Paper 68-26

Let's Go to School and Discover the OOH in SCL

Michael A. Mace, SPS Software Services, Inc., Canton, OH

ABSTRACT

Call it SAS® Screen Control Language or the SAS® Component Language, SCL is the programming language which helps to make SAS/AF® and SAS/FSP® applications so powerful. This paper will provide an introduction to the many features of SCL, starting with its elements and rules and structure (doesn't that sound like we are in school again) and moving on to its use in SAS/AF® and SAS/FSP@ applications. There are an amazing number of things that SCL can do – and we will not try to cover them all; this paper is simply a primer, hopefully taking away some of the mystery, and providing a good basis for getting started, further study, and exploration.
Now, let the class begin …

Lesson I:
What are the Elements of SCL ?

SCL and the base SAS® language share many of the same elements, such as: statements, functions, operators, variables, macro capability, and CALL routines. SCL provides additional elements to manage the interaction between the user and the application, manipulate SAS data sets and external files, and create routines and programs that can be called from other SCL, SAS/AF and SAS/FSP applications. In the SAS system, version 8, many SCL functions have been ported to the DATA step – that is a topic for another day.

Like the base SAS system, SCL statements can be executable, declarative, and comments. Executable statements, which must be in a labeled section, are complied into intermediate code and perform some action when the SCL program is executed; these include: assignment statements, IF-THEN-ELSE, and CURSOR. Declarative statements, which can appear anywhere in the SCL program, provide information to the SCL compiler but are not executed; examples are: ARRAY, LENGTH, and ENTRY. Comments, which can and should appear anywhere in an SCL program, provide information to the programmer. Remember that could be you six months hence or someone else who is trying to decipher what the code is suppose to do.

Standard DATA step expressions are supported identically in SCL except for the IN operator. In SCL, if no match is found, the IN operator returns 0; else the index of the matched element is returned. In a DATA step, IN returns 1 or 0 to indicate if a match is found or not, respectively.

SCL supports all DATA step functions, except LAGn and DIFn. Functions are used to manipulate argument values, such as REVERSE; or perform some action and return a value indicating success or failure; that value is called a return code variable. Unless the return code is needed for other processing, it is more efficient to nest the function within statements. Also the same return code variable can be used multiple times for different function calls.

When using macros and macro variables in SCL, you must remember that they are resolved when the SCL program is compiled, not during execution. To delay the resolution until during execution, use the SYMPUT, SYMPUTN, SYMGET and SYMGETN functions.

To construct valid SCL program statements:
- ❖ End each statement with a semi-colon
- ❖ Begin statements in any column (i.e. use intents for clarity)
- ❖ Separate words with blanks or operators
- ❖ Statements may continue on to succeeding lines as long as no keyword is split
- ❖ Place arguments for SCL functions and CALL routines in parentheses and separate multiple arguments with commas
- ❖ Arguments can be literal values or the names of variables that contain the desired values

Lesson II:
SCL Variables

SCL variables have the following attributes:
name, type(numeric character), and length.
These attributes are determined by the category of the variable and the application.

There are three categories of SCL variables:
system – generated by SCL, provide application status information
window – linked to fields in the application window, pass values between the window and the SCL program. Each field in the application window has a corresponding variable in the SCL porgram.
nonwindow – defined in the SCL program, hold temporary values that do need to be displayed to the user

Several system variables are:
BLANK - character – indicates whether a window variable contains a value or sets the value to blank (SAS/AF only)
MSG - character – text displayed in the window's message line
STATUS - character – status of program execution

> You can check for the following values:
> C – the user issued the CANCEL command
> E – the user issued the END command
> You can set _STATUS_ to the following vaules:
> R – to resume execution. This allows you to control whether the user can exit the application or the current observation.
> H – to terminate the current window without further user input
> Note that setting the value of _STATUS_ does not cause an immediate action because the value is queried only after the SCL program returns control to the application

For window variables in SAS/AF applications:
The name and type of each window variable are determined by the PROGRAM entry fields. Note that while SAS/AF has several special types (ACTION, INPUT, etc), the corresponding SCL window variables will be either character or numeric. Numeric window variables always have a length of eight (8). Character window variables have a length equal to the width of the corresponding field in the application window.

For window variables in SAS/FSP applications:
There is a one-to-one correspondence between the variables in the SAS data set (displayed and unwanted) and the SCL window variables. The name, type, length, format, and informat of each window variable is determined by the corresponding SAS data set variable.

Nonwindow variables can be explicitly defined in a LENGTH statement or the name and type are determined by the first assignment statement in which the variable appears. Their names can be up to 32 characters long. They have no informat or format.
Numeric nonwindow variables always have a length of eight (8) and character nonwindow variables have a default maximum length of 200. Therefore it is more efficient to explicitly define a variable with a LENGTH statement.

As previously noted, variables can be grouped into arrays. Temporary arrays are defined to hold values without creating variables by adding the _TEMPORARY_ argument to the ARRAY statement, such as: array things(100) _temporary_ ;
Temporary array elements must be referenced using subscripting,
i.e. things(33), not things33.

Lesson III:
SCL Program Structure

SCL programs for SAS/AF PROGRAM entries and FSEDIT SCREEN applications execute in at least three phases or program blocks or sections: initialization, main processing, and termination or clean-up. The beginning of each section is designated by a reserved label, INIT:, MAIN:, and TERM: respectively, and each program block ends with a RETURN statement. Labeled sections determine when statements are executed. Executable statements are only executed if they are in a labeled section. Declarative statements can appear anywhere in the program.

SCL programs for SAS/AF programs perform tasks such as:
validating user input, calculating field values, providing messages, menus, and selection lists to users, reading and writing to SAS data sets and external files, and interfacing with other software including the SAS System.

An SCL program for a SAS/AF application would have this form:

INIT:
Statements to initialize the application:
- Initialize window variables
- Import values of macro variables
- Display initial messages
- Open SAS data sets and external files

RETURN ;

MAIN:
Statements to process user input
- Validate user input
- Calculate values for computed variables
- Invoke secondary windows
- Query and execute user commands
- Retrieve values from SAS data sets and external files

RETURN ;

TERM:
Statements to terminate the application
- Update and close SAS data sets and external files
- Export values of macro variables
- Submit statements to the SAS System for execution

RETURN ;

INIT executes only once, for each invocation of the PROGRAM entry, before the entry's window opens. For entries without a window, MAIN executes immediately after INIT completes. For entries with a window, MAIN executes each time the ENTER, RETURN, or any function key is pressed, provided that any modified fields contain valid values based on their attribute specifications. After MAIN completes, any computed variable values are displayed in the corresponding window fields. For entries without a window, TERM executes immediately after MAIN completes. For entries with a window, TERM executes when a user enters an END or CANCEL command. If any fields have been modified since the last execution of MAIN, MAIN executes again before TERM; and if any fields contain invalid values, control is returned to the application and TERM does not execute.

