An Introduction to the Power of PROC FSEDIT

Mic Lajiness, The Upjohn Company

ABSTRACT

PROC FSEDIT is a powerful and versatile data editing tool available in SAS/FSP software. This interactive procedure can be used to create new datasets, facilitate data entry, allow modification of existing entries, and retrieve data based on ad hoc queries. This tutorial will provide an introduction to this procedure and will cover the following topics.

- The PROC FSEDIT statement and selected options;
- Customizing screen appearance to facilitate accurate data editing;
- Assigning special attributes to data entry fields ;
- Modification of general parameters to customize the FSEDIT environment;
- Searching for data in a SAS dataset using FSEDIT commands;
- Using simple SCL statements to create pop up windows to aid data entry;

Many examples will be shown illustrating the indicated topics and the ease with which you can gain access to the power of PROC FSEDIT.

INTRODUCTION

There is much to be gained from the use of PROC FSEDIT through the myriad of ways it can be used and the options that control its use. In fact, there is so much to FSEDIT that an in depth coverage is beyond the scope of this present treatment. The purpose of this paper is not to duplicate the information in the SAS/FSP manual but to call attention to those features and functions that the author feels provide the biggest bang for the buck! The interested reader is referred to past SUGI proceedings for more information, for example, the excellent paper by Fain, et al entitled "An Introduction to SAS/FSP Software" [1].

PROC FSEDIT is an interactive, full screen data editing procedure that is one of the five procedures that comprise the SAS/FSP family of procedures. FSEDIT can be used for creating SAS data sets and entering/changing information in SAS datasets or SAS /ACCESS views. The FSEDIT procedure is distinguished from the FSVIEW procedure by the fact that it operates on *one* observation at a time. Typically, one uses FSEDIT to add to or modify data in an already existing dataset or view.

FSEDIT can be invoked in interactive mode from the Program Editor Window in interactive mode. It can also be used in non-interactive mode using a PROC statement. It is this latter method that we will be focusing on in this paper and is illustrated below. It should be noted that in the following examples FSEDIT is run in command line mode rather than in menu mode.

```
PROC FSEDIT DATA=TEST.DATA
VAR variables-list;
LABEL variable='label';
FORMAT list;
INFORMAT list;
WHERE ...;
```

These statements simply start the FSEDIT procedure on the last dataset created. Since we are assuming a dataset already exists, we can also assume the existence of appropriate LABEL FORMAT, and INFORMAT statements. The VAR statement merely serves to limit the variables supplied to FSEDIT and is typically unnecessary. The WHERE statement is particularly useful in quickly identifying a subset of observations for modification -- more on this later.

THE PROC FSEDIT STATEMENT

There are three options that particularly useful: SCREEN= , PRINTALL, and the WHERE options.

Screen = Option

The statement::

```
PROC FSEDIT Data=TEST.DATA Screen=TEST.SCREEN1;
```

for example, creates a permanent catalog entry called TEST.SCREEN1 where customized screen information will be or has been stored. If the Screen= statement is not used any customization will ***not be saved***.

Printall Option

The statement::

```
PROC FSEDIT Data=TEST.DATA Screen=TEST.SCREEN1
        PRINTALL;
```

for example, uses the previously customized screen, TEST.SCREEN1, to print out all the observations in the dataset TEST.DATA. That is, each observation in the TEST.DATA dataset will be printed out one observation/page in the customized screen format stored in the TEST.SCREEN1 entry. This method of printout is very useful in many situations and extremely quick as well -- no fooling around with PROC REPORT or PUT statements to create customized printouts.

Where command

The WHERE command is not really an option but it is included here because it is very useful in editing subsets of observations and can greatly increase the speed with which one can access and then change data. The command:

```
PROC FSEDIT Data=TEST.DATA Screen=TEST.SCREEN1;
WHERE DATE='11/10/94';
```

for example, only brings in data from the TEST.DATA dataset with a date of 11/10/94 into the FSEDIT environment for editing. This has several advantages. First, since only the subset of interest was retrieved for editing, one cannot erroneously change data for other dates, thus providing an enhanced level of data security. Secondly, when TEST.DATA is large, using the WHERE statement greatly facilitates processing, in many cases dramatically reducing the time it takes to retrieve observations. Processing is especially enhanced when the where statement variable (in this case DATE) has an ***index***. It should be noted that one can used the command line version of the WHERE command which is discussed in a subsequent section.

