Automating Creation of Subsets in SAS/FSP® Using Pulldown Menus, SCL and Macros
Steve Hinnenkamp, Johns Hopkins University

Abstract
While browsing and updating datasets using SAS/FSP software, users frequently wish to create subsets containing specific records of interest. This may be a cumbersome task, however, if users are unfamiliar with field names and the syntax requirements of the WHERE command. To simplify the creation of subsets, you may use pulldown menus as a front-end to WHERE clause processing. This paper will show a way to automate execution of WHERE commands through the use of pulldown menus, SCL and macro commands. Upon completion of the paper, you will have templates you may use to implement your own subsetting menus.

Introduction
SAS Institute’s FSP products enable users to browse and edit datasets in either single-record or table format. While browsing and editing datasets, users frequently need to issue temporary WHERE clauses, either to find a specific record or to display only those records meeting specific criteria. By default, these WHERE clauses must be issued from the SAS command line or using the search item on the SAS menu bar. Both of these methods require that users know variable names and values as well as syntax requirements that differ by variable type. While this may be manageable for experienced SAS programmers, it is cumbersome and confusing for users with limited experience.

As the individuals responsible for managing users’ access to SAS datasets, we can simplify temporary WHERE clause processing using a combination of pulldown menus, screen control language and macro language. This article shows one way that you may use these tools to implement a subsetting front-end for your users.

As you will see below, in our subsetting system we use pulldown menus to supply variable names, values and operators to two templates, a macro template that is used to process subsetting commands while browsing databases using FSVIEW, and a screen control language template for use while using FSEDIT and FSBROWSE.

To adapt this system to your own applications, you need only modify the pulldown menus so they contain the variable names and values for your database(s). The underlying SCL and macro templates are usable in their current format and require no initial changes. As you expand the scope of your subset menu structure, you may have to add or modify template modules, but in their current form, the templates may be used to automate WHERE clause processing for a wide range of data elements.

Example Application
The subsetting system referenced in this article was initially developed to allow staff at Johns Hopkins University to browse databases containing information on research proposals and awards from/to JHU faculty. We will use this application to provide a conceptual framework for the article, although one should note at the outset that our primary objective is to show the reader how they might modify the routines to implement their own subsetting interface.

Selecting a Browse Type
While browsing databases it is helpful to have the option of browsing data in table or single-record format because each browse type serves different purposes. For example, when users want to view all database elements for a specific record, it is easiest to browse the database in single-record format using FSEDIT and a custom screen designed to cleanly display the data fields. On the contrary, when users want to view lists of information, such as total award amounts for all of an investigator’s projects, it is easiest to browse the data in table format using FSVIEW.

Display 1: Selecting a Browse Method
Display 1 shows a block menu that allows users to select their desired browse method (see Appendix 1 for complete code). When users select View Tables, the following code is submitted to initiate an FSVIEW session that sets piname as an identifying field and activates the pmenu paper.screens.fsview.

    call execcmd('fsview paper.sampledb; show id piname; setmenu paper.screens.fsview');

Display 2 is an example of the resulting FSVIEW session. Although our menu bar only contains the Subset and Goback items, when you generate your own menus you may want to add additional items for scrolling and related commands.
If the user selects View Records, an fsedit call is processed to begin a single-record browse session. Notice that we do not specify the pulldown menu in the fsedit call. Rather, we list the name of the custom screen and, as shown in Appendix 3, we activate the pmenu from within the screen SCL.

```plaintext
_dataset="paper.sampledb";
call fsedit(_dataset,
  "paper.screens.browse.screen", "browse");
```

Display 3 shows the resulting FSEDIT browse screen. Using this display, let's review the database variables and corresponding WHERE clauses users may want to issue while browsing the dataset. This will provide the foundation for our discussion of subsetting menus and templates.

### Sample WHERE Clauses

#### Numeric Fields

While looking at this screen, consider the types of subsets users may want to see while browsing the database. Next, think how you would issue a WHERE clause to obtain the desired subset. Finally, consider how the WHERE clauses would differ by variable type, but would be nearly identical for similar variables. As you will see below, we have used this similarity between variables to construct a limited number of subsetting modules that create WHERE statements for a wide-range of data elements. First, let’s look at some sample WHERE clauses.

