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.

Data Set Options
Wherever 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.
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 and, lastly, SASHELP.FSP.

Modification Menu
Option 6, a new option, has been added to the FSEDIT window. 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

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

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 original 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 if a WHERE clause is in effect '(subset) win display only. It does not permanently alter the data set.

The SPELL command can be used to customize the standard dictionary sent with the spell checker or create a new dictionary of specialized words. It can be used to 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.

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 withholdings 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.

**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.

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.
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 TERML 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
tobs = 0;
00014 do while (fetch(dsid) = 0);
00015 nols = nobs + 1;
00016 income(nobs) = getvarn(dsid, 1);
00017 ftax(nobs) = getvarn(dsid, 2);
00018 stax(nobs) = getvarn(dsid, 3);
00019 end;
00020 return;
00021
00023 inlt:
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 fics = monsal * 0.055;
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.

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.

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.

After the INIT label is executed the FSEDIT screen is displayed. All of the values are correct at this time.
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: 333-44-0555
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 CALC TAX label the debugger stops the program and gives us control. Let's check the value of MONSAL again.

Command == >

00040 calc tax:
  if (salary <= income{ i } ) then
  leave;
  end;
  fica = mensal * 0.065;
  sttax = mensal * stax{ i };
  fedtax = mensal * ftax{ i };

Command == >

Break at line 41 in A.DEMO.fsedit.SCREEN
examine mensal
MONSAL = 2333.33333333333

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.

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