

Stupid SCL Tricks: Techniques for Manipulating Screens and Views and Other Cute Stuff

Betty Smith, Computer Programmer
Bureau of Business Research
University of Texas at Austin

This paper describes a number of SCL coding techniques one can do to get an application to perform a number of tasks such as fetching (information retrieval, fishing (table lookup), swimming (navigation), creating and raising kids (creating/updating/appending SAS data sets in SCL), and other stuff like the mysterious METHOD routine and SUBMIT blocks.

The introduction of the SCREEN CONTROL LANGUAGE to VERSION 6 of SAS has empowered data entry and interactive applications with almost unimaginable abilities. For example, from within an FSEDIT display you have the ability to access other data sets and external files, and display them in any form you desire. You can perform calculations and tasks on specific fields in your display. Suppose while in an FSEDIT session you would need to copy a phone field to other phone fields in the observation. You could design an SCL routine which allows the user to press a function key and it will copy the phone field to the other phone fields in the display. You can affect the behavior of the display, if a certain condition is present. You could design an SCL routine that will issue an alarm and a message to the user when a certain field needs particular attention payed to it. This routine could even transport the display to the appropriate screen on which the field is located in multi-screen applications. The possibilities are endless. This paper deals with only a few possibilities and describes some techniques you can use to create some spectacular effects.

Trick #1 -- The Submit Block: Executing SAS code from within SCL

The SUBMIT block allows SAS/AF applications to generate SAS code statements. SAS code stored in an AF program will only execute after a SUBMIT command has been issued from within an application. There are two ways this command can be issued. One way is to do this is to issue a command from the command line of an AF program DISPLAY screen or any valid FSP screen which in turn calls the AF program where the SUBMIT block is stored; or from within the AF program's SCL using an EXEC CMD statement as is done in the following example:

```
init:
ALARM;
*sets up data files for this application;
call execcmd('SUBMIT');
submit ;
%macro go;
PROC DISPLAY CATALOG=ORDER.CAT.SETUP.HELP;RUN;
*set up pub.list;
data pub.list;
length pubnum $14 pub $80 price 8 qty 8 select $1 ordnum 8;
RUN;
proc fsedit data=pub.list screen=order.cat.pub.screen;run;
%mend go;
%go
PROC DISPLAY CATALOG=ORDER.CAT.ORDER.MENU;RUN;
ENDSUBMIT;
RETURN;
MAIN:
RETURN;
TERM:
RETURN;
```

Any standard SAS code can exist within the SUBMIT block of an SCL program including MACRO routines as is demonstrated in the above code; however, you cannot integrate SCL code and SAS code within the SUBMIT block. SCL code may precede or follow a SUBMIT block in the program. Although if you intend that SCL code following a SUBMIT block be executed, you must branch around the SUBMIT block by using SUBMIT CONTINUE as the label for the block. When you do this, all SCL code in the program is executed before the SUBMIT block statements are executed. For this reason it is not wise to put SCL code following a SUBMIT block which is dependent

upon the results of code executed within in the SUBMIT block. For example let's say you create a data set in the SUBMIT block and then following the SUBMIT block want to retrieve the number of observations in the data set:

```
INIT:
call execcmd('submit');
SUBMIT CONTINUE;
data temp;
do i = 1 to 10;
output;
end;
run;
endsubmit;
dsid=open('temp','i');
num=attrn(dsid,'nobs');
return;
main:
return;
term:
return;
```

This program would cause an error, because the SUBMIT block will not execute until all SCL statements preceding and following the block are executed first. Since the data set TEMP is created in the SUBMIT block, it is non-existent at the time SCL tries to retrieve observations from it. If you use any other WHEN statement next to the SUBMIT label such as PRIMARY, IMMEDIATE, or TERMINATE the SCL code following the SUBMIT block would be ignored. Consequently, if your program was triggered by a call from an AF menu or any other AF or FSP application that you intend to return to, you will need to issue the proper procedure to get back to that application from within the SUBMIT block as was done in this instance:

```
data pub.list;
length pubnum $14 pub $80 price 8 qty 8 select $1 ordnum 8;
RUN;
proc fsedit data=pub.list screen=order.cat.pub.screen;run;
%mend go;
%go
PROC DISPLAY CATALOG=ORDER.CAT.ORDER.MENU;RUN;
ENDSUBMIT;
```

Notice that we included a PROC DISPLAY prior to exiting the SUBMIT block, which returns to the menu that initiated the AF program where the SUBMIT block is stored.

Trick #2: Giving Birth: Creating Data Sets from Within an SCL Program

For this demonstration we will look at a program called SETUP.PROGRAM, which is also an AF program. This program first creates a blank data set and then displays the blank data set in an FSEDIT session:

```
init:
ALARM;
*sets up data files for this application;
dsid=open('pub.list','n');
rc=newvar(dsid,'pubnum','c',14,'string');
rc=newvar(dsid,'pub','c',80,'string');
rc=newvar(dsid,'price','n',8);
rc=newvar(dsid,'qty','n',8);
rc=newvar(dsid,'select','c',1,'string');
rc=newvar(dsid,'ordnum','n',8);
call close(dsid);
call fsedit('pub.list','order.cat.pub.screen','add');
RETURN;
MAIN:
RETURN;
TERM:
RETURN;
```

In order to setup a null data set you must first use an OPEN function as we did in the above code. Using a NEWVAR function, you then define what variables are to be in the data set. It is important that you describe at least the variable name, type, and length; but, you may also assign a label, format, and informat for the variable at this time. You cannot define multiple variables in a single NEWVAR function. Each variable must be created through a separate NEWVAR function. Finally you must close the data set, using the CLOSE statement in order to do anything else to the data set. When the above code is executed, it creates a blank or null data set. If you want to look at this data set, you will need to use an FSEDIT function or FSVIEW function. In this case, an FSEDIT session is used to display the data set, which uses an ADD statement. When the data set is displayed, a blank screen will display a new record ready for data entry.

Trick #3: Customizing Function Keys in an FSEDIT Session using SCL

Suppose you want to customize the definitions assigned to the KEYS window of an FSEDIT application temporarily without compromising the permanent set up stored in the SASUSER profile. You can do this using the GETFKEY function and the SETFKEY routine.

The following example illustrates this technique in a typical FSEDIT SCL program:

```
Fseinit:
temp1=getfkey(fkeyname(3));
call setfkey(fkeyname(3),'end');
temp2=getfkey(fkeyname(19));
call setfkey(fkeyname(19),'add');
return;
Init:
return;
Main:
return;
Term:
return;
Fseterm:
call setfkey(fkeyname(3),temp1);
call setfkey(fkeyname(19),temp2);
return;
```

As you can see in order to accomplish this task, the SCL code which performs the temporary replacement of keys must be housed in the FSEINIT block. The FSEINIT block executes only once upon the initiation of the FSEDIT session. In this case, we replaced the key definition for F3 with the GLOBAL command END, and F19 with ADD. The old definitions are stored in SCL variables TEMP1 and TEMP2. Upon completion of the FSEDIT session, the old definitions are replaced using the SETFKEY routine in the FSETERM block.