#### Date Fields

When issuing a WHERE clause for numeric, non-date variables, the command syntax is simple, and requires only that you provide the variable name and the subsetting value. So, if we wanted to find all records where the value of the variable `palltot` is greater than $1,000,000 we would issue the command:

```plaintext
where palltot > 1000000
```

#### Character Fields

When issuing WHERE commands for character variables, the primary syntax requirement is that character strings must be enclosed in single or double quotes. With this in mind, let’s look at three subsetting commands we may want to execute while browsing the dataset.

**One Variable, Multiple Values.** Because there are a variety of valid values for the status field including Not Funded, Withdrawn and Transferred, users might want to limit the browse session so that only Awarded/Active and Pending records are displayed. To implement this filter one would issue the following WHERE command:

```plaintext
where status = "Pending" or status = "Awarded/Active"
```

**One Value, Multiple Variables.** In other situations, users might want to browse only those records that are contracts or subcontracts. Because this information is recorded in two different data elements, we must include both variable names in our subset command as follows:

```plaintext
where contract = 'X' or subcont = 'X'
```
**String Search.** Finally, users might want to browse only those records where the variable `projdept` contains the letter M. To limit the browse session to records containing this string we would issue the following command.

```
where projdept contains 'M'
```

Note that each of these examples represent a different type of `WHERE` command, but that each command could be used for other database variables simply by changing the variable name, value(s) and, possibly, the operator. For example, to browse only those records where the project ended before January 1, 1997 we would issue the following command.

```
where pallto < '01JAN97'd
```

Similarly, to view only those records with a project purpose of research or training, we would issue the following command.

```
where research = 'X' or instruc = 'X'
```

As we explain below, our system provides a means to dynamically create these `WHERE` clauses based upon a user’s subset criteria. This is done by creating a series of pulldown menus and dialog boxes that contain variable names, values and operators. Users simply click or enter their selection in the dialog box and all subset parameters are passed to the subsetting templates, where the subsetting commands are created and executed. We will begin with a discussion of the pulldown menus, and proceed with a review of the `subview` macro that is used to subset `FSVIEW` browse sessions. Finally, we will look at the SCL routine used to execute `WHERE` commands within `FSEDIT`.

**Creating the Pulldown Menu**

The `PMENU` procedure is used to create a menu system that provides user’s subset criteria to command templates. By embedding variable names, values and operators within the pulldown menu structure, we simplify `WHERE` clause creation for users because they don’t need to know variable names, enter values (except when entering values in text boxes) or adhere to proper syntax. Users simply indicate their selections in dialog boxes and from there SCL and macro programs create and issue the `WHERE` commands.

Let’s begin by looking at the basic menu structure with its corresponding code. We will then look at example dialog boxes for each type of subsetting command noted above.

**Note:** For the purposes of this article we will demonstrate how to create menu bars, pulldown menus and dialog boxes that may be used to obtain subset criteria from users. Although we will briefly describe `PMENU` syntax and commands, we will not provide an exhaustive summary of the `PMENU` procedure. For that we refer readers to relevant documents and articles listed below.

**Menu Bar and Pulldown Menus.**

Display 4 shows the pulldown menu structure during an `FSVIEW` browse session. Notice that the first pulldown contains options to the `WHERE` command. These allow users to create a new subset, refine an existing subset, undo existing subset criteria or clear all subsetting conditions. If `New` or `Also` are selected, second- and third-level menus are activated that contain links to individual dialog boxes. These menus are identical in appearance, although the underlying `pmenu` code for `New` and `Also` varies slightly so we may distinguish between the options.

```
Display 4: Top level Pull-Down Menus
```

The second-level pulldown represents categories of variables based upon data type. When any items are selected from this pulldown, a third-level menu is activated that contains entries for each of the variables. For example, display 5 shows the third-level menu activated when the `General` item is selected.

```
Display 5: The General Pulldown Menu
```

Display 6 shows the initial portion of the `pmenu` code used to create the above menu structure. Let’s review the relevant segments.

1. Here we define a menu called `fsview` that will be saved as `paper.screens.fsview.pmenu`. The menu bar contains two items, `Subset` and `Goback`.

