ABSTRACT

The introduction of Screen Control Language (SCL) with version 6.03 provides new measures of control for those developing data entry systems with PROC FSEDIT. SCL can be used to check for specific values during data entry, rather than checking only within a given range. Cursor control under SCL can provide for the conditional skipping of items, rather than advancing one item at a time. Values can be crosschecked for consistency as they are entered, rather than waiting to find inconsistencies later in a separate DATA step. Customized messages can provide instructions for specific situations, rather than relying on standard messages provided by the procedure.

Such tools are essential in developing a data entry system for an interview containing multiple skip patterns and many possible inconsistencies. This paper explores two concerns: 1) what are the difficulties in testing and developing such a system with SCL; and 2) what are the limits of control SCL provides. For comparison, these same issues are addressed in developing a similar data entry system with Microsoft QuickBASIC 4.0. These comparisons provide the basis for recommendations on how SCL might be improved in future releases.

SCREEN DEVELOPMENT

Screen development under FSEDIT combines flexibility with ease of use. In screen modification mode, text can be typed exactly as it will appear when the application is run. By turning NUMBER ON in the editor, the developer can see how many rows of text will constitute one screen. This number is thirty-one rows on my IBM monochrome monitor. I could then juggle the text from each page of the interview to fit on a single screen. This juggling act was extensive, however, since each printed page from the interview consisted of up to sixty-six lines of text and white space. I achieved this reduction by eliminating the listing of possible answers and valid codes which follows each question in the printed interview. Further reduction was also achieved by displaying only the question number and the first couple of lines to lengthy questions. This allowed a blank line to separate each question, and the page number to be displayed at the bottom of the screen. Such a screen provides a reasonable facsimile of the printed page and displays sufficient information to the data entry editor to easily identify the corresponding location in the interview.

Not surprisingly, the devising of similar screens with Microsoft QuickBASIC 4.0 proved to be much more cumbersome, since QuickBASIC has no built-in features designed explicitly for screen development. A file containing the text of questions was read into a QuickBASIC Array and printed to the screen. It was necessary to count lines in the text file to determine whether one page of information would fit on one screen. Adjusting the format of the screen required running the program, viewing the outputted screen, editing the text file, and re-running the program.
PROGRAM CONTROL

An important issue in developing this data entry system is the point at which the SCL program takes control of the process. By default, statements in the MAIN section of the SCL program are executed only after a field is modified with a valid value AND the user presses the ENTER or a function key. The CONTROL statement in SCL can modify this so that the ENTER or a function key is sufficient to execute the MAIN label. Under either circumstance, SCL does not gain control of the process until the ENTER or a function key is pressed.

This is critical in an application that wants to control the positioning of the cursor and how it advances. Under FSEDIT the cursor can be moved in numerous ways without ever pressing the ENTER or a function key. If a value fills the field provided, the cursor automatically proceeds to the next unprotected item on the screen. The TAB key also advances the cursor to the next unprotected field, and the back TAB backs up the cursor in a similar fashion. The arrow keys can position the cursor anywhere on the screen. Since these cursor changes occur without the ENTER or function keys, the SCL program exerts no control.

To increase the chances of the SCL program being executed immediately after a field has been modified, I left justified the fields and provided more spaces than needed for each value. This leaves the cursor on the field after the value has been typed in. The LEGEND window can be used to instruct the data entry person to 'press ENTER after each value' to make sure that the SCL program finds variables later in the interview to be missing when earlier variables are being entered.

The QuickBASIC program can provide much greater control of the cursor. The INKEY$ function reads whatever character is typed from the keyboard. The program can specify what should happen to the cursor next based solely on the characters typed. Since the ENTER key is not required to activate these statements, the data entry person is saved from that extra key stroke for each item.

CHECKING VALID VALUES

Most of the items in the interview have codes of 1, 5 or 9. This limits the usefulness of assigning MIN and MAX values as special attributes during modification mode. Screen Control Language is needed to check for specific values when the possible codes are not consecutively numbered. SCL statements can be designed to mimic what happens when the MIN/MAX attributes are violated during an FSEDIT session. The use of the Macro Language greatly reduces the number of statements necessary to achieve this effect for all the appropriate variables. The following macro checks whether a value has an acceptable value. If not, the computer beeps, the error flag is set for that variable, an error message is written to the screen, and the cursor is placed on the field in error.

```
%macro code159 (var);
  if &var not in (.,1,5,9) then do;
    alarm;
    erroron &var;
    _msg=# ERROR: Invalid Code 
    _cursor &var;
  end;
%mend code159;
```

The macro must allow missing as a valid code, since the SCL program should be activated after each field is modified. This will insure that the cursor advances correctly with each entry, but it also means that the SCL program will find variables later in the interview to be missing when earlier variables are being entered.