SCL programs for FSEDIT SCREEN entries perform tasks such as:
calculating field values, providing messages, validating user input against values of other fields, values of other SCL variables, and values in other SAS data sets or external files, and interfacing with other software including the SAS System.

An SCL program for an FSEDIT SCREEN entry would have this form:

FSEINIT: (OPTIONAL)
Statements to initialize the application:
- Initialize nonwindow variables
- Import values of macro variables
- Open other SAS data sets or external files
- Specify an initial message to be displayed when the first observation is displayed

RETURN ;

INIT:
- Statements to execute before displaying each observation:
- Manipulate values of the SAS data set before display
- Initialize computed fields for current observation
- Specify an initial message for each observation

RETURN ;

MAIN:
Statements to process user input:
- Validate user input
- Calculate values for computed variables
- Invoke secondary windows
- Query and execute user commands
- Retrieve values from SAS data sets and external files

RETURN ;

TERM:
Statements to execute before leaving each observation:
- Manipulate values of the SAS data set before updating
- Update other SAS data sets

RETURN ;

FSETERM: (OPTIONAL)
- Statements to terminate the application:
- Close other SAS data sets and external files
- Export values of macro variables

RETURN ;

FSEINIT executes only once, when the FSEDIT application is invoked. INIT executes before each observation is displayed and each time an observation is displayed, and also when the SAVE command is issued. Before INIT executes, FSEDIT reads the current observation from the data set and initializes SCL window variables with the values of the associated data set variables; thereby making those values available for processing. MAIN executes each time fields are modified and the ENTER key is pressed, provided that any modified fields satisfy the attributes specified in the FSEDIT Attributes or Names window. After MAIN completes, updated values of window variables are displayed in the corresponding fields. TERM executes when one or more fields have been modified, or the observation is new, and the user attempts to leave the observation by: attempting to move to another observation (i.e. FORWARD or BACKWARD), search for another observation (i.e. FIND, SEARCH, LOCATE, WHERE), or enters one of these commands: ADD, DUP, SAVE, END, or CANCEL. If any fields have been modified since the last execution of MAIN, MAIN executes again before TERM; and if any fields contain invalid values, control is returned to the application and TERM does not execute. For new observations only, those added using ADD or DUP, variables with the REQUIRED field attribute are checked; if any are missing they are flagged in error and values must be entered before TERM executes. FSETERM executes when the user issues the END command. If any fields have been modified on the current observation, MAIN and TERM are executed before FSETERM. After FSETERM completes, the FSEDIT window closes and control returns to the process that invoked the entry.

Lesson IV: Program Labels and Controlling Program Flow

In addition to the reserved labels described in the previous lesson, you may use other labels to designate the destinations for LINK or GOTO statements. These nonreserved labels must have unique, valid SAS names and be followed by a colon. A RETURN statement is used to delineate the end of these labeled sections. Note that the code within these sections is only executed when called.

For SAS/AF applications with a window and SAS/FSP applications, it is possible to use window variable blocks of code. Each section is labeled with the same name as the window element/variable and the code is executed when the corresponding window element/variable is modified, provided the value entered is valid according to the informats and attributes specified.

The CONTROL statement, with one or more options, can be used to control the flow of a program. It is usually placed in the INIT or FSEINIT section.
CONTROL LABEL – must be present for window variable blocks to execute.
CONTROL ENTER – will force the MAIN section to execute, even if no fields were modified.
CONTROL ERROR – will force the MAIN section to execute, even if one of more field values is in error based on the informats or attributes specified. Note that using this option puts the onus of field validation on the developer.
CONTROL ALWAYS and CONTROL ALLCMDS – combine the effects of ENTER and ERROR, and in addition, allow the interception of global or application-specific commands. Control is passed to the MAIN section before further action is taken.
CONTROL TERM – will force the TERM section to execute whenever a command is entered to leave an observation, even if no fields were modified.

The _STATUS_ system variable mentioned earlier may be used to:

- stop the execution of the SCL program, after which control returns to the invoking program or window, or
- resume execution of the SAS/AF application or remain on the current SAS/FSEDIT observation.

Lesson V: Accessing SAS Data Sets

Of course the first step to accessing a SAS data set, is to assign a libref with the LIBNAME statement or function. Next, the data set must be opened. The SAS/FSP products automatically open the data set being displayed; for other applications, the OPEN function is used.

```
dsid = OPEN( 'fileref.dsname', 'mode' )
```

where dsid is a unique, numeric data set identifier that is used to reference the data set throughout the rest of the program; and mode is the access control level assigned to the data set, indicating Read-Only or Update. In addition, the ***data set data vector*** (DDV) is created. It is important to note that there also exists an ***SCL data vector*** (SDV), which contains the variable values manipulated by the SCL program. The variable values of the data set and the program transfer back and forth between the DDV and SDV via SCL functions, such as FETCH, GETVARC, GETVARN, PUTVARC, PUTVARN, and UPDATE. If the data set and SCL variables have the same names and types, they can all be linked automatically using the SET statement.

It is possible to limit the number of observations accessed by using the WHERE= data set option in the OPEN statement, thereby creating a permanent WHERE clause – permanent for as long as the data set is open. Alternatively, the WHERE function can apply a temporary WHERE clause.

```
rc = WHERE( dsid, 'Namevar :=' || letter ) ;
```

This WHERE condition can be augmented by use of the ALSO keyword.

```
rc = WHERE( dsid, 'ALSO agevar > 65' ) ;
```

The last applied WHERE clause can be removed by the UNDO keyword.

```
rc = WHERE( dsid, 'UNDO' ) ;
```

To undo all temporary WHERE clauses:

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

New observations are added programatically with the APPEND function. Observations are deleted using the DELOBS function.

Using SCL functions, SAS data sets can be created, copied, sorted, indexed, deleted, data set and variable attributes can be accessed.

When you are done with the data set, you should close it:

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

Lesson VI:
Accessing External Files

SCL programs can also access external files and directories of files. The first step here is to assign a fileref using the FILENAME statement or function. Next, the file is made available to the program using the FOPEN function,; this creates a unique, numeric file identifier and a temporary storage buffer called the *file data buffer* (FDB). Data is transferred between the file and the FDB with FREAD; and then to the SDV using FGET. The data can be handled as an entire record or separate values. The FPUT function writes data from the SDV to the FDB; the FWRITE or FAPPEND functions then update the external file. FCLOSE and DCLOSE are used to close files and directories, respectively.

