An Introduction To Screen Control Language with the FSEDIT Procedure

Annette T. Harris, SAS Institute Inc., Cary, NC
Ann E. Carpenter, SAS Institute Inc., Cary, NC
Glenn H. Itano, The Fair, Isaac Companies, San Rafael, CA

ABSTRACT

The FSEDIT procedure is used to create and edit SAS® data sets and prepare customized data entry windows. In Version 6 of SAS/FSP® software, you can write programs using Screen Control Language (SCL) to increase the functionality of the FSEDIT procedure as a data entry tool.

INTRODUCTION

The FSEDIT procedure in SAS/FSP software is used as a data entry and data editing tool for SAS data sets. Using the FSEDIT procedure, you can create, browse, and modify a SAS data set in an interactive windowing environment.

In previous releases, the FSEDIT procedure enabled the user to create a customized data entry window and perform simple data validation by specifying field attributes. These attributes enabled you to indicate such restrictions as minimum and maximum values for a field or a requirement that a field be filled in before the user could leave the FSEDIT window.

In Version 6 of the SAS System, you also have the ability to write a program that is integrated with the FSEDIT window to perform sophisticated data validation and to interact with the user as data are entered. This is done with Screen Control Language (SCL), a language whose syntax is very similar to that of the SAS DATA step. Many DATA step language statements and functions are available in SCL. In addition, several special features of SCL are designed specifically to help you design powerful data entry systems.

This tutorial is intended for a person who has used the DATA step and is somewhat familiar with the FSEDIT environment. The following topics will be discussed:

• how to invoke the FSEDIT Program window
• SCL statements and functions
• SCL reserved statement labels
• SCL statement execution sequence
• examples of PROC FSEDIT SCL programs.

INVOKING THE FSEDIT PROGRAM WINDOW

The SCL program is stored with the modified screen of a PROC FSEDIT application. To access a sample SCL program, invoke the FSEDIT procedure with the data set PERM.ADDRESS and modified screen PERM.MYSCRN by submitting the statements in Screen 1 to the SAS System for execution.

INVOKING THE FSEDIT PROGRAM WINDOW

These statements initiate an FSEDIT session and place you in edit mode on the first observation of the data set specified. On the command line of the FSEDIT window, issue the MODIFY command to open the FSEDIT Menu window. See Screen 2. You can select options from the Menu window to modify the appearance of the FSEDIT window, create and compile a Screen Control Language program, define attributes of variable fields, or define general parameters of your FSEDIT application.

Select option 3, Edit Program Statements and Compile, to access the FSEDIT program window. See Screen 3. Here you will enter your SCL program. When you press END to exit this window, your SCL program will automatically be compiled and stored with the modified screen specified in the PROC FSEDIT statement. Any error messages generated during compilation will be written to the SAS log.
An SCL program is composed of SCL statements, many of which are identical to DATA step statements. Some examples of SCL statements are assignment statements, DO groups, and IF-THEN/ELSE statements. Many SCL functions are comparable to their DATA step counterparts, including TODAY, INT, UPCASE and others.

Like DATA step variables, SCL variables contain either character or numeric data. They can be grouped into three types:

- **screen variables** are variables that appear in the FSEDIT window. In the FSEDIT procedure, these are generally the variables in the SAS data set you are editing but may also be special fields. Special fields are repeated or computational fields.

- **nonscreen variables** are temporary variables that can be created and used by the SCL program but do not appear in the FSEDIT window.

- **system variables** are variables with reserved names that check or update system information. _MSG_ is an example of a system variable that enables a customized message to be written to the message line of the FSEDIT window.

**STATEMENT LABEL EXECUTION SEQUENCE**

The reserved SCL labels in an FSEDIT session follow this general sequence of execution. When the FSEDIT procedure is first invoked:

1. **FSEINIT** executes
2. **INIT** executes
3. the first observation is displayed.

If a screen variable is modified and ENTER or a function key is pressed, the following sequence occurs:

1. **MAIN** executes
2. the current observation is redisplayed.

If a screen variable is modified and you scroll off the current observation, the sequence is as follows:

1. **MAIN** executes
2. **TERM** executes
3. **INIT** executes for the new observation
4. the new observation is displayed.

