ABSTRACT
This paper discusses the use of the Release 6.06 SAS/FSP® FSVIEW procedure to develop end user data entry applications. Some topics to be discussed are:

- Comparing the FSVIEW and FSEDIT procedures
- Customizing the FSVIEW display
- Using formula entries to create enhanced displays
- Using Screen Control Language (SCL) in formula entries
- Accessing SAS/AF® entries within FSVIEW
- Using the WHERE processor within FSVIEW
- Sorting data within FSVIEW
- Creating new data sets within FSVIEW
- Modifying the behavior of FSVIEW by setting the control level

INTRODUCTION
Release 6.06 of the FSVIEW procedure has undergone significant changes between Releases 5.18 (mainframe and minicomputers) and 5.03 (microcomputers), not the least of which is the changing of its name from FSPRINT to FSVIEW to more closely describe its new functionality. With this new release of PROC FSVIEW, an application developer now has, in addition to PROC FSEDIT, a new tool to provide an interactive windowing environment for browsing or editing SAS data sets. However, unlike FSEDIT, FSVIEW displays data sets as a table with rows of observations and columns of variable values. FSVIEW has a rich set of commands that allow the applications developer to customize the data entry environment to suit the specific application needs.

Comparing the FSVIEW and FSEDIT procedures
The following lists some of the major differences between the FSVIEW and FSEDIT procedures:

FSVIEW:
1. Displays multiple observations at a time.
2. Provides ability to create customized screens by moving, dropping, and rearranging variables and by creating computed variables.
3. Allows use of most of SAS/FSP Screen Control Language in the formula entries except for a small subset.
4. Requires use of Screen Control Language to create an interface with the FSLETTER procedure.
5. Provides limited ability to create an ad hoc report via the cut and paste facility.
6. Writes observation to data set after any changes, such as observation or formula modifications, etc.
7. Does not allow a search for a specific variable value.
8. Provides the ability to create a new data set based on the open data set.
9. Provides the ability to dynamically sort the open data set.

FSEDIT:
1. Displays a single observation at a time.
2. Provides greater ability to create customized screens.
3. Allows use of the entire SAS/FSP Screen Control Language.
4. Provides an interface with the FSLETTER procedure to create form letters.
5. Provides the PRINTALL option to create ad hoc reports.
6. Writes observation to data set after scrolling off observation.
7. Allows a search for a specific variable value.
8. Does not provide the ability to create a new data set based on the open data set.
9. Does not provide the ability to sort the open data set.

Customizing the FSVIEW display
The FSVIEW procedure provides a variety of commands to customize the display. A summary of these commands follows:

- **VAR**: displays a subset of the data set variables (procedure invocation statement).
- **ID**: defines ID variables to be displayed (procedure invocation statement).
- **DROP**: drops variables from the FSVIEW display.
- **SHOW**: redisplays dropped variables or defines variables as either ID or scrolling variables.
- **MOVE**: redefines display order of variables.
- **FORMAT**: associates formats with a displayed variable.
- **INFORMAT**: associates informat with a displayed variable.
- **PROTECT**: protects a displayed variable from editing.
- **INITIAL**: sets initial values for an autoadd observation.
- **CURSOR**: sets cursor variable for an autoadd observation.
- **SETWSZ**: stores the FSVIEW window size.
- **DEFINE**: creates computed display variables.
- **WHERE**: subsets displayed observations.
- **SETCR**: controls cursor movement upon pressing <ENTER>.

A major advantage of FSVIEW over FSEDIT is that it displays multiple observations on one screen. FSVIEW, however, does not allow the screen customization currently available with FSEDIT. Nonetheless, the above commands can be used to customize the FSVIEW display to a much greater degree than the default display. A typical screen customization would consist of the following steps:

```sas
libname sasdata 'physical sas data library';
libname formula 'physical sas formula library';
proc fsview data=sasdata.s.sdata;
   var <Varlist>; /* subset desired variables */
   id <Varlist>; /* define id variables */
run;
```

The formula is stored in the catalog FORMULA.FORMULA with an entry name of SASDATA.

