Building a Data Entry and Correction System by Synchronizing the Data Table and the Data Form

Hung X Phan, U.S. Bureau of the Census

Abstract

The Data Set Entry classes display the contents of SAS data sets as follows:

Data Form displays one row of a SAS data set at a time using SAS/AF widgets to represent the columns.

Data Table provides a tabular display of multiple rows of a SAS data set.

This paper will illustrate a technique of combining the best of both, Data Table and Data Form, to build a Data Entry and Correction System. This system will have a Data Table widget displayed on the top half of a frame and a Data Form widget on the bottom half. When a row within the Data Table is clicked, the same record is displayed on the Data Form where data may be entered and/or modified. Buttons on a toolbar allow the users to perform various functions: FORWARD, BACKWARD, TOP, BOTTOM, ADD, DELETE, UPDATE, SORT, FIND, HIDE/UNHIDE, END, RESET. For example, when the FORWARD button is clicked, the next record in the SAS data set is shown on the top row of the Data Table, which is synchronized with the same record shown on the Data Form, where data can be edited. This system is currently running in a multi-user environment on a SAS6.12/Windows95 Novell server at the U.S. Bureau of Census.

Introduction

The Alphabetical Index of Industries and Occupations was developed primarily for use in classifying a respondent's industry (employer's kind of business) and occupation (employee's kind of work) as reported in the 1990 Census and other surveys conducted by the U.S. Bureau of the Census.

Each title has been assigned a number code by the coding specialists. The coding scheme was based on the Standard Industrial Classification (SIC), a system that federal, state, and local governments, the business community, and the general public have used since the 1930s.

As a result of the North American Free Trade Agreement (NAFTA), the United States, Canada, and Mexico, in order to better compare economic and financial statistics, will replace the countries' separate systems for classifying industries with one uniform system--the North American Industry Classification System (NAICS)--by early 1999.

Our division, Housing and Household Economics Statistics, in compliance with the new coding system, needed to build a GUI system which provides the coding specialists with easy point and click access to data so they can view the old SIC Industry and Occupation code and assign the new NAICS Industry and Occupation code.

Building the Data Entry and Correction System interface

First, create a new blank frame called dt_entry by submitting the following SCL code.

```
libname wuss99 'c:\wuss99';
proc build
c=wuss99.myapp.dt_entry.frame;
run;
```

Second, create a data table in browse mode and place it on the top half of the frame as shown in Figure 1.

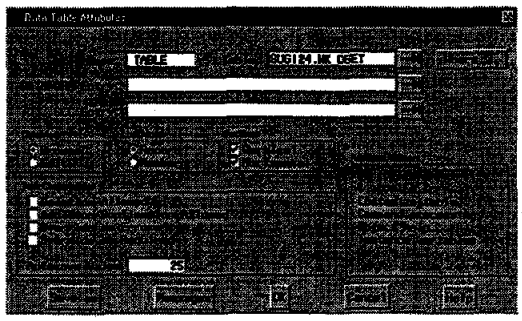

Figure 1: Data Table Attributes Window

Next, create a data entry form in edit mode and place it on the bottom half of the frame as shown in Figure 2.

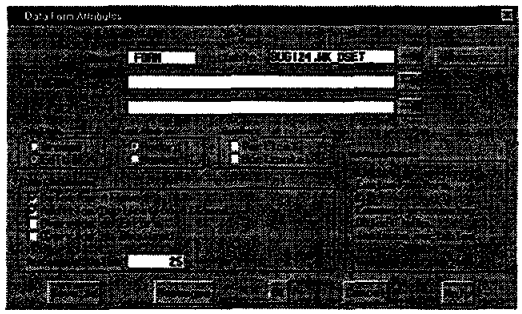

Figure 2: Data Form Attributes Window

Lastly, create a tool bar which consists of 12 buttons: FORWARD, BACKWARD, TOP, BOTTOM, ADD, DELETE, UPDATE, SORT, FIND, HIDE/UNHIDE, END, RESET to help the users navigate throughout the system as shown in Figure 3.