Since we were only replacing two keys in the above example, we used SCL variables. If you need to replace a slew of keys, you might use the following technique:

```
FSEINIT:
array temp(24) $20;
array key(24) $20 ('help' 'database' 'end' 'left' 'right' 'blue' 'back'
'forward' 'compdata' 'newdata' 'sic' 'homeoff' 'phonepa' 'dupthisfield'
'dupfirmmail' 'duppepasa' 'dupallfields' 'getinfo' 'add' 'search1' ' '
' ' ' 'alt');
do i = 1 to 2;
temp(i)=getfkey(fkeyname(i));
call setfkey(fkeyname(i),key(i));
end;
return;
```

```
init:
return;
main:
return;
term:
return;
FSETERM:
do i = 1 to 24;
call setfkey(fkeyname(i),temp(i));
end;
return;
```

You will notice in the above example that some of the key definitions are not SAS commands, which brings us to a discussion of:

Trick #4: Creating Customized Commands for Use in FSEDIT sessions:

It is possible to create customized commands that SCL will recognize and perform defined tasks. You can store your customized commands in the KEYS window as we did in the previous example, in PULL-DOWN MENUS, or you can type them in on the command line. How the command is issued or where it is stored is up to you. What is important is how to get SAS to accept these commands without causing an error. You can do this by putting a CONTROL ALWAYS statement as the last statement in an INIT block or as the first statement prior to the MAIN block in those cases when other labeled blocks precede the MAIN block:

```
INIT:
control always;
return;

OR

INIT:
Control label;
return;
If:
if il ne '' then do;
    SCL statements
    ...
end;
return;
Control always;
Main:
    SCL statements
    ... etc
```

The CONTROL ALWAYS statement allows the issued command to pass directly to SCL without evaluating it as a valid command. At this point you might be wondering if the the CONTROL ALWAYS causes SAS to ignore valid commands. The answer is NO. SAS will continue to execute valid commands whether CONTROL ALWAYS is in effect or not. For this reason, you should not use valid command names as you customized command names. For example, if you wanted to trigger a key word search application based on the issuance of the command SEARCH, SAS would cause an error because SEARCH is a valid FSEDIT command. The SAS system would be expecting to see the issuance of a STRING command preceding the issuance of the SEARCH command, and would therefore issue an error message on the display screen. The best solution for this senario would be to assign another name to the command such as SEARCH1 or SRCH.

When a command is issued in FSEDIT, it is stored in a command buffer. In order to proceed, you must get SCL to retrieve the command from the buffer. You can do this be using the WORD function. The WORD function

must be in the MAIN block of the SCL program:

```
MAIN:
DC=WORD(1,'U');
```

In the above example everytime the MAIN block executes, the WORD function retrieves the first word from the command buffer and then the 'U' parameter converts the command to upper case. The function then stores the command in the SCL variable DC.

Next you will need to set up a check to test the value of DC:

```
MAIN:
DC=WORD(1,'U');
IF DC = 'DATABASE' THEN DO;
    SCL STATEMENTS ...
END;
IF DC = 'BLUE' THEN DO;
    SCL STATEMENTS ...
END;
```

Using an IF statement, if the value of DC is interpreted as one the customized commands SCL will perform the task set forth in the IF block.

Trick #5 -- Swimming: Using SCL to Move Around in Multi-Screen FSEDIT Sessions

There is nothing more frustrating for an user than to have to page left and right through multiple screens in a FSEDIT session when all he needs to do is change only a few fields in the observation. Unfortunately these fields are very often spread through out several screens. As developers, we must be aware of what we can do to help speed up the entry process. Of course, you can bunch the most modified fields on the first screen of the application; but, in most real time applications this usually not a very good idea. For one reason this is not a good idea is that the screens you create should follow as closely as possible the flow of the document the user will be processing; the chances that the document will have the most modified fields bunched in one section would be unusual. For another reason suppose there is not enough consistency in the frequency of modified fields from one edit session to the next to justify placing these fields on the same screen. Luckily we can create ways, using SCL, to navigate from one screen to another without having to page through unwanted screens. The following illustrations show two ways this can be accomplished:

1.

```
+FSEDIT ALPHA.EXTDTM-----Obs 1 Screen 1--+
Command ==>
NOTE: This application uses 8 screens.
Status Code: 1      # Bluecards: ____      Flag: ____      Id: 7707099
-----
Delete this entry: ____      No Changes to This Entry: ____
Does this Company Also Wholesale? ____
-----
Go to:

_ Mailing Information          x Statistics/Plant Executives
_ Location Information        _ Alternate/Double Names
_ Phone Numbers              _ Siccodes/Text
_ Home Offices               _ Twin Plants
_ Notes to the Editor         _ Messages to (BETTY, SYLVIA, IDA)
(will be sent with entry)    (Sent to message file on an account)
-----
You may also reach the above areas by using the:
      F4 to view previous screen
      F5 to view next screen
You can only reach the next/previous records by:
      F7=Previous  F8=Next
You can always return to this screen by press f1:
+-----+
```

```

Command ==>
You can return to the first screen by pressing F1 key

                                Phones                                Id: 7707099
-----
Plant Phone:          ( 512 ) 722 - 9075
Toll Free (Texas):   1-800 - _____
Toll Free (Outside Texas): 1-800 - _____
FAX:                  ( ____ ) ____ - ____ ext ____
-----
                                Plant Statistics
Annual Sales: _ Not Reported      Organization: S Sole Proprietorship
Distribution: L Within Town or City  Yr Estab: 1946 # Employees:      5
-----
                                Plant Officers
Plant Executive: J E Saenz
Title:          owner                      Phone: ( ____ ) ____ - ____
Purchasing Agent: _____
Title:          _____                Phone: ( ____ ) ____ - ____
Sales Agent   : _____
Title:          _____                Phone: ( ____ ) ____ - ____
-----
Goto: _ F4 Previous Scr   F5 Next Screen   F7 Previous Rec   FB Nxt REC
+-----

```

[illegible]

+FSEDIT WORK.PROCESS-----Obs 1 Screen 2--+

Command ==>

Order # _____

Customer Id: _____

Shipping Information

Ship by: x UPS _ Mail _ Federal Express

Shipping Address: 310 N Louisiana _____

City: Amarillo _____ State: TX

zip: 79106 _____

Contact Addressee: Mildred Smith _____

----->

In the first example, we designed the navigation area on the first screen of the application and created some computed variables while in screen modification and design which will be used as GO TO triggers for the application. Computed variables are variables which appear on the screen but are not part of the data set. Beside each of the variables is a description of the section of the observation which SCL will go to, if its accompanying variable is modified. We then entered the SCL code for the screens as follows:

```
INIT:
CONTROL LABEL;
    SCL statements...
RETURN;
GM:
IF GM NE '' THEN CURSOR MAILNAME;
GM='';
RETURN;
GL:
IF GL NE '' THEN CURSOR FIRM;
GL='';
RETURN;
```

```

GP:
IF GP NE '' THEN CURSOR PHCD;
GP='';
RETURN;
GH:
IF GH NE '' THEN CURSOR HONUM1;
GH='';
RETURN;
GN:
IF GN NE '' THEN CURSOR NOTE1;
GN='';
RETURN;
GS:
IF GS NE '' THEN CURSOR SALEVOL;
GS='';
RETURN;
GA:
IF GA NE '' THEN CURSOR ALTER1;
GA='';
RETURN;
ALT1:
IF ALT1 NE '' THEN CURSOR ALTER1;
ALT1='';
RETURN;
ALT2:
IF ALT2 NE '' THEN CURSOR ALTER1;
ALT2='';
RETURN;
GT:
IF GT NE '' THEN CURSOR SIC1;
GT='';
RETURN;
GTP:
IF GTP NE '' THEN CURSOR NOTE;
GTP='';
RETURN;
GMS:
IF GMS NE '' THEN CALL DISPLAY('DTMEXT.CAT.SNDMSG.PROGRAM');
GMS = '';
RETURN;

```

A CONTROL LABEL statement is used in the INIT block. CONTROL LABEL directs SCL to execute blocks whose labels are the same name as the names of screen variables in FSEDIT session. Therefore SCL checks to see if any of the variables we set up in our navigation screen have been modified and executes the code in the corresponding block before it executes code in the MAIN block. To cause SCL to display the correct screen, we used the CURSOR statement followed by a the name of a screen variable which is located on the screen we want to display. When SCL encounters the CURSOR statement, it positions the cursor on the field whose names follows the word CURSOR.

