SAS/FSP® SOFTWARE: Highlights of the 6.06 release

Carl LaChapelle, Annette Harris
SAS Institute Inc., Cary, NC

Abstract

This paper highlights the new features in release 6.06 of SAS/FSP software. These features include the use of the WHERE clause with the FSEDIT procedure and FSVIEW procedure, examples of using SCL with PROC FSEDIT and PROC FSVIEW and the use of the SCL source level debugger to assist in the development process.

Introduction

Version 6.06 of SAS/FSP software introduces many enhancements that make the full-screen procedures more flexible and more powerful. This paper documents changes that have been made to the Global Commands as well as to the FSEDIT, FSVIEW, FSLIST and FSLETTER procedures. The debugging process using the SCL source level debugger is also illustrated.

Global Commands

Global Commands are now available that invoke the SAS/FSP procedures. The syntax for these global commands begins with the name of the respective procedure and takes as arguments dataset, letter or filenames as appropriate for the procedure.

If you do not specify a dataset, letter or filename, or specify a question mark, a selection list of datasets, letters, or filenames will be displayed. From the selection list you can choose the dataset, letter, or filename you want to work with. For example:

FSEDIT	< dataset|? screen >
FSBROWSE	< dataset|? screen >
FSVIEW	< dataset|? formula >
FSLETTER	< letter|? >
FSLIST	< fileref|? >

The FSEDIT, FSBROWSE and FSVIEW command allows optional specification of a dataset name and a screen or formula name. However, if you specify a screen or formula catalog, you must specify the data set.

Data Set Options

Where ever you can specify a SAS data set in a SAS/FSP procedure, you can specify SAS data set options. The data set options should be specified in parenthesis after the SAS data set name. For example, if you wanted to use FSEDIT to view a SAS data set containing all of the employees in your company, but you only wanted to look at employees in the sales department, you could use the FSEDIT command and specify the WHERE data set option. The command might look like:

```
FSEDIT employee(where=(dept='SALES'))
```

For a complete list of SAS data set options refer to the *SAS Language Guide*.

WHERE Clause Processing

The WHERE clause creates a subset of observations for processing by the procedure. The WHERE clause is any valid SAS expression involving one or more of the variables in the data set. Observations must satisfy the specified conditions to be processed by the procedure. The data set itself is not subsetted. The WHERE clause is available in the FSBROWSE, FSEDIT, FSVIEW, and FSLETTER procedures.

A WHERE clause can be specified at the invocation of the procedure or from the command line once the procedure is active. If specified at invocation, the WHERE clause cannot be removed from within the procedure. A WHERE clause specified by the WHERE command can be added to, or removed while in the procedure.

The syntax of the WHERE command is:

WHERE [option] [where expression]

The available options are:

ALSO — The ALSO option allows you to edd the conditions specified in the expression to any existing where expressions.

CLEAR — The CLEAR option removes all temporary where expressions.

UNDO — The UNDO option removes the most recently added where expression. You can also remove the previous where expression by just issuing the WHERE command by itself.

If none of the options are specified, the where expression will replace any temporary where expression previously in effect.

The following example shows WHERE clause processing in FSVIEW.

Command ===> where origin = 'GMC'

OBS	ORIGIN	MODEL	MPG
1	GMC	ELDORADO	3
2	GMC	CHEVETTE	5
3	GMC	CITATION	4
4	GMC	MALIBU	3
5	FORD	FAIRMONT	3
6	FORD	MUSTANG	3
7	FORD	PINTO	4
8	JAPAN	CIVIC	5
9	CHRYSLER	GRAN FURY	2
10	CHRYSLER	HORIZON	4
11	CHRYSLER	VOLARE	2
12	GMC	FIREBIRD	1
13	EUROPE	DASHER	5
14	EUROPE	RABBIT	5
15	EUROPE	DL	4
16	FORD	ESCORT	4
17	JAPAN	PRELUDE	4

Command = = = > where also mpg > = 4

OBS	ORIGIN	MODEL	MPG
1	GMC	ELDORADO	3
2	GMC	CHEVETTE	5
3	GMC	CITATION	4
4	GMC	MALIBU	3
12	GMC	FIREBIRD	1

Command = = = >

OBS	ORIGIN	MODEL	MPG
2	GMC	CHEVETTE	5
3	GMC	CITATION	4

PROC FSEDIT

Keys

You can now identify modified key settings by using the KEYS= option on the PROC FSEDIT statement.