The QuickBASIC program reads in the acceptable codes for a question into an array and compares the value entered against the acceptable codes. There is no need to consider a missing value, since the program knows from the array index variable where the cursor is and what variable is about to be entered. This exceeds what an SCL program can know about the cursor. SCL can position the cursor with the CURSOR statement, but there is no function or automatic variable to identify the position of the cursor upon entering the SCL program. There is a CURROW automatic variable which identifies the row being processed in an extended table, but such a table is only used in SAS/AF applications.

CHECKING SKIP PATTERNS

The checking of skip patterns presents a much more involved task, since there are no general rules to be applied as in the case of valid codes. Each of the fourteen skips in the interview has its own logic, so separate statements have to be tailored to each condition.

For example, the first question in the section asks the age at which the respondent was first drunk (variable DIS149). If the respondent reports this...
age to be under 15, he is asked whether this happened more than once (variable DIS149B). If the reported age is 15 or over, the DIS149B question should be skipped and the next question is asked (variable DIS150). The following SCL code can be used for the appropriate skipping.

```sas
if 15 <= DIS149 <= 99 and DIS150 = . then do;
    DIS149B = .;
    cursor DIS150;
end;
```

Thus, when the age is over 15 and question DIS150 has not yet been answered, give DIS149B a skip code (.s) and position the cursor at DIS150. This duplicates how the interview should have proceeded. The second part of the IF statement is needed to prevent the cursor from being stuck on DIS150 once that question has been answered. Of course, all of this assumes the ENTER key is pressed after each value is entered, so the SCL program gains control before entering the next question.

A more general approach was used with the QuickBASIC program. In the text file containing the valid codes for each question, a corresponding target question was listed for each code. Consequently, the program knew not only if a code was valid, but also where the cursor should be placed next.

**CHANGING ANSWERS**

FSEEDIT provides the person entering data much flexibility in moving the cursor to previous fields if it is necessary to change a value already entered. This flexibility, however, poses a threat to maintaining the logical flow of the interview. This is particularly true since the SCL statements are not activated until the ENTER key or a function key is pressed. The back TAB or arrow keys allow the cursor to be moved to previous fields and answers can be changed, but the SCL program has no control until ENTER or a function key is pressed.

Our above example with DIS149, DIS149B, and DIS150 demonstrates the problem. If the age of 21 is entered for DIS149, the next question is skipped and the cursor is placed at DIS150. Nothing prevents the data entry person from using the back TAB to enter a value for DIS149B. This creates an inconsistency in the data.

There are two ways to provide for such situations. One way is to write additional statements which will flag such an inconsistency once the SCL program regains control. In our example, such a statement would be:

```sas
if DIS149B gt 0 and 15 <= DIS149 <= 99 then do;
    alarm;
    erroron DIS149B;
    _msg_="Item should be skipped ";
    cursor DIS149B;
end;
```

The drawback to this approach is that it doubles the amount of programming needed to handle the skips in the interview. Not only does the SCL program have to provide for the proper advancing of the cursor moving forward through the interview, it also has to catch inconsistencies caused by moving backward through the interview.

The other way to handle this problem is to create a backup command and assign it to a function key. The WORD function in SCL can be used to create a new command. This function returns words from the command line. When a command is issued that SAS(R) software does not recognize, the SCL program can specify what should happen next. For example, if we want to call the new command BACKUP the SCL program can use the WORD function as follows:

```sas
if word(1)= 'backup' then do;
    /* specify what should happen with the BACKUP command */
end;
```

The drawback to this approach is that it doubles the amount of programming needed to handle the skips in the interview. Not only does the SCL program have to provide for the proper advancing of the cursor moving forward through the interview, it also has to catch inconsistencies caused by moving backward through the interview.

**CHANGING ANSWERS**

FSEEDIT provides the person entering data much flexibility in moving the cursor to previous fields if it is necessary to change a value already entered. This flexibility, however, poses a threat to maintaining the logical flow of the interview. This is particularly true since the SCL statements are not activated until the ENTER key or a function key is pressed. The back TAB or arrow keys allow the cursor to be moved to previous fields and answers can be changed, but the SCL program has no control until ENTER or a function key is pressed.

Our above example with DIS149, DIS149B, and DIS150 demonstrates the problem. If the age of 21 is entered for DIS149, the next question is skipped and the cursor is placed at DIS150. Nothing prevents the data entry person from using the back TAB to enter a value for DIS149B. This creates an inconsistency in the data.