The second example uses an AF program to accomplish the same task. In order to perform this task we use the technique described in trick #3 and #4 to start off:

```

FSEINIT:
temp=getfkey(fkeyname(1));
call setfkey(fkeyname(1), 'MENU1');
INIT:
length cmd $8;
Control Always;
return;
Main:
dc= Work(1,'U');
if dc = 'MENU1' then do;
CALL display('order.cat.moveme.program',cmd);
if cmd = 'ID' then cursor ID;
if cmd = 'UPS' then cursor UPS;
if cmd = 'TOTPRI' then cursor TOTPRI
end;
    MORE SCL Statements ...
return;
term:
return;
FSETERM:
call setfkey(fkeyname(1),temp);
return;

```

The set up above assigns a custom command to F1 which is called MENU1. Whenever F1 is pressed, SCL uses a DISPLAY function to call up MOVEME.PROGRAM, and passes the variable CMD to the program. When SCL encounters a DISPLAY statement it executes the program. MOVEME looks like this:

```

+BUILD: DISPLAY MOVEME.PROG+ E)-----+
| Command ==>          |          |
|                      |          |
| &I1 Goto Customer Info | iption          Updated |
| &I2 Enter Publications |          |
| &I3 Goto Shipping INfo | W formulas for WORK.ORDER.    09/16/92 |
| &I4 Look at Totals     | .HELP          08/25/92 |
|                      | .MENU          08/24/92 |
| press f3 to leave.    | PROGRAM        09/15/92 |
+-----+ SG.PROGRAM        09/15/92 |
| _ CUSTMSG2 PROGRAM CUSTMSG.PROGRAM    09/15/92 |
| _ END      PROGRAM  END.PROGRAM      09/11/92 |
|                      |          |
+-----+

```

The SOURCE window for MOVEME looks like this:

```
Entry Cmd $8;
INIT:
call setfkey(fkeyname(3),'END');
Control Label;
return;
i2:
if i2 ne '' then link pub;
cmd='totpri';
return;
i3:
if i3 ne '' then cmd='ups';
return;
i4:
if i4 ne '' then cmd = 'totpri';
return;
main:
return;
term:
return;
PUB: .. More SCL statements
```

MOVEME will display a screen which will overlap the FSEDIT screens. So it is necessary to design the display screen first before entering the SCL code in the SOURCE window of the application. When we first began to set up MOVEME, we accessed the catalog where it was to be stored. Once inside the catalog, we entered EDIT MOVEME.PROGRAM on the command line and pressed the enter key. At that point, the DISPLAY window appeared and we entered GATTR to access the GATTR window. The GATTR window allowed us to set the window size of the display window as well as other attributes. After setting up the parameters which govern the nature of the DISPLAY window, we left the GATTR window and returned to the DISPLAY window where we entered the variables I1-I4 and the direction dialog. After doing this, we accessed the SOURCE window by typing SOURCE on the command line of the display window and pressing enter. At this point we entered the SCL code you see above.

The first thing you notice in the SCL code is the the ENTRY statement. The ENTRY statement causes SCL to receive the variable passed to it in the DISPLAY routine we set up in our FSEDIT screen program. The SCL program checks to see if any of the variables I1-I4 have been modified and then executes the block whose label name matches the name of the modified variable. If for instance the user modifies I3, then the I3 block executes. SCL checks to see if I3 is blank. If I3 is not blank then it assigns the value of 'UPS' to CMD. When the user presses F3, which contains an END command, then control is returned to the FSEDIT screen program and the value in CMD is also returned to data set vector. The

statements following the DISPLAY routine are then executed. These statements query the value of CMD:

```
CALL display('order.cat.moveme.program',cmd);  
if cmd = 'ID' then cursor ID;  
if cmd = 'UPS' then cursor UPS;  
if cmd = 'TOTPRI' then cursor TOTPRI  
end;
```

Since CMD is equal in this instance to 'UPS', SCL moves the cursor to UPS field, which results in the display of the screen where the field USP is located.

Trick #6 -- Raising Kids -- Updating, Appending, and Deleting Observations to Datasets Using SCL

You can update, append, and delete observations in a data set or a group of data sets while in an FSEDIT session or AF program. Suppose that you want to create an application which performs maintenance on one or more data sets based on information entered in a single observation in a data entry session. This is possible using SCL. For example, suppose you want to create an application that updates information in a customer file as the customer's order for publications is being processed. When the customer calls and gives his name to the user, the user searches the customer file to see if the customer exists in the current customer file. After finding the customer's name, the user transfers the information from the customer file to the order file he is working on in FSEDIT. After some query, the customer informs the user that the company will be moving soon and would like the publications to go to the new address. Rather than re-accessing the customer file, the user can enter the new addresses (shipping and mail) on the order processing data set and SCL then updates the customer file based on the customer id.

In order to accomplish this feat, you will need to make sure that the data set variables that contain shipping and mailing information have the same name, type and length as the corresponding variables in the customer data set. Once this is done then all you have to do is this:

```
term:
*update info to customer file;
ID9="||TRIM(LEFT(ID))||";
call symput('id9',id9);
DSID=OPEN('CUST.DATA (WHERE=(ID=&ID9))','U');
RC=FETCH(DSID);
CALL SET(DSID);
RC=UPDATE(DSID);
RC=CLOSE(DSID);
Return;
```

In the TERM block of the application, we first did a little modification to the the id. We surrounded the id in quotes and assigned the the modified id to the SCL variable ID9. Next we fed ID9 in to a macro variable also called &ID9 so it could be used in an WHERE clause later on. We then opened the customer file and subsetting it using the WHERE clause in the OPEN statement. The OPEN statement set the customer data set up for updating using a 'U' parameter. The WHERE clause used in the open function made available to SCL only those observations that have the same customer id as the id currently displayed in FSEDIT screen. We then retrieved the observation in the customer file and put it in the data set vector using a FETCH function. We used a SET routine to associate the data set variables in the retrieved observation with matching variables in SCL (which includes all screen variables of the same name as well as any other SCL variables of the same name). Next we used the UPDATE function, which updated the retrieved observation's variables with the values of their matching SCL variables. Finally we close the data set using the CLOSE statement. Every

time the user leaves the current observation of the order process data set to get to the next observation, the TERM block will update the customer file.

By now you may be wondering what we would do about new customers. With a little modification to the example above, the task of adding new customers to the customer file can also be accomplished. For example, somewhere in previous SCL statements we set up a way of assigning new ids to new customers. At the time the new id is assigned, SCL will set a variable with a flag value. When the user begins to move to the next observation, the TERM block would execute the following code:

```
term:
*update info to customer file;
If flag = 0 then do;
ID9=" "||TRIM(LEFT(ID))||" ";
call symput('id9',id9);
DSID=OPEN('CUST.DATA (WHERE=(ID=&ID9))','U');
RC=FETCH(DSID);
CALL SET(DSID);
RC=UPDATE(DSID);
RC=CLOSE(DSID);
end
if flag = 1 then do;
dsid=open('cust.data','u');
call set(dsid);
Rc=append(dsid);
Rc=close(dsid);
end;
Return;
```

The above code checks to see if the variable FLAG is equal to '0' or '1'. Flag is set to '1' only when the order file observation contains a new customer id. To append the new customer information to the customer data set, we opened up the customer file for UPDATE using the OPEN statement much as we did before in the update process, except without subsetting the data set. Since we were adding information to the data set, it was not necessary to fetch any of the observations in the data set; therefore, the next step was to use the SET routine. Instead of an UPDATE function, we used the APPEND function to add the observation to the data set and of course we closed the data set with the close function.