PROC FSEDIT DATA=PERM.MYDATA KEYS=UPDATED;

The above statement indicates that UPDATED.KEYS is to be read from your SASUSER.PROFILE; or if you are creating a new set of keys, they should be written to SASUSER.PROFILE. Default key settings are stored in the SASHELP.FSP catalog under the name FSEDIT.KEYS. If you specify a screen name on the FSEDIT statement, the name of the keys to be used is stored with the screen. The search sequence is the current screen catalog, the SASUSER.PROFILE and, lastly, SASHELP.FSP.

Modification Menu

Option 6, a new option, has been added to the FSEDIT screen modification menu. It allows you to browse your Screen Control Language (SCL) program without forcing the program to be compiled when you exit the source screen.

WHERE Clause Processing

When a WHERE command is in effect, the SEARCH, FIND and LOCATE commands work only within the context of the subset. BACKWARD and FORWARD commands move between the previous and next observation within the subset.

The FSEDIT window will help you identify when a WHERE condition is in effect. The word WHERE will appear in the upper right-hand corner if a temporary condition is in effect. If a WHERE condition has been specified at procedure invocation, the word (subset) will follow the dataset name in the window.

PMENU Function

Customized Command Menus can be created using PROC PMENU, available in Base SAS software. These command menus provide your user the ability to make choices by cursor positioning. Because you create these menus, they are specific to your application. There are two ways to take advantage of these customized menus. The first way is to use the SCL PMENU function in your program. The other way is to issue the SETPMENU command followed by the name of the pmenu you want displayed.

PROC FSVIEW

PROC FSPRINT has been renamed PROC FSVIEW, and many new features have been added. The changes have been concentrated in two areas. The first is data entry facilities, the second is allowing you to create variables whose values are computed from other variables in the data set. Also, you can save information about your session in a FORMULA entry.

Data Entry Features

Several new features have been added to enhance FSVIEW as a data entry tool. Some of the enhancements are:

1. FSVIEW automatically adds new observations to the data set as you enter values.
2. You can protect individual variables from modification.
3. You can automatically position the cursor on a specific variable when a new observation is displayed.
4. You can rename variables for display purposes.
5. You can change from browse mode to edit mode without having to exit the procedure.

Computed Variables

To create a computed variable, you give FSVIEW a formula to use in determining the value for the variable. For example, using the following data set, suppose you want to create a new variable that is the sum of QTR1, QTR2, QTR3, QTR4. Issue the DEFINE command.

Command = = = > define total = qtr1 + qtr2 + qtr3 + qtr4

OBS	QTR1	QTR2	QTR3	QTR4
1	$25,000	$25,600	$28,000	$30,000
2	$34,000	$37,000	$33,000	$37,100
3	$8,000	$8,900	$9,350	$9,350

The new variable TOTAL is added to the display, and its values are the sums of QTR1 - QTR4 in each observation.

Command = = = >

OBS	QTR1	QTR2	QTR3	QTR4	TOTAL
1	$25,000	$25,600	$28,000	$30,000	$108,600
2	$34,000	$37,000	$33,000	$37,100	$141,100
3	$8,000	$8,900	$9,350	$9,350	$35,600

RESET enables you to remove any formulas you have assigned, and SAVE saves the dataset or formula used to define new variables.

You can use the REVIEW command to see all of the formulas you have assigned.

Formula Entries

FSVIEW now saves a great deal of information about the display and the variables in a formula entry. Even though the type of the entry is FORMULA, there do not have to be any formulas for the information to be saved.

The information that is stored consists of:

1. Any formulas that have been assigned to a variable by the

DEFINE command.

2. The order in which the variables are displayed.
3. The current variables and observations displayed.
4. The window size and position when the SETWSZ command was issued. The window colors are also saved.
5. The default values used by several of the commands.

Other New Features

The WHERE clause can also be used with PROC FSVIEW to subset data for the duration of your display only. It does not permanently alter the data set. As with the FSEDIT procedure, if a where clause is in effect '(subset)' will display as part of the window name.

The CREATE command creates a new SAS dataset with some or all of the variables in the current data set.