Other Options

There are several other options available that will not be discussed here which include:

NEW=	OBS=
KEYS=	PMENU
LABEL	NC=
ADD	NR=
MOD	TAB=

Please refer to the SAS/FSP manual for details concerning these options.

STEPS IN CREATING FSEDIT APPLICATIONS

The usual steps involved in creating an FSEDIT application include

- Creating a customized screen
- Assigning special attributes to entry fields
- Creating SCL programs to augment standard FSEDIT
- Modifying appropriate FSEDIT general parameters

One can initiate these steps by accessing the FSEDIT Menu. This can be accomplished by entering the command MOD or MODIFY on the command line (or selecting MODIFY under the Locals menu), after executing the FSEDIT command, to obtain the menu illustrated in figure 1.

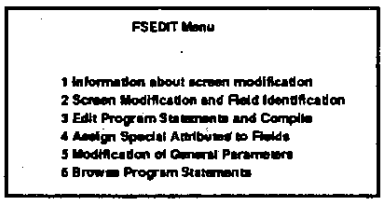

Figure 1. FSEDIT Menu

CUSTOMIZING SCREEN APPEARANCE

The Default Screen

The default screen that comes up after invoking PROC FSEDIT isn't very pretty. An example of a default screen for TEST.DATA appears in Figure 2.

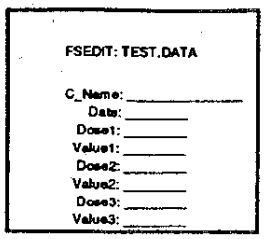

Figure 2. Default FSEDIT Screen

Creating a Customized Screen

Customizing the appearance of a screen display allows one to not only make the data entry screen look more appealing, but can actually improve the accuracy of the entered data. You can use the various capabilities of your terminal and keyboard to add effects to your screen. These attributes include blinking, underlined text, as well as reverse video and up to 8 colors. While these features help one make very attractive screens, use of these special attributes can also call attention to particular entry areas and can enhance the data entry process. On the other hand, misuse of color and the other attributes can actually adversely affect the use of FSEDIT. Therefore, make sure that your use of these special capabilities adds to an overall improvement in the look and feel of the editing environment.

Screen Modification Mode: To create a customized screen one needs to use the SCREEN= option discussed earlier. One needs to use a two-level name to create a permanent screen entry. Then, after accessing the FSEDIT Menu, one can select the second option (Screen Modification and Field Identification). One should note that there are actually two components to customizing screens. One part is the actual "painting" of the entry screen and the second part is identifying where the entry for each variable is to occur.

Once in Screen Modification mode a good idea in general is to turn line numbers on by entering the command **NUM ON**. One can then copy, delete, and move lines on the screen using text editor commands. One can change the screen in any way imaginable using underscores to represent where data should be entered. It is often helpful to have any PF-keys in effect defined on the bottom of the screen. Please note that there must be a space before and after an entry field for it to be recognized. Figure 3 contains an example of a screen after customization.

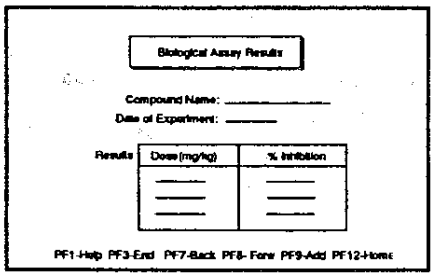

Figure 3. Customized Screen

Field Identification Mode: When finished one needs to END the session (via PF3 for example). One then automatically enters Field Identification Mode. This is where one tells the program where the various entry fields are by pressing the ENTER key on the entry field for each requested variable. One can use the UNWANTED command to tell FSEDIT you don't want to tell it where a variable is

to be entered. When all variables are identified or flagged as UNWANTED you will receive the message:

NOTE: All fields are identified

After receiving this message you should END and then return to the FSEDIT Menu. You may then select another option or return to the Data Entry Screen.