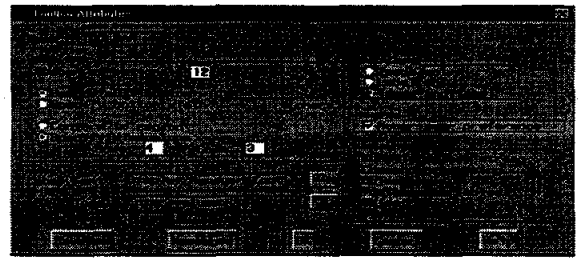

Figure 3: Tool Bar Attributes Window

After we add a text entry and a graphic text for displaying information about the working data set and the title for the application, we have the Data Entry and Correction System screen as seen in Figure 4.

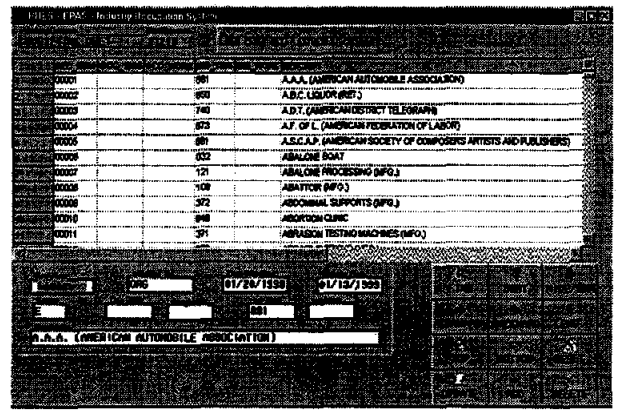

Figure 4: Data Entry and Correction System screen

Synchronizing the Data Table and the Data Form:

Listing 1 shows the DT_ENTRY.SCL code which executes the INIT section as the DT_ENTRY.FRAME first displayed. It is important to understand the process that goes on in this segment of the code. It highlights the concept of synchronizing the data table and the data form technique presented in this paper.

First we load the working data set called WK_DSET into the data table in browse mode. Data table is where the users may make a selection by clicking on a row.

Initially, we want to synchronize the first record. In the data table, the '_set_dataset_' method opens the WK_DSET in browse mode and displays the first record in the top row by default. In the data form, the '_set_dataset_' method opens the WK_DSET in edit mode and displays values of the first record, where the users are allowed to enter data and/or make changes to the data

```
****************************************
INIT:
 lib_is  = 'WUSS99'; ds_is  = 'WK_DSET';
 lib_dsn = lib_is||'.'||ds_is;

/* initially set up a record counter,
rec_num=1, open the WK_DSET, and fetch
the first obs into the DDV. The call
set(dsid) establishes an automatic link
between the SCL variables on the data
form and the WK_DSET data set varialbes.
We need this information for the
implementation of the tool bar. */
 rec_num=1;
 dsid=open(lib_dsn);
 rc=fetchobs(dsid,rec_num);
 call set(dsid);

/* retrieve the widget id(s) for the data
table and the data form. We need this
info for the call send in the TABLE bcl*/
call notify ('.','_get_widget_','table',
            t_id);
call notify ('.','_get_widget_','form',
            formid);
/* set the WK_DSET in the data table -
   read only mode */

 call notify ('table','_set_dataset_',
  lib_dsn,'browse','record','bronly',
         'noadd','nodelete');

/* set the WK_DSET in the data form -
   edit mode */

call notify ('form','_set_dataset_',
lib_dsn,'edit','record');

return;  /* INIT */
******************************
```

Listing 1. DT_ENTRY.SCL (INIT)

With a few lines of code in the INIT section we managed to synchronize the data table and the data form of the first observation in the WK_DSET data set. In fact, we will show throughout this paper that we only need to use a few commands, a few functions, and a few methods to accomplish what may seem a complicated task.

