from the editor command line of Display Manager or in a batch program external to Display Manager. Clearly in the development of a system when much of the testing can be done within the menu or program entries, programmer efficiency is increased.

When first approaching dataset manipulation, it is important to understand the different data structures involved. There are three types of SCL variables: window (screen), nonwindow and system. If a program has an associated application screen, as in the screens shown above, this screen will contain fields where variable values can be displayed. SCL communicates with the application window through these variables. Nonwindow variables are not associated with a field in a screen but are merely used within that individual program. System variables provide special types of information which can be used as checks within the program. For instance, _STATUS_ is a system variable which reports the current status of the application. This system variable can be reset within the program thus allowing for control within the application. These variable types do not include SAS dataset variables; they remain a separate entity.

When running SCL, data storage areas are also important. There may be two separate types of data buffers in use during an SCL session. When a dataset has been opened, dataset variable values can be moved with SCL code into a data buffer called the Dataset Data Vector or DDV. A separate DDV will be associated with each open dataset. Screen control variable values are stored in a data buffer called the SCL Data Vector or SDV. Ordinarily there is no communication between these two buffers so SCL code is needed to bridge the vectors when manipulating a dataset from window applications.

Dataset Manipulation

Before reading from or writing to the dataset in any way, SAS requires that the dataset be opened. The SCL OPEN function returns a unique dataset identifier which is then used when performing many other operations. The code is as follows:

```
dataset-id = OPEN(dataset-name,mode);
```

Dataset-id is a numeric variable which stores this unique identifier. When the identifier is equal to 0 or below, the dataset was not successfully opened. If the dataset is a hard-coded value, this value will appear in quotes, while if the dataset name is input into a screen variable, then the variable need not be in quotes. The mode is the mode in which the dataset is opened. Valid modes used for dataset manipulation are I for input mode (read only), which is the default, and U for update (read-write). Although many datasets can be opened at the same time, it is best to open the minimum number of datasets, since each opened dataset requires memory and each open dataset will access its own DDV. Since several datasets can be opened at the same time, make sure that each dataset has been assigned a unique identifier. The dataset need not be opened with each new program. The identifier can be passed from one program to the next. Throughout the following examples, dataset identifier will be stored in the variable name, DSID.

Once the dataset has been opened, some of the normal dataset functions can take place. For instance, the CUROBS function will return the relative observation number:

```
obsno = CUROBS(dsid);
```

When operations are complete on a dataset, the CLOSE function will remove the dataset from memory and discontinue access. The syntax of the function is:

```
rc = CLOSE(dsid);
```

where rc is a system result code. If the dataset has been successfully closed, this code should be set to 0. Any errors will cause the code to be equal to some other number and coding should be written accordingly.

Assuming the OPEN operation was successful, before any dataset manipulation occurs, the data must be moved between the DDV to the SDV. The easiest method to use is the SET command. By SETting a dataset once in a program, data is automatically moved between these two vectors. Dataset variables are matched by name and type to the SCL variables. Although a dataset can remain open from one program to another, each program must SET the dataset. While there are other functions which serve a similar purpose, the SET function is the easiest and most comprehensive. As a rule the SET command should immediately follow the OPEN command. The syntax of the SET command is:

```
CALL SET(dsid);
```

Parameter Passing

When the user chooses to add an observation, the first option on the Main Menu, a university name will be entered and passed to the ADD program. This field value can be passed in two ways, through CALL DISPLAY and ENTRY or CALL SYMPUT and SYMGET.

CALL DISPLAY is used to move from one AF window to another. This command can also pass parameters to the CALLed program. The parameters may include a variable name or a static value. At the receiving end, the called program will include an ENTRY statement prior to the INIT section. This statement must include an argument list which matches the CALLing arguments, including a variable name, a $ if a character variable is to be received, and/or the length of the value. The actual variable names do not have to be the same. In the sample case, the variable passed is named UNIV by both the CALLing and receiving programs.

```
CALLing program (MainMenu.Program):
   CALL DISPLAY('Add.Program',univ);
Receiving Program (Add.Program):
   ENTRY univ $;
```

To use the macro function SYMPUT in the term section of the CALLing program, the value input into the screen variable UNIV will be stored in a macro variable. These variables can have the same name or different names. In this case, the names will be the same so the code would be:

```
CALL SYMPUT('UNIV',univ);
```

This will create a macro variable with the same variable name and value as the screen variable. The Menu program will move to the ADD program by using the command:

```
CALL DISPLAY('Add.Program');
```

In the ADD Program INIT section, a SYMGET function will read this macro variable:

```
univ = SYMGET('univ');
```

Since this function has been placed in the INIT section, when the ADD screen appears, the University name entered on the menu will appear.