ASSIGNING SPECIAL ATTRIBUTES

The 4th entry in the FSEDIT Menu involves assigning special attributes to entry fields. Selection of this option allows one to cycle through the 15 available attributes. Move forward/backward using PF8/PF7 for example. There are a variety of attributes one can define for each of the variables on your screen. While all are useful, the ones that are most useful in the author's experience include:

JUSTIFY - This allows one to specify Left/Right/Center/Default justification. Note: Use Left (L) to left justify numeric entry fields to make it easier to "type over" incorrect entries.

INITIAL - This allows one to specify an initial value for selected fields. This saves time entering data that rarely changes.

MAXIMUM - This attribute allows one to specify the maximum value a particular numeric variable can take.

MINIMUM - This attribute allows one to specify the minimum value a particular numeric variable can take.

Other Attributes one can select appear below.

Attribute	Description
INITIAL	Defines initial values for a field
REQUIRED	Specifies a field must be filled in
CAPS	Indicates that field entries are capitalized
FCOLOR	Specifies the color of a field
ECOLOR	Specifies the color of a field when in error
FATTR	Assigns special highlighting attributes
EATTR	Assigns special highlighting attributes for fields in error
PAD	Sets Pad characters for blank entry fields
PROTECT	Indicates which fields should be protected from entry
NODISPLAY	Specifies which fields should not be displayed
NOAUTOSKIP	Controls whether the cursor will skip to the next field automatically
NOAUTOBLANK	Controls whether a field is blanked automatically

SPECIFYING GENERAL PARAMETERS

One can set parameters via option 5 that will affect the colors of your application, whether deletes/additions are allowed, whether errors can be overridden, whether the MODIFY command requires a password, default variables for searching, and many other settings as well. The corresponding screen can be found in figure 4.

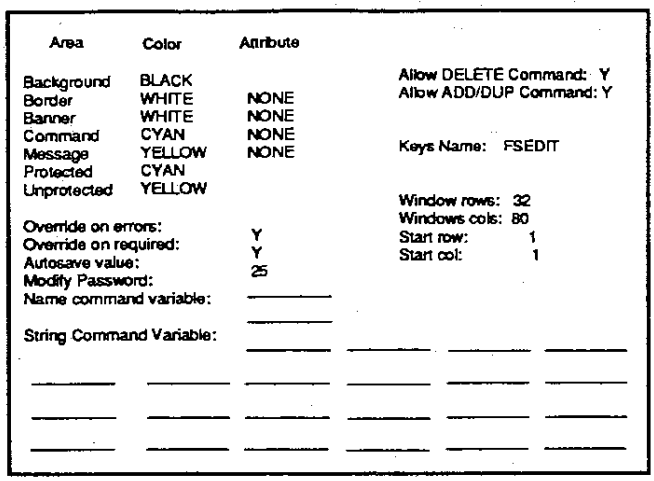

Figure 4. General Parameters Settings Screen

Allow DELETE command - Setting this parameter to N prevents the removal of observations from the dataset.

Allow ADD/DUP command - Setting this parameter prevents observations from being added.

Keys name - This name points to a PF-key definition file that maps PF-key to functions/commands. It is quite often convenient to simply modify the local FSEDIT key definitions by entering the command KEYS and then changing the key-settings to your liking. The author recommends that if one is using a command-line environment it is useful to have a PF-key defined as "?" which will retrieve past commands.

Modify Password - Setting a password will prevent modification of the customized screen by curious (but nefarious) users.

String Command Variables - One can include lists of variables here that can be searched by the SEARCH or S command. Note: Use of the SEARCH command is preferable to the LOCATE command (another way of locating observations) since substrings can be used (See Locating Observations in the next section).

