Counting in a PROC FSEDIT Subset
John Paul Elrod, Johns Hopkins University

INTRODUCTION
When using the WHERE command to display a subset of data in PROC FSEDIT or PROC FSBROWSE, there is normally no way to know how many observations are in the subset being viewed without scrolling through all the records. The techniques shown in this Code Corner paper provide a way to simply issue a COUNT command to have SAS/FSP return a note on the message line saying:

Command ===> count
NOTE: 524 Observations are in this Subset.

This SAS/FSP Screen Control Language program implements a COUNT custom command and an interception of the PROC FSEDIT WHERE command that facilitates the display of the observation count.

SCL CODE
The first step in the code is to issue the CONTROL ALWAYS command, which is needed to create any custom command in SAS/FSP.

The next step is to open a second data set identical to the one that is being displayed by the SAS/FSP procedure. The FSEINIT and FSETERM sections contain the code to open and close the second data set. To do this, the SCL open function must know the name of the data set that is currently being displayed by the SAS/FSP procedure. Because there is still no SCL function or method for determining the name of the data set being used by the SAS/FSP procedure, the data set name must be either passed to the SCL program via a macro variable and SYMGGETO function, as shown in the example below, or hard-coded in the SCL program.

%LET dsname = sample.data;
PROC FSEDIT DATA = sample.data
SCREEN = sample.screen.fsedit.screen;
RUN;

WHERE Command Interception
In order to count the records in a subset, the WHERE clause entered on the command line must first be applied to the secondary copy of the data set. This is done by intercepting the WHERE command issued by the end user and then applying it to the secondary data set. The first WORDO function and a DO loop are used to detect and process the WHERE command issued by the end user. (This differs from the technique previously presented by this author at NESUG '92, which required the use of a SUBSTITUTE WHERE command with a more complex syntax due to limitations in SAS® version 6.07.)

Once the code has detected a WHERE command, it first parses the command arguments, i.e. the WHERE clause, by using a DO UNTIL loop to concatenate each argument word retrieved with a WORDO function into the wclause variable. The NEXTWORDO function call clears each word from the command stack as it is processed. The complete WHERE clause is then applied to the secondary copy of the data set using a WHEREO function.

In the process of intercepting the WHERE command above, the code has cleared the command stack. If control was returned to PROC FSEDIT at this point, the procedure would never see the original WHERE command that was issued and the data set being displayed would not have the WHERE clause applied to it. To avoid this problem, the code must place the complete original WHERE command back on the command stack before returning control to PROC FSEDIT. To do this, the WHERE clause is appended to the word "WHERE" and stored in the variable wcommand. The CALL EXECCMDI(wcommand,"noexec") function is used to pass the contents of wcommand back to the command stack for processing against the primary data set that is being displayed.

The EXECCMDI() routine is used instead of EXECCMD() so that the command will be processed immediately before control returns to the SCL. This does not normally, however, prevent the immediate re-execution of the MAIN section. If the MAIN section is allowed to re-execute, it would cause an infinite loop as each WHERE command was intercepted, processed, and passed back to the command stack to be re-intercepted again. The "noexec" option to EXECCMDI() is an undocumented feature that allows the command stack to be processed immediately but prevents re-execution of the MAIN section, and therefore the infinite loop.

The COUNT command is placed on the command stack via a normal CALL EXECCMD() routine so that the record count will be automatically displayed after each use of the WHERE command. This line can be omitted if this is not a desired feature.

Count Command
The code for the COUNT command simply determines if a WHERE clause is in effect for the secondary data set using
the `attrn(dsid,"issubset")` function. If none is in effect, the
code uses an `attrn(dsid,"nobs")` function to retrieve the
data set record count. If a subset is in effect, the code
uses a `do while` loop and `fetch()` function to actually read
and count all observations that are in the subset, and
displays the message to the end user via the `_msg_`
variable. Note that it is actually the records in the
identical secondary data set that are counted, not the
primary data set being displayed by PROC FSEDIT.

PERFORMANCE
Depending on the size of the data set and the size of the
subset, the time required to fetch and count the records
may vary greatly, from instantaneous to an unacceptably
long delay. Performance can be greatly increased,
however, if the data set and the variables used in the
where clause are indexed.

RESILIENCE
The previous version of this paper warned of a potential
problem whereby incorrect count figures could be
displayed. This was caused by the situation where the
data set being displayed and the secondary data set
used by the SCL program became accidentally "out of
clock." Because the actual SAS/FSP `where` command is
now used instead of a substitute command, it is virtually
impossible for the two data sets to become out of sync.
The count command is therefore now much more
reliable.

CONCLUSION
This Code Corner paper has shown a way to count
records in a PROC FSEDIT `where subset` that is easy to
use and automatic. The code exploits new as well as
undocumented features of SAS® version 6.09 Screen
Control Language.