2. When the `Subset` menu item is selected, a submenu called `subset` is activated. This submenu contains four items: `New`, `Also`, `Undo` and `Clear`. If `New` or `Also` are selected, additional submenus, called `first` and `second`,
are activated. If the user selects Undo or Clear, the appropriate command is executed to remove the most recent WHERE clause or to clear all existing temporary WHERE clauses.

If New is selected a submenu called first is activated to display categories of data elements. Notice that each category references another menu that is activated upon selection.

When General is selected, an additional menu is activated to display a list of data elements. Any of the items selected from the General pulldown activate a dialog box in which the user may specify their subset criteria.

Dialog Boxes
The meat of this system lies in the individual dialog boxes, and you must define a dialog box for each data element to be used in subsetting. Below we provide examples of five types of dialog boxes. These examples correspond to the various WHERE clauses that we discussed above.

Character Variables using _charvar
To select all records with a status of Awarded/Active or Pending one would select Status from the pull-down menu. This activates the dialog box shown in Display 7. This dialog is comprised of a series of checkbox entries that represent all possible values for the variable status. Users may select one or more items for their subset simply by clicking the desired value or, if using a character-based interface, scrolling to the item and pressing <Enter>.

libname paper '/adm/slh/work/nesug';
proc pmenu cat=paper.screens;
  menu fsview;
    item 'Subset' menu=subset;
    item 'Goback' selection=G;
    selection G 'end';
  menu subset;
    item 'New' menu=first;
    item 'Also' menu=second;
    item 'Undo' selection=R;
    item 'Clear' selection=U;
    selection R 'where undo';
    selection U 'where clear';
  menu first;
    item 'General' menu=general;
    item 'Dates' menu=dates;
    item 'Dollars' menu=dollars;
    item 'Codes' menu=codes;
  menu general;
    item 'Record ID' dialog=id;
    item 'Status' dialog = status;
    item 'Sponsor Types' dialog=sptypes;
    item 'Purpose' dialog=purp;
    item 'Type' dialog=typ;

display status '%subfview(_charvar,status,,
41|42|43|44|45|46|47|48|49)',
  text #2 @6 'Select Project Status';
  checkbox #4 @8 'Awarded/Active';
  checkbox #5 @8 'Awd/Not-Begun';
  checkbox #6 @8 'Awd/Terminated';
  checkbox #7 @8 'Pending';
  checkbox #8 @8 'Disapproved';
  checkbox #9 @8 'Not Funded';
  checkbox #10 @8 'Withdrawn';
  checkbox #11 @8 'Unknown';
  checkbox #12 @8 'Transferred';
  text #14 @2 'Press <Enter> to make a Selection';

display status '%subfview(_charvar,status,,
41|42|43|44|45|46|47|48|49)';
  text #2 @6 'Select Project Status';
  checkbox #4 @8 'Awarded/Active';
  checkbox #5 @8 'Awd/Not-Begun';
  checkbox #6 @8 'Awd/Terminated';
  checkbox #7 @8 'Pending';
  checkbox #8 @8 'Disapproved';
  checkbox #9 @8 'Not Funded';
  checkbox #10 @8 'Withdrawn';
  checkbox #11 @8 'Unknown';
  checkbox #12 @8 'Transferred';
  text #14 @2 'Press <Enter> to make a Selection';

Display 6: Defining the Pulldown Menus

If New is selected a submenu called first is activated to display categories of data elements. Notice that each category references another menu that is activated upon selection.

When General is selected, an additional menu is activated to display a list of data elements. Any of the items selected from the General pulldown activate a dialog box in which the user may specify their subset criteria.

Display 7: Dialog Box for the Status Variable

The pmenu code used to create the status dialog is listed in Display 8. The code consists primarily of a series of checkbox statements, each of which contain a possible value for the variable status. The guts of the code lies in the dialog statement listed on the first two lines.

Display 8: Pmenu Code to Define Status Dialog Box

Dialog Statement. After stating the name of the dialog box (status), the dialog statement references a command macro, subfview, that takes four parameters. The parameters are used to create a WHERE clause as follows. (Note: the double percent sign in the macro call is required because a single % percent sign is used to reference radiobox selections in PROC PMENU).