There are two ways to provide for such situations. One way is to write additional statements which will flag such an inconsistency once the SCL program regains control. In our example, such a statement would be:

```sas
if DIS149B gt 0 and 15 <= DIS149 <= 99 then do;
    alarm;
    erroron DIS149B;
    _msg_="Item should be skipped ";
    cursor DIS149B;
end;
```

The drawback to this approach is that it doubles the amount of programming needed to handle the skips in the interview. Not only does the SCL program have to provide for the proper advancing of the cursor moving forward through the interview, it also has to catch inconsistencies caused by moving backward through the interview.

The other way to handle this problem is to create a backup command and assign it to a function key. The WORD function in SCL can be used to create a new command. This function returns words from the command line. When a command is issued that SAS(R) software does not recognize, the SCL program can specify what should happen next. For example, if we want to call the new command BACKUP the SCL program can use the WORD function as follows:

```sas
if word(1)= 'backup' then do;
    /* specify what should happen with the BACKUP command */
end;
```

The drawback to this approach is that it doubles the amount of programming needed to handle the skips in the interview. Not only does the SCL program have to provide for the proper advancing of the cursor moving forward through the interview, it also has to catch inconsistencies caused by moving backward through the interview.

The other way to handle this problem is to create a backup command and assign it to a function key. The WORD function in SCL can be used to create a new command. This function returns words from the command line. When a command is issued that SAS(R) software does not recognize, the SCL program can specify what should happen next. For example, if we want to call the new command BACKUP the SCL program can use the WORD function as follows:

```sas
if word(1)= 'backup' then do;
    /* specify what should happen with the BACKUP command */
end;
```

The drawback to this approach is that it doubles the amount of programming needed to handle the skips in the interview. Not only does the SCL program have to provide for the proper advancing of the cursor moving forward through the interview, it also has to catch inconsistencies caused by moving backward through the interview.

The other way to handle this problem is to create a backup command and assign it to a function key. The WORD function in SCL can be used to create a new command. This function returns words from the command line. When a command is issued that SAS(R) software does not recognize, the SCL program can specify what should happen next. For example, if we want to call the new command BACKUP the SCL program can use the WORD function as follows:

```sas
if word(1)= 'backup' then do;
    /* specify what should happen with the BACKUP command */
end;
```

The drawback to this approach is that it doubles the amount of programming needed to handle the skips in the interview. Not only does the SCL program have to provide for the proper advancing of the cursor moving forward through the interview, it also has to catch inconsistencies caused by moving backward through the interview.

The other way to handle this problem is to create a backup command and assign it to a function key. The WORD function in SCL can be used to create a new command. This function returns words from the command line. When a command is issued that SAS(R) software does not recognize, the SCL program can specify what should happen next. For example, if we want to call the new command BACKUP the SCL program can use the WORD function as follows:

```sas
if word(1)= 'backup' then do;
    /* specify what should happen with the BACKUP command */
end;
```

The drawback to this approach is that it doubles the amount of programming needed to handle the skips in the interview. Not only does the SCL program have to provide for the proper advancing of the cursor moving forward through the interview, it also has to catch inconsistencies caused by moving backward through the interview.

The other way to handle this problem is to create a backup command and assign it to a function key. The WORD function in SCL can be used to create a new command. This function returns words from the command line. When a command is issued that SAS(R) software does not recognize, the SCL program can specify what should happen next. For example, if we want to call the new command BACKUP the SCL program can use the WORD function as follows:

```sas
if word(1)= 'backup' then do;
    /* specify what should happen with the BACKUP command */
end;
```

The drawback to this approach is that it doubles the amount of programming needed to handle the skips in the interview. Not only does the SCL program have to provide for the proper advancing of the cursor moving forward through the interview, it also has to catch inconsistencies caused by moving backward through the interview.

The other way to handle this problem is to create a backup command and assign it to a function key. The WORD function in SCL can be used to create a new command. This function returns words from the command line. When a command is issued that SAS(R) software does not recognize, the SCL program can specify what should happen next. For example, if we want to call the new command BACKUP the SCL program can use the WORD function as follows:

```sas
if word(1)= 'backup' then do;
    /* specify what should happen with the BACKUP command */
end;
```

The drawback to this approach is that it doubles the amount of programming needed to handle the skips in the interview. Not only does the SCL program have to provide for the proper advancing of the cursor moving forward through the interview, it also has to catch inconsistencies caused by moving backward through the interview.

The other way to handle this problem is to create a backup command and assign it to a function key. The WORD function in SCL can be used to create a new command. This function returns words from the command line. When a command is issued that SAS(R) software does not recognize, the SCL program can specify what should happen next. For example, if we want to call the new command BACKUP the SCL program can use the WORD function as follows:

```sas
if word(1)= 'backup' then do;
    /* specify what should happen with the BACKUP command */
end;
```

The drawback to this approach is that it doubles the amount of programming needed to handle the skips in the interview. Not only does the SCL program have to provide for the proper advancing of the cursor moving forward through the interview, it also has to catch inconsistencies caused by moving backward through the interview.
use them. It is possible to PROTECT fields when values are entered so that the data entry person could not change them. This would force the use of the backup command which could include an UNPROTECT statement. All of this still assumes that the ENTER key is being pressed after each value.

