Effectively Using the FSEDIT Procedure
Howard Levine, Levine Software Systems

Introduction

Proc FSEDIT is an extremely useful procedure for editing SAS data sets interactively. Use of the Screen Control Language (SCL) with version 6 of the SAS® System makes the procedure much more powerful. In this paper, I will cover the following points:

- Introduction to Proc FSEDIT
- Proc FSEDIT Options
- Proc FSEDIT Commands
- Proc FSEDIT Screens - Basic
- Using Formats and Informats - Basic
- Restructuring data for easier editing
- Using SCL

The most important thing to remember with any interactive editing technique is that there are three basic characteristics that are desirable. The editing should be quick, easy, and accurate. Any data editing programs should be judged on those criteria.

Quick editing is desirable because it saves people time. This is particularly true for experienced data entry personnel.

Easy editing is desirable because it reduces training costs and encourages people to fill in the entry screens accurately. Difficulty with using data entry systems causes high levels of user frustration which discourages proper data entry.

Accuracy is desired to prevent data bases from containing garbage data. Accuracy can be improved by using relational data bases to insure data integrity. SCL is well suited for this task, but post-edit programming steps can perform data validation (even if they are more awkward and less convenient than SCL).

Introduction to Proc FSEDIT

In order use a procedure effectively, it is necessary to understand what its options and statements are and what it can do. Only points of interest that I feel important will be covered. This is not meant to be a substitute for the procedure documentation in the SAS/FSP® Users' Guide.

The following points are the distinguishing features of Proc FSEDIT:

- Used for editing SAS data sets in a Full Screen mode
- Is observation (or record) oriented
- Allows validation of fields (limited in Version 5)
- Allows extensive validation and cross validation of fields (Version 6 SCL)
- Calculated fields can be displayed (Version 6 SCL)

Full screen editing is a minimum requirement for an effective data entry program. This allows for easy editing and entry because the end user can fill in the blanks for new data and type over data that needs to be updated. Fields for each variable are set up, so keeping track of column numbers is not necessary as when inputting data into a text file.

FSEDIT is an observation oriented procedure. The user sees and edits one observation at a time. This is in contrast to Proc FSPRINT which shows several variables and observations from the data set - as may as the computer terminal will display will allow. The data is displayed in a tabular format in FSPRINT. In contrast, FSEDIT puts as many variables on the screen as it can fit. More than one screen might be needed to display all the variables. But all information a screen will be from the same observation.
To facilitate accurate data entry, it is possible to validate fields in a very limited way. This is done by setting minimum and maximum values and using SAS formats and informats. This can be done in versions 5 and 6.

Version 6 FSEDIT allows far more extensive field validation and cross validation of fields when using SCL. Version 6 is a great improvement over version 5 in terms of the ability to perform data integrity checks.

In addition to field validation, SCL allows version 6 FSEDIT to display computed fields which can make data entry easier. Displaying totals and sub-totals of variables on the observation might let a user do a quick visual check of the data. This is especially important when entering things like hours worked this week (it will usually add up to 40). The computer does not need the total since it can recalculate it when needed, but it can make catching typographical errors much easier.

In this paper, I will use a simple time tracking system to illustrate some of the points that will be made.

**Uses of Proc FSEDIT**

One of the simplest uses is to quickly examine a data set to find out what variables are on it and what their values are. It is often more convenient than using a simple Proc PRINT. Of course, in addition to looking at the values, you may change variable values, add new observations, and delete existing observations. Proc FSEDIT will supply a default screen for your data set if you are not using a customized screen. Here is an example of using this technique on the HOURS data set used in the time tracking system.

```
PROC FSEDIT DATA=SASUSER.HOURS;
RUN;
```

Command=>

```
PROJECT: ABC 011
BILLABLE: _
DATE: 04DEC89
HOURS: 5
```

As you can see, all four variables are displayed with their values.

Although this is a rather unsophisticated use of Proc FSEDIT, it is a very effective one.

In general, these are the things you can use Proc FSEDIT for:

- Edit data sets
- Create new data sets
- Print data sets in screen format
- Print form letters (with Proc FSLETTER)

Editing data sets is, by far, the leading reason for using Proc FSEDIT. However, if you use Proc FSLETTER, you may use FSEDIT and FSLETTER together for form letters as your primary FSEDIT application.