PROC FSLETTER

New enhancements to the FSLETTER procedure include a spelling checker, the availability of WHERE clause processing, commands to open and close datasets, and a new field attribute.

Spelling Checker

The SPELL and DICT commands controls the interactive spell checker. The SPELL command controls spell checking, suggestions and substitution of suggestions. Options to the SPELL command allow you to check the spelling of the current word, a previous or subsequent word, or all words in your text. The spell checker has a 50,000 word dictionary.

The DICT command maintains and accesses auxiliary dictionaries. It can be used to customize the standard dictionary sent with the spell checker or create a new dictionary of specialized words.

WHERE Clause

The WHERE command allows you to print letters using observations that meet specific conditions. For example, if you wanted to send a letter to all employees in a certain department just issue the appropriate WHERE command and then send your letters.

DATA and CLOSE Commands

The DATA command opens a SAS dataset for use by the SEND command. A dataset name is optional, but if no name is specified on the DATA command, the name of the currently open dataset is displayed.

The CLOSE command closes datasets opened with the DATA command. It does not close a dataset being used by PROC FSEDIT.

FLOW LINE Attribute

The FLOW LINE attribute allows you to flow the text of a single line. Flowing will begin with the first column of your entry field and continue to the end of the line. This is particularly helpful with names that appear in the address section of a letter. When the first and last name values are stored as different variables, flow line will permit appropriate spacing of the names without affecting any following address lines.

PROC FSLIST

Performance enhancements have been made to PROC FSLIST that result in quicker display of the screen. Also, variable length records can now be viewed without truncation of data. Additionally, new options allow you to indicate the type of carriage control characters to be used in formatting the data for display.

The carriage control options are:

CC Use the native carriage control characters.

FORTCC Use FORTRAN-style carriage control.

NOCC Treat carriage control characters as regular text.

SOURCE LEVEL DEBUGGER

The SCL Source Level Debugger has been created to ease your job of application debugging. Logic errors can be determined at run time. Features include displaying the source program while indicating which line is being executed and watching values of variables as they change. You can modify variable values, control further program execution, and suspend program execution as errors are encountered. Many other commands for locating problem areas exist. There is even a menu-driven command facility if you are unfamiliar with specific debugger commands.

A partial list of available commands and their functions follow:

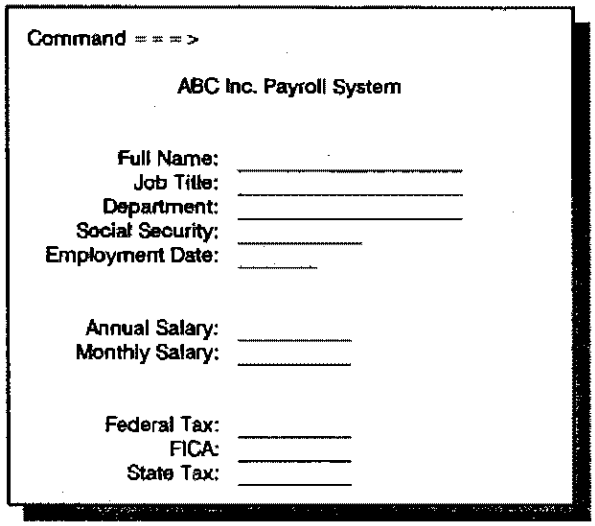
COMMAND	FUNCTION
ENTER	Redefine the ENTER key.
EXAMINE	Examine a variable's value.
SET	Modify variable's value.
BREAK	Set breakpoints, suspend program execution.
GO	Resume execution after program suspension.
JUMP	Resume execution from a given executable statement.
STEP	Monitor program statement by statement.
WHERE	Trace active programs.
TRACE	Trace program flow.

If you want to assign a debugger command to a function key you must precede the debugger command with SCL. For example, if you wanted to set a function key to examine the variable amount, you would need to set the function key to 'SCL EXAMINE amount'.

To activate the debugger, use the DEBUG option on the PROC FSEDIT statement or DEBUG ON can be specified on the command line once inside the procedure. The debugger is terminated with the QUIT command.

Let's walk through a short example. This example shows a simple payroll system, for a company which pays its employees on a monthly basis. The system just allows you to change employee's salaries. When an employee's salary is updated the SCL program determines the new witholding amounts for each check.

The FSEDIT screen for the application is shown below. The monthly salary variable is a computed variable; all other variables are in the data set.