The second problem is that this may be an overly punitive method for changing answers. If only one answer on the first screen had to be changed when the cursor was on the fifth screen, this method would erase all the answers in between one at a time and the data would have to be reentered once the backing up was completed.

The only fool-proof method for insuring clean data requires both forward and backward logic in the SCL program. This would eventually catch any inconsistencies created by backing up with something other than the newly defined function key. It would also allow the data entry person greater flexibility in moving around the interview, especially when a destructive backup is not necessary.

The QuickBASIC program provides only for a destructive backup. This guarantees clean data and eliminates the need for backward logic, but it has the drawbacks noted above. To match all the FSEDIT capabilities for moving about the interview, however, would entail very extensive programming in QuickBASIC.

**PROGRAM DEVELOPMENT TOOLS**

FSEDIT provides a menu system for entering the SCL environment. After entering the FSEDIT procedure, the developer can execute the MODIFY command and then is presented with the modification menu. Option 3 on this menu is "Edit Program Statements and Compile." With this option, all of the SAS editor's features are available for entering, copying, and moving the text of the SCL statements.

The only hindrance in this process comes in testing the SCL program, since it is necessary to compile the statements before using them with the application. As the SCL program grows, this recompiling can take several minutes. If some statements have syntax errors, a warning is printed on the modification menu to look at the log for errors. Once the errors are found, it is necessary to reinvoke option 3 of the modification menu, correct the errors, recompile the program, and test the application again. If errors in logic are found in testing the application, the same multi-step process is required to change the SCL program. It is particularly annoying that this time-consuming recompilation is done even when no SCL statements have been altered.

Since the time involved for recompiling the program becomes substantial as the number of statements grows, it is helpful to restrict testing to a section at a time. This can be done by entering statements in the PROGRAM window of the Display Manager. When a SCL section is completed, the statements can be MARKed and CUT into the paste buffer. After entering option 3 of the modification menu in FSEDIT, one can PASTE this section into the FSEDIT Program window. In this way the compiling, testing and recompiling of code can be limited to a piece of the entire program. Properly working sections can be filed and combined for later use.

By comparison, the program development tools available with QuickBASIC are truly superb. Version 4.0 combines a text editor, a special kind of compiler, and a debugger into a single smart editor. This smart editor finds many syntax errors as they are being typed. The editor also standardizes the format of each line as it is entered, by automatically capitalizing all BASIC keywords and providing spaces before and after all BASIC operators. This standardization can alert the programmer to other errors before run time, such as typing print instead of PRINT. If the syntax of the statement is correct, it is translated into executable form and is ready to run immediately.

Code can be tested simply by pressing a function key. Since statements have already been translated into executable code, this process takes seconds rather than minutes. To test ideas interactively before incorporating them into the main program, the developer can access an "Immediate Window" in which statements are executed immediately after being entered. After a program has been tested and debugged, it can then be compiled as an .EXE file executable from the DGS prompt.

**SUGGESTED SCL IMPROVEMENTS**

PROC FSEDIT has many powerful tools for screen development, data entry, editing, and data retrieval. These tools combine flexibility with ease of use. It would be a daunting task to write a QuickBASIC program which matched all of FSEDIT's features. Yet, a comparison with QuickBASIC shows two areas of needed improvement: 1) SCL can be strengthened...
with more program control, and 2) SCL program development can be made more efficient with a better programming environment.

1. More Program Control
   a. Put more control in the CONTROL statement.
      The tightest control provided by the CONTROL statement still requires an ENTER or function key to be pressed before the SCL statements are executed. An alternative to this would be to have SCL statements executed whenever a data field is filled. Thus, during screen development, the developer could identify fields with just enough underscores to contain the value. Then, when data is being entered, the SCL statements would be executed after each value was typed. This would eliminate the need for the extra ENTER keystroke after each value, and control would pass immediately to the SCL statements.
   
   b. Provide the option to selectively disable keys.
      The ability to disable the TAB, back TAB, and arrow keys would provide additional control to the SCL program. In this way, the cursor could only move by filling a field, issuing commands from the command line or with function keys, or by pressing ENTER — all of which would return control to the SCL program (if the first recommendation was implemented).

2. Better Programming Environment
   Programming in SCL involves a cycle of editing, compiling, running, debugging, and editing again. By way of contrast, the smart editor in QuickBASIC 4.0 combines these processes into a single integrated environment. The more the SCL programming environment can approach such a model, the easier and more efficient program development will become.