What about deletions? Well suppose in the course of talking to the customer, the customer tells the user he wants his name stricken from the customer file. Again without having to re-access the customer file, the user can delete the customer from the customer while still in the order processing file. Suppose that we have a variable in the order processing database called DELCUS. If the user modifies DELCUS then

the variable FLAG can be set to '2'. Now that the variable flag contains a value of '2', when the user leaves to go to the next observation in the data set the following code would execute:

```
term:
*update info to customer file;
If flag = 0 then do;
ID9="||TRIM(LEFT(ID))||";
call symput('id9',id9);
DSID=OPEN('CUST.DATA (WHERE=(ID=&ID9))','U');
RC=FETCH(DSID);
CALL SET(DSID);
RC=UPDATE(DSID);
RC=CLOSE(DSID);
end
if flag = 1 then do;
dsid=open('cust.data','u');
call set(dsid);
Rc=append(dsid);
Rc=close(dsid);
end;
if flag = 2 then do;
ID9="||TRIM(LEFT(ID))||";
call symput('id9',id9);
DSID=OPEN('CUST.DATA (WHERE=(ID=&ID9))','U');
RC=FETCH(DSID);
RC=DELOBS(DSID);
RC=CLOSE(DSID);
end
Return;
```

In the last IF-THEN-DO block, you will notice the code is similar to the UPDATE setup; however, instead of using a SET function and an UPDATE function, we used the DELOBS function. Remember the FETCH retrieves the observation and places it in the data set vector. DELOBS deletes whatever is the current observation in the vector from the the data set.

Trick #7 -- Teaching SCL to Fetch -- Retrieving Information from a Data Set and Displaying it on the Screen.

Now that you have become aware of how easy it is to access other data sets with SCL, it is only logical that you would be able to transfer information from an external data set to your display screen. There are several ways that you can initiate this set up. For example, you could use a customized command to retrieve information or trigger execution of the application by modifying a screen variable. This is pretty much up to you; however, you must have a way of identifying the information you want to retrieve. In the following example, when the user enters a valid HOME OFFICE number in the field called HONUM1, the SCL program goes to the HOME OFFICE file and retrieves all the pertinent fields:

```
+FSEDIT ALPHA.EXTDTM-----Obs 1  Screen 5--+
| Command ==>
| You can return to the first screen by pressing F1 key
|
| Home Office #: 01004      Flag: _      Goto: _      Id: 7707099
|
| ALPH: _____
|
| Name: _____
|
| City: _____
|
| MailName: _____
| MailAttn: _____
| MailDiv: _____
| Address: _____
| City: _____ St: _ Zip: _____
|
| Phone: _____ FAX: _____
| Country: _____
|
|
| PRESS F5 TO SEE NEXT HOME OFFICE
+-----+
```

```

+FSEDIT ALPHA.EXTDTM-----Obs 1 Screen 5--+
| Command ==>
| You can return to the first screen by pressing F1 key
|
| Home Office #: 01004      Flag: 1      Goto: _      Id: 7707099
|
| ALPH: Founders Newspapers Group
|
| Name: Founders Newspapers Group
|
| City: Pasadena TX
|
| MailName: FOUNDERS NEWSPAPERS GROUP
| MailAttn: _____
| MailDiv: _____
| Address: BOX 6192
| City: PASADENA      St: TX Zip: 77506
|
| Phone: _____ FAX: _____
| Country: _____
|
| PRESS F5 TO SEE NEXT HOMEOFFICE
+-----+

```

In order to achieve this effect, we entered the following code in the MAIN section of the SCL program of our FSEDIT screen:

```

IF MODIFIED(HONUM1) AND HONUM1 NE '' THEN DO;
RC=SYSTEM('LINK BRAE051 191 499 RR DATABASE');
RC=SYSTEM('ACC 499 M');
HONM1="" TRIM(LEFT(HONUM1)) "";
CALL SYMPUT('HNM',HONM1);
DSID8=OPEN('DTM.HOMEOFF','I');
RC=WHERE(DSID8,'HONUM=&HNM');
DO WHILE(FETCH(DSID8) = -1);
HONN=GETVARC(DSID8,VARNUM(DSID8,'HONUM'));
IF HONN NE HONUM1 THEN CONTINUE;
HONAM1=GETVARC(DSID8,VARNUM(DSID8,'HONAME'));
HOCITY1=GETVARC(DSID8,VARNUM(DSID8,'HOCITY'));
HOFAX1=GETVARC(DSID8,VARNUM(DSID8,'HOFAX'));
HOPHON1=GETVARC(DSID8,VARNUM(DSID8,'HOPHONE'));
MAILNAM1=GETVARC(DSID8,VARNUM(DSID8,'MAILNAME'));
MAILADD1=GETVARC(DSID8,VARNUM(DSID8,'MAILADDR'));

```

```

MAILATN1=GETVARC(DSID8,VARNUM(DSID8,'MAILATTN'));
MAILCTY1=GETVARC(DSID8,VARNUM(DSID8,'MAILCITY'));
MAILDIV1=GETVARC(DSID8,VARNUM(DSID8,'MAILDIV'));
COUNTRY1=GETVARC(DSID8,VARNUM(DSID8,'COUNTRY'));
HOZIP1=GETVARC(DSID8,VARNUM(DSID8,'MAILZIP'));
HOST1=GETVARC(DSID8,VARNUM(DSID8,'MAILST'));
HOFLG1=GETVARC(DSID8,VARNUM(DSID8,'FLAG'));
FAXNAME1=GETVARC(DSID8,VARNUM(DSID8,'FAXNAME'));
ALPHNAM1=GETVARC(DSID8,VARNUM(DSID8,'ALPHNAME'));
LEAVE;
END;
RC=WHERE(DSID8);
CALL CLOSE(DSID8);
RC=SYSTEM('REL M(DET)');
END;

```

As you can see we triggered the execution of these statements based on the modification of the variable HONUM1. In the above example, SCL checks to see if HONUM1 has been modified. If it has been modified, then we link over to the disk which contains the HOME OFFICE data set using the SYSTEM functions. Since our operating system is CMS, we use CMS host commands within the SYSTEM functions to executes the link. We then surround HONUM1 with single quotes and assign the result to a variable called HONM1. Next we feed HONM1 into a macro variable called &HMN. The next step is to open the HOME OFFICE data set with an OPEN function for INPUT indicated by using the parameter 'I' in the function. We then issue a WHERE function which subsets the HOME OFFICE data set, so that the only retrievable observations are those whose HOME OFFICE number match the value in &HMN. We then FETCH each observation in the subset. Next we retrieve the appropriate variables from the subset and assign them to corresponding variables in our displayed data set using the GETVARC and GETVARN functions. When we have finished retrieval, we close the data set with a CLOSE statement and then release and detach the disk we accessed using SYSTEM functions.