Lesson VII:
SCL Lists

SCL lists are ordered collections of data. What makes them different than arrays and so powerful is that the collection can be a mixture of types: numeric, character, and even other lists, the size of a list is dynamic, able to grow or shrink at run time, and lists are resident in memory only. The items of the list can be assigned names and referenced by name or index. Like data sets and external files, SCL lists are accessed via a unique, numeric identifier.

```
        alist = MAKELIST() ;
```

Entries can be inserted or replaced using the SETITEMN, SETNITEMC, SETITEMC, SETNITEMN, SETITEML, and SETNITEML functions. List items are retrieved using these functions: GETITEMC, GETNITEMC, GETITEMN, GETNITEMN, GETITEML, and GETNITEML. Items are deleted from a list using DELITEM and DELNITEM.

Note that items can be referenced from the end of the list simply by using a negative index.

Using INSERTC, INSERTN, INSERTL, POPC, POPN, and POPL functions, a list can function as a stack or queue.

Lists can be sorted, reversed, rotated, searched, and copied. The CLEARLIST function clears all values of a list, leaving the list with a length of zero. The SAVELIST function will save a list in a SAS catalog entry (SLIST) or an external file. You delete a list with the DELLIST function.

LESSON VIII: Performance, Productivity, and Maintenance

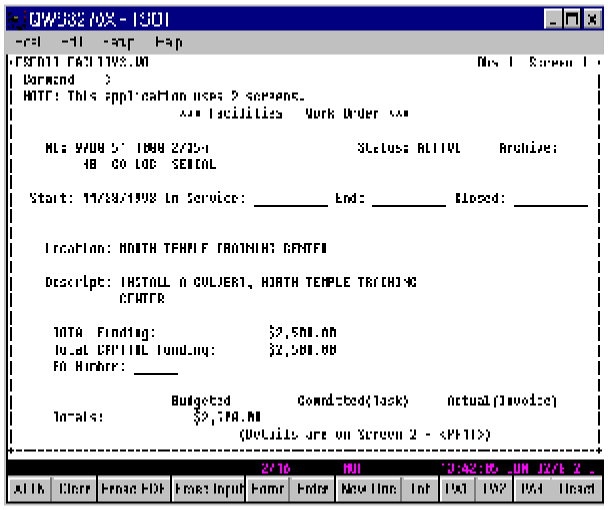
Please see Chapter 13, Optimizing Application Performance in the *SAS® Screen Control Language: Reference, Second Edition* manual for a list of efficiency techniques. One technique that I regularly use when working with SCL is to store the SCL program as a separate entity, either as a text file or an SCL entry in a SAS catalog; and then simply code a %INCLUDE statement in the Source of the SAS/AF PROGRAM or SAS/FSP SCREEN entry. This allows me to easily examine and modify the SCL program without locking out the users from using the application. When the changes are complete, and the application is available for maintenance, I can allocate the application's SAS catalog with exclusive access, edit the Source, compile and save, and be done in less than a minute.

Lesson IX:
Examples

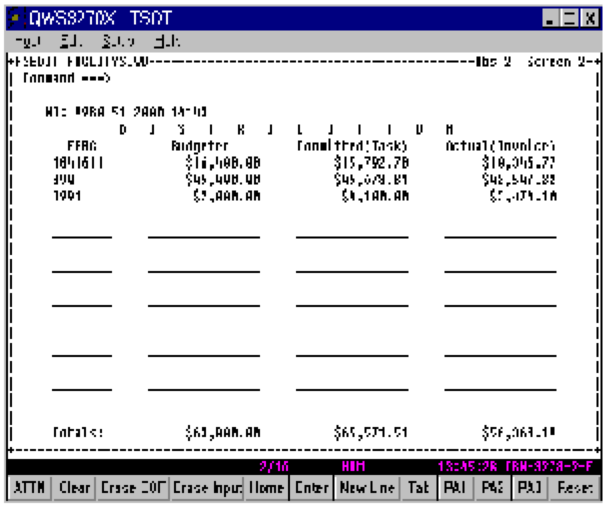
Okay class; now that we have covered the basics of the SAS Screen Control Language (SAS Component Language), let's examine three SCL programs from a SAS/FSP application. I have inserted comments to highlight topics covered in our lessons as well as several other interesting techniques. Documentation, as we all know, is good programming practice.

This specific application was written with version 6.09 of the SAS system on an MVS operating system, but would work equally well on another operating system. These FSEDIT sessions were initiated using PROC FSEDIT, however they could have been started from a SAS/AF program with CALL FSEDIT statements. In fact, you will see examples of this statement as we 'jump' from one screen to another.

Work Order


QWS327XX - TSOT
File Edit Setup Help
2/16 NUM
ATTN Clear Erase EOF Erase Input Home Enter New Line Tab PA1 PA2 PA3 Reset

```
+FSEDIT FACILITYS.WO                                         Obs 1  Screen 1 +
| Command ===>
| NOTE: This application uses 2 screens.
|                    *** Facilities   Work Order ***
|
|   WO: 9708 51 1998 27154                   Status: ACTIVE    Archive:
|        48  CO LOC  SERIAL
|
|  Start: 11/28/1998 In Service: ________ End: ________ Closed: ________
|
|   Location: NORTH TEMPLE TRAINING CENTER
|
|   Descript: INSTALL A CULVERT, NORTH TEMPLE TRAINING
|             CENTER
|
|    DATA Funding:                  $2,500.00
|    Total CAPITAL Funding:         $2,500.00
|    PO Number: ______
|
|                  Budgeted        Committed(Task)     Actual(Invoice)
|    Totals:         $2,700.00
|                            (Details are on Screen 2 - <PF11>)
+--------------------------------------------------------------------------+
```


QWS327XX TSOT
File Edit Setup Help
2/16 NUM
ATTN Clear Erase EOF Erase Input Home Enter New Line Tab PA1 PA2 PA3 Reset

```
+FSEDIT FACILITYS.WO-------------------------------------Obs 2  Screen 2-+
| Command ===>
|
|   WO: 9708 51 2000 14703
|            D  O  L  L  A  R     A  M  O  U  N  T
|     FERC          Budgeted        Committed(Task)     Actual(Invoice)
|    1041611       $10,400.00       $15,792.70          $10,345.77
|    400           $45,400.00       $45,678.81          $42,547.82
|    1001           $7,200.00        $4,100.00           $5,474.18
|
|    ______     ____________     ____________     ____________
|
|    ______     ____________     ____________     ____________
|
|    ______     ____________     ____________     ____________
|
|    ______     ____________     ____________     ____________
|
|    ______     ____________     ____________     ____________
|
|    Totals:       $63,000.00       $65,571.51          $58,363.10
+-------------------------------------------------------------------------+
```