1. The first item, _charvar, is the keyword we send to the subset templates to specify the type WHERE clause to create.
2. The second item, status, is the variable name that will be included in the WHERE clause.
3. In this example, the third parameter is null. When the user has the option of selecting an operator, it will be sent to the macro as the third parameter.
In pmenu code, checkbox selections are referenced using an ampersand and the checkbox number. The fourth parameter (&1|&2|...|&n) is a pipe-delimited string that will contain all the values selected by the user. Our subsetting templates will parse the string and use the values to populate the WHERE clause.

Character Variables using _charx
The previous example demonstrates subsetting when using one or more values for the same variable. This example demonstrates creating a subset when we want to view records where one or more variables contain a specific value. Consider the case where a user wants to browse only records that are contracts or sub-contracts. To do so, they would select Type from the General pulldown, activating the dialog box shown in Display 9.

![Display 9: Project Type Dialog](image)

When we look at the code used to create the dialog we see that is quite similar to the previous example (see Display 10). There are several notable differences, however.

1. The dialog statement references a new keyword, _charx, indicating that this WHERE clause will be piped through a different module within our subsetting templates.

2. In this instance, both the second and third parameters are null. The third parameter is null because the user does not have the option of selecting an operator. The second parameter is null because the variable names (grant, contract and subcont) are being submitted to the subview macro as the fourth parameter.

3. We introduce a new pmenu option here, substitute. This option allows us to display the word Sub-Contract in our checkbox, but still submit the variable name, subcont, to the subsetting macro.

```
<dialog type='%%subfview(_charx,,&1|&2|&3)'>
  text #2 @6 'Select Project Types';
  checkbox #4 @10 'Grant';
  checkbox #5 @10 'Contract';
  checkbox #6 @10 'Sub-Contract' substitute='subcont';
  text #8 @2 'Press <Enter> to make a Selection';
</dialog>
```

![Display 10: Pmenu Code to Define Project Type Dialog Box](image)

Numeric Variables using _numeric
Dialog boxes for subsetting numeric variables require a text box for entering a value, and a series of radiobuttons that allow users to select an operator. For example, if a user wants to create a subset containing records where total funding exceeds $1,000,000, they would select Total Project from the Dollars pulldown, activating the dialog shown in Display 11. Using the dialog box, they would enter the dollar value and toggle the desired operator as shown.

![Display 11: Total Dollars Dialog Box](image)

The code to create this dialog is shown in Display 12. There are several notable additions in this example.

1. The user has an opportunity to select an operator. Their selection will be referenced in the dialog statement with %1. We set the default operator to Equal To and use a series of rbutton statements to list all valid operators. We use rbutton rather than checkbox in this case because rbutton only allows one selection, while checkbox allows multiple selections.

2. The macro quoting function %str is used in the dialog statement to quote the dollar value in case the user submits special characters with the value (i.e., $ and ,). The dollar value is entered into a 15 character textbox that we defined, and is referenced in the dialog statement as @1.

```
data dollars;
  item 'Total Project ' dialog=prptot;
  item 'Total Direct ' dialog=prpdird;
  item 'Total Indirect' dialog=prpindir;
<dialog type='%%subfview(_number, palltot, %1, %%str(@1))'>
  text #2 @1 "Please enter the desired amount:"
  text #2 @34 LEN=15;
  radiobox default=1;
  rbutton #4 @13 "Equal To" substitute="eq";
  rbutton #5 @13 "Less Than" substitute="lt";
  rbutton #6 @13 "Less/Eq To" substitute="le";
  rbutton #7 @13 "Greater Than" substitute="gt";
  rbutton #8 @13 "Greater/Eq To" substitute="ge";
  rbutton #9 @13 "Not Equal" substitute="ne";
</dialog>
```

![Display 12: Pmenu Code to Define Project Total Dialog Box](image)
Date Variables using _date

When specifying subset criteria for dates, the dialog box is almost identical to that used when subsetting other numbers. The only difference is the keyword listed in the dialog statement. Instead of piping dates to _numeric, we note the keyword _date in the dialog to ensure proper formatting of our date variables. An example dialog box and the corresponding code are shown in Displays 13 and 14.