The CALL DISPLAY/ENTRY option is preferable when passing from one screen to another. CALL SYMPUT/SYMGET is useful when the same variable is passed to several screens.

Choice 1: Adding an Observation

To add an observation to an opened dataset, use the APPEND command. This function adds an observation using the values stored in the DDV. The code is:

```
rc = APPEND(dsid);
```

Again, the result-code will be set to 0 if the append was successful. Remember that if the dataset variable names are different from the screen variable names, reset the dataset variables to the screen values.

Using these statements and assuming that the university name and the dataset-id are passed into the program from MainMenu.Program with a CALL DISPLAY function, bare-bones code for the Add.Program module would be:

```
ENTRY univ $ dsid 8;
INIT:
  *Initialize screen variables;
RETURN;

MAIN:
    *Validate screen values;
    *If error, return;
RETURN;

TERM:
    CALL SET(dsid);
    rc = APPEND(dsid);
RETURN;
```

Add Option

Selecting an Observation

Before discussing deletion of an observation, it is important to study the selection of a specific observation to delete. The delete (DELOBS) function merely deletes the last read observation. Rarely would deletion occur on 'any' observation in the dataset. Most often, the observation to be deleted must be selected from the dataset. Therefore, the specific observation must be chosen. If the observation number of the record to be deleted is known, then a FETCHOBS statement can be used. FETCHOBS reads the observation specified by the relative observation number from a SAS dataset into the Dataset Data Vector. Again, prior to using this command, the dataset must be opened and set. The code might be:

```
rc = FETCHOBS(dsid,obs-number);
```

If the operation is performed without error, the result-code is set to 0. If end-of-file is reached, then the code is -1, and if an error occurred, the code is not equal to 0 or -1.

The FETCH function reads the next non-deleted observation from the SAS dataset into the DDV. Place this function in a loop that reads some number of observations and deletes them or that reads to end of file and deletes all those observations. For example:

```
dsid = OPEN('Paper.Sample','U');
CALL SET(dsid);
rc = FETCHOBS(dsid,10);
rc = DELOBS(dsid);
DO WHILE (FETCH(dsid) ^= -1);
   rc = DELOBS(dsid);
END;
rc = CLOSE(dsid);
```

In the above example the dataset will be opened and observation number 10 will be fetched and deleted. Then all observations until end of file, as denoted by a return code of -1, will be deleted.

Since the observation number is not always known, there must be ways of identifying an observation by variable values. If the observation contains a unique value for a variable, then two functions can be used to identify that observation. The VARNUM function returns the number of a variable within the dataset. So the code:

```
varnum = VARNUM(dsid,variablename);
```

will return the number of the variable. If the variable name cannot be found, the varnum will be set to 0.

The VARNUM function can be used in association with the LOCATEC function. This function is a character-type specific function; LOCATEN is the numeric-type counterpart. LOCATEC will search the dataset for an observation containing a value of a particular variable. This function requires several arguments; the dataset id, the variable number and the value must all be specified. Additional arguments including sort information can be included. The code specifications are:

```
rc = LOCATEC(dsid,varnum,value);
```

If a match is found, data from that observation is moved to the DDV. The result code will be set to greater than 0 if the observation has been found.

In the sample dataset, to delete the observation where the university name is 'UNIVERSITY OF MARIELAND', the code might be:

```
dsid = OPEN('Paper.Sample','U');
CALL SET(dsid);
vnum = VARNUM(dsid,'univ');
rc = LOCATEC(dsid,vnum,
   'UNIVERSITY OF MARIELAND');
rc = DELOBS(dsid);
rc = CLOSE(dsid);
```

Functions can be collapsed to one line as in:

```
rc = LOCATEC(dsid,VARNUM(dsid,'univ'),
      'UNIVERSITY OF MARIELAND');
```

To restate the functions which allow for selection of an observation from a dataset which has been opened and set previously:

FETCHOBS -- selects by observation number
FETCH -- selects next available observation
VARNUM -- returns number of a variable
LOCATEC -- locates an observation by variable value

Choice 2: Deleting an Observation

To delete an observation, the steps are similar to the APPEND function, substituting a DELOBS statement for the APPEND statement. The syntax of the DELOBS function is:

```
rc = DELOBS(dsid);
```

This sample dataset does not have unique identifiers in that one university may in fact have several schools offering informatics programs. Therefore, all observations which include the chosen university name must be deleted, so the Delete.Program might appear as:

```
ENTRY univ $ dsid 8;
INIT:
RETURN;

MAIN:
RETURN;

TERM:
   CALL SET(dsid);
   vnum = VARNUM(dsid,'univ');
   DO WHILE
     (LOCATEC(dsid,vnum,univ) > 0);
       rc = DELOBS(dsid);
   END;
RETURN;
```

Delete Option

Advanced Selection Criteria

As stated above, the dataset Paper.Sample in actuality could contain entries which had duplicate University names. Searching for a particular entry was made more efficient by storing all university values as upper case, thus eliminating the need for case-sensitive search. Still, searches posed a sticky problem.