Trick #8: -- SCL Goes Fishing -- Techniques for Setting Up a Table-Look Up Application

It is apparant that SCL's interactive capabilities are impressive. This trick utilizes these capabilities to the max as well as uses some of the tricks mentioned above. This technique helps to set up a key word search application that can be attached to an FSEDIT session or an AF program. The key word search application will search a database for the occurance of up to four key words and then display the result in an FSVIEW DISPLAY. The user can then select an observation to look at in its entirety which is then displayed on an FSEDIT screen. At this point, the user may select to transfer the information back to the display screen of the FSEDIT or AF display from where the search orginated:

```
+FSEDIT ALPHA.EXTDTH-----Obs 1 Screen 4--
```

| Command ==> USER POSITIONS CURSOR ON A TEXT 2 FIELD AND PRESSES F20 KEY.

| You can return to the first screen by pressing F1 key

| Sic: 2099 Text: Corn tortillas

| _____

| _____

| _____

| Sic: ____ Text: _____

| _____

| _____

| _____

| Sic: ____ Text: _____

| _____

| _____

| _____

| Sic: ____ Text: _____

| _____

| _____

| _____

Goto _

```
| Command ==>USER ENTERS KEYWORDS AND PRESSES |
| ENTER KEY |
| Enter Keywords you wish to look up: |
| |
| 1. cotton |
| 2. modules |
| 3. |
| 4. |
| |
| |
| Press f3 to abort. Press enter when you |
| complete entry. |
| |
+-----+
```

. Sic: ____ Text: _____ .
. _____ .
. _____ .
. _____ .
. Sic: ____ Text: _____ .
. _____ .
. _____ .
. _____ .
. Goto _____ .

```
| Command ==> USER CHOOSE TO LOOK AT METAL COTTON MODULES AND PRESSES ENTER |
```

```
| PRESS F3 TO RETURN |
```

OBS	USE	MSIC	DESCRIP
1		2394	Canvas cotton modules
2		2394	Cotton canvas modules
3		2394	Canvas products, including tar
4	x	3499	Metal cotton modules
5		2394	Covers for cotton modules, tar

+FSEDIT WORK.LOOK1 (Subset)-----Obs 4--+

| Command ==> USER SELECTS TO USE THIS DESCRIPTION IN EDIT FILE |

| Use this Sic # and Description: x -5672 |

| Sic: 3499 Metal cotton modules |

| _____ |
| _____ |
| _____ |

+FSEDIT ALPHA.EXTDTM-----Obs 1 Screen 4--+

| Command ==> INFORMATION TRANSFERS TO SIC TEXT 2 |

| Cursor must be on sic text to access this function |

| Sic: 2099 Text: Corn tortillas |

| _____ |
| _____ |
| _____ |

| Sic: 3499 Text: Metal cotton modules |

| _____ |
| _____ |
| _____ |

| Sic: ____ Text: |

| _____ |
| _____ |
| _____ |

| Sic: ____ Text: |

| _____ |
| _____ |
| _____ |

| Sic: ____ Text: |

| _____ |
| _____ |
| _____ |

Goto _

This particular example of table look-up is currently in a real time application used by the entry operators who process the DIRECTORY OF TEXAS MANUFACTURERS. The ability to look up and use descriptions already in the DIRECTORY OF TEXAS MANUFACTURERS on new entries helps to provide a consistency in the format of SIC descriptions in the directory. It also helps novice coders become more secure about assigning SIC codes to new entries, because it provides the ability to code by example. Finally, it frees the user of having to use voluminous amounts of texts and printouts to accomplish the same goal. Other possible uses of this technique could include providing on-line dictionaries for educational applications and an on-line procedures manual for processing insurance forms.

In the above example, we used a set up in FSEDIT to initiate the application:

```
FSEINIT:
array temp(24) $20;
array key(24) $20 ('help' 'database' 'end' 'left' 'right' 'blue' 'back'
'forward' 'compdata' 'newdata' 'sic' 'homeoff' 'phonepa' 'dupthisfield'
'dupfirmmail' 'duppepasa' 'dupallfields' 'getinfo' 'add' 'search1' ' '
' ' ' 'alt');
do i = 1 to 2;
temp(i)=getfkey(fkeyname(i));
call setfkey(fkeyname(i),key(i));
end;
return;
INIT:
array sicfld(5) $8 ('sictxt1' 'sictxt2' 'sictxt3'
'sictxt4' 'sictxt5');
array sic(5) sic1-sic5;
array sictxt(5) $200 sictxt1-sictxt5;
    More SCL statments ...
CONTROL ALWAYS
return;
MAIN:
    SCL statements ...
if dc = 'SEARCH1' then do;
fldon=curfld();
do i = 1 to 5;
if fldon = upcase(sicfld(i)) then do;
sicnum=sic(i); text=sictxt(i);
call display('dtmext.cat.search1.program',sicnum,text);
sic(i)=sicnum; sictxt(i)=text;
leave;
pass=0;
end;
else pass=1;
end;
if pass=1 then do;
alarm;
_msg_='Cursor must be on sic text to access this function';
end;
end;
    More SCL ....
```

In previous steps we made sure that F20 was defined as a command called SEARCH1. In the MAIN block, we checked to see if the command SEARCH1 was passed to the command buffer. If it had been passed then we executed the ensuing statements. In order that the application may transfer information from a selection in our table to a field in the entry screen, we used the CURFLD function to return the name of the field the cursor was positioned on. Then we checked to make sure that the field is a valid field to receive the transfer of data. In this particular application, there are five valid fields the cursor could be positioned on in order to initiate the application. These fields names were assigned to an array called SICFLD in the INIT block as were the values of the TEXT and SIC number fields assigned to arrays SIC and SICTXT. If the cursor is positioned on one of the TEXT fields and the F20 is pressed then we assigned the value of the corresponding SIC and TEXT to SCL variables SICNUM and TEXT. We then issue a DISPLAY routine and pass SICNUM and TEXT to a KEY WORD SEARCH PROGRAM called SEARCH.PROGRAM which is an AF program. SEARCH.PROGRAM is designed as follows:

```
+BUILD: DISPLAY SEARCH1.PROGRAM (E)-----+
| Command ==>                               |
|                                             |
| Enter Keywords you wish to look up:         | Updated |
|                                             |         |
| 1. &KEY1_____                           | 09/14/92 |
| 2. &KEY2_____                           | 09/21/92 |
| 3. &KEY3_____                           | 07/09/92 |
| 4. &KEY4_____                           | 08/04/92 |
|                                             | 08/17/92 |
|                                             | 08/04/92 |
| Press f3 to abort. Press enter when you    | 09/01/92 |
| complete entry.                            | 09/04/92 |
|                                             | 09/04/92 |
+-----+                                07/28/92 |
| _ TITLE  SCL   TITLE.SCL                  | 09/04/92 |
| _ ALPCOR  SCREEN FSEDIT SCREEN DESCRIPTION | 08/21/92 |
| _ ALPHA   SCREEN FSEDIT SCREEN DESCRIPTION | 09/23/92 |
| _ ALPHA1  SCREEN FSEDIT SCREEN DESCRIPTION | 08/26/92 |
| _ ALPHON  SCREEN FSEDIT SCREEN DESCRIPTION | 08/05/92 |
| _ LOOK1   SCREEN FSEDIT SCREEN DESCRIPTION | 08/07/92 |
|                                             |
+-----+
```