Display 13: Example Dialog for Date Fields

```
menu dates;
item 'Project Start' dialog=pallbeg;
item 'Project End' dialog=pallend;
dialog pallbeg '%%subfview(_date, pallfrom, %1, %%str(@1))';
text #2 @1 "Please enter the date in MMDDYY format:";
text #2 @41 LEN=8;
radiobox default=1;
rbutton #4 @13 "Equal To" substitute="eq";
rbutton #5 @13 "Less Than" substitute="lt";
rbutton #6 @13 "Less/Equal To" substitute="le";
rbutton #7 @13 "Greater Than" substitute="gt";
rbutton #8 @13 "Greater/Equal To" substitute="ge";
rbutton #9 @13 "Not Equal" substitute="ne";
```

Display 14: Pmenu Code to Define Date Dialog

FSVIEW and the subfview macro

Now that we have provided examples of the various types of dialog boxes, let’s look at the subfview macro that is used to process users’ requests. The macro, shown in Display 17, is explained below.

The macro subfview takes four parameters from the PMENU dialog statement. These parameters are used to create a WHERE command. The parameters are:

- **Keyword** - specifies the module used to process the code. For example, if subsetting by a date field, keyword will be _date.
- **Variable** - contains the variable name to use while constructing the WHERE command. In some instances (i.e., the _charx module) this value will be null because the variable names are provided in the values macro-variable.
- **Operator** - contains the operator to use in the WHERE command. This will be a null value if the user does not have the option of selecting an operator.
- **Values** - contains the values to be used in the WHERE clause. In some instances, multiple values will be provided in pipe-delimited format. In these instances, each value is parsed from the macro-variable and appended to the WHERE clause.

Miscellaneous character variables with _code

In the character variable examples cited above we never allowed the user to select a subsetting operator. There are instances when subsetting character variables, however, where users may want to choose an operator. This may be done by using the subfview macro with _code as the keyword. For example, in some instances users may want to list all records where the department value begins with the letter M, while in other situations they may want to view only those records from department M.21. For these situations, we use a dialog box that allows users to specify a search operator (see Displays 15 and 16).
Here we initialize the temporary variables I, ARG and ARG1 for use in do loops.

Here we determine if the user wants to issue a new WHERE clause or modify an existing subset using WHERE ALSO. The pmenu dialog statements are constructed so keyword will be begin with one underscore (i.e., _charvar) if a new subset or with two underscores if a modified subset (i.e., __charvar). If alsoflag is set, keyword is shortened using the %substr function.

This module is used to process generic numeric variables, such as dollar amounts. First, we use the compress function to remove dollar signs and commas. Then we construct the WHERE clause using the value provided by the user.

This module is used to process date variables. First, we use the compress function to remove unnecessary characters such as dashes, slashes, underscores and periods. Next, we use inputn to identify the value as a date. Finally, we convert the value from mmddyy6. format to date7. format, which is the required format when issuing WHERE statements using date values.

The _charvar module is used to create WHERE clauses when we want to match specific character strings for a single variable. For this module, the values provided from the pmenu may look something like this:

```
_charvar,status,,|||Pending|Withdrawn|||
```

Because there may be multiple values in the value macrovariable, we use %scan to parse the individual values. We then enclose the respective values in quotes using the %sysfunc and quote() functions, combine the value with the variable name and generate a where clause as follows:

```
status= "Pending" or status = "Withdrawn"
```

The _charx routine is used when we want to match non-null fields for one or more variables. For example, we may want to find all records where the research field has been marked with an X. For this module, the input from the pmenu might look as follows:

```
_charx,,,Research||
```

Note in this case that &variable and &operators are null values. This is because the WHERE clause operator is provided directly within the module and the variable names are actually included in the &values macrovariable.

The _code routine is used to process general text entries such as department code and sponsor number. The pmenu dialog statement might send the following:

```
_code,projdept,eq,M.20
```
The routine begins by upcasing and removing periods from the value. Then the WHERE clause is constructed using the variable name, operator and value provided by the user.

**Case insensitive searches.** Note here that we have simplified text entry for the user by making the query case-insensitive and also by removing periods from the search string. To do this we have upcased and compressed the incoming value, and we have issued a WHERE command that will upcase and compress the database elements while processing the WHERE command. In this way we allow users to enter m21 as their search criteria when they are really searching for values of M.21. This makes using the system even easier for our users.