/* MSMS.SASPROG.PERM.SAS(SCLWO) */

/* it is good practice to declare the length of SCL variables */
/* response accepts the answer from the DELETE confirmation window – see the MAIN section */

```
LENGTH response $ 1
        invctmp1 invctmp2  8  ;
```

```
FSEINIT:   /* this section executes ONCE per session */
/* even though the data set is opened by FSEDIT, we need
this identifier to check for duplicates – see the MAIN section
*/
     dsiself = OPEN('faclitys.wo','i') ;

/* enable screen variable labels and
ensure that the MAIN section is always executed */
     CONTROL LABEL ALWAYS ;
RETURN ;


INIT:   /* this section executes for each observation */
/* arrays can be used for SCL and screen variables  */
    ARRAY woparts(4) $8 ('wohb','woco','woloc','woserial') ;
    ARRAY wo19x(15) $19 wo19_1-wo19_15 ;
    ARRAY woferc(15) $8  woferc1-woferc15 ;
    ARRAY commit(15) 8  commit1-commit15 ;
    ARRAY invoic(15) 8  invoic1-invoic15 ;
RETURN ;


/* the next three sections execute when the corresponding
screen variables are modified – because we used CONTROL
LABEL */
status:
/* determines the size of the next window opened */
    CALL WREGION(10,35,15,45,'') ;

/* if the user enters a '?' or a [SPACE] in a field, display a
pop-up list, created with SAS/AF® */
   IF status IN('?',' ')
   THEN status = LISTC('status.list','','Y',1) ;
RETURN ;

archive:
   IF archive NE ''
  THEN archive ='X' ;
RETURN ;

woco:
   CALL WREGION(10,35,15,45,'') ;
   IF woco IN('?',' ')
   THEN woco = LISTC('company.list','','Y',1) ;
RETURN ;


MAIN:
/* get the first word on the command line and upper case it */
   cmdword = WORD(1,'U') ;
/* this tests for the text or the function key definition */
   IF cmdword =: 'DEL'
   THEN DO ;       /* here is the confirmation displayed earlier */
                   response = 'N' ;
                   CALL WREGION(8,10,8,60,'') ;
/* Execute a SAS/AF program and accept the response  -
if 'Y', then immediately delete this observation */
                   CALL DISPLAY('confirm.program',response) ;
                   IF response = 'Y'
                   THEN CALL EXECCMDI('DELETE','NOEXEC') ;
               END ;

    ERROROFF _ALL_ ;
    IF WORD(1,'U') =: 'DUP' AND archive ='X'
    THEN DO ;
/* turn error condition on for all variables and effectively
pause execution */
                   ERRORON _ALL_ ;
/*  conditionally display a message to the user … */
                   _MSG_ ="An ARCHIVED observation
                                   cannot be duplicated" ;
               END ;
```

```
    ERROROFF wohb woco woloc woserial ;
    DO i = 1 TO DIM(woparts) ;    /* woparts is an array */

/* the FIELD function reports on the status of or performs an
action on a field */
          IF FIELD('MODIFIED',woparts(i))
          THEN DO ;

/* the parts of the work order are needed, as well as the whole
number, as well as the first eleven characters */
                   wo =
LEFT(COMPRESS(wohb||woco||woloc||woserial)) ;
                   wo11 = LEFT(COMPRESS(woco||woloc||woserial)) ;

/* this is one technique to perform a real-time check for
duplicate key field values – another method uses the
FETCHOBS function after the WHERE clause */

/* apply a where condition and test for any observations */
               rc = WHERE(dsiself,"wo11=' "||wo11||" ' ") ;
               IF ATTRN(dsiself,'ANY') GT 0
               THEN DO ;

                              ERRORON wohb woco woloc woserial ;
/* position the cursor on a specific variable */
                              CURSOR wohb ;
                              _MSG_ ="This is a duplicate WO number" ;
                         END ;
                    END ;   * FIELD('MODIFIED',woparts(i)) ;
    END ;   * i = 1 TO DIM(woparts) ;


    DO f = 1 TO 15 ;   /* for each of the ferc variables */
         IF INDEX(woferc(f),'?') > 0 THEN
            DO ;
/* open another data set for searchiing only – no updates */
                 dsiferc = OPEN('library.woferc','i') ;
                 CALL WREGION(3,15,20,65,'') ;

/* the data set which is the basis for a format is also used as
a lookup table - here the DATALISTC function displays the
start and label variables in a dialog box with a title of
"FERC", and closes the dialog box when one choice is made
*/
                 woferc(f)=
                      DATALISTC(dsiferc,'start label','FERC','Y',1) ;
                 rc= CLOSE(dsiferc) ;
            END ;
/* note how concatenation is used to get the variable name */
         ELSE IF FIELD('MODIFIED','woferc'||LEFT(PUT(f,2.)))
                      AND woferc(f) NE '' THEN
                   DO ;
/* determine if the frec number entered is valid */
                        dsitest= OPEN('library.woferc(WHERE=
                                             (start=" '||woferc(f)||' "))') ;
                        IF ATTRN(dsitest,'ANY') GT 0 THEN
                           DO ;
                              woferc(f)= LEFT(COMPRESS(woferc(f),'.')) ;
                              wo19x(f) = TRIM(LEFT(wo11||woferc(f))) ;
                           END ;
                        ELSE _MSG_="That FERC is not defined
                                          in the table" ;
/* at this point, the user could enter the word 'FERC' on the
command line in order to add a new ferc value - see code on
next page */
                      rc= CLOSE(dsitest) ;
                   END ;
```

/*an example of capturing an application-specific command*/
```
     IF cmdword EQ 'REFRESH'  AND woferc(f) NE ''
     THEN DO ;
           dsitask= OPEN('faclitys.taskinvc(WHERE=
                                    (wo19='"||wo19x(f)||"'))');
            IF ATTRN(dsitask,'ANY') GT 0 THEN
               DO ;
```
/* real-time summarization from the task/invoice data set */
/*the sum of committd is put in the array element */
```
                    rc= VARSTAT(dsitask,'committd','SUM',commit(f)) ;
```
/* the sum of total is put in an SCL variable */
```
                    rc= VARSTAT(dsitask,'total','SUM',invctmp1) ;
               END ;
            ELSE invctmp1 = 0 ;
            rc= CLOSE(dsitask) ;
```

/* another data set is opened and another sum is determined */
```
            dsilpocc= OPEN('faclitys.lpocc(WHERE=
                                   (wo19='"||wo19x(f)||"'))');
            IF ATTRN(dsilpocc,'ANY') GT 0
            THEN rc=VARSTAT(dsilpocc,'amount','SUM',invctmp2) ;
            ELSE invctmp2 = 0 ;
            rc= CLOSE(dsilpocc) ;
```