1. Use the VAR statement on procedure invocation to list a subset of variables to be displayed.
2. Use the ID statement on procedure invocation to define ID variables to be displayed in lieu of observation numbers. Note, however, that only the first sixty columns of the display may be used for ID variables.
3. Use the DROP command to drop any additional variables, although the variable list specified on procedure invocation should have been sufficient to subset the variables as desired and is easier than multiple DROP commands.
4. Use the SHOW command to reshow dropped variables or redefine ID variables back to display variables and vice versa. The SHOW command places the reshow variables at the far right of the current display in the order the variables are specified on the command line.
5. Use the MOVE command to further redefine the order in which variables are displayed. FSVIEW defaults to displaying the variables in the order in which they are defined in the data set.
6. Use the FORMAT and INFORMAT commands to associate display and input formats with variables as desired. Note that it is very important that the formats and informats are compatible. Also, in many cases it is essential that an informat be associated with a variable if a format is used.

7. Use the PROTECT command to prevent a display variable from being edited.

8. Use the INITIAL command to set initial values for an autoadded (new) observation. This causes new observations to assume the values of the INITIAL observation (similar to the DUP command except these values are used throughout the autoad process until the initial values are reset).

9. Use the CURSOR command to position the cursor on the specified variable when adding a new observation.

10. Use the WSHRINK and WGROW commands to change the window size if a reduced FS VIEW window is desired, then use the SETWSZ command to save the window size to the formula entry.

11. Finally, use the PARMS command to open the PARMS window to set colors, autoad, scrolling values, autosave value, and cursor variable. Note that changes made with the equivalent command line commands will be reflected in the PARMS window.

Using formula entries to create enhanced displays

Now that the data set variables have been moved, formatted, and so on, you are ready to define any computed variables. The DEFINE command allows you to define computed variables based on either data set variables or other computed variables. Any valid SAS expression, as well as most SAS and SCL functions, can be used in defining the computed variable. Also valid are multiple expressions separated by semicolons.

Note that computed variables are evaluated based on the order in which they appear in the data set. Thus, if a computed variable is dependent on the value of another computed variable, the dependent variable MUST appear after the independent variable or its value will be set to missing. However, computed variables can appear either before or after any data set variables that they reference. Computed variables can also be used as ID variables. Unlike computed variables in FS EDIT, computed variables in FS VIEW are always protected from editing.

The above commands are all saved to the formula entry when you end the FS VIEW procedure. Thus, the state of the FS VIEW session is preserved across procedure invocations. For example, if the application is ended in autoad mode, autoad will be active the next time the application is invoked. While this ability is desirable when creating the formula entry, it can also cause problems later if the formula entry is not protected. And, of course, the formula risks corruption or deletion if it is not protected. Furthermore, unlike FS EDIT modified screens, FS VIEW formula entries cannot be password protected. Thus, once the formula is completed, it is advisable to protect the formula from further changes by placing it in a read-only library. Note that the above formula entry was created in a different SAS library than the data set. Thus, to protect the formula entry from further change, issue the following MVS LIBNAME statement in the application invocation code:

```
libname formula 'physical name for formula library'
```

The share disposition on the above LIBNAME statement will prevent further changes to the formula entry. (Other operating systems may have a different syntax for specifying read only access to a SAS library.)

You can also use two additional commands to modify the FS VIEW display. The WHERE command subsets the displayed observations based on the values of data set variables. The SETCR command defines the behavior of the <ENTER> key, specifying cursor movement, control passing, and field modification. However, the conditions imposed by these two commands are NOT saved in the formula entry. Consequently, these two commands must be understood and entered by the end user of the application each time he or she enters the application. See below for additional explanation of these two commands.

Using Screen Control Language (SCL) in formula entries

Screen Control Language (SCL) can also be used in FS VIEW formula entries to greatly enhance the functionality of an FS VIEW application. Although a thorough discussion of all the SCL available in the FS VIEW procedure is beyond the scope of this paper, the following are those SCL statements and functions that are not allowed in FS VIEW:

- Statements and functions valid only in SAS/AF software
- ERRORMON
- ERROOFF
- PROTECT
- UNPROTECT
- CURSOR

Refer to SAS Screen Control Language: Usage, Version 6, First Edition for details on the Screen Control Language. Also see below for examples of using SCL in FS VIEW formula entries.

Note that the SCL in a computed variable will execute even if the variable is dropped from the display. This is very convenient for using a non-displayed computed variable to execute an SCL function. For example, let's say you wanted to sort a data set whenever an observation was added or modified. You could issue the following command:

```
Formula 1:
```

```c
define temp
define TEMP as a character variable with the formula:
temp = '#if modified(var1) or modified(var2)
or ... modified(varn) then
call execcmd('sort var1 var2 ... varn')
```

Now drop TEMP from the FS VIEW display. Even though TEMP is not displayed on the data set, the SORT call will execute whenever the specified variables in an observation are modified.

An FS VIEW formula executes for all observations displayed on the screen whenever the screen is displayed or redrawn. Therefore, you must be very careful in defining formula definitions that cause a screen redraw or you run the risk of creating an user-coded infinite loop. For example, if the above formula definition had been coded as:

```
Formula 2:
```

```c
temp = '#call execcmd('sort var1 var2 ... varn')
```

FS VIEW would have entered a user-coded infinite loop: on ending from the define window, FS VIEW would have attempted the sort, which would have caused a redraw of the screen, which would have caused the formula to be reexecuted, and so on. The conditional logic in the first formula definition prevents the loop since the logic is false after the sort executes. In general, unconditional execution of an SCL command that causes the screen to be redrawn should be avoided if possible.

Accessing SAS/AF entries within FS VIEW

If your site has licensed the SAS/AF product, you can create an AF entry to be accessed via the FS VIEW formula by using the "CALL DISPLAY" function in SCL. This capability greatly increases the functionality of the FS VIEW procedure. See SAS/AF Software: Usage and Reference, Version 6, First Edition for more details on creating a SAS/AF entry. Also see below for an example of calling a SAS/AF entry from an FS VIEW formula.

Using the WHERE processor within FS VIEW

Release 6.06 of the SAS System includes a WHERE processor that implements the ANSI Standard SCL "where" expressions. This standard contains a rich set of expressions and operators to subset
data according to the value of one or more variables. This processor is extremely useful and adds tremendous functionality to this latest version of the SAS System, especially when used with interactive procedures. The WHERE processor will dynamically subset observations as they are read from the physical data set, selecting only those observations that meet the WHERE criteria. This allows the application developer to subset the data the end user sees without having to create an intermediate, temporary data set. However, the WHERE processor must use actual data set variables; it cannot process computed variables within FSVIEW.

There are two types of WHERE conditions: temporary and permanent.

A temporary WHERE condition is created by the WHERE command. This WHERE condition, as its name implies, is temporary in that it can be changed, augmented, or cleared by the end user. This command is useful to a knowledgeable end user who desires to limit the display to those observations meeting certain criteria. Also, since the FSVIEW procedure does not currently have the ability to search for a specific observation, the temporary WHERE command is very useful for subsetting the data set so that the desired observation(s) can be easily located.

A permanent WHERE condition is created by either the WHERE statement or the WHERE data set option. The permanent WHERE condition, as its name implies, is permanent and cannot be modified or cancelled by the end user during this FSVIEW session. This WHERE condition is useful to the application developer who desires to impose a subsetting of data on the end user. Since the end user cannot remove this WHERE condition, this is an effective means of having one data set but limiting observations based on the WHERE criteria. For example, an applications developer could impose various permanent WHERE conditions based on the need of the end user. See the SAS Language: Reference, Version 6, First Edition for more details on the WHERE processor.

Sorting Data Within FSVIEW

The FSVIEW procedure allows the user to dynamically sort the data set while still editing it. However, since the sorted data set is actually written to the physical storage device, the data set must be in edit mode. Furthermore, since the sort updates the data set in place, the observation numbers will change if the data set contains any deleted observations. Also, you cannot sort the data set while a WHERE condition is active, nor can you sort an indexed data set.

You can sort on any combination of data set variables, just as in PROC SORT, but, as with the WHERE processor, they must be actual data set variables not computed variables. Finally, in order to accomplish the sort, FSVIEW must be able to obtain exclusive access to the data set, that is, a control level of MEMBER. FSVIEW will not be able to accomplish the sort if another window or user has shared access with your open FSVIEW data set.

Creating new data sets within FSVIEW

The FSVIEW procedure has two commands that allow the user to create new data sets: NEW and CREATE.

The NEW command in FSVIEW creates a new, empty data set by bringing up the NEW window where the user may define a new data set. This window is completely independent of the current data set being edited by FSVIEW and gives the user full latitude in creating the new data set. Once the user closes the NEW window, another FSVIEW window will be opened in edit mode with control level of MEMBER on the new, empty data set. Ending this FSVIEW session returns the user to the previous FSVIEW session. You may also specify the LIKE= option with the NEW command. This operand will pass the variable name, type, length, label, format, and informat characteristics from the LIKE= data set to the NEW window. However, this option will not work if another process has member lock on the LIKE= data set since FSVIEW will be prevented from reading the characteristics of that data set.

The CREATE command in FSVIEW creates a second data set based on the data and structure of the current, open data set. This

This command brings up the VARSORT window from which you may select or exclude any of the variables from the open data set. Note that computed variables are included in the variable list and will be written out to the data set. Also, any active WHERE condition on the open data set will affect the observations written out to the new data set. Thus, the CREATE command is very flexible and powerful in giving the user total control over the created data set: variables can be included or excluded, computed variables can be written to the new data set, and only observations that meet the WHERE condition will be written out. For example, if you were editing a data set called OLD and wanted to sort the data set on a computed variable, you could execute the following commands:

1. create new /* create data set called "new" */
2. end / * select both data sets and computed vars */
3. edit new /*
4. sort cold computed variables
5. swap /* rewrite to old FSVIEW session */
6. end / close old data set */
7. create old replace /* replace old with "new" */
8. drop the old (computed) variable, then end out
9. view new old /* return to "new" data */
10. next fsvlew /* return to "new" data */
11. end / close fsview session on "new" data */

This example merely creates a new data set containing the computed variable, sorts the data set by that variable, and copies the new data set to the old data set. Note that this example does an update in place on the old data set (step 7 above) which could be potentially dangerous. However, realize that this procedure is as safe as a DATA step, especially if the new data set is analyzed for number of observations and variables (step 3 above).

Modifying the behavior of FSVIEW by setting the control level

There are two control levels in FSVIEW: record-level locking and member-level locking.

Record-level locking (the default) means that FSVIEW only locks a single record from the data set at one time. This allows the data set to be shared with other processes (and other users if SASSHARE has been licensed). However, because FSVIEW displays multiple observations at one time, you must first lock the record (i.e., gain exclusive control of the record) before you can edit it. To do so, simply place the cursor on any character of the record and press <ENTER>. The record will then be highlighted to indicate that it is locked from other processes and users. You can now update the record. Note that you also must lock the record before you can delete it. Until you lock a record the entire FSVIEW display is protected. This ensures that you cannot edit a record that another process or user is also editing and vice versa.

Member-level locking means that FSVIEW has exclusive control of the entire data set. No other process or user may have access to the data set at this time, even in browse mode. Since no other process or user can access the data set while you have member-level locking, the danger of data collision is eliminated. Consequently, FSVIEW exhibits different characteristics with member-level locking than with record-level locking. With member-level locking the entire display is unprotected. You may tab to any variable on any observation and edit the data. You may also delete any observation or multiple observations by specifying an observation range with the delete command.

FSVIEW exhibits additional differences between record-level versus member-level editing. First, FSVIEW sets an internal "locked" flag whenever an observation has been locked. In fact, this is how FSVIEW determines the record to highlight. This also means that a formula entry will always execute whenever a record is locked since formulas execute upon screen redisplay. Second, once a record has been locked, any activity, including pressing <ENTER>, will set the modified flag. This means that a formula entry that queries the modified flag or execute whenever <ENTER> is pressed on a locked observation, whether or not the observation was actually modified. Third, FSVIEW will add a new record to the data set only if the modified flag has been set. Consequently, with record-level locking, FSVIEW will add new records to the data set...
whenever the cursor is on the autoadd observation and <ENTER> is pressed whether data was entered in the new observation or not. This can result in blank observations being added to the data set.

With member-level locking, the modified flag is set only when a variable has actually been changed. This is true for both existing and new observations. Thus, with member-level locking a new observation will not be added to the data set unless the new observation is actually modified. Also, a formula entry will execute only upon screen redisplay or actual modification of an observation.

The best way to get a good understanding of the above characteristics is to create a data set and enter the following formula, where $X$ is a variable on the data set.

**Screen 1:**

```
FS VIEW: Define command
Command ===>
```

Now add, edit, and lock observations under both record-level and member-level locking and notice the value of `CHANGED` and the results of the formula execution.

You can change the control level of the data set you are editing by issuing the MODIFY command. If you are in browse mode and issue the MODIFY command with no operands, the data set will assume record-level locking by default. Once you are in edit mode, you can specify either the RECORD or MEMBER operands with the MODIFY command to change the control level of the data set. Issuing the MODIFY command with no operands while in edit mode displays a message indicating the current control level.

**General Information**

When FS VIEW writes out its data:

It is very important to understand when the FS VIEW procedure writes out data. Because FS VIEW displays multiple observations at one time, it must write observations to disk whenever a change takes place. Therefore, modifying an observation or formula and pressing enter or a function key will cause the current screen's data to be written to the disk data set. Because of this, you cannot cancel changes made to an observation as you can in FS EDIT. Also, if data is flagged as in error, the data has already been written to the data set as a missing value. The reverse video on the screen only indicates the display value that is in error. You cannot restore the data to its previous values by entering the CANCEL command. In fact, the CANCEL command in FS VIEW will close down the procedure and save the data without updating the formula entry; this is a much different definition of the CANCEL command than in FS EDIT. Finally, be extremely careful in defining a data set variable as a computed variable. First of all, this action will cause the data set variable to be protected. However, if an error is made in the formula definition, missing values will be immediately written to the data set for the observations displayed on the screen.

The SETCR command and its effect on data entry and editing:

As mentioned earlier, the user can enter the SETCR command to define the behavior of the <ENTER> key, specifying cursor movement, control passing, and field modification. The syntax of the command is:

```
SETCR <STAY|HTAB|VTAB|NEWL|CMDPN
RET|NORET
MOD|NOMOD>
```

The operands are defined as follows:

- **Cursor Movement**
  - **STAY** cursor does not move.
  - **HTAB** cursor moves to the next unprotected field below the current line.
  - **VTAB** cursor moves to the next unprotected field below the current field.
  - **NEWL** cursor moves to the first unprotected field below the current line.
  - **CMDPN** cursor moves to the command line.

- **Control Passing**
  - **RET** returns control to the application.
  - **NORET** does not return control to the application.

- **Field Modification**
  - **MOD** marks the field as modified.
  - **NOMOD** does not mark the field as modified unless the field value is changed.

**EXAMPLES:**

**Example 1:**

This example illustrates how you can use a formula entry to capitalize character variables in FS VIEW:

```
data chartest;
  length lname fname addr1 addr2 city $20 state $2;
run;
```

```
proc fsview data=chartest;
  fsview form=chartest;
run;
```

Define the computed variable `SCL` as shown below:
Screen 2:

FSVIEW: Define command
Command ===>
Name: SCL
Type: NUMERIC
Format:
Informat:
Label:
Enter the formula below:
SCL = "";
array upperi| leame from add1
add2 city state;
do i=1 to 6; upperi(1)=upcase(upperi(1));end;

Now drop the computed variable from the display. Now whenever you enter data into the variables of the data set, they will be converted to upper case.

Example 2:

This example illustrates how you can use FSVIEW as a menu generator. You should note that this technique is merely meant to show you how to use SCL within FSVIEW. The actual example could be accomplished more effectively by using SAS/AF to create an extended table.

libname formula 'physical sas formula library';
data a b c d e: /* demonstration data sets */
do x=1 to 10;
    output;
end;
rw;
proc contents data=a b c d e out=directory (keep=mamin name)
directory a b c d e;
run;
data directory (keep=mamin name)
directory a b c d e;
where mamin name='DIREC TRI';
run;
proc fsvie data=directory (on=author)
formula=formula.direc tory
edit noadd nodelate;
Define the computed variable SCL as shown below:

Screen 3:

FSVIEW: Define command
Command ===>
Name: SCL
Type: NUMERIC
Format:
Informat:
Label:
Enter the formula below:
SCL = "";
when ('E') call fsview(filename);
otherwise _msg_="Choose E or V and Press Enter";

Now protect the FILENAME variable:
====> protect on filename

Note that a knowledgeable, vandalistic user could unprotect the filename variable and cause the application to fail. However, even with a read-only formula, you cannot disable the "protect off filename" command.

Now make the variable S the ID variable:
====> show id s

Now drop the SCL computed variable:
====> drop SCL

Your finished screen should now look like this:

Screen 4:

FSVIEW: WORK.DIRECTORY (E) command ===>
S
FILENAME
WORK.A
WORK.B
WORK.C
WORK.D
WORK.E

Now try entering values in the selection field. You will see that entering an 'E' will bring up the selected data set using FSEDIT and a 'V' will bring up the selected data set using FSVIEW. Any other characters are ignored and converted back to an underscore. Multiple selections are also permitted since the formula will execute for all displayed observations.

Example 3

This example illustrates how you can call a SAS/AF program entry from an FSVIEW formula definition and take advantage of two very useful SCL functions to validate data entry based on the values in a lookup data set. This application will validate the data entered in two fields and, if missing or incorrect, will provide the end user with a selection list of valid values.

First create the needed data sets:

libname sdata 'physical sas data library';
libname formula 'physical sas formula library';
data sdata.project (keep=status probcode descript)
sdata.verystat (keep=status sort)
sdata.veryprob (keep=probcode sort);
length status probcode $ 10 descript $ 40 sort $;
stop;
rw;
proc build cat=formula.project.verify.program;
run;
The SORT variable in the two verification data sets allow you to keep the data sorted in any desired order by matching up a sort variable to each data element.

Now create the SAS/AF program entry:

proc build cat=formula.project.verify.program;
run;
There is no need for any screen variables, so enter the following command on the command line of the DISPLAY window
====> source
then enter the following SCL program:

***** DEFINE VARIABLES TO VERIFY *****
%let var1=status;
%let var2=probcode;
%let msg1=Please Select Status;
%let msg2=Please Select Problem Code;
***** END VARIABLE DEFINITION *****
entry \%eval var1 var2 $ 20;
Since you are doing virtually the same thing for each variable to verify, placing the verification code in a macro call eliminates the need placed in an autocall library rather than the SAS/AF program entry to be recompiled for each observation displayed on the FSVIEW window. This saves code duplication and will greatly assist in program maintenance. It is very easy to use this technique. Note, however, that this macro would normally be listed together.

FSVIEW: Define command

Command =>

Now create the formula entry that will call the SAS/AF program entry:

Define the computed variable VERIFY as shown below:

Screen 6:

When you end out of the define screen, the formula will be compiled and will execute for each observation displayed. Since your project data set is empty, you automatically entered AUTOADD mode when you first invoked the application. So, enter any character in either the status or problem code field to have the application prompt you with valid values for both these fields. Within the selection list, simply tab to the desired value and press <ENTER>. The new observation will then be added to the data set.

This application illustrates some areas of the FSVIEW procedure that warrant further explanation:

1. The FSVIEW screen display routine executes the formula entry(s). Once the data for the computed variables are calculated, the screen is then written as a whole to the terminal. However, if the formula entry causes a display of another window (as this example does with the selection list window), the FSVIEW display routine may pause while the formula executes. This can create the situation where, on initial invocation of the application, the selection list is displayed before the remainder of the FSVIEW screen. This occurs because the screen display routine has not completed and is waiting to return from the formula entry. In this situation, you will have a selection list to update observations that have not yet been displayed! In this example, this occurs when the project data set is edited outside the application and bad data that do not meet the selection list criteria are entered in the first screen's display. If this situation occurs, issue the RESHOW command within the selection list window to refresh the screen and display the FSVIEW window.

2. Earlier in this paper, the timing of the formula calculations and the subsequent danger of user-coded infinite loops was explained. This application has the potential for an annoying "mini loop" (two iterations). This phenomena occurs during add mode when the SAS/AF program entry does not take into account the
NEW observation that is added automatically. To illustrate: you enter autoadd mode and enter a character into the status field. The formula executes and displays the selection list. You make a selection for both status and probcode. Now, however, the display must be updated with your new values and the next NEW observation that will be added automatically. So, the formula reexecutes for the new observation and, since the NEW observation contains missing values, the selection list is redisplayed. Since the actual display is not updated until the formula ends, these two displays of the selection list occur without an update of the FSVIEW display. Thus, the impression is that it requires two iterations of the selection list in order to update the NEW observation when, in fact, the first iteration updates the first NEW observation and the second iteration values for the next NEW observation are dropped. The line in the SAS/AF entry

```
do while (not ne and (status eq ' '));
```
takes into account that the values for both STATUS and PROBCODE will be missing for a NEW observation. Checking the concatenation of STATUS and PROBCODE rather than each one individually allows the end user to only modify one of the variables to get a selection list for all variables.

3. Because of the need to ignore checking the NEW observation, the user is able to enter an observation with missing values for STATUS and PROBCODE by pressing the space bar and then pressing <ENTER>. However, because the application is using member-level locking, the NEW observation must be modified in order for a blank observation to be added to the data set.

4. Conditional logic could have been added to the formula entry to call the AF program only on modified observations. However, the modified flag is not set upon initial invocation or when scrolling to a new screen. Thus, the formula would not automatically execute if there were bad data in the data set from external editing. More importantly, the modified flag is not set for the NEW observation; even if data are entered into its fields, the modified flag remains false when the observation is written to the data set. Thus, adding conditional logic to the AF program call would disable the data checking during add mode.

5. Because of the do loop in the AF program entry, you cannot cancel out of data entry as long as there are bad data on the screen display. This can be particularly disconcerting if the selection lists are displayed without the FSVIEW display (see #1 above) but otherwise is often desirable. Note that this is a factor of the application design not of the FSVIEW procedure.

6. Note the commented lines

```
  * put 'before verify' _all_; /* debugging code */
  * put 'after verify' _all_; /* debugging code */
```
The PUT statement is very helpful in debugging your FSVIEW formula entries and AF program entries. The above statements allow you to trace the logic and execution of the formula entry and program call. FSVIEW sends the output of this command to the MSG window by default. After the initial invocation of FSVIEW and output to the MSG window, you may want to execute the AUTOPOP OFF command in the MSG window to prevent future output from popping the MSG window to the foreground. The output from the above lines can be lengthy and is proportional to the number of variables and observations displayed.

CONCLUSION

The changes and enhancements made to the FSVIEW procedure in Release 6.06 SAS software have greatly increased the functionality of the procedure over the FSPRINT procedure from earlier releases. FSVIEW is an ideal complement to the FSEDIT procedure and helps make the SAS/FSP product line well-suited to your interactive data entry needs.

Acknowledgements

The author wishes to thank Patti Brideson, Carl LaChapelle, Ann Carpenter, and Anna Kushner for technical review, Liz Malcom for editing, and Bud Whitmeyer and Maureen Fitzgerald for aid in typesetting the paper.

SAS, SAS/FSP, SAS/AF, and SAS/SHARE are registered trademarks of SAS Institute Inc., Cary, NC.