Listing 2 shows the SCL code for the TABLE section that is executed when a row is clicked from the data table. The '_get_current_row_number_' method returns the row number of which the users clicked on in the data table. We use this information to display the selected row (record) in the top row of the data table using the '_goto_absolute_row_' method. Similarly the '_goto_row_number_' method displays values of the selected record on the data form. Hence, the synchronization is accomplished.

```
**********************************
TABLE:
call send
(t_id,'_get_current_row_number_',rec_num)
call send
(t_id,'_goto_absolute_row_',rec_num);
call send
(formid,'_goto_row_number_',rec_num);

return; /* TABLE */
****************************************
```

Listing 2. SCL code for the TABLE section

Synchronization of the data table and the data form also occurs when the users point-and-click on a button of the toolbar. For example, to view and/or enter data on the next record from the current record displayed on the data form, the users can alternatively click on the FORWARD button, rather than clicking on the next row in the data table.

One important thing we need to understand is what goes on behind the scenes and how data set variables relate to the SCL variables implemented on a data form.

When SAS compiles a Data Step, it sets up an area of memory called the Program Data Vector (PDV), into which each line is read, before being written out to the data set.

In a similar way, when an SCL program is compiled, an area of memory called the SCL Data Vector (SDV) is opened. When an SCL program opens a data set, the SAS System automatically opens a temporary storage area to hold the values of the data set variables. This storage area, which is unique for each open data set, is called the data set data vector (DDV).

To write values of SCL variables to data set variables, the procedure is reversed: Values of SCL variables are copied from the SDV to the DDV. The values of variables in the DDV then update the specified observation in the data set.

Figure 5 shows the transfer of data between a data form AF application and the WK_DSET data set.

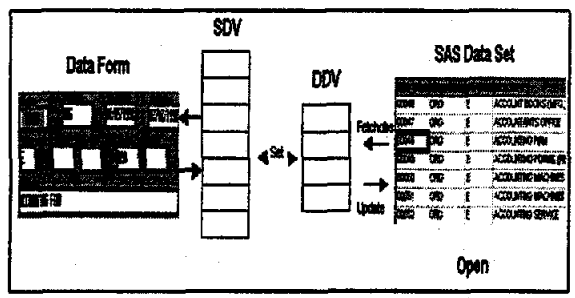

Figure 5. Transfer of data between data form AF application and WK_DSET data set

Note that in the INIT section, the OPEN function opens the data set and creates a DDV, which is unique to the WK_DSET data set. The FETCHOBS function copies the specified observation, obsnum=1, into the DDV. The CALL SET routine establishes an automatic link between the SCL variables on the data form and the variables in the data set.

Listing 3 contains the SCL code which is used to implement the FORWARD and BACKWARD buttons. The logic used here is quite simple. We add or subtract 1 from the rec_num counter depending on whether we want to implement the forward or backward function. Again, the '_goto_absolute_row_' method is used to place the specified record in the top row of the data table and the '_goto_row_number_' method is used to display its values on the data form. Use the system variable _msg_ to display appropriate messages on the window's message line. Note that _msg_ is not displayed if the window has its BANNER attribute set to NONE.

```
*****************************************
/* implement the FORWARD button. */
FORWARD:
/* if rec_num is less than the number of
    record in the data set then
    increment it by 1 and go forward
   else
    _msg_ = At bottom of the data set.
 */
 if (rec_num < maxrec)
  then
   do;
    rec_num=rec_num+1;
    call send(t_id, _goto_absolute_row_',
              rec_num);
    call send(formid,'_goto_row_number_',
              rec_num);
      _msg_="Go forward one record.";
    end;
  else
    do;
      _msg_='At Bottom of Data Set';
end;
return; /* FORWARD */
```

```
/* implement the BACKWARD button. */
BACKWARD:
/* if rec_num is greater than 1 then
    subtract 1 from it and go backward
   else
     _msg_ = At top of the data set.
 */
 if (rec_num >1)
    then
     do;
      rec_num=rec_num-1;
    call send(t_id,'_goto_absolute_row_',
              rec_num);
    call send(formid,'_goto_row_number_',
              rec_num);
      _msg_="Go backward one record.";
     end;
   else
     _msg_='At Top of Data Set';
 return; /* BACKWARD */
*****************************************
```