/* now a screen variable is updated with the partial sums */
```
            invoic(f) = SUM(invctmp1, invctmp2) ;
                 END ;
```

/* allow the user to jump to the specific task/invoice details -- - after typing ‘TSKINV’ on the command line and placing the cursor on a ferc field, CURFIELD function, and then pressing [ENTER] */
```
     IF cmdword EQ 'TSKINV'
         AND CURFLD() EQ 'woferc'||LEFT(PUT(f,2.))
     THEN DO ;
                    dsitest =OPEN('faclitys.taskinvc(WHERE=
                                 (wo19=" ' ||wo19x(f)|| ' "))');
                    IF ATTRN(dsitest,'ANY') GT 0
```
/* open the data set with a where condition and a specific, formatted screen, whose Source is another SCL program */
```
                    THEN CALL FSEDIT(
                                'faclitys.taskinvc(WHERE=
                                          (wo19=" ' ||wo19x(f)|| ' "))',
                                'affspscl.budscrns.taskinvc.screen') ;
                    ELSE _MSG_="There are no TASKs/INVOICEs
                                    with that WO and FERC" ;
                    rc= CLOSE(dsitest) ;
               END ;
   END ;   * FINALLY!, DO f = 1 TO 15 ;
```

/* opdate three screen variables */
```
   bdgtttl = SUM(OF budget1-budget15) ;
   cmmtttl = SUM(OF commit1-commit15) ;
   invcttl = SUM(OF invoic1-invoic15) ;
```

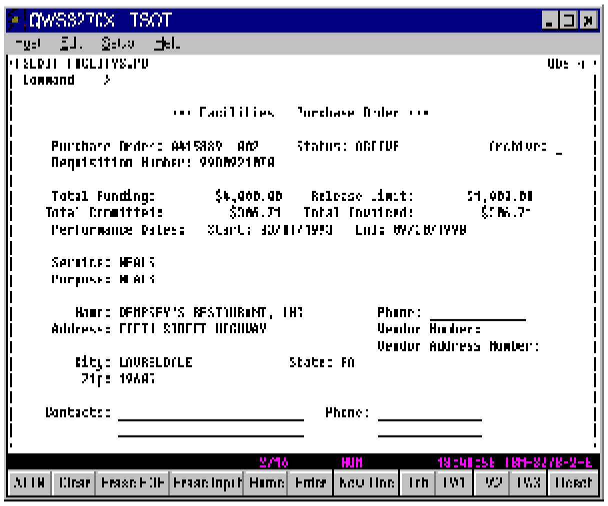
/* the following snippets, allow the user to jump to the other data sets with associated screens – BE CAREFUL! Each of these opens a distinct interface, from which the user can jump to another data set --- yes, even back to here. It can be very easy to get lost in the open windows */
```
    IF WORD(1,'U') EQ 'POX'
    THEN CALL FSEDIT('faclitys.po',
                          'affspscl.budscrns.po.screen') ;
    IF WORD(1,'U') EQ 'TSKINV'
    THEN DO ;
               dsitest =OPEN('faclitys.taskinvc(
                              WHERE=(wo11=" ' ||wo11|| ' "))') ;
               IF ATTRN(dsitest,'ANY') GT 0
               THEN CALL FSEDIT(
                    'faclitys.taskinvc(WHERE=(wo11='"||wo11||"'))',
                    'affspscl.budscrns.taskinvc.screen') ;
               ELSE _MSG_="There are no TASKs/INVOICEs
                              with that WO" ;
               rc= CLOSE(dsitest) ;
          END ;

   IF WORD(1,'U') EQ 'LPOCC'
  THEN DO ;
              dsitest= OPEN('faclitys.lpocc
                            (WHERE=(wo11='"||wo11||"'))') ;
              IF ATTRN(dsitest,'ANY') GT 0
              THEN CALL FSEDIT(
                     'faclitys.lpocc(WHERE=(wo11='"||wo11||"'))',
                     'affspscl.budscrns.lpocc.screen') ;
             ELSE _MSG_="There are no LPOCCs with that WO" ;
             rc= CLOSE(dsitest) ;
          END ;

   IF WORD(1,'U') EQ 'FERC'
   THEN  CALL FSVIEW('library.woferc','ADD') ;
```
/* note the use of FSVIEW and ADD mode */
```
RETURN ;
```
/* this is the end of the MAIN section --- finally */

```
TERM:
```
/* nothing needs to be done when leaveing an observation */
```
RETURN ;
```

/* this section executes when the FSEDIT session closes */
```
FSETERM:
  rc = CLOSE(dsiself) ;
RETURN ;
```

Purchase Order

/* A purchase order has no direct tie to a work order; however a task/invoice, which is tied to a work order can only be created from a purchase order screen. */



```
FSEDIT FACLITYS.PO                                          Obs 1
Command ===>

                  *** Facilities  Purchase Order ***

   Purchase Order: AA15889  A02     Status: ACTIVE          Archive: _
   Requisition Number: 990M021NTA

   Total Funding:         $4,000.00    Release Limit:       $1,000.00
  Total Committed:          $506.71   Total Invoiced:         $506.71
   Performance Dates:   Start: 10/01/1993   End: 09/30/1998

   Service: MEALS
   Purpose: MEALS

      Name: DEMPSEY'S RESTAURANT, INC         Phone: ____________
   Address: FIFTH STREET HIGHWAY              Vendor Number:
                                              Vendor Address Number:
      City: LAURELDALE            State: PA
       Zip: [illegible]

  Contacts: ____________________      Phone: __________
            ____________________             __________
```

```
/* MSMS.SASPROG.PERM.SAS(SCLPO) */

LENGTH response $ 1 ;  * for the DELETE confirmation window;

FSEINIT:
```
/* establish data set identifiers for this data set and another */
```
  dsiself = OPEN('faclitys.po','i') ;
  dsivdrs = OPEN('faclitys.vendors','i') ;
  CONTROL LABEL ALWAYS ;
RETURN ;
```

```
INIT:
ERROROFF po ;
/*OBSINFO returns information about the current
observation*/
  IF OBSINFO('NEW')
  THEN DO ;     /* check for duplicates */
             rc = WHERE(dsiself,"po='"||po||"'") ;
             IF ATTRN(dsiself,'ANY') GT 0
             THEN DO ;
                        ERRORON po ;
                        CURSOR po ;
                        _MSG_ ="This is a duplicate PO number" ;
                      END ;
         END ;
RETURN ;


status:
   CALL WREGION(10,35,15,45,'') ;
   IF status IN('?',' ') THEN status = LISTC('status.list','','Y',1) ;
RETURN ;

archive:
   IF archive NE '' THEN archive ='X' ;
RETURN ;


MAIN:
/*  confirm a deletion request */
   cmdword = WORD(1,'U') ;   * command line,first word ;
   IF cmdword =: 'DEL'
   THEN DO ;
              response = 'N' ;
             CALL WREGION(8,10,8,60,'') ;
             CALL DISPLAY('confirm.program',response) ;
             IF response = 'Y'
             THEN CALL EXECCMDI('DELETE','NOEXEC') ;
          END ;

/* disallow duplication of archived observations */
   ERROROFF _ALL_ ;
   IF WORD(1,'U') =: 'DUP' AND archive ='X'
   THEN DO ;
             ERRORON _ALL_ ;
             _MSG_ ="An ARCHIVED observation
                         cannot be duplicated" ;
           END ;

/* the PO field is editable – ensure no duplicates */
   ERROROFF po ;
   IF FIELD('MODIFIED','po')
   THEN DO ;
              rc = WHERE(dsiself,"po='"||po||"'") ;
              IF ATTRN(dsiself,'ANY') GT 0
              THEN DO ;
                         ERRORON po ;
                         CURSOR po ;
                        _MSG_ ="This is a duplicate PO number" ;
                       END ;
          END ;

/* the following code will search for vendor information even
when only part of the vendor name has been entered */
   _MSG_ = " " ;
/* another way to set the error condition of a field */
   rc = FIELD('ERROROFF','vndrname') ;
   IF FIELD('MODIFIED','vndrname') AND vndrname NE ' '
   THEN DO ;
         rc = WHERE(dsivdrs,
                        'vndrname CONTAINS " ' || vndrname || ' "') ;
```

