Enhancing Data Entry Screens for the Processing of the Directory of Texas Manufacturers Using SAS Screen Control Language®, SAS/AF®, and SAS/FS®

Betty Smith
Bureau of Business Research, University of Texas at Austin
Chaparral Systems, Santa Fe, NM.

ABSTRACT

The processing of the Directory of Texas Manufacturers (DTM), which has been produced by the Bureau of Business Research since 1933, has always been lengthy and somewhat complicated. Over the past few years, funding cuts have forced reductions in staff which even more complicate the process. Luckily, the advent of Screen Control Language with the release version 6.07 for CMS has been a great help in alleviating some of the pressure in producing the 1993 edition. Using Screen Control Language (SCL), we have been able to do key word searches of reference data sets which help with Standard Industrial Classification (SIC) coding, perform automatic abbreviations of words in fields such as the firm and personnel title fields which conform with DIRECTORY standards, allow the entry operator on-line access to the DTM data sets for reference in processing entries, among other tasks which seem to automate the entry process. This paper deals with a few of the enhancements we have made to the system.

The General Flow of the Entry Process

There are four main applications which make up the annual processing of the directory:

1. The Check In Process. Update and New Research forms, mailed out previously, are received, batched in groups of 200, and their id numbers are entered in to a Check In data set. Each batch is assigned a batch number. The batches are set aside for pick up by the entry personnel.

2. Initial Entry: The data entry personnel retrieve the batches and then using the batch number retrieves observations from the Check In and DTM data sets in SAS®. The resulting data set contains observations which match each form in the batch in its exact position in the batch. The electronic batch is displayed in an FSEDIT session. The entry personnel then process the information from the forms thereby updating the information on the screen. After completing data entry, the data set is appended to a temporary holding data set where it awaits retrieval by the verifier. The physical batch is then put on a shelf for the verifier to retrieve.

3. Verification: The verifier retrieves the batch and using the batch number extracts the electronic batch from the holding data set. The entries are then checked for accuracy and conformity. After completing verification, the electronic batch is split and appended to three data sets which reside on the update disk. One data set contains updates, one deletes, and one no changes. The physical batch is then filed under its batch number.

4. Update: Consists of series of batch programs which update the DTM database.

This paper deals primarily with enhancements designed for FSEDIT and FSVIEW displays in the first three applications.

Dictating Screen Behavior in FSVIEW in the CHECK IN Application

SCL is used in both FSVIEW formula entries and in AF programs called from the formula to catch common keying errors such as number transposition, to flag duplicate forms, to control the scrolling environment of the FSVIEW session, and to assign batch numbers to each entry in the newly created batch as well as position number and other criteria.

Here is a demonstration of how the formulas work while someone is checking in a batch:

```
*FSVIEW: WORK.ENTRY (1)------------------------------------
| COMMAND =#
| NOTE: AS IN...
| OBS ID MIN FLAG IMPORT SPNS BATCH |
|  |
| 1 |  |
| 2 |  |

*FSVIEW: WORK.ENTRY (2)------------------------------------
| COMMAND =#
| NOTE: AS IN...
| OBS ID MIN FLAG IMPORT SPNS BATCH |
|  |
| 1 |  |
| 2 |  |
```
In the above demonstration, we entered the id number and the FSVIEW formula retrieved the first twenty characters of the firm name, assigned a batch number and a position number to the entry.

When we entered the third entry, the screen scrolled forward one observation so that the fourth entry is positioned at the top of the view. When we entered the fourth entry, the formula detected this entry as a duplicate of an entry in batch 26, pos 63. As the user of this application runs into duplicates such as these, he/she usually checks to see if the entry is a miskey and if not, he/she puts a note on the form so that later we can check the form in batch 26 to see if this form is a duplicate, photocopy, fax, or a new company put on a photocopy of the original form. The formula also checks for invalid entries:

A picture of the review window while in the application reveals the formula entries which drive this FSVIEW application:

![Diagram](image-url)

The entry ascribed to the computed variable CHANGED calls an AF program called CHECKIN.PROGRAM which is put in the DTM1.CHECKIN catalog. This program checks to see if the id number entered is valid, assigns the batch and returns the first twenty characters of the firm name from the source data sets. The first thing the program attempts to do after passing the values sent from the DISPLAY function in the CHANGE entry is to establish the legitimacy of the id number:
Once the matches the id contained is available to the data set vector. The name is then shortened from its original length and assigned to \texttt{nam}.