Listing 3. FORWARD & BACKWARD SCL section

TOP and BOTTOM buttons are quick ways for the users to go to the top or bottom records in the data set.

Here we illustrate the data table and the data form are synchronized when the BOTTOM button is clicked. We first get the number of records in the WK_DSET data set, i.e, maxrec=attrn(dsid,'nlobs'). The call send (t_id,'_goto_absolute_row_', maxrec) displays the last record of the WK_DSET in the top row of the data table. Similarly, the call send (formid, '_goto_row_number_',maxrec) displays its values on the data form.

Listing 4 contains the SCL segment of code which is used to implement the TOP and BOTTOM buttons.

```
*****************************************
/* implement the TOP button. */
TOP:
/* set rec_num = 1 for first record */
     rec_num = 1;
   call send
(t_id,'_goto_absolute_row_',rec_num);

call send
(formid,'_goto_row_number_',rec_num);
    _msg_='At top of the data set.';
return; /* TOP */

/* implement the BOTTOM button. */
BOTTOM:
    maxrec = attrn(dsid,'nlobs');
    call send(t_id,'_goto_absolute_row_',
              maxrec);
    call send(formid,'_goto_row_number_',
              maxrec);
      _msg_='At bottom of the data set.';
   end;
return; /* BOTTOM */
*****************************************
```

Listing 4. TOP & BOTTOM SCL section

The FIND, SORT, and HIDE/UNHIDE buttons provide powerful tools to the users. The users have the ability to search and locate record(s) that meet a specific criteria. The users can sort the data in the WK_DSET data set in a specify order. The HIDE/UNHIDE button allows the users to choose which column(s) in the data table they want to view. The users may want to hide nonessential column(s) and concentrate on the more critical column(s) to work on without having to scroll back and forth horizontally.

We include the following SCL code, Listing 5, the MAKELIST function creates an empty SCL list whose identifier is stored in the AVAILIST variable. The ATTRN function returns the number of variables in the data set. The VARNAME function returns the name of the variables in the data set. The INSERTC function inserts the variable names into the AVAILIST. We use this AVAILIST in the pop-up variables dialog box to allow the users to select the variables they want to perform the FIND, SORT, or HIDE/UNHIDE function on.

```
*****************************************
  availist = makelist();
  nvars    = attrn(dsid,'nvars');
  do v = 1 to nvars;
    varn = varname(dsid,v);
    rc   = insertc(availist,varn,-1);
  end;
*****************************************
```

Listing 5. SCL code to make availist

When the FIND button is clicked, the find dialog box pops up. The users can select from one of the variable names in the AVAILIST and specify a criteria in the text box as shown in Figure 5.

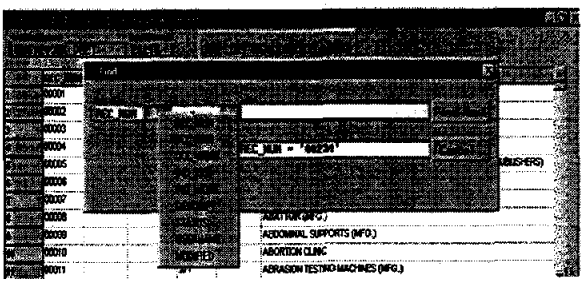

Figure 5. the FIND Dialog Box

When the SORT button is clicked, the sort dialog box pops up. The users can select from one of the variable names in the AVAILIST and specify the sorting order criteria as shown in Figure 6.