Since all entries with a particular university name were to be deleted, the VARNUM/LOCATEC functions dealt with this situation satisfactorily. This was not the case when a particular observation was to be updated. Two matches were necessary to choose a specific observation for update: university and school.

The WHERE clause proves to be most useful in these situations. A WHERE clause is a set of conditions which observations within the dataset must meet to be processed. The result code will be set to 0 if the WHERE operation was successful. The basic syntax of a WHERE clause is as follows:

```
rc = WHERE(dsid,clause1,...);
```

and the basic syntax to clear any existing WHERE clauses is:

```
rc = WHERE(dsid);
```

Additional WHERE clauses imposed without an AND or ALSO statement will overwrite any previous conditions. The basic WHERE syntax seems simple enough, but when the variables are character string variables, special care must be taken. Screen variables values should not be enclosed in quotes, but WHERE clauses must be enclosed in quotes, so the syntax to subset the sample dataset by a value stored in the UNIV variable would appear as:

rc = WHERE(dsid,'UNIV = "'||*univ*||'"');

where UNIV refers to the dataset variable and *univ* refers to the screen variable.

Add to this clause another set of conditions:

```
rc = WHERE(dsid,'UNIV = "'||univ||'" AND
     SCHOOL = "'||school||'"');
```

It is important to note that careful testing of the actual operation of each WHERE clause is imperative. At times the clause might compile yet be in error either syntactically or logically. Check the result code of this option often during the testing phase.

When a WHERE clause is in effect, take care in using regular dataset functions. For example the NOBS function will normally return the number of observations in the dataset. This function will continue to return the total number of undeleted observations in the dataset, not the number available after the WHERE condition has been imposed.

The ATTRN function does allow for a useful option for dealing with WHERE conditions. The ANY attribute of this function will return a numeric value which is set to -1 if no observations or variables exist in the dataset, 0 if the dataset has no observations, and 1 if the dataset has both observations and variables. To use this function:

```
attrib-value = ATTRN(dsid,'ANY');
```

In the sample system, an intermediate program was developed which allowed for input of a school name to be checked for existence in the dataset. If the option to add new university/school was chosen, the school could not already exist in the dataset. Conversely, if the option chosen were to update an existing observation, the school should be found in the dataset. The module's screen and program are:

```
Enter the School: &school_______________

For University:  &univ_______________
```

Screen #3

```
ENTRY univ flag $ dsid 8;
INIT:
RETURN;

MAIN:
RETURN;

TERM:
   CALL SET(dsid);
   rc = WHERE(dsid,'UNIV = "'||univ||'"
      AND SCHOOL = "'||school||'"');
   rc1 = ATTRN(dsid,'ANY');

   IF flag = 'A' THEN DO;
      IF rc1 = 1 THEN DO;
         *error processing and return;
      END;
      ELSE DO;
        CALL DISPLAY('Add.Program'
          univ,school,dsid);
      END;
   END;

   IF flag = 'C' THEN DO;
      IF rc1 = 0 THEN DO;
         *error processing and return;
      END;
      ELSE DO;
        CALL DISPLAY('Update.Program'
          univ,school,dsid);
      END;
   END;

RETURN;
```

School Choice

As shown, the School.Program will receive 3 parameters from the Main Program: university name (univ), a flag designating the choice of Add or Change (flag) and the dataset id (dsid). If all validity checks prove correct as tested by the ATTRN function using the ANY attribute, the program will pass three parameters (univ, school, dsid) to the appropriate application windows.

Choice 3: Updating an Observation

The UPDATE function writes values from the DDV to the current observation of the dataset. The basic syntax of the UPDATE function is:

```
rc = UPDATE(dsid);
```

As with other operations, a result code of 0 denotes a successful update.