- Here we execute the WHERE clause which creates our subset. Depending upon the value of alsoflag, we either create a new subset or modify an existing subset.

**FSEDIT Sessions and SCL**

The above narrative explains how to use the `subview` macro to generate WHERE clauses while browsing tables. Now we will demonstrate how the code can be adapted to handle subsetting in FSEDIT sessions. All of the code listed below is SCL code that runs behind the FSEDIT session. To create this code you must first define a screen that will be used for the FSEDIT session. After specifying the screen, you may access the SCL window using the `modify` command from within your FSEDIT session.

**FSEDIT vs. FSVIEW Pulldown Menus**

The pulldown menus used in FSEDIT are nearly identical to those used in FSVIEW. Because we listed and explained these above, we will not cover `fsedit.pmenu` in this section. The complete code for `fsedit.pmenu` will be available from the FTP site listed below, and readers may peruse that code if interested.

Let us note, however, that the primary difference between `fsview.pmenu` and `fsedit.pmenu` lies in the dialog statements. Whereas our `fsview.pmenu` dialog statements send parameters to the `subview` macro, `fsedit.pmenu` only needs to send the parameters to the command line. From there the SCL processes the items and creates the WHERE command. So, to modify your `fsview.pmenu`, simply change your dialog statements to remove the macro call as listed below. Also note that null values should now be sent as empty double-quotes, rather than as empty comma-delimited fields.

```sas
dialog typ '%%subview(_charx,,&1|&2|&3)';
dialog typ '_charx "" &1|&2|&3';
```

**FSEDIT SCL**

The full code for the FSEDIT SCL is displayed in Appendix 3. Due to space limitations, we will only briefly explain the code in the following section.

1. To use custom commands in FSEDIT it is necessary to issue the `control always` statement, which allows `MAIN` to execute whenever a user presses <Enter>, even if the command line contains items unrecognized by the SAS system.

2. Here we associate our menu with the window and execute the command to turn on pulldown menus.

3. Here we open a copy of the dataset we are browsing. We use this to count the number of records meeting the subset criteria. If no records meet the criteria, we proceed no further. Otherwise, we issue the WHERE clause and display a message stating the number of records included in the subset.

4. Here we execute UNDO and CLEAR commands from the `pmenu`. Notice how we execute the command on the background dataset with `where(dsid,“undo”)` and then we issue the WHERE command to the FSP dataset with `call execcmd(wclause)`. This allows us to count the records in the subset and display a message to the user.

5. If the command string begins with an underscore we assume that it is a subsetting command from the `pmenu`. The command line is parsed and values are assigned to the SCL variables `keyword`, `variable`, `operator` and `values`.

6. The `CHECK` routine ensures that at least one valid value was submitted as part of the subsetting command.

7. If a valid value was submitted, we call the `MKWHERE` section to create the WHERE command based upon the values for the respective variables.

8. Here we issue the WHERE command against the background dataset and test to determine if any records meet the subset criteria. If the subset contains records, we execute the WHERE command against the FSEDIT dataset we are viewing and issue the COUNT command to display to the message line a count of the number of records in the subset. (Note: The COUNT command is not shown in Appendix 3, but it is included in the SCL available via FTP).

9. If no records meet the subset criteria, we write a note to the message line, and remove the WHERE clause from the background dataset.
Conclusion
In this article, we have shown one way to automate subset creation for your users. If you like this system, you can easily adapt it to your own needs by modifying the pulldown menus so they contain the variable names and values for your database. No initial changes are required to the SCL and macro subsetting templates, although as you expand your menu structure to meet specific needs, you will likely need to add additional modules to your subsetting templates. Given the background provided in this article, you should be able to do that with relative ease.

Code Availability
You may obtain copies of the code related to this article by connecting via anonymous FTP to resource.ca.jhu.edu. Login using the logname anonymous, and send your email address as the password. All related programs will be available in the directory pub/nesug97.