LOCATING OBSERVATIONS

Locating observations in a dataset efficiently is extremely important and FSEDIT provides several ways to do this. Please note that when executing any type of search command the search starts at the current observation and will continue until the end of the file. It is possible to cycle back through the data easily by using the RFIND command which is typically accessed by PF-key. There are basically three ways to search through a dataset and locate certain observations of interest. These involve the

FIND command
NAME and LOCATE commands
STRING and SEARCH commands

FIND - There are basically 3 parts to the find command: the variable name; the comparison operator; and the value of interest. For example, assuming we are starting at the first observation of a dataset, the command:

FIND NUMVAR >= 20

will find the first observation where the variable NUMVAR has a value greater than or equal to 20. The comparison operators are all the standard ones such as =,>=,<=,<,>. One indicates "not equal" by using a ¬ = or NE comparison operator (please note that ¬ symbol is not supported by all keyboards while using NE is supported on all systems). If we are interested in character variable searches, the search value needs to be enclosed in single quotes. Thus, the following command:

FIND CHARVAR = 'TEXT'

will find the first observation where the variable CHARVAR exactly matches the value TEXT. Please note that the FIND command is useful for finding exact matches and one should use the SEARCH or LOCATE commands for substring searching. The search can be repeated by using the RFIND command which will recycle the search once the end of the file is reached.

special case: DATES Date values are numeric but one needs to enclose it in quotes as in the following example.

FIND DATE>'1/1/95'

NAME / LOCATE - The NAME and LOCATE or LOC commands are another way to search for *exact matching* character and numeric values. Usually the NAME/LOCATE commands are used for searching character values. One uses the NAME command to specify a *single* variable to be used by subsequent LOCATE commands. One can use LOCATE to find exact matches to specified numeric values for numeric variables. The values need not be enclosed in quotes. Character values need to be enclosed in single quotes, as do date values. For example, the command:

NAME CHARVAR

indicates that the character variable CHARVAR is the current target for LOCATE searches. Now the command:

LOCATE 'TEXT'

will find the first observation that exactly matches 'TEXT'. RFINDs will repeat the search as discussed previously.

NAME / LOCATE: - One can use the LOCATE: or LOC: command to find character strings that ***start*** with the indicated text. In the previous example if we used the command:

LOC: 'TEX'

we would have found the first occurrence of the substring TEX in the variable CHARVAR. One cannot use the substring feature on numeric variables.

STRING / SEARCH - The STRING and SEARCH or S commands can ONLY be used for searching character values. The basic difference from the NAME/LOCATE commands is that one can search *several* variables simultaneously for substrings that can occur *anywhere* in the text string. For example, the command:

STRING CHARVAR1 CHARVAR2 CHARVAR3

indicates that subsequent SEARCH commands will find text embedded in any of the character variables listed. Now the command:

S EX

will *not* get you a listing of local massage parlors but will instead find the first occurrence of the text string EX embedded anywhere in any of the variables CHARVAR1, CHARVAR2 or CHARVAR3. It should be noted that SEARCH@ and S@ are also valid commands that perform the same functions as the commands without the @.

OTHER USEFUL COMMAND LINE COMMANDS

There are a number of commands that are useful in the FSEDIT environment and several of these are listed below, along with a short description of their use.

N - One can access observations by observation number. For example, no matter where you are in a dataset, issuing the command:

1

will immediately send you to the beginning of the dataset. Likewise issuing the command:

999999

will send you immediately to the end of the file unless you work for the census bureau. Please note that one cannot access observations by observation number when using FSEDIT on SAS/ACCESS views.

WHERE - A WHERE statement can be issued on the command line and will have the effect of causing the subsequent display of observations that match the where condition. For example, the command:

WHERE STUDYKEY > 111

will cause the display of only those observations that have a value of the numeric variable STUDYKEY greater than 111. Please note that this "subsetting" will stay in effect until this WHERE condition is cleared by issuing the command:

WHERE

without any conditions. Also note that once you are operating in a WHERE-clause environment, access by observation number is denied. There are ways, however, to obtain a count of the number of observations in a given subset [1].