The ability to know how many observations are in a
subset has been a feature much appreciated by our end
users. This algorithm has been successfully used at
Johns Hopkins for the past three years. The code can
easily be added to any existing SCL program.

AUTHOR
John Paul Elrod
Johns Hopkins University
300 Whithead Hall
Baltimore, Maryland 21218
(410) 516-8732
jpe@jhuspo.ca.jhu.edu

SAS® and SAS®/FSP® are registered trademarks of SAS Institute
Inc., Cary, North Carolina

SCL SOURCE CODE

```sas
length dsname $ 18;
length wclause $ 70;
length wcommand $ 84;
length count $ 5;

FSEINIT:
  control always;
  dsname = symget("dsname");
  dsid = open(dsnarne,"i~");
  return;

INIT:
  return;

MAIN:
  if word(l,"u-"="WHERE") then do:
    call nextword();
    wclause = "";
    do until (word(l) = "");
      wclause = wclause || word(l) || " ";
      call nextword();
      end;
    sysrc = where(dsid,wclause);
    if sysrc then _msg_ = sysmsg();
    wcommand = "WHERE " || wclause;
    call excecmd(wcommand,"noexec");
    call excecmd("COUNT");
    end;
  if word(l,"u")="COUNT" then do;
    call nextcmd();
    insubset = attrn(dsid,"issubset");
    if insubset then do;
      numobs = attrn(dsid,"nobs");
      count = trim(left(put(numobs,S.}));
      _msg_ = "NOTE: " || count ||
              " Observations are in this File."
    end;
    else do;
      sysrc = rewinds(dsid);
      counter = 0;
      do while (fetch(dsid,'NOSET') ^= -1);
        counter + 1;
        end;
      count = trim(left(put(counter,5.}));
      _msg_ = "NOTE: " || count ||
              " Observations are in this Subset.";
      end;
  end;
  return;

TERM:
  return;

FSETERM:
  call close(dsid);
  return;
```

SCL SOURCE CODE

1 This code is available via anonymous FTP from
jhuspo.ca.jhu.edu as /pub/nesug/subset.scl
Coder's Corner Coding Challenge Problems
(Solutions to these problems can be found on page 1695.)

Problem 1.
You have a unique numeric id value or key (e.g., SSN) and you need to map that to a valid SAS name (e.g., to the name of SAS variable or to a catalog entry of the form SASUSER.NOTES.xxxx.SOURCE where xxxx is the mapped value). How would you do it?

Problem 2.
In an SCL program you have issued a DATALIST to get a list of values that the user has selected. There is no limit on the number of selections. You need to subset the data with a WHERE clause limiting it to the values the user selected. Since you can't assume that the clause will fit into 200 characters, or even three sets of 200 characters (since the SCL WHERE functions can have three clauses passed to it), how can you subset the data set and use it in your SCL program to include only the selected observations?

Problem 3.
How can you ensure the elimination of trailing and leading blanks when using a PUT statement with leading text, a variable length numeric variable, followed by continued text, like this:

   put 'Leading Text' the-variable-goes-here 'Following Text';

Problem 4.
You need to display two graphs on the same page and they have the same X axis. You only want the axis on the graph on the top half of the page to display axes values but need to specify the axes range for both. How would you do this?

Problem 5.
A variable has a length of $11. The first 7 characters in the variable must be numeric and must be in the form of a Julian date, the last 4 must be character (not alphanumeric). Provide edit code that can check that the value meets these conditions.
Problem 6.
PART A. How would you create a data set containing the names of the SAS data sets in a data library?

PART B. How would you create a data set containing the names of a catalog and the catalog entries in a data library?

Problem 7.
How would you change a character variable with a length of $10, to a shorter value, $7, dropping the last 3 characters and any trailing 0's? For example,

<table>
<thead>
<tr>
<th>Change these values</th>
<th>to these values</th>
</tr>
</thead>
<tbody>
<tr>
<td>C225430000</td>
<td>C22543</td>
</tr>
<tr>
<td>C201000001</td>
<td>C201</td>
</tr>
</tbody>
</table>

Problem 8.
You need to create a macro variable whose value is a list of all the values in a data set. For example, from the following data set

<table>
<thead>
<tr>
<th>OBS</th>
<th>DSVAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
</tr>
<tr>
<td>2</td>
<td>George</td>
</tr>
<tr>
<td>3</td>
<td>Paul</td>
</tr>
<tr>
<td>4</td>
<td>Ringo</td>
</tr>
</tbody>
</table>

you want to make a macro variable whose value is the character string 'John George Paul Ringo' but you can not assume that the total length of all the values will be less than 200. In these cases you cannot use CALL SYMPUT because the second argument of SYMPUT is a standard data step character expression and thus cannot exceed 200 characters. What is the solution?

Problem 9.
What do you do when you have an SCL program (or any other program for that matter) running locally, you're using REMOTE SUBMIT to run a program remotely, and you have a piece of information in the remote environment (such as a macro variable) that you'd like to access and check locally?