**Proc FSEDIT Options**

In order to do anything with a SAS procedure, it is necessary to know at least some of the options associated with it. Listed below are most of the options you would ever need to know about with brief explanations. For more details, consult the SAS/FSI Users' Guide.

- **DATA=** Specify SAS data set to edit. Not used with NEW=.
- **ADD** Immediately edit new observation at end of data set. The same as entering FSEDIT without the ADD option and entering the ADD command immediately.
- **NEW=** Name new SAS data set to create. Must not be used with DATA=.
LIKE:

- **LIKE=** Specify a SAS data set with a structure you want to use for a new data set. Can only be used with NEW=.

- **CMENU** Command menu is used instead of a command line. This is useful for people who only occasionally edit SAS data sets and do not remember FSEDIT commands.

- **PRINTALL** Print data set the way it looks on the screen. This provides a quick alternative to Proc PRINT for printing data sets.

- **SCREEN=** Names SAS catalog entry that stores a customized screen. An SCL program can be stored as part of the screen.

- **MOD** Directly edit the Screen before editing data. It is the same as entering FSEDIT without the MOD command and immediately entering the MOD command.

- **OBS=** Determines the observation that is initially displayed.

- **LABEL** Labels are displayed instead of variable names. This should not be used with the SCREEN= option because SCREEN= overrides LABEL. LABEL is only effective when you want the default screen with labels instead of variable names.

- Other options: **NC=, NR=, STCOL=, STROW=, TAB=, LETTER=, PRINT=, SEND=**. These can be looked up in the manual at your convenience. They are used for setting up the "window" used by FSEDIT on your screen (first 4 options), setting up your default screen (TAB=), and with FSLETTER (last three options).

Statements used with Proc FSEDIT are the usual ones: **LABEL, VAR, FORMAT, and INFORMAT.** There are two basic things to keep in mind. One is that default formats and informats are used if the format and informat statements are not used. The other is that the LABEL statement has meaning only if the LABEL option is used on the Proc FSEDIT statement.

Some things to avoid with Proc FSEDIT are the FIRSTOBS= and OBS= data set options and the WHERE statement. Since Proc FSEDIT accesses a data set one observation at a time, FIRSTOBS and OBS have no meaning to it. You directly access each observation as you want it. The WHERE statement, which is used only in version 6 of the SAS System, does not work with SAS/FSP products in version 6.03.

### Command Line Commands

Once you have started editing a data set with Proc FSEDIT, there are many commands that can be used for entering new data and locating existing data. Many of the commands you can use are listed below along with a brief description of what the command does.

#### Scrolling

- **BACKWARD** Go to previous observation
- **FORWARD** Go to next observation
- **LEFT** Go to previous screen of current observation
- **RIGHT** Go to next screen of current observation
- **n** Go to observation n
- **=n** Go to screen n of the current observation
- **=name** Go to the field of the variable name specified for the current observation

#### Editing

- **ADD** Add a new observation at the end of the data set
• CANCEL Cancel editing changes to the current observation

• CURSOR Sets initial cursor position for the screen

• DELETE Deletes the current observation by setting all variables to missing. The observation still physically exists in the data set. (Exists with missing values Ver. 5.) (Flagged for deletion in Ver. 6) These are very important points. In version 5, you must have a data step after your FSEDIT step which looks for observations having all variables set to missing; those observations should be deleted from the data set. In version 6, it is necessary to create a new data set by setting the edited data set into a data step or by sorting the edited data set. If the deleted observations are not physically deleted, you may get strange results in subsequent processing.

• DUP Duplicates current observation as a new observation at the end of the data set

• OVERRIDE Used in version 6 to override error checks. Can be disabled by programmer.

• SEARCH Searches for an observation containing all of the strings specified in variables identified by the STRING command

• SEARCH@ Searches for an observation containing any of the strings specified in variables identified by the STRING command

• RFIND Repeats the last FIND, FIND@, LOCATE, LOCATE@, SEARCH, or SEARCH@ command

**Screen Modification**

Now that you have seen how an end user can use Proc FSEDIT, it is time to look at how a programmer can set up a screen to make editing data easier.