DELETE - Issuing the DELETE command from the command line will delete the currently displayed observation from the dataset.

CANCEL - One can use the CANCEL command to cancel any changes that have been made to the currently displayed observation. If changes have been made to an observation and then you move to another observation, the change cannot be revoked.

CURSOR - One can use the CURSOR command to place the cursor on any valid entry field to facilitate data entry/changes. The

way to use this command is to type the command CURSOR on the command line, move the cursor to the place you want it, and then hit the ENTER key.

SIMPLE THINGS TO DO WITH SCL

There are lots of simple things that can be done using EDIT program statements (option 3 in the FSEDIT Menu). We will be looking at a couple of examples where we will

- Set up "coded" entry fields
- Set up pop-up windows to allow selection of entry values
- Set up a simple consistency checking routine to catch errors

But first we will talk about setting up FSEDIT programs in general,

FSEDIT Programs

After accessing option 3 in the FSEDIT Menu one can define an FSEDIT program. FSEDIT programs utilize Screen Control Language (SCL). While it is possible to perform very complex functions using complicated SCL programs, there are lots of **simple and powerful** things you can do as well. There are five execution sections that can be defined in FSEDIT SCL programs. Without going into too much detail these are

FSEINIT:	which does initialization prior to displaying observations
INIT:	which does initialization prior to each observation being displayed
MAIN:	which executes every time a user modifies a field or presses the ENTER key
TERM:	which executes once for each observation before the next observation is displayed.
FSETERM:	executes when the END command has been issued

Now, let's consider some of the examples

Defining Coded Entries

Coded entries can be used to simplify the entry of long text strings. Essentially one uses a letter or number to represent a predefined response or value. For example, suppose there was a field called ACTIVITY and the acceptable values for ACTIVITY included

ACTIVE
NEARLY ACTIVE
ALMOST NEARLY ACTIVE
INACTIVE

It is in many case important to have consistent information in a field of interest. In other words, for those observations for which the third response in the above list was appropriate one would always want to find the value: ALMOST NEARLY ACTIVE. But given the length of the value it is almost certain that at some point an entry error would occur (e.g. misspelling). Thus the use of coded entries standardizes response and ensures a greater degree of consistency in data values. The following SCL code will perform this coding for the ACTIVITY example.

```
INIT: return;

MAIN:
if activity=' A'    then activity='ACTIVE'
if activity='I'     then activity='INACTIVE'
if activity='NA'    then activity='NEARLY ACTIVE'
if activity='ANA'   then activity='ALMOST NEARLY ACTIVE'

TERM: return;
```

Figure 5. FSEDIT Program for Coded Entry

After the above program is typed in, executing the END command will automatically compile the program. Warnings or errors will be indicated at this time. Please note, if one is NOT in a Display manager (DMS)environment it is not possible to review messages associated with the problem. Thus, operating in DMS mode is recommended at least until the program is debugged.

So, after the above SCL program is compiled one can return to the entry screen. Now, whenever an A,I,NA, or ANA is placed in the ACTIVITY field the value will automatically change to the appropriate response.

Creating Pop-up Windows to Aid Data Entry

An alternative to using coded entries to enhance the data entry process is to use pop-up selection windows. One simple way to do this is to first create a data set containing the acceptable entries. This dataset can be static (never changing) or dynamic (changing in response to new entries) but that is a subject for another paper. Figure 6. contains a program to create a dataset called SELECT.LIST which contains the possible values for ACTIVITY in the previous example.

```
Data SELECT.LIST; informat ACTVAL $20.;
ACTVAL='ACTIVE'; OUTPUT;
ACTVAL='INACTIVE'; OUTPUT;
ACTVAL=' ALMOST ACTIVE'; OUTPUT;
ACTVAL='ALMOST NEARLY ACTIVE'; OUTPUT;
```

Figure 6. SAS Code to Create SELECT.LIST

Once this dataset has been created we can create and test SCL code to pop-up the selection list. The SCL code in Figure 7 reads in the SELECT.LIST dataset and then displays the selection list in response to a "?" placed in the entry field for ACTIVITY.

```
INIT: dsid=OPEN('Select.List', 'I');

MAIN:
if substr(activity,1,1)='?' then
   activity=datalistc(dsid,'actval');

TERM: rc=close(dsid);
```

Figure 7. FSEDIT program to display the Selection list

It is important in Figure 7 to note that the dataset SELECT.LIST is opened in the INIT section and closed in the TERM section. The SCL function DATALISTC does the pop-up window for character variables (DATALISTN does it for numeric variables).