References and Additional Information


SAS and SAS/FSP are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Contact Information
Steve Hinnenkamp
Johns Hopkins University
Research Information Systems
708N Wyman Park Center
Baltimore MD 21218
slh@jhu.edu

Appendix 1

```
INIT:
return;

MAIN:
choice=1;
do while (choice ne 0);
  choice=block('Nesug 97 Paper','Select Browse Method',1,'View Records', 'View Tables', ' ', ' ', 'End', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 110, 54, 0, 0, 0, 116, 0, 0, 0, 0, 0, 0);
  if _status_ ^= "C" and _status_ ^= "E" then select(choice);
    when(0);
    when(1) do;
      _dataset="paper.sampledb";
      call fsedit(_dataset,
           "paper.screens.browse.screen",
           "browse");
    end;
    when(2) do;
      call execcmd('fsview paper.sampledb;
                   show id piname;
                   setpmenu paper.screens.resviews');
    end;
    when(6) choice=0;
    otherwise;
  end;
return;

TERM:
return;
```
Appendix 3 - FSEDIT SCL

length wclause $200
count $5
errmsg $70
keyword $10
variable $8
operator $10
values $200
arg $30;

FSEINIT:
  control always;
  rc = pmenu('paper.screens.fsedit.pmenu');
  call execcmd('pmenu on');
  dsid = open("paper.sampledb","IS");
  return;
INIT:return;

MAIN:
  if word(1,"u") = "UNDO" then do;
    sysrc = where(dsid,"undo");
    wclause = "where undo;"
    call execcmd(wclause);
  end;
  if word(1,"u") = "CLEAR" then do;
    sysrc = where(dsid,"clear");
    wclause = "where clear;"
    call execcmd(wclause);
  end;
  if substr(word(1),1,1) = "_" then do;
    keyword = word(1);
    variable = word(2);
    operator = word(3);
    call nextword();
    values=word(3);
    errflag=0;
    link check;
    if errflag ne 1 then link mkwhere;
    if errflag = 1 then do;
      _msg_ = errmsg;
      refresh;
      goto nosubset;
    end;
    /* Subset only if records meet criteria */
    sysrc = where(dsid,wclause);
    obs = attrn(dsid,"any");
    if obs > 0 then do;
      wclause = "where " || wclause || ";
      call execcmd(wclause);
    end;
    else do;
      _msg_ = "NOTE: No records meet the subset criteria. Try again.";
      refresh;
      sysrc = where(dsid,"undo");
      goto nosubset;
    end;
    if sysrc then _msg_ = "SCL " || sysmsg();
    else call execcmd("COUNT;");
    NOSUBSET:
    end;
TERM: return;
FSETERM:
  rc=where(dsid);
  rc=close(dsid);
  return;

MКWHERE:
  alsoflag = 0; i=0;
  if substr(keyword,2,1) = "_" then do;
    alsoflag = 1;
    keyword=substr(keyword,2);
  end;
  select(keyword);
  when("_number") do;
    value=inputn(compress(values, ',','best15.'));
    errflag = 1;
    else wclause=variable||' '||operator||' '||left(put(value,best15.));
  end;
  when("_date") do;
    value=inputn(values, 'mmddyy8.'));
    if value = . then do;
      errmsg = "NOTE: Invalid date. Please reenter.;"
      errflag = 1;
      end;
      else wclause=variable||" "||operator||" "||put(value, date7.);||"u;"
    end;
  when("_charvar") do;
    do until (arg = "");
      i=i+1;
      arg = scan(values, i, '|');
    if i = 1 then
      wclause=variable||"="||" '"||arg||"'";
    else
      if arg = " " then
        wclause=wclause||" or "||variable||" "||value||"'";
      end;
    end;
  when("_charx") do;
    do until (arg = "");
      i=i+1;
      arg = scan(values, i, '|');
    if i = 1 then
      wclause=variable||"="||" "||operator||" "||arg||"'";
    else
      if arg = " " then
        wclause=wclause||" or "||variable||" "||value||"'";
      end;
    end;
  when("_code") do;
    values = upcase(compress(values,'.'));
    wclause="upcase(compress(" || variable ||" ','.''))|| operator ||" "||quotes(values);
    end;
  otherwise;
  end;
  if alsoflag = 1 then wclause = "also " ||wclause;
  if length(wclause) > 199 then do;
    errmsg = "NOTE: Where clause too long.;"
    errflag = 1;
    end;
    return;
CHECK:
  check = compress(values,'|');
  if check = "" then do;
    errmsg = "NOTE: Please enter a subsetting value.;"
    errflag = 1;
    end;
    return;