We designed the window size for the display on the GATTR screen, and then entered the fields for the key words and the surrounding text. We then entered the source window and entered the following:

```

entry sicnum 8 text $200;
init:
RC=SYSTEM('SET CMSTYPE HT');
RC=SYSTEM('CP SET INMSG OFF');
alarm;
rc=system('fi look1 disk d d d');
array key(4) $ key1-key4;
return;
control always;
main:
whr='Where=';
whr2='Where=';
if (modified(key1) and key1 ne '') or
    (modified(key2) and key2 ne '') or
    (modified(key3) and key3 ne '') or
    (modified(key4) and key4 ne '') then do;
dc=word(1,'U');
dsid99=open('look1','n');
rc=newvar(dsid99,'use','c',1,'string');
rc=newvar(dsid99,'msic','c',4,'string');
rc=newvar(dsid99,'descri','c',200,'string');
rc=newvar(dsid99,'descrip','c',30,'string');
rc=newvar(dsid99,'n','n',8);
rc=close(dsid99);
* build where statement;
do i = 1 to 4;
if trim(left(key(i))) ne '' then do;
ky="" upcase(trim(left(key(i)))) "";
whr=trim(left(whr)) ' upcase(descri) ? ' ky ' and';
whr2=trim(left(whr2)) ' upcase(sictext) ? ' ky ' and';
end;
end;
whr=whr " msic ne ''";
whr2=whr2 " sicCODE ne .)";
call symput('whr',whr);
call symput('whr2',whr2);
rc=system('link brae053 191 699 rr all');
rc=system('acc 699 1');
rc=system('fi looksic2 disk d d l');
rc=system('fi look2 disk d d d');
dsid98=open('looksic2.data (&whr)','i');
do while(fetch(dsid98) = -1);
use='';
msic=getvarc(dsid98,varnum(dsid98,'msic'));
descri=getvarc(dsid98,varnum(dsid98,'descri'));
descrip=substr(trim(left(descri)),1,30);
n=getvarn(dsid98,varnum(dsid98,'n'));
dsid99=open('look1','u');

```

```

call set(dsid99);
rc=append(dsid99);
rc=close(dsid99);
end;
rc=close(dsid98);
rc=system('link brae051 191 499 rr database');
rc=system('acc 499 M');
rc=system('fi DTM disk d d M');
rc=system('fi look2 disk d d d');
dsid98=open('DTM.SIC (&whr2)', 'i');
do while(fetch(dsid98) = -1);
use='';
msic=TRIM(LEFT(getvarN(dsid98, varnum(dsid98, 'siccode'))));
descri=getvarc(dsid98, varnum(dsid98, 'sictext'));
descri=substr(trim(left(descri)), 1, 30);
n=curobs(dsid98)*-1;
dsid99=open('look1', 'u');
call set(dsid99);
rc=append(dsid99);
rc=close(dsid99);
end;
rc=close(dsid98);
rc=copy('look1', 'look2');
call fsview('look2 (cntllev=member)', 'edit',
'dtmext.cat.look2a.formula');
dsid99=open("look1 (where=(use ne ''))", 'i');
do while(fetch(dsid99) = -1);
sicnum=getvarc(dsid99, varnum(dsid99, 'msic'))*1;
text=getvarc(dsid99, varnum(dsid99, 'descri'));
leave;
end;
rc=close(dsid99);
rc=system('rel l(det');
RC=SYSTEM('REL M(DET');
call execcmd('end');
end;
return;
term:
return;

```

In the above example we use the ENTRY statement to pass SICNUM and TEXT into the program. Then we assign the values of KEY1-KEY4 to an array called KEY. In the MAIN block, we assign the beginnings of a WHERE statement to variables WHR and WHR2. WHR and WHR2 will later be used as macro variables as WHERE clauses in a couple of OPEN functions. Using an IF-THEN-DO structure, we check to see if any of the window variables have been modified and if they contain any values in them. If they have been modified then we proceed to create a blank template in which we will store our subsetting data set. Using an OPEN function, we create the empty data set called LOOK2. LOOK2 contains the same variables found in our source data sets with the exception of the variable USE. USE is a variable that

the user will mark in order to select the observations he wants to view. The creation of a variable such as USE is essential for reference data sets. Next we begin to build our WHERE statements using the DO-END block. For this particular application we will be retrieving information from two data sets. One data set contains a list of SIC codes and descriptions created from an SIC manual created by one of our editors that is taken from reference texts. The other is the data set which contains the actual entries in the DIRECTORY OF TEXAS MANUFACTURERS database. In the DO-END block, we check to see if the value of our window variables are blank. If a window variable is not blank then we append the following texts to WHR and WHR2:

```
if trim(left(key(i))) ne '' then do;
ky=""||upcase(trim(left(key(i))))||"";
whr=trim(left(whr))||' upcase(descri) ? '||ky||' and';
whr2=trim(left(whr2))||' upcase(siccode) ? '||ky||' and';
end;
```

If we use the keywords described in the example sited at the beginning of this section, the first pass of the DO-END would cause WHR and WHR2 to contain the following text:

```
WHERE= ( UPCASE(DESCRI) ? 'COTTON' AND
WHERE= ( UPCASE(SICCODE) ? 'COTTON' AND
```

After the second pass through the DO-END loop, WHR and WHR2 would contain the following text:

```
WHERE= ( UPCASE(DESCRI) ? 'COTTON' AND UPCASE(DESCRI) ? 'MODULE' AND
WHERE= ( UPCASE(SICCODE) ? 'COTTON' AND UPCASE(SICCODE) ? 'MODULE' AND
```

After completing the DO-END loop, we tag the the following text to WHR and WHR2:

```
whr=whr||" msic ne ''";
whr2=whr2||" siccode ne .)";
```

Thus WHR and WHR2 would contain the following texts:

```
WHERE= ( UPCASE(DESCRI) ? 'COTTON' AND UPCASE(DESCRI) ? 'MODULE' AND
MSIC NE '')

WHERE= ( UPCASE(SICTEXT) ? 'COTTON' AND UPCASE(SICTEXT) ? 'MODULE' AND
SICCODE NE .)
```

After building WHR and WHR2, we feed both variables into macro variables also called &WHR and &WHR2. We then extract the necessary observations from the first dataset:

```
call symput('whr',whr);
call symput('whr2',whr2);
rc=system('link brae053 191 699 rr all');
rc=system('acc 699 1');
rc=system('fi looksic2 disk d d 1');
rc=system('fi look2 disk d d d');
dsid98=open('looksic2.data (&whr)','i');
do while(fetch(dsid98) = -1);
use='';
msic=getvarc(dsid98,varnum(dsid98,'msic'));
descri=getvarc(dsid98,varnum(dsid98,'descri'));
descrip=substr(trim(left(descri)),1,30);
n=curobs(dsid98);
dsid99=open('look1','u');
call set(dsid99);
rc=append(dsid99);
rc=close(dsid99);
end;
rc=close(dsid98);
```

We link to the disk where the data set is stored using a SYSTEM function. Using an OPEN function, we access the first data set. Within the OPEN function, we incorporate &WHR. When SCL encounters this macro it reads the OPEN statement as follows:

```
DSID=OPEN("LOOKSIC.DATA (WHERE=(UPCASE(DESCRI) ? 'COTTON' AND
UPCASE(DESCIR) ? 'MODULE' AND MSIC NE ''))","I");
```

Therefore, the observations which we will be accessing are only those observations whose TEXT field contains our key words we specified and whose manufacturing SIC is not blank. Using a DO-WHILE loop, we retrieve each observation in the subsetted data set, pull out the fields we want to put in our table using GETVARC and GETVARN functions, assign the value of the current observation to the variable N using a CUROBS function, and using the structure described in trick #6, append the observations to LOOK1. We

close both data sets and proceed to extract the observations using the same structure as described above from our second source data set:

```
rc=system('link brae051 191 499 rr database');
rc=system('acc 499 M');
rc=system('fi DTM disk d d M');
rc=system('fi look2 disk d d d');
dsid98=open('DTM.SIC (&whr2)','i');
do while(fetch(dsid98) = -1);
  use='';
  msic=TRIM(LEFT(getvarN(dsid98,varnum(dsid98,'siccode'))));
  descri=getvarc(dsid98,varnum(dsid98,'sictext'));
  descrip=substr(trim(left(descri)),1,30);
  n=curobs(dsid98)*-1;
  dsid99=open('look1','u');
  call set(dsid99);
  rc=append(dsid99);
  rc=close(dsid99);
end;

rc=close(dsid98);
```