Updating an observation requires that the observation in question can be located. In the sample dataset which has been opened and set, the code to update an observation where the university name is 'UNIVERSITY OF MARYLAND' and the city should be changed from 'Batlimore' to 'Baltimore' might be:

```
vnum = VARNUM(dsid,'univ');
rc = LOCATEC(DSID,vnum,
   'UNIVERSITY OF MARYLAND');
city = 'Baltimore';
rc = UPDATE(dsid);
```

It is important to track the value of the variables. When the observation is read, the value of CITY is 'Batlimore'. Since the correct value is read from a screen into a screen variable of the same name, the value should be stored in a nonwindow variable. After the LOCATEC function occurs, CITY will be reset to this hold variable value. Different variable names can be used for the screen and dataset variables, but then in the INIT section, the screen variable must be set to the value of the dataset variable. The core code for the Update.Program is:

```
ENTRY univ school $ dsid 8;
INIT:
   CALL SET(dsid);
   rc = LOCATEC(dsid,VARNUM(dsid,
         'univ'),univ);
RETURN;

MAIN:
   *Data Validation checks;
   *If error, return;
RETURN;

TERM;
   rc = UPDATE(dsid);
RETURN;
```

Update Option

The LOCATEC function is used in the INIT section to select the observation and pass the values to the screen variables. Since the dataset has already been subset to the correct university and school in the CALLing program, School.Program, the correct observation will be displayed.

Main.Program Code

The MainMenu module appears as:

```
INIT:
   dsid = OPEN('PAPER.SAMPLE','U');
   CALL SET(dsid);
RETURN;

MAIN:
 IF choice ^= 9 THEN DO;
   IF univ = _BLANK_ THEN DO;
      *error messages and return;
   END;

   *undo WHERE clauses;
   rc = WHERE (dsid);
   vnum = VARNUM(dsid,'univ');
   rc = LOCATEC(dsid,vnum,univ);

   IF choice > 1 AND rc <= 0 THEN DO;
     *error--university isn't found;
   END;

   IF choice = 1 THEN flag = 'A';
   IF choice = 2 THEN flag = 'C';

  SELECT(choice);
   WHEN(1) CALL DISPLAY
      ('School.Program',univ,flag,dsid);
   WHEN(2) CALL DISPLAY
      ('School.Program',univ,flag,dsid);
   WHEN(3) CALL DISPLAY
      ('Delete.Program',univ,dsid);
   OTHERWISE RETURN;
  END;

 END;
RETURN;

TERM:
   IF choice = 9 THEN DO;
        rc = CLOSE(dsid);
        RETURN;
   END;
RETURN;
```

Main Menu Code

In the sample system, the university name must exist for the Change or Delete options. Validity checking will occur when the user has input the choice and the university name. As discussed earlier, the School.Program module will contain further validity checks. The SELECT statement is used to CALL menu items. This statement allows for ease of adding additional options to the menu. The TERM section of this program is used to finally CLOSE the dataset and exit the system. As is apparent in this code, the Add module would need amendment in order to receive the school value.

Data Vectors Revisited

With several SCL dataset functions defined, further description of the vectors should be useful. Again, the two data buffers involved are the DDV (Dataset Data Vector) and the SDV (SCL Data Vector). This diagram shows the movement of data values between these vectors and into screens or open datasets:

```
             A--->        C--->        D--->
Open Dataset    DDV          SDV          Screen
             <---B        <---C        <---E
```

After the dataset has been properly opened, the following actions (A through E) are accomplished by:

A	FETCH, FETCHOBS, LOCATEC
B	APPEND, UPDATE
C	SET
D	REFRESH, RETURN
E	ENTER, END, CANCEL

In studying this diagram, it is apparent that no single simple manipulation function occurs with a single command. If one command is missing or misplaced, the system does not work.

Debugging Hints

Return codes are used throughout Screen Control Language code. In most cases, codes of 0, >0 and -1 are returned. There are times errors occur and further definition of the code is needed. This is ever the case when using the WHERE clause which will seemingly compile without error but will create inaccurate or invalid subsetting of the dataset. Use the system function, SYSMSG, to return a text to the return code specified. Use this function in conjunction with a PUT statement, and the error message will appear in the message window. For example:

```
    rc = WHERE(dsid,'UNIV = "'||univ||'"');
    message = SYSMSG();
    PUT 'message' message;
```

This piece of code is extremely useful in debugging SCL code. Remember to remove this code before implementation.

Conclusion

This paper still only scratches the surface of dataset manipulation with screen control language. None of the above programs are complete. They lack validity and error checks, key responses, etc. But these programs do begin to acquaint you with the power of SCL. Screen Control Language has options for almost any instance, but locating the correct syntax can be an arduous road. In addition, errors that are not apparent during compile may occur during execution. Tracking these mystical beasts may cause nightmares. But in the midst of your frustration, remember it can be done!

SAS^{TM} is a registered trademark of SAS Institute, Inc., Cary, NC

References: SAS Screen Control Language, Release 6.06 Edition, SAS Institute, Inc., Cary, NC.

If you have questions or comments:
Marge Scerbo
University of Maryland at Baltimore
Information Services
100 N. Greene St., Room 211
Baltimore, MD 21201

Phone: (410)328-8424

Email: MSCERBO@UMAB.UMD.EDU