Important considerations when designing screens are the placement of variables on the screen, the use of formats and informats, performing data validation, and, in version 6, making use of calculated fields and other features of Screen Control Language (SCL).

A useful screen will have fields to be edited most often on the left side of the screen and toward the top. This makes data entry faster because rarely changed fields do not have to be tabbed over.

It is possible to put data set variables anywhere on the screen and label the variables in any way that is desired. But remember that the end user must understand which variable each field represents in order to effectively use the FIND, SEARCH, and LOCATE commands. In simpler terms: they must know the variable name that goes with a field on the screen. This is not necessarily important if the programmer sets up a NAME and/or STRING command to be executed by screen upon entering FSEDIT.

One major difference between version 5 and version 6 is that version 6 has SCL. Other than this very substantial difference, most other aspects of the procedure are common to both versions of the SAS System.

To get into the screen modification mode, type MOD on the command line. A screen will appear asking what you wish to modify. In version 6, the screen looks like this:
Field Attributes

Listed below are some attributes that can be set for variables by using the field attribute screen. In version 6, all of these attributes could be set using SCL, but that would require programming rather than just filling in some fields on a screen. One advantage of using SCL over the menu is that using the menu is similar to hard coding your attributes. Using SCL allows attributes to be fed into an edit screen by using data sets and macro variables. The field attribute screen provides a quick, convenient way to set up some basic attributes, but it limits the error message and could be more difficult to maintain later on. Using SCL involves more effort initially, but screens can be more customized and, in some cases, more easily maintained. For example, default, minimum, and maximum values could be set or changed for several screens all at once by assigning macro variable values. Here are the attributes:

- Minimum and Maximum values
- Default values
- Video attributes - colors, reverse, blinking, etc.
- Error video attributes - attributes the field has when an erroneous entry has been made
- Required Field - must be non-missing for observation to be accepted
- Protected Field - can be seen but not edited
- Visible Field (Ver. 6) - Allows field to be made invisible. Useful for confidential data and passwords.
- Capitalized

General Attributes

In addition to attributes for each field, there are attributes that determine the overall behavior of the FSEDIT screen. Some of them are listed here:

- AUTOSAVE Number of edits or additions before the data set is automatically saved. The default is 25. It should be set at a level appropriate for the user. A fast data entry person may find 25 or even 30 acceptable. Someone who does very little data entry may find that 1, 2, or 3 is a more comfortable number.
- NAME Sets default NAME variable for use with the LOCATE command
- STRING Sets default STRING variable for use with the SEARCH command
- ERROR OVERRIDE Version 6 only. Allows the user to override validation checks implemented by the programmer.

Using Formats and Informats

Formats and informats are useful for displaying data in a useful way and entering data in a format that is easy to understand. In FSEDIT, they are essential for SAS date and time values. Date and time values are simply numeric variables, but formats and informats provide an easy way to translate SAS values into values people can easily understand. One thing that should always be remembered is that the format and informat used with a particular variable must be compatible. For example, if you use the DATE7. format for a date value, use the DATE7.
informat as well. Failure to do this will make data entry either difficult or impossible.

Another use of formats is for simple data entry checks. In the example shown below, values that are not legitimate are displayed as a question mark. Unfortunately, the bad data is still in the data set. The question mark only serves as a warning to the data entry person that the data is invalid.

```
proc format;
value $proj 'abc' = 'abc'
    'ghr' = 'ghr'
    other = '?'
;
run;
```

```
proc fsedit data=hours;
var project hours date;
format project $proj.;
run;
```

Restructuring Data to Make Editing Easier

Another technique that can make FSEDIT more effective is to restructure the data set into one that can be edited more easily. Here are some reasons you might want to do this:

- **Speed**
  Allows several observations of the original data set to be entered on one screen

- **Accuracy**
  Cross checking values across several observations of the original data set is easier.

- **Code Edit**
  The process of restructuring the data sets before and after the edit provides an opportunity for pre-edit and post-edit checks and corrections to the data using standard SAS code.