Figures 8,9, and 10 display the sequence of screens one would expect from execution of this code.

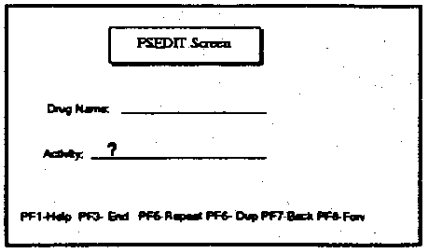

Figure 8. FSEDIT Screen with '?' entry

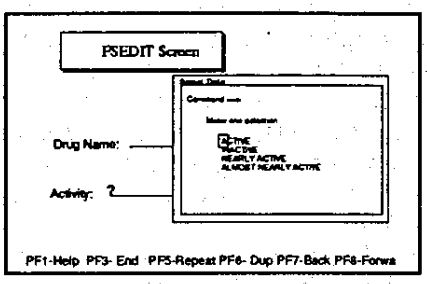

Figure 9. FSEDIT Screen with pop-up window

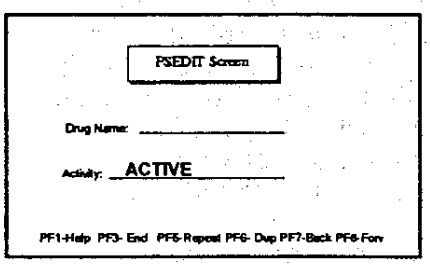

Figure 10. FSEDIT Screen with 'ACTIVE' entry

Thus, it is really not difficult to enhance your FSEDIT display with pop-up windows. The benefit in using them is to increase the accuracy and consistency of the data entry as well as improving the overall look and feel of the application.

Consistency (or other) Checking

It is possible to check the accuracy or validity of responses using simple SCL code. The classic example of a consistency error that is often given is in the case of a form that asks for the sex of the subject and whether the subject is pregnant or not. Clearly, one cannot be both male and pregnant (unless you're Arnold Schwartzenagger). So to illustrate the way one performs consistency the code in figure 11 is used.

```
INIT: return;

MAIN:
if sex='M' and pregnant='Y' then do;
  erroron sex;
  _msg_='Unless you are Arnold you can't be male and Pregnant!';
  end;
else erroroff sex;
return;

TERM: return;
```

Figure 11. FSEDIT Program to check for consistency

The above code simply checks for the subject being male and the variable pregnant being Y (yes) and then sets the erroron flag and displays a message. Only when one response or the other has been changed will the erroron flag be cleared.

SUMMARY

PROC FSEDIT is an extremely powerful feature of SAS software. One can access much of that power with just a little bit of knowledge. A useful strategy in dealing with FSEDIT, and most of the other SAS procedures, is to first learn usage at the simplest level. Then after mastering the simple options gradually build up to the complex as needs and time allow.

REFERENCES

[1].Fain, T., Gareleck, C. (1994). *An Introduction to SAS/FSP Software*. Proceedings of the Nineteenth Annual SAS Users Group International Conference. SAS institute Inc. Cray, NC 27513

[2].Elrod, JP (1994). *Counting in a PROC FSEDIT Subset.* Proceedings of the Nineteenth Annual SAS Users Group International Conference. SAS institute Inc. Cray, NC 27513

Comments and questions can be sent to the author at

Mic Lajiness
Computer Aided Drug Discovery
The Upjohn Company
Kalamazoo, MI 49001
(616) 385-7494 (work)
MSLAJINE@UPJ.COM