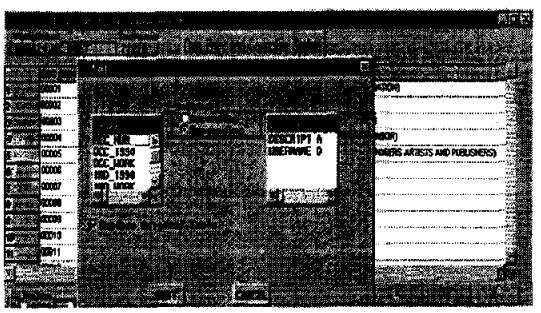

Figure 6. the SORT Dialog Box

Similarly, when the HIDE/UNHIDE button is clicked, the hide/unhide dialog box pops up. The users can select from one of the variable names in the AVAILIST to have the specified column(s) hidden or unhidden as shown in Figure 7

Figure 7. the HIDE/UNHIDE Dialog Box

Listing 6 shows the SCL code to implement the FIND, SORT, HIDE/UNHIDE buttons.

```
*****************************************
FIND:
/* return the number of the row that
   meets the find request.
   The find request should contain one or
   more character list items.  Each
   item can use standard WHERE clause
   syntax and is handled as an additional
   request.
*/
choice = 0;  /* initialize choice */

/* invoke the pop up FIND dialog box.
If choice = 0 means the cancel button is
selected, var is the variable used in
find request, oper is the comparison
operators, text is the string entered,
srchtext is the find-request str */

call display('wuss99.myapp.find.frame',
 choice,availist,var,oper,text,srchtext);

if (choice)  /* find */
 then
  do;
   rec_num=rec_num+1; /*searching point*/
   list=makelist();
   list=insertc(list,srchtext);
  /* list contains the find request string */
call
send(t_id,'_find_row_',list,rec_num);
if (rec_num > 0)       /* record found */
    then
     do;
       call send
(formid,'_goto_row_number_',rec_num);
end;
else               /* record not found */
 do;
  _msg_ = srchtext||' not found';
 end;
end;
return; /* FIND */
```

```
SORT:
/* sort the data table by one or more
columns. The order of the column names
reflects the order in which to sort. The
column name should be preceded by the
keyword DESCENDING when the column is to
be sorted in descending order. Ascending
order is assumed unless otherwise
specified. The users may specify as many
columns as they like. */

  choice = 0;   /* initialize choice */

/* invoke the pop up SORT dialog box */
  call display('wuss99.myapp.sort.frame',
       choice,availist,oper,sort_str);

if (choice) /* sort button is selected */
   then
    do;
      call send(t_id,'_sort_',sort_str);
      _msg_ = 'Sort '||sort_str;
return; /* SORT */

HIDE:
/* hide a list of columns. The hidden
columns are not removed from the data
table but rather are removed from the
display */

choice = 0;  /* initialize choice */

/* invoke the pop up HIDE dialog box */
call display('wuss99.myapp.hide.frame',
             choice,displist,hidelist);
if (choice)
 then
  do;
    hide_str = '';
    numvars = listlen(displist);
    do i = 1 to numvars;
      vname=getitemc(displist,i);
      call send(t_id,'_hide_column_',
                vname);
     end;
  end;
return; /* HIDE */

UNHIDE:
/* unhide a list of columns */
choice = 0;   /* initialize choice */
/* invoke the pop up UNHIDE dialog box */
call display('wuss99.myapp.unhide.frame',
             choice,displist,unhidels);
if (choice)
 then
  do;
    hide_str = '';
    numvars = listlen(unhidels);
    do i = 1 to numvars;
      vname=getitemc(unhidels,i);
      call send(t_id,'_unhide_column_',
                vname);
     end;
  end;
return; /* UNHIDE */
*************************************
```

Listing 6. FIND, SORT, HIDE/UNHIDE SCL

The Data Entry