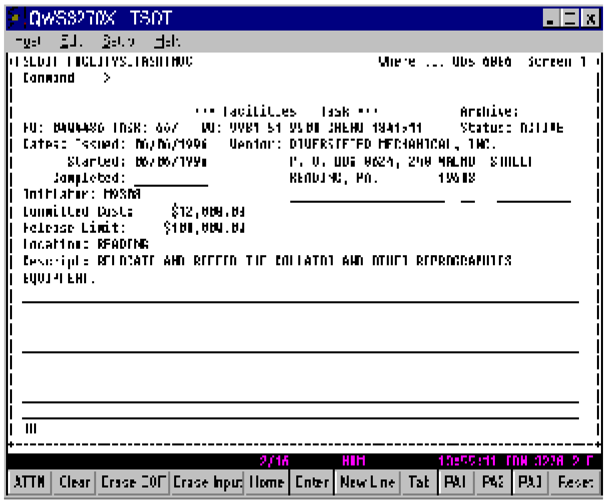
```
       /* test for ANY observations */
          IF FETCHOBS(dsivdrs,1) = -1
          THEN DO ;
/* remove the WHERE clause in preparation for next try  */
                     rc = WHERE(dsivdrs) ;
                     _MSG_ = "NO matches found on the
                              vendor address table!";
                     rc = FIELD('CURSOR','vndrname') ;
                  END ;

       /* test for more than one, if NOT, gather the data */
          ELSE IF FETCHOBS(dsivdrs,2) = -1
                THEN DO ;
                     rc = FETCHOBS(dsivdrs,1);
  vndrname = GETVARC(dsivdrs,VARNUM(dsivdrs,'vndrname'));
   vndrnmbr = GETVARC(dsivdrs,VARNUM(dsivdrs,'vndrnmbr')) ;
    vndraddr = GETVARC(dsivdrs,VARNUM(dsivdrs,'vndraddr')) ;
   vndradd1 = GETVARC(dsivdrs,VARNUM(dsivdrs,'vndradd1'));
   vndradd2 = GETVARC(dsivdrs,VARNUM(dsivdrs,'vndradd2')) ;
       vndrcity = GETVARC(dsivdrs,VARNUM(dsivdrs,'vndrcity')) ;
          vndrst = GETVARC(dsivdrs,VARNUM(dsivdrs,'vndrst')) ;
       vndrzip = GETVARC(dsivdrs,VARNUM(dsivdrs,'vndrzip')) ;
                END ;

          ELSE DO ;   /*get selection from list of matches */
                    CALL WREGION(10,1,24,76);
                     vndrname = DATALISTC(dsivdrs,
                                    'vndrname vndradd1 vndrcity',
                                    'Vendor Names','Y') ;
/* NOTE returns an identifier for the current observation */
                     noteid = NOTE(dsivdrs) ;
/* POINT locates the observation identified by NOTE */
                     rc = POINT(dsivdrs, noteid) ;
/* FETCH reads the data */
                     rc = FETCH(dsivdrs) ;
  vndrnmbr = GETVARC(dsivdrs,VARNUM(dsivdrs,'vndrnmbr')) ;
    vndraddr = GETVARC(dsivdrs,VARNUM(dsivdrs,'vndraddr')) ;
   vndradd1 = GETVARC(dsivdrs,VARNUM(dsivdrs,'vndradd1')) ;
   vndradd2 = GETVARC(dsivdrs,VARNUM(dsivdrs,'vndradd2')) ;
       vndrcity = GETVARC(dsivdrs,VARNUM(dsivdrs,'vndrcity')) ;
          vndrst = GETVARC(dsivdrs,VARNUM(dsivdrs,'vndrst')) ;
       vndrzip = GETVARC(dsivdrs,VARNUM(dsivdrs,'vndrzip')) ;
            END ;
      END ;


    IF WORD(1,'U') EQ 'WO'
    THEN DO ;
              CALL FSEDIT('faclitys.wo',
                             'affspscl.budscrns.wo.screen') ;
          END ;

    IF WORD(1,'U') EQ 'TSKINV'
    THEN DO ;
              dsitest =OPEN('faclitys.taskinvc(
                               WHERE=(po="'||po||'"))') ;
              IF ATTRN(dsitest,'ANY') GT 0
              THEN CALL FSEDIT(
                         'faclitys.taskinvc(WHERE=(po="'||po||'"))',
                         'affspscl.budscrns.taskinvc.screen') ;
              ELSE _MSG_="There are no TASKs/INVOICEs
                               with that PO" ;
              rc= CLOSE(dsitest) ;
           END ;
```

/* this is how a task/invoice is created --- macro variables are created to hold values that must be passed to the task/invoice, then a CALL FSEDIT is performed with the ADD option */