Example of Time Keeping Data Screen

No Restructuring

Command ==> 

<table>
<thead>
<tr>
<th>PROJECT:</th>
<th>ABC 011</th>
</tr>
</thead>
<tbody>
<tr>
<td>BILLABLE:</td>
<td>_</td>
</tr>
<tr>
<td>DATE:</td>
<td>04DEC89</td>
</tr>
<tr>
<td>HOURS:</td>
<td>5</td>
</tr>
</tbody>
</table>

- Reduce or eliminate typing the values of certain variables (project code in the example you will see in this paper)
Example of Time Keeping Data Screen With Restructuring

Press F1 for help and F10 to exit.

<table>
<thead>
<tr>
<th>OBS</th>
<th>PROJECT</th>
<th>BILLABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DEF 012</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DEF 012</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>GHI 321</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VACATION</td>
<td>N</td>
</tr>
</tbody>
</table>

The code that restructures the data is shown in Appendix A. The code that transforms the data back into its original form is shown in Appendix B.

As you can see, the restructured data set is much easier to edit.

Two examples of SCL are shown below. The first one uses a SAS format based on the list of legitimate project code/billable combinations to determine if an entry is legitimate. If it is not, a screen is displayed showing the legitimate combinations. The second example is similar except that the data set containing the legitimate combinations is directly examined, and the user is able to change the list of legitimate project code/billable combinations.

One very important factor to note is that macros are used in the SCL programs. It is important to remember that SCL is compiled code. A macro that is defined outside the SCL program and used in the SCL program will use the macro definition at the time of SCL code compilation. If the macro changes later, the change will not be reflected in the compiled SCL.

Look-up Table Used to validate entries

Here is a print-out of the data set containing legitimate project code/billable combinations for the simple time tracking system.

LEGITIMATE PROJECT CODES

SCL examples 1 and 2 show the SCL code used to create useful screens. Both examples use calculated fields. They both access the project data, too, but they do so in different ways. In this way, you can look at two different techniques that can be used to accomplish similar objectives.

Conclusion

We have seen how FSEDIT is used and some techniques for using it effectively. Attribute screens for variables and general attribute screens were examined for their useful features. The uses of formats, informats, restructuring data, and SCL have also been examined.

What has been demonstrated is that Proc FSEDIT is a powerful data editing tool that can be made even more effective by a programmer who is willing to try different techniques to make data editing faster, easier, and more accurate.

References

- Technical Report P-175 (User Defined Informats)
- SAS® Procedures Guide Release 6.03
SAS and SAS/FSP are registered trademarks of SAS Institute Inc., Cary, NC.

If you have any questions or comments, please feel free to contact the author at

Levine Software Systems
5525 Queen Avenue South
Minneapolis, MN 55410
(612) 926-6967

---

**SCL Example #1**

- Adds columns and rows for sub-totals
- Calculates grand total for the week
- Validates Projects
- User sees allowed projects if a bad one is entered

---

**SCL Example #2**

- Adds columns and rows for sub-totals
- Calculates grand total for the week
- Validates Projects
- User sees allowed projects if a bad one is entered
- User can add new projects
Appendix A - Preparing Data

FILENAME MKWIDE 'MKWIOE.SAS';

/* ASSIGN NUMBER OF ROWS AND COLUMNS FOR FSEDIT SCREEN. */
%LET COLS = 7;
%LET ROWS = 8;
/* ASSIGN NUMBER OF ROWS AND COLUMNS FOR FSEDIT SCREEN. */
OPTIONS NOMPRINT NOMRACENOSYMBOLGEN;
%MACRO MKWIDE;

RUN;

DATA TPLATE1;

PROC FREQ DATA=TPLATE1;
RUN;

DATA TPLATE3;

DATA INDATA;

RUN;

PROC SUMMARY DATA=INDATA MISSING NWAY;
RUN;

DATA 1EMPWD1;

DATA TEMPW2;

PROC FSEDIT DATA=TEMPW2 SCREEN=SCREEN;

Appendix B - Put data back into data set

FILENAME MKNARR 'MKNARRSAS';

/* ASSIGN NUMBER OF ROWS AND COLUMNS FOR FSEDIT SCREEN. */
%LET COLS = 7;
%LET ROWS = 8;
/* ASSIGN NUMBER OF ROWS AND COLUMNS FOR FSEDIT SCREEN. */
OPTIONS NOMPRINT NOMRACENOSYMBOLGEN;
%MACRO MKNARR;