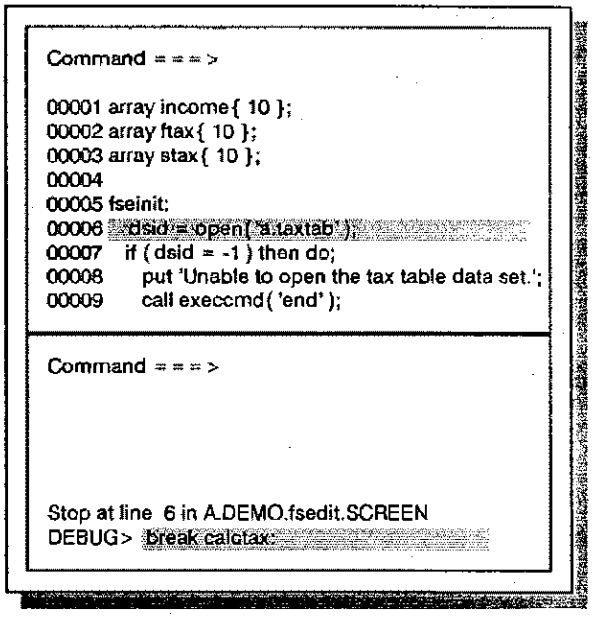


The SCL program is listed below. The FSEINIT label is executed when the procedure is first invoked. The INIT label is executed before each observation is displayed. The MAIN label is executed when a field on the screen is modified and the ENTER key is pressed. The TERM label is executed when you leave an observation. The FSETERM label is executed when you leave the procedure.

```
00001 array income{ 10 };
00002 array ftax{ 10 };
00003 array stax{ 10 };
00004
00005 fseinit:
00006   dsid = open( 'a.taxtab' );
00007   if ( dsid = -1 ) then do;
00008     put 'Unable to open the tax table data set.';
00009     call execcmd( 'end' );
00010     return;
00011     end;
00012
00013   nobs = 0;
00014   do while ( fetch( dsid ) = 0 );
00015     nobs = nobs + 1;
00016     income( nobs ) = getvarn( dsid, 1 );
00017     ftax( nobs ) = getvarn( dsid, 2 );
00018     stax( nobs ) = getvarn( dsid, 3 );
00019     end;
00020
00021   return;
00022
00023 init:
00024   monsal = salary / 12;
00025   link calctax;
00026   return;
00027
00028 main:
00029   link calctax;
00030   return;
00031
00032 term:
00033   return;
00034
00035 fseterm:
00036   if ( dsid ^= -1 ) then
00037     call close( dsid );
00038   return;
00039
00040 calctax:
00041   do i = 1 to nobs;
00042     if ( salary < = income{ i } ) then
00043       leave;
00044     end;
00045
00046   fica = monsal * 0.065;
00047   sttax = monsal * stax{ i };
00048   fedtax = monsal * ftax{ i };
00049
00050   return;
```

After you have compiled the program with the DEBUG option, you are returned to the MOD screen. When you end from the MOD screen, the debugger will be invoked. The following screen shows how your screen will look when the debugger is running. The upper half of the screen shows the source of your SCL program. The bottom half of the screen is where you enter the debugger commands.

From the listing of the program source above, you can see that the amount of withholding is calculated by the CALCTAX label. Let's set a break point at the CALCTAX label so that we can watch what is happening.



Now that we have set the break point let's tell the debugger to execute the program until it gets to the CALCTAX label. To do this issue the GO command.

```
Command ===>

00001 array income{ 10 };
00002 array ftax{ 10 };
00003 array stax{ 10 };
00004
00005 fseinit:
00006   dsid = open( 'a.taxtab' );
00007   if ( dsid = -1 ) then do;
00008     put 'Unable to open the tax table data set.';
00009     call execcmd( 'end' );
```

```
Command ===>


Stop at line  6 in A.DEMO.fsedit.SCREEN
break calctax;
Set breakpoint at line 41
DEBUG> go
```