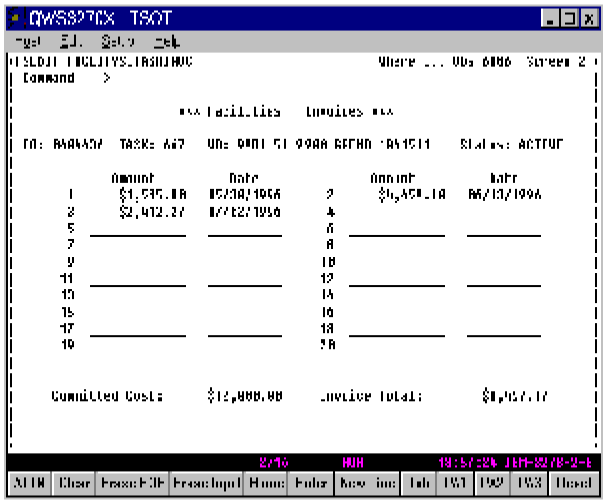
```
IF WORD(1,'U') EQ 'XTASK'
   THEN DO ;
/* CALL SYMPUT is used because this is run-time */
                 CALL SYMPUT('po',po) ;
                 CALL SYMPUT('status',status) ;
                 CALL SYMPUTN('release',release) ;
                 CALL SYMPUT('vndrname',vndrname) ;
                 CALL SYMPUT('vndradd1',vndradd1) ;
                 CALL SYMPUT('vndradd2',vndradd2) ;
                 CALL SYMPUT('vndrcity',vndrcity) ;
                 CALL SYMPUT('vndrst',vndrst) ;
                 CALL SYMPUT('vndrzip',vndrzip) ;
                 CALL SYMPUTN('vndrphon',vndrphon) ;
                 CALL FSEDIT('faclitys.taskinvc',
                             'affspscl.budscrns.taskinvc.screen',
                             'ADD') ;
            END ;

   IF WORD(1,'U') EQ 'LPOCC'
   THEN CALL FSEDIT('faclitys.lpocc',
                        'affspscl.budscrns.lpocc.screen') ;

   IF WORD(1,'U') EQ 'FERC'
   THEN CALL FSVIEW('library.woferc','BROWSE','','BRONLY') ;

RETURN ;    /* end of MAIN section */


TERM:     /* nothing to do here */
RETURN ;


FSETERM:     /* clean up before we leave */
   IF dsivdrs > 0 THEN rc = CLOSE(dsivdrs) ;
   rc = CLOSE(dsiself) ;
RETURN ;
```