The formula entries \texttt{CC2} and \texttt{CC3} control scrolling behavior of the FSVIEW session (see Figure A):

\texttt{CC2} checks to see if the variable \texttt{id} has been modified, and then issues the 'FORWARD 1' command using the EXECMD statement. In order to get the screen to move forward only one observation, the HSCROLL value in the PARMS window was set to '1'.

In order to cause the first observation to not scroll out of sight, variable \texttt{CC} contains the value of the current observation in the data set.

After leaving the FSVIEW session, the value in the data set \texttt{DTMBAT.ID} which contains the batch number is incremented by one.

\texttt{CC4} creates an alias data set to the data set being created in the FSVIEW. This data set is used to find duplicates within the same batch when the formula entry \texttt{CC6} calls the \texttt{AF} program, \texttt{DUP.PROGRAM}.

\texttt{DUP.PROGRAM} first checks for duplicate entries in the same batch by checking a sister data set, \texttt{ENT}, and then checks to see if the id is a duplicate of an entry in another batch by checking the check in data set, \texttt{DTM1.CHECKN}. If duplicates are found in either data set then flag is set to '\@', dupbat to the batch number where the original id is found and dpos the position in dupbat where the id is found:

\begin{verbatim}
entry id $7 cr $8 nam $20 flag $1 dupbat $5 dpos $8 source $8;
init:
  length name $100;
  return;
main:
  id1 = "" | trim(left(id)) | "";
  dt = date0;
  curny = year(dt);
  curny = trim(left(curny)) | '00000';
  curny = substr(curny,3,2);
  nxtyr = trim(left(curny));
  yr2 = trim(left(curny)) | '00000';
  yr2 = cutyr+1;
  retum;
end;

entry id $7 cr $8 nam $20 flag $1 dupbat $5 dpos $8 source $8;
init:
  length name $100;
  return;
main:
  id1 = "" | trim(left(id)) | "";
  dt = date0;
  curny = year(dt);
  curny = trim(left(curny)) | '00000';
  curny = substr(curny,3,2);
  nxtyr = trim(left(curny));
  yr2 = trim(left(curny)) | '00000';
  yr2 = cutyr+1;
  retum;
end;

SCL stored in CHECKIN.PROGRAM

Once the program establishes that the id is valid, then it opens up a copy of the source data set which houses the firm name. To do this an OPEN function in SCL is used and then the program subracts the data set with a WHERE function so that only the observation which matches the id entered is available to the data set vector. This WHERE function contains a variable created at the beginning of the MAIN block previously, which contains the id number surrounded by quotes. The variable is called \texttt{id1}.

The program then gets the firm name from the first matching observation in the data set using the GETVARC function. The name is then shortened from its original length and assigned to \texttt{nam}.

The formula entries \texttt{CC2} and \texttt{CC3} control scrolling behavior of the FSVIEW session (see Figure A):

\texttt{CC2} checks to see if the variable \texttt{id} has been modified, and then issues the 'FORWARD 1' command using the EXECMD statement. In order to get the screen to move forward only one observation, the HSCROLL value in the PARMS window was set to '1'.
Here are examples of the results returned by DUP.PROGRAM:

```
+PRIOR TRNS.DMY (2) +
| Command +
| DMS 20 NAV  FLAG DUMP CPU DSP 
| 91 007963 Kansas Equipment & So . 46
| 02 007963 Kansas Equipment & So . THIS 91 46
```

+PRIOR TRNS.DMY (1) +
| Command +
| DMS 20 NAV  FLAG DUMP CPU DSP 
| 91 007963 Kansas Equipment & So . 46
| 02 007963 Kansas Equipment & So . THIS 91 46

Enhancing the Data Entry Screens for the Initial Entry and Verification Applications

In the following example, the entry operator indicates a wish to go to the PLANT STATISTICS/PERSOONNEL section of the observation:

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Writing Information</td>
<td>Statistics/Plant Executives</td>
<td></td>
</tr>
<tr>
<td>Location Information</td>
<td>Alternate/DOUBLE Names</td>
<td></td>
</tr>
<tr>
<td>Phone Numbers</td>
<td>Site/Company/ерт</td>
<td></td>
</tr>
<tr>
<td>Main Office</td>
<td>List Plants</td>
<td></td>
</tr>
<tr>
<td>Notes to the Editor</td>
<td>Messages to (MINTY, SILFIA, EMA)</td>
<td></td>
</tr>
<tr>
<td>Will be sent with entry</td>
<td>Text to MESSAGE File on an account</td>
<td></td>
</tr>
</tbody>
</table>
```

After the enter key is pressed, SCL transports the user to the following screen and places the cursor on the Sales Volume Variable:

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Screen 3:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Command +
| You can return to the first screen by pressing F1 key | |
| Phone | 
```

The section of the SCL code which triggered this demonstration looks like this:

```
INIT:
CONTROL LABEL
SCL statements
return;
SCL Statements
GS:
IF GS NE " THEN CURSOR SALEVOL;
GS = ";
RETURN;
```

In the SCL excerpt above, a CONTROL LABEL statement is used so that SCL is forced to execute the GS block which corresponds to the name of the computational variable marked by the user. The GS block uses a CURSOR statement to move the view to the third screen of the application and places the cursor on the variable SALEVOL.

In addition to being able to navigate multiple screens, we used a number of customized commands to perform tasks such as looking at other data sets, and copying and moving fields and the like. We stored these customized commands in the keys window by using array processing in the FSEINIT and FSETERM blocks in our FSEEDIT session:

```
FSEINIT:
array newkey(24) $20 ('Ootor' 'database' 'end' 'left' 'right' 'blue' 'back' 'forward' 'commdata' 'newdata' 'sic' 'homeoff' 'phoacent' 'dupthisfield' 'dupfirmmail' 'duppepass' 'dupallfields' 'getinfo' 'add' 'search!' 'getblue' 'left' 'right' 'alt');
array oldkey(24) $20;
do i = 1 to 24;
oldkey(i) = getfkey(fkeyname(i));
call setfkey(fkeyname(i),oldkey(i));
end;
RETURN;
SCL blocks and statements
FSETERM:
do i = 1 to 24;
CALL SETFKEY(fkeyname(i),oldkey(i));
end;
RETURN;
```

In order to engage these customized commands and to disengage the error facility in SAS when any of the customized commands are initiated by pressing their corresponding function key, we place a CONTROL ALWAYS as the last statement before the MAIN BLOCK in SCL. When this statement is in effect, SAS does not cause an error condition to occur when a customized command word such as 'SEARCH!' is entered in the command buffer, but will execute any valid command line command.
Once CONTROL ALWAYS is in place, the program can then be set to retrieve issued commands from the buffer. This is done by causing the SCL program to continually poll the command buffer for commands by using the WORD function in the MAIN block:

```
DC = WORD(1,'U');
```

The WORD function in the above example, retrieves the first word from the command buffer every time the MAIN block executes. The 'U' parameter in the function converts the command name to upper case. The command is stored in the variable DC. DC can then be checked for content and if it contains a reserved command then a specific task can be triggered. For example, when the user presses the PF2 key, the program displays the DTM database's main data set called DTM.DATA:

```
TITLE.SCL:DMIN.
```

The METHOD feature was used in this application to perform abbreviations on the titles for the Plant Executive, Purchasing Agent, and Sales Agent. In the following example, the Plant Executive's title and the Purchasing Agent's title have been spelled out:

```
1. Plant Executive: W.W. Smith:
   Title: President:
   Purchasing Agent: Cheryl W. Baker:
   Title: Corporate Secretary:
```

After the user pressed the enter key, a method block triggered by the modification of either field was executed causing the following to occur:

```
---+-----------------------------------+----------------+---
| Plant Executive: W.W. Smith:  |   |                         |
| Title: President:              |   |                         |
| Purchasing Agent: Cheryl W. Baker: | | |
| Title: Corporate Secretary:   |   |                         |
```

The METHOD calls which triggered this effect look like this:

```
if modified(petit) then do;
  call method('title.scl','title1',petit,pename);
end;
```

In the method block stored in our catalog under TITLE.SCL, we stored the names we would need to abbreviate in an array and their corresponding abbreviations in another. The block then scans the title field passed to it by the METHOD CALLS listed above and translated any titles needing abbreviation to their corresponding abbreviations.