After appending the observations to LOOK1, we use a COPY function to create LOOK2. LOOK2 is the data set we will view in FSVIEW and LOOK1 will be the data set displayed later in the FSEDIT screen. After copying the data set, we use an FSVIEW to display our table. Within the FSVIEW routine, we specify a formula which at this point is nonexistent and will be described later in the process. After returning from FSVIEW, we retrieve values for SICNUM and TEXT (the variables that we passed into the program earlier from our FSEDIT program) from the data set LOOK1.

```
rc=copy('look1','look2');
call fsview('look2 (cntllev=member)','edit',
'dtmext.cat.look2a.formula');
dsid99=open('look1 (where=(use ne ''))','i');
do while(fetch(dsid99) = -1);
  sicnum=getvarc(dsid99,varnum(dsid99,'msic'))*1;
  text=getvarc(dsid99,varnum(dsid99,'descri'));
  leave;
end;
```

When the program returns control to the FSEDIT program from whence the search application was launched, the values of SICNUM and TEXT are assigned to the appropriate fields and displayed on the screen:

```
call display('dtmext.cat.search1.program',sicnum,text);
sic(i)=sicnum; sictxt(i)=text;
leave;
pass=0;
end;
else pass=1;
end;
if pass=1 then do;
alarm;
_msg_='Cursor must be on sic text to access this function';
end;
end;

More SCL ....
```

In order to set up the formula we assigned in the FSVIEW statement in the SEARCH program, we had to execute the key word search from the FSEDIT screen, placing the cursor on a the TEXT field and pressing F20. After conducting the key word search which resulted in pulling up our table in FSVIEW, we defined the positions of the variables we wanted to display and then defined a formula entry as follows:

```
.FSVIEW: WORK.LOOK2 (E).....
. Command ==>
.
. OBS      USE MSIC DESCRIP
. +FSVIEW: Define command-----+
. | Command ==>
. |
. |      Name: CC
. |      Type: NUMERIC  CHARACTER
. |      Format: BEST12.
. |      Informat: 12.
. |      Label: _____
. |
. | Enter the formula below:
. |
. |      CC = modified(use);if cc and use ne '' then do;call symputn('n
. |              1',n);call fsedit('look1 (where=(n=&n1))','dtmext.cat.loo
. |              k1.screen','edit');use='';END;ALARM;
. |              MSG_='PRESS F3 TO RETURN';
. |
. +-----+
.
.
.....
```

In order to enter the formula entry line we issued a DEFINE CC command from the command line and pressed enter. The result of this action creates a computed variable called CC. In FSVIEW screens, SCL is stored in the define windows of computed variables. The above code illustrates the importance of having a SELECT variable (in this case USE) in the table. The variable CC receives a '1' if the variable USE is modified. The formula then checks to see if CC = '1' and the variable USE has been marked. If the condition is true then we assign the value of the variable N to the numeric macro 'N1' using a SYMPUTN statement. N is a unique identifier assigned to observation at the time of extraction in the SEARCH.PROGRAM. We then use N in the WHERE clause in the FSEDIT routine in order to subset the sister data set LOOK1, so that the observation displayed before the user is identical to the one he/she selects in LOOK2. After entering the code above, we returned to the FSVIEW screen and placed an 'x' in the first observation to pull up the FSEDIT screen. We then entered the modification section of the FSEDIT screen and designed the screen to look as follows:

```

+FSEDIT WORK.LOOK1 (Subset)-----Obs 1+
|
| Command ==>
|
| Use this Sic # and Description: _ 1834
|
| Sic: 2394 Canvas cotton modules
|
| _____
| _____
| _____
|
|-----
|
| .
| .
| .
| .
| .
| .
| .
| .
|

```

After leaving the modification menu of the screen, we thus completed the development of the application. All that was left to do was test it, so we placed an 'x' in USE on this screen and when we returned to our data entry screen the SIC code and TEXT resided in TEXT field from which we initiated the SEARCH.

Of course, how you display the table that has been extracted is a matter of style. However, tables basically must have these characteristics:

1.	The data set must contain a SELECT variable or a way the USER can select one or more observations from the table.
2.	The data set must contain a variable that is a unique identifier for each observation

You can also use one table instead of two if you choose to display it in only one mode, i. e. FSEDIT or FSVIEW. In this case if you choose to use FSVIEW, variables whose fields exceed the length of the screen will not be displayed in their entirety in which case you might want to display the table in FSEDIT instead.

In the above example, the values returned by the application were displayed in an FSEDIT session. If you choose to return the values of a table look-up to an AF program display, you might need to modify the returned variables to adjust to size allowed for AF displays. For example, if one of the return variables bigger than the display allows, you may need to design several screen variables and break the string up into several parts then assign the parts to the corresponding screen variables.

Trick #9: The Method -- Using Method Entries to Enhance Data Entry

The final trick this paper will address is the mysterious METHOD feature. METHOD is to Version 6.07 SCL, what MACRO routines are to standard SAS code. The feature allows you to design blocks of code that are stored in AF catalogs in an SCL entry. Usually the entry is defined with an .SCL extension in the catalog. The METHOD block contains code which performs a task or calculation in which several elements in the SCL program may need executed. For example, we used the METHOD feature to perform abbreviation on three fields in the following example:

```
+FSEdit ALPHA.EXTDTM-----Obs 2 Screen 3-|
| Command ==>|
| You can return to the first screen by pressing F1 key|
|          Phones                               Id: 7706124|
|-----|
| Plant Phone:      ( 806 ) 274 - 6347|
| Toll Free (Texas): 1-800 - _____|
| Toll Free (Outside Texas): 1-800 - _____|
| FAX:              (  )  -  -  ext  _____|
|-----|
|          Plant Statistics|
| Annual Sales: _ Not Reported      Organization: C Corporation|
| Distribution: I Export           Yr Estab: 1929 # Employees: 125|
|-----|
|          Plant Officers|
| Plant Executive: Larry Donovan|
| Title:          plant manager      Phone: ( 806 ) 274 - 6347|
| Purchasing Agent: J I Lee|
| Title:          Manager purchasing  Phone: ( 806 ) 274 - 6331|
| Sales Agent    : C M McGuire|
| Title:          manager marketing   Phone: ( 216 ) 836 - 9591|
|-----|
| Goto: _ F4 Previous Scr  F5 Next Screen  F7 Previous Rec  F8 Nxt REC|
+-----+
```

In the above example, we spelled out the titles for the PLANT EXECUTIVE, PURCHASING AGENT, and SALES AGENT. In the SCL program for the screen we called a METHOD ROUTINE each time one of the fields were modified. In other words, there are three METHOD calls in the SCL program:

```
if modified(petitl) then do;
call method('title.scl','title1',petitl,pename);
end;
if modified(patitl) then do;
call method('title.scl','title1',patitl,paname);
end;
if modified(satitl) then do;
call method('title.scl','title1',satitl,saname);
end;
```

The METHOD call consists of the following elements:

1.	The name of entry where the method block is stored
2.	The name of the label the block of code can be found under
3.	Parameters necessary for the operation to be performed