If a screen variable is modified and the END command is issued, the sequence is as follows:

- displaying initial messages for an observation.
If you leave an observation by scrolling or issuing the END command and a screen variable has not been modified, the TERM section is not executed.

**SCL FOR DATA CHECKING**

SCL provides special statements, functions and system variables to support the process of verifying data as they are entered. Some of these include:

- **MODIFIED** a function that indicates whether a value in a field has been updated.
- **ERROR** a function that tells the program whether a field's error attribute has been set.
- **ERRORON** a statement that enables the application builder to set the error attribute for a field.
- **ERROROFF** a statement that enables the application builder to turn off the error attribute for a field.
- **ALARM** a statement that sounds the terminal bell.

**DATA VALIDATION EXAMPLE USING AN ARRAY**

The example in Screen 4 checks to see if a value has been entered into the DEPT field, which appears in the FSEDIT window. If the field is blank, a message is written to the message line. If the value entered is in error, a message is written, the cursor is positioned, the error flag is set, and the alarm is sounded.

```scil
/* Check to see if department is blank */
if dept = '' then
desc dept;
  /* If dept is invalid, issue a more severe warning */
  if dept = 'SALES' and dept = 'SALES' and dept = 'SALES'
  and dept = 'EACH' and dept = 'EACH'
  then
correct dept;
  alarm;
  /* Send system variable _MSG_ to user */
  if _MSG_ = 'Invalid department code. Please reenter.'
  then
do;
  /* Send new message */
  if _MSG_ = 'Product code is invalid. Please reenter.'
  then
do;
  /* End */
do;
end;
return;

3. FSETERM executes

4. the FSEDIT procedure is terminated.

Screen 4 Simple Data Validation Example

This same example might also be written as shown in Screen 5.

**DATA VALIDATION EXAMPLE USING A SECONDARY DATA SET**

In this application, the user will enter a value for product code in the FSEDIT window. The SCL code will then search a secondary data set to verify that the code entered is valid and, if so, will display a description of that product to the user. This process is called cross-validation because the SCL code verifies that a valid product code has been entered. In addition, by seeing the expanded description, the user can verify that the intended product code was entered.

```scil
/* Screen 5 Alternative Code for Simple Data Validation */

/* FSEDIT program */

/* Command */

/* INIT */

/* INIT */

/* TERM */

/* FSETERM */

/* TERM */

/* TEST.SECOND is opened in input mode since we will only be reading from the data set. The second statement uses the VARNUM function to map the location of the variable CODE in the secondary data set for later use. A RETURN statement is then issued to close the FSEINIT section and return control to the FSEDIT procedure. */

In the FSEINIT section of this example, the secondary data set TEST.SECOND is opened in input mode since we will only be reading from the data set. The second statement uses the VARNUM function to map the location of the variable CODE in the secondary data set for later use. A RETURN statement is then issued to close the FSEINIT section and return control to the FSEDIT procedure.

The INIT section executes when each observation is initially displayed. In Screen 5, it uses the system variable _MSG_ to send a message to the user indicating that a product code must be entered.

In Screen 5, it uses the system variable _MSG_ to send a message to the user indicating that a product code must be entered.
The MAIN section executes each time a user modifies a field in the FSEDIT window and presses ENTER. The statements here are in a DO group so they are executed if the user modifies the variable CODE. Once CODE has been modified, SAS software uses the LOCATEC function to search the secondary data set to see if the specified product code is valid.

If the search returns a 0, the value was not found. The SCL statements then put the CODE field in error and display a descriptive error message to the user. See Screen 7.

If successful, the search returns a value greater than zero and the user sees an appropriate message. Also, the GETVARC function retrieves the value of DESCRIPT from the secondary data set and displays it in the FSEDIT window as further verification to the user as shown in Screen 8.

The TERM section executes when the user exits the current observation and displays the next observation or exits the FSEDIT session. Since no statements need to be executed in this section, it is simply closed with a RETURN statement.

The FSETERM section is the complement to the FSEINIT section and executes when you exit from the FSEDIT procedure. Here it is used to close the secondary data set.

CONCLUSION
The Screen Control Language in Version 6 is a very powerful tool for enhancing FSEDIT applications. Functions, statements and automatic system variables available in SCL provide new flexibility and control in an interactive windowing environment.

SAS and SAS/FSP are registered trademarks of SAS Institute Inc.