/* PUT ALL PROJECTS WORKED ON DURING PREVIOUS 3 WEEKS ON SCREEN */
DATA TPLATE;

PROC FREQ DATA=TPLATE;
RUN;

DATA TPLATE;

PROC FSEDIT DATA=TFMPWD2 SCREEN=SCREEN;

CREATE A DATA SET THAT INCLUDES ALL THE HOURS DATA, PLUS THE
/* TEMPLATE AND AN ENTRY FOR VACATION FOR THE CURRENT DAY TO */
/* INSURE THAT THERE WILL BE AN OBSERVATION FOR THE CURRENT WEEK */
DATA INDATA;

/* SORT DATA FOR PROCESSING. */
PROC SORT;
BY DESCENDING START PROJECT BILLABLE DATE;
RUN;

/* MAKE FILE FOR DATA ENTRY. */
DATA TEMPW2;
DROP PROJECT BILLABLE HOURS DAYNUM PNUM;
IF FIRST THEN SASUSER.HOURS=
ARRAY HR(&COLS,&ROWS) %DO i = 1 %TO &COLS;
%DO j = 1 %TO &ROWS;
HR&I.C&J = .;
%END;
END;

ARRAY PROJ (&ROWS) PROJ = PROJ&ROWS;
ARRAY BILL (&ROWS) BILL = BILL&ROWS;

DATA TEMPW2;
BY DESCENDING START PROJECT BILLABLE;
IF FIRST THEN PNUM = 0;
%DO i = 1 %TO &ROWS;
%DO j = 1 %TO &ROWS;
PROJ&I.C&J = .;
BILL&I.C&J = .;
%END;
END;

/* MAKE FILE FOR DATA ENTRY. */

/* MAKE FILE FOR STORING DATA */

/* MAKE FILE FOR STORING DATA */

/* INSURE UNIQUE DATE PROJECT BILLABLE COMBINATIONS */
PROC SUMMARY DATA=INDATA MISSING NWAY;
CLASS DATE PROJECT BILLABLE;
VAR HOURS;
OUTPUT OUT = SUMHOURS SUM=;
RUN;

/* INSURE UNIQUE DATE PROJECT BILLABLE COMBINATIONS. */
PROC SUMMARY DATA=INDATA MISSING NWAY;
CLASS DATE PROJECT BILLABLE;
VAR HOURS;
OUTPUT OUT = SUMHOURS SUM=;
RUN;

/* CALCULATE START AND END OF WEEK VARIABLES. */
DATA TEMPW1;
SET SUMHOURS KEEP=PROJECT BILLABLE DATE HOURS;
START = INDEX(WEEK,DATE);
END = START + 6;
RUN;

/* CALCULATE START AND END OF WEEK VARIABLES. */

73
DO J = 1 TO ROWS;
   PROJECT = FRA(J);
   BILLABLE = BILL(J);
   HOURS = HHR(J);
   IF PROJECT NE. "" AND HOURS NE. THEN OUTPUT;
   END;
END;
RUN;

/* MAKE FILE FOR STORING DATA */

/* INSURE UNIQUE DATE PROJECT BILLABLE COMBINATIONS */
PROC SUMMARY DATA=TEMPNAMES MISSING NWAY;
   CLASS DATE PROJECT BILLABLE;
   VAR HOURS;
   OUTPUT OUT = TEMPNAMES SUM;
RUN;
/* INSURE UNIQUE DATE PROJECT BILLABLE COMBINATIONS */

/* KEEP ONLY DESIRED VARIABLES AND ASSOCIATE FORMATS AND LABELS */
DATA SASUSER.HOURS;
   IF 0 THEN SET SASUSER.HOURS;
   SET TEMPNAMES KEEP=DATE PROJECT BILLABLE HOURS;
   IF HOURS = 0 THEN DELETE;
   IF DATE = "" THEN DELETE;
   IF PROJECT = "" THEN DELETE;
RUN;
/* KEEP ONLY DESIRED VARIABLES AND ASSOCIATE FORMATS AND LABELS */
%MEND MKNARS;
%MKNARS