Since the FSEDIT screen has not yet been displayed we know that we have come to the CALCTAX label through the INIT label. Since all tax calculations are made using the variable MONSAL let's check its value to make sure it's correct at this point.

```
Command ===>

00040 calctax:
!       do i = 1 to nobs;
00042     if ( salary < = income{ i } ) then
00043       leave;
00044     end;
00045
00046   fica = monsal * 0.065;
00047   sttax = monsal * stax{ i };
00048   fedtax = monsal * ftax{ i };
```

```
Command ===>

Stop at line  6 in A.DEMO.fsedit.SCREEN
break calctax;
Set breakpoint at line 41
go
Break at line 41 in A.DEMO.fsedit.SCREEN
DEBUG> examine monsal
```

The value of MONSAL is correct at this point. So let's issue the GO command again. The debugger will continue executing the program until the CALCTAX label is called again.

```
Command ===>

00040 calctax:
!       do i = 1 to nobs;
00042     if ( salary < = income{ i } ) then
00043       leave;
00044     end;
00045
00046   fica = monsal * 0.065;
00047   sttax = monsal * stax{ i };
00048   fedtax = monsal * ftax{ i };
```

```
Command ===>

Set breakpoint at line 41
go
Break at line 41 in A.DEMO.fsedit.SCREEN
examine monsal
MONSAL = 2333.33333333333
DEBUG> go
```

After the INIT label is executed the FSEDIT screen is displayed. All of the values are correct at this time.

```
Command ===>

                ABC Inc. Payroll System


          Full Name:  VANCE, MARTHA S.
           Job Title:  PROGRAMMER
          Department:  RESEARCH & DEVELOPMENT
      Social Security:  333-44-5555
    Employment Date:  15SEP84


       Annual Salary:  $28,000.00
      Monthly Salary:  $2,333.33


        Federal Tax:  $350.00
               FICA:  $151.67
          State Tax:  $256.67
```

Now let's change Martha's salary, and press ENTER.

```
Command = = = >

                ABC Inc. Payroll System


          Full Name:  VANCE, MARTHA S.
          Job Title:  PROGRAMMER
         Department:  RESEARCH & DEVELOPMENT
     Social Security:  333-44-5555
  Employment Date:     15SEP84


      Annual Salary:  30500.00
     Monthly Salary:  $2,333.33


        Federal Tax:  $350.00
               FICA:  $151.67
          State Tax:  $256.67
```

When a field on the screen is modified and the ENTER key is pressed, the MAIN label is executed. Since the MAIN label calls the CALCTAX label the debugger stops the program and gives us control. Let's check the value of MONSAL again.

```
Command = = = >

00040 calctax:
!       do i = 1 to nobs;
00042     if ( salary < = income{ i } ) then
00043       leave;
00044     end;
00045
00046  fica = monsal * 0.065;
00047  sttax = monsal * stax{ i };
00048  fedtax = monsal * ftax{ i };
```

```
Command = = = >

Break at line 41 in A.DEMO.fsedit.SCREEN
examine monsal
MONSAL = 2333.33333333333
go
Break at line 41 in A.DEMO.fsedit.SCREEN
DEBUG> examine monsal
```

As you can see MONSAL's value has not changed. If you look at the source you can see that MONSAL's value is only calculated by the INIT label. So, to fix our bug we should move the calculation of MONSAL's value to the CALCTAX label to make sure we always use the correct value.

```
Command = = = >

00040 calctax:
!       do i = 1 to nobs;
00042     if ( salary < = income{ i } ) then
00043       leave;
00044     end;
00045
00046  fica = monsal * 0.065;
00047  sttax = monsal * stax{ i };
00048  fedtax = monsal * ftax{ i };
```

```
Command = = = >

MONSAL = 2333.33333333333
go
Break at line 41 in A.DEMO.fsedit.SCREEN
examine monsal
MONSAL = 2333.33333333333
DEBUG> quit
```

As you have seen, the SCL Source Level Debugger makes it much easier to find and correct bugs in your applications.

CONCLUSION

Version 6.06 SAS/FSP software contains many new features that result in more power and flexibility. Major changes to each of the full-screen procedures have been discussed, in addition to global changes that affect all of the procedures. The FSCALC procedure is not available in the 6.06 release of SAS/FSP software. It will be available, with many new enhancements in a subsequent release.

NOTE: For specifics on changes described here as well as other enhancements, consult the appropriate version 6.06 reference manual. For more information on the SCL debugger, see the 'SCREEN CONTROL LANGUAGE USERS GUIDE AND REFERENCE'. Note that this debugger is also available for your SAS/AF applications.