Task / Invoice

/* the unique thing to savor with the SCL of task/invoices is how a new one is created; it is required that there be a purchase order first, before any task/invoice can be created. Once this relationship exists, it is possible to duplicate a task/invoice, however the integrity of the system depends upon this initial link. */



```
FSEDIT FACLTYS.TASKINVC                          Where ... Obs 6066   Screen 1
 Command ===>

                     *** Facilities   Task ***              Archive:
 PO: [illegible] TASK: 667   WO: [illegible]                Status: ACTIVE
 Dates: Issued: [illegible]/1996   Vendor: DIVERSIFIED MECHANICAL, INC.
        Started: 06/06/1996               P. O. BOX [illegible], 290 [illegible]
     Completed: ________                  READING, PA.        196[illegible]
 Initiator: [illegible]
 Committed Cost:     $12,000.00
 Release Limit:     $100,000.00
 Location: READING
 Descript: RELOCATE AND REFEED THE COLLATOR AND OTHER REPROGRAPHICS
 EQUIPMENT.
```



```
FSEDIT FACLTYS.TASKINVC                          Where ... Obs 6066   Screen 2
 Command ===>

                    *** Facilities   Invoices ***

 PO: [illegible]   TASK: 667   WO: [illegible]            Status: ACTIVE

              Amount        Date                Amount        Date
     1     $1,535.00    05/30/1996      2     $4,450.10    06/13/1996
     3     $2,412.27    07/02/1996      4
     5    __________    __________      6    __________    __________
     7                                  8
     9                                 10
    11    __________    __________     12    __________    __________
    13                                 14
    15                                 16
    17    __________    __________     18    __________    __________
    19                                 20

    Committed Cost:     $12,000.00     Invoice Total:      $[illegible]
```

```
/* MSMS.SASPRORG.PERM.SAS(SCLTSKIN) */

LENGTH response $ 1 ;  *for the DELETE confirmation window ;

FSEINIT:
   dsiself = OPEN('faclitys.taskinvc','i') ;
   CONTROL LABEL ALWAYS ;
/* remind these hard-working people how to get to where they want to go */
   _MSG_="          * * * INVOICEs are on Screen 2 <F11> * * *" ;
RETURN ;
```

```
INIT:
/* a LENGTH statement can be placed anywhere in the
program */
   LENGTH lasttask 8   cmdword $8 ;
   cmdword = WORD(1,'U') ;   * command line,first word ;

/* existing tasks/invoices CAN be duplicated in order to save
re-keying data.  However a NEW task/invoice must come
from a purchase order therefore . . . */

   IF OBSINFO('NEW') AND cmdword NOT IN('DUP','ADD')
   THEN DO ;
                po = SYMGET('po') ;   /* from PO */

                IF dsiself > 0 THEN rc= CLOSE(dsiself) ;
                dsiself = OPEN('faclitys.taskinvc(
                                     WHERE=(po="'||po||'"))','i') ;

  /* calculate the next sequential task number */
                rc = VARSTAT(dsiself,'task','MAX',lasttask) ;
                task = MAX(lasttask + 1,1) ;
                potask = po||PUT(task,Z3.) ;

  /* immediately SAVE the observation so that that NEW task
number is available for another user */
                CALL EXECCMDI('SAVE') ;

/* SYMGET is used because this is run-time */
/* these macros variables were created from the PO */
                status = SYMGET('status') ;
                release = SYMGETN('release') ;
                vndrname = SYMGET('vndrname') ;
                vndradd1 = SYMGET('vndradd1') ;
                vndradd2 = SYMGET('vndradd2') ;
                vndrcity = SYMGET('vndrcity') ;
                vndrst = SYMGET('vndrst') ;
                vndrzip = SYMGET('vndrzip') ;
                vndrphon = SYMGETN('vndrphon') ;

  /* fill in these fields automatically */
                taskstrt = DATE() ;
                initator = SYMGET('sysuid') ;

  /* once created the work order number is protected, since
this is a new observation, open the fields for input */
                UNPROTECT wohb woco woloc woserial woferc ;
                CURSOR wohb ;
           END ;

 /* for duplicated tasks/invoices,
 re-initialize the fields that are unique to each */
   ARRAY invdatx(*) invdat1-invdat15 ;
   ARRAY invamtx(*) invamt1-invamt15 ;
   IF OBSINFO('NEW')  AND  WORD(1,'U') =: 'DUP'
   THEN DO ;
                DO i = 1 TO 15 ;
                     IF invdatx(i) NE . THEN invdatx(i) = . ;
                     IF invamtx(i) NE . THEN invamtx(i) = . ;
               END ;
               committd = . ;
               invoiced = . ;
               UNPROTECT woferc ;
               CURSOR woferc ;
           END ;
RETURN ;    /* end of INIT section */


status:
   CALL WREGION(10,35,15,45,'') ;
   IF status IN('?',' ')
   THEN status = LISTC('status.list','','Y',1) ;
RETURN ;
```

```
woco:
   CALL WREGION(10,35,15,45,'') ;
   IF woco IN('?',' ')
   THEN woco = LISTC('company.list','','Y',1) ;
RETURN ;

woferc:
   IF INDEX(woferc,'?') > 0
   THEN DO ;
                dsiferc = OPEN('library.woferc','i') ;
                CALL WREGION(3,15,20,65,'') ;
               woferc=DATALISTC(dsiferc,'start label','FERC','Y',1);
                 rc= CLOSE(dsiferc) ;
            END ;
   ELSE DO ;
               dsitest =OPEN('library.woferc(
                                   WHERE=(start="'||woferc||'"))') ;
               IF ATTRN(dsitest,'ANY') GT 0
               THEN ;   /* A FERC is defined */
               ELSE _MSG_="That FERC is not defined
                                   in the table" ;
               rc= CLOSE(dsitest) ;
           END ;

                /* remove any spaces, commas, and underlines */
    woferc = LEFT(COMPRESS(woferc,'.,_')) ;
RETURN ;


MAIN:
   IF cmdword =: 'DEL'
   THEN DO ;
                response = 'N' ;
                CALL WREGION(8,10,8,60,'') ;
                CALL DISPLAY('confirm.program',response) ;
                IF response = 'Y'
                THEN CALL EXECCMDI('DELETE','NOEXEC') ;
            END ;

/* the FIELD function allows a maximum of three field names,
so... */
   IF FIELD('MODIFIED','wohb')
      OR  FIELD('MODIFIED','woco')
      OR  FIELD('MODIFIED','woloc')
      OR  FIELD('MODIFIED','woserial')
      OR  FIELD('MODIFIED','woferc')
   THEN DO ;
/* create the non-displayed variable that link this task/invoice
to a work order */
                wo=LEFT(COMPRESS(
                           wohb||woco||woloc||woserial||woferc,'.,_'));
                wo19=LEFT(COMPRESS(
                              woco||woloc||woserial||woferc,'.,_'));

wo11=LEFT(COMPRESS(woco||woloc||woserial,'.,_'));

                 dsitest =OPEN('faclitys.wo(
                                   WHERE=(wo11="'||wo11||'"))') ;
                 IF ATTRN(dsitest,'ANY') GT 0
                 THEN ;  /*a work order exists for this
task/invoice*/
                 ELSE _MSG_="There are no WOs with that
                                   WO(CO/LOC/SERIAL)" ;
                  rc= CLOSE(dsitest) ;
            END ;

   invoiced = SUM(OF invamt1-invamt20) ;
```

```
/* if the user attempts to ADD a new observation */
   ERROROFF _ALL_ ;
   IF WORD(1,'U') =: 'ADD'
   THEN DO ;
/* PROTECT prevents fields from being modified */
                PROTECT _ALL_ ;
                ERRORON _ALL_ ;
/* HOME positions the cusrsor on the command line */
                HOME ;
                _MSG_ ="To ADD a TASK, start at PO screen" ||
                        " and type XTASK - " <F2> to Cancel" ;
            END ;

   ERROROFF _ALL_ ;
   IF WORD(1,'U') =: 'DUP' AND archive ='X'
   THEN DO ;
                  ERRORON _ALL_ ;
                  CURSOR po ;
                   _MSG_ ="An ARCHIVED observation cannot
                           be duplicated" ;
            END ;

IF WORD(1,'U') EQ 'WO'
   THEN DO ;
               dsitest =OPEN('faclitys.wo(
                             WHERE=(wo11='"||wo11||"'))') ;
               IF ATTRN(dsitest,'ANY') GT 0
               THEN CALL FSEDIT(
                      'faclitys.wo(WHERE=(wo11='"||wo11||"'))',
                      'affspscl.budscrns.wo.screen') ;
               ELSE _MSG_="There are no WOs with that WO" ;
/* if we wanted to permit the editing of the whole data set  */
            /* ELSE CALL FSEDIT('faclitys.wo',
                                 'affspscl.budscrns.wo.screen');*/
               rc = CLOSE(dsitest) ;
            END ;

   IF WORD(1,'U') EQ 'POX'
   THEN DO ;
                dsitest =OPEN('faclitys.po(WHERE=(po='"||po||"'))') ;
                IF ATTRN(dsitest,'ANY') GT 0
                THEN CALL FSEDIT(
                          'faclitys.po(WHERE=(po='"||po||"'))',
                          'affspscl.budscrns.po.screen') ;
                ELSE _MSG_="There are no POs with that PO" ;
             /* ELSE CALL FSEDIT('faclitys.po',
                                 'affspscl.budscrns.po.screen');*/
                rc = CLOSE(dsitest) ;
            END ;

   IF WORD(1,'U') EQ 'LPOCC'
   THEN CALL FSEDIT('faclitys.lpocc',
                    'affspscl.budscrns.lpocc.screen') ;

   IF WORD(1,'U') EQ 'FERC'
   THEN CALL FSVIEW('library.woferc','BROWSE','','BRONLY') ;

RETURN ;


TERM:
RETURN ;


FSETERM:
  IF dsiself > 0 THEN rc= CLOSE(dsiself) ;
RETURN ;
```

CONCLUSION

Oh, there goes the bell! It is time to move on to your next class. I hope that this paper has provided you with a good primer to SCL, its language, its structure, and several of its uses. We have not covered every aspect or use of SCL, such as submit blocks, extended tables, use with FRAME entries, and data step interface. Your homework is to take what you learned, expand upon that with the manuals and on-line help, and put some ooh and awe in your next SAS/AF® or SAS/FSP® application. Class dismissed!

REFERENCES

SAS Institute Inc. (1994), *SAS® Screen Control Language: Reference, Version 6, Second Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1989), *SAS/AF ®Software: Usage and Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1989), *SAS/FSP ®Software: Usage and Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1991), *SAS ®Technical Report P-222, Changes and Enhancements to Base SAS ® Software, Release 6.07*,
Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1999), *SAS OnLineDoc®, Version Eight,*
Cary, NC: SAS Institute Inc.

Michael A. Mace
1152 E 344 ST
Eastlake, OH   44095-2942
440-953-8924 (H)
330-796-3981 (W)
e-mail: **michael.a.mace@worldnet.att.net**