After the User presses enter in the above example, the method block returns the following:

```
+FSEdit ALPHA.EXTDTM-----Obs 2 Screen 3-|
| Command ==>|
| You can return to the first screen by pressing F1 key|
|          Phones                               Id: 7706124|
|-----|
| Plant Phone:          ( 806 ) 274 - 6347|
| Toll Free (Texas):    1-800 - _____|
| Toll Free (Outside Texas): 1-800 - _____|
| FAX:                  ( ____ ) ____ - ____ ext ____|
|-----|
|          Plant Statistics|
| Annual Sales: _ Not Reported      Organization: C Corporation|
| Distribution: I Export            Yr Estab: 1929 # Employees: 125|
|-----|
|          Plant Officers|
| Plant Executive: Larry Donovan|
| Title:          plt mgr          Phone: ( 806 ) 274 - 6347|
| Purchasing Agent: J T Lee|
| Title:          mgr purchasing    Phone: ( 806 ) 274 - 6331|
| Sales Agent   : C H McGuire|
| Title:          mgr mktg          Phone: ( 215 ) 636 - 9991|
|-----|
| Goto: _ F4 Previous Scr  F5 Next Screen  F7 Previous Rec  F8 Nxt REC|
+-----+
```

Notice that the title fields for the plant officers are now abbreviated.

The METHOD block which is stored in an .SCL entry in the DTTEXT.CAT catalog which performed the operation looks like this:

```
array titnam(*) $30 ('chief executive officer'
'chief operating officer' 'chairman of the board'
'accountant' 'accounting' 'accounts' 'account' 'acting'
'administrator' 'administrating' 'administration' 'administrative'
'advertising' 'assistant' 'bookkeeper' 'branch' 'business'
'chairman' 'clerk' 'committee' 'comman' 'commercial' 'comptroller'
'coordinator' 'controller' 'department' 'director' 'district'
'distribution' 'division' 'editor' 'engineering' 'engineer'
'equipment' 'executive' 'general' 'information' 'laboratories'
'laboratory' 'merchandise' 'mechanical' 'mechanic' 'manager'
'management'
'manufacturing' 'manufacturer' 'marketing' 'market' 'officer'
'office' 'partners' 'partner' 'plant' 'president' 'production'
'publication' 'publisher' 'research & development' 'refinery'
'regional' 'representative' 'resident' 'secretary' 'superintendent'
'supervisor' 'services' 'service' 'technical' 'technician'
'treasurer' 'vice president' 'vice pres' 'warehouse' 'works');
array titabr(*) $12 ('ceo' 'coo' 'chmn' 'acct' 'acctg' 'acct'
'acct' 'actg' 'admin' 'admin' 'admin' 'admin' 'adv' 'asst'
'bkkpr' 'br' 'bus' 'chmn' 'clk' 'com' 'committeeman' 'coml'
'comp' 'coord' 'contr' 'dept' 'dir' 'dist' 'distr' 'div'
'ed' 'eng' 'eng' 'equip' 'exec' 'gen' 'info' 'lab' 'lab' 'mdse'
'mech' 'mech' 'mgr' 'mgmt' 'mfg' 'mfr' 'mktg' 'mkt' 'off' 'off'
'parts' 'part' 'plt' 'pres' 'prod' 'pub' 'pub' 'r&d' 'ref'
'reg' 'rep' 'res' 'sec' 'supt' 'supvr' 'svc' 'svc' 'tech' 'tech'
'treas' 'vp' 'vp' 'whse' 'wks');
title:
method title $27 nam $50;
j=0;
wrđ='hi';
do until(wrd = '');
j+1;
wrđ=scan(title,j);
do i = 1 to dim(titnam);
title=lowcase(title);
title=tranwrđ(title,trim(left(titnam(i))),trim(left(titabr(i))));
end;
end;
indof = index(title,' of ');
if indof > 0 then do;
title=tranwrđ(title,' of ', ' ');
title=compbl(title);
end;
indor=index(nam,' or ');
indan=index(nam,' and ');
indanl=index(nam,' & ');
indsls=index(nam,'/');
```

```

if indor = 0 and indan = 0 and indan1 = 0 and inds1s=0 then do;
title=trim(left(title));
if title = 'chmn/ceo' or title = 'chmn ceo' then title='chmn';
if title = 'chmn/gen mgr' or title = 'chmn gen mgr' then title='chmn';
if title = 'chmn/pres' or title = 'chmn pres' then title='chmn';
if title = 'ceo/chmn' or title = 'ceo chmn' then title='chmn';
if title = 'ceo/exec vp' or title = 'chmn exec vp' then title='ceo';
if title = 'ceo/vp' or title = 'chmn vp' then title='ceo';
if title = 'ceo/pres' or title = 'ceo pres' then title='ceo';
if title = 'pres/ceo' or title = 'pres ceo' then title = 'ceo';
if title = 'pres/chmn' or title = 'pres chmn' then title = 'chmn';
if title = 'pres/dir mfg' or title = 'pres dir mfg' then title='ceo';
if title = 'dir mfg/pres' or title = 'dir mfg pres' then title='ceo';
end;
endmethod;

```

Notice you can declare arrays and you can use variables or constants in the performance the tasks in the METHOD block that are unique to the METHOD routine. However, information that is part of the SCL program which calls the METHOD block and variables that receive the action in the block must be passed as parameters in the METHOD block. For example, we sent the variables PENAME and PETITL to the METHOD block for calculation in the following statement:

```

if modified(petitl) then do;
call method('title.scl','title1',petitl,pename);
end;

```

The METHOD block receives these parameters as dummy variables in the METHOD statement within the METHOD block stored under TITLE1 in TITLE.SCL:

```

title1:
method title $27 nam $50;

```

The METHOD statement must be the first statement to follow the LABEL in the block, the number of parameters listed must match the number of parameters sent by the CALL METHOD, and the parameters must have a length and type specification. In this case, the parameters sent were variable values; but, you can also send string constants and numbers in the CALL METHOD ROUTINE. You can store several METHOD blocks in a single SCL entry. METHOD blocks can be called and used by multiple applications which use SCL.

The METHOD block need not be reserved just for calculations, you can also perform functions such as SORT, FSEDIT, FSVIEW, or DISPLAY calls from within METHOD blocks. For example, suppose you have several applications

which require you to display a data set in FSEDIT the following scenario could be used:

From within Application #1, you want to display a file called CUST.DATA then your SCL would look like this:

```
Call Method('ORDER.CAT.FSSEE.SCL','EDIT1','CUST.DATA','ORDER.CAT.CUST.SCREEN');
```

From within Application #2, you want to display a file called ORDER.DATA then your SCL would look like this:

```
Call Method('ORDER.CAT.FSSEE.SCL','EDIT1','ORDER.DATA','ORDER.CAT.ORDER.SCREEN');
```

The METHOD block stored in FSSEE.SCL would look like this:

```
EDIT1:  
METHOD DATAST $10 SCREN $30;  
CALL FSEDIT(DATAST,SCREN,'EDIT');  
ENDMETHOD;
```

As you can see the METHOD feature saves code and therefore space in the SCL program. Imagine how much space the abbreviation routine in our example would take up if much of the code had to be duplicated for each of the titles in the application.

Conclusion

As you can see from the few demonstrations presented in this paper, the SCREEN CONTROL LANGUAGE feature is very powerful and has provided tools which enable developers to create powerful interactive applications. There is so much more beyond the few tidbits addressed by this paper not only in regard to enhancing data entry applications; but, in regard to developing system applications as well. This paper barely touched on a few of the functions discussed in the manual. There are all sorts of things you can do with directories, external files, graphics, and much more. It is hoped that this paper will encourage you to use SCL to enhance current and future interactive applications.