The final enhancement to the data entry screen that this paper will deal with involves providing a key word search application to the FSEDIT screen for table look-up purposes. This enhancement is used to look up SIC codes for products added to Update and New Research forms. The feature is engaged from the product description fields from the data entry screen:

```
---+-----------------------------------+----------------+---
| Search Program:                  |   |                         |
| Enter keywords you wish to look up: |   |                         |
| 1. plastic                         |   |                         |
| 2.                        |   |                         |
| 3.                        |   |                         |
| Press F2 to abort. Press enter when you complete entry: |   |                         |
```

In the method block stored in our catalog under TITLE.SCL, we stored the names we would need to abbreviate in an array and their corresponding abbreviations in another. The block then scans the title field passed to it by the METHOD CALLS listed above and translated any titles needing abbreviation to their corresponding abbreviations.
After a few seconds the following table was displayed:

```
*BEGIN WORK.LOCK (1)

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRESS F3 TO RETURN</td>
</tr>
<tr>
<td>USE BASIC RECORD</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

*END WORK.LOCK (1)*
```

The user then selected 'Pickled peppers' and the entire observation is displayed on an FSEDIT screen:

```
*BEGIN WORK.LOCK2 (EDIT)*

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>USE BGN TIMES</td>
</tr>
<tr>
<td>Sic: 2005 Pickled peppers</td>
</tr>
</tbody>
</table>

*END WORK.LOCK2 (EDIT)*
```

The application is triggered by a customized command called 'SEARCH1!', which is stored in PF20 in the FSEINIT block. In the MAIN block, the program checks the value of DC to see if it is equal to 'SEARCH1!':

```
if dc = 'SEARCH1' then do;
    fidon=curfld();
    do i = 1 to 5;
    if fidon = upcase(sicfld(i)) then do;
        sicnum=sic(i);text=sictxt(i);
        call display('dtmext.cat.search1.program',sicnum,text);
        sic(i)=sicnum;sictxt(i)=text;
        leave;
        pass=0;
    end;
```

The AF program SEARCH1.PROGRAM creates a data set from DTM.SIC and from another data set called LOOKSIC2.DATA. DTM.SIC contains current entries in the DTM database from the last processing of the update application. LOOKSIC2.DATA contains a list of SIC codes and descriptions taken from a reference manual. When the keys words are entered for the search, a WHERE clause is created which is then concatenated to the string containing the data set name in an OPEN statement in SCL. Observations are then read from the respective source data sets and appended to the temporary data set called LOOK1. LOOK1 is then copied into another data set called LOOK2. LOOK2 is presented in an FSVIEW session for the user. When the user marks a record, the formula retrieves the current observation number using a CUROBS command and then calls an FSEDIT session on LOOK1 subsetting the data set using a WHERE clause. The result is the view of marked observation's product description in full display. The user can then mark the 'USE' variable and the product description and SIC code will be transported back to the data entry screen and placed in the appropriate fields.

Conclusion

It is important to note that the above paper has only dealt with a few of the enhancements used on data entry screens in the processing of the directory. In addition, to those tasks explained above, the programs also abbreviate FIRM and LOCATION fields, function keys have been set to perform such tasks as copying the plant phone into the Plant Officers phone fields, copying the Plant Executive fields into the Purchasing and Sales Agents fields, looking at other data sets, copying the firm name to the mail name and vice versa, copying information from the previous observation to the current observation, and much more. Unfortunately, due to the requirements of space, the paper copy of this presentation shows only a few examples of the SCL structures which drive the applications discussed in the paper. Nonetheless, it is hoped that by looking at how SCL works in FSP and AF through these applications those who read this paper will get some idea how to use SCL in their own applications.

For Information Contact: Sylvia Cook
Bureau of Business Research
University of Texas at Austin
Box 7459
Austin, TX 78713
512-471-1616

Betty Smith
Chaparral Systems
Santa Fe, NM
505-983-5594

SAS, SAS/AF, SAS/FSP, and SAS Screen Control Language are registered trademarks or trademarks of SAS Institute Inc and other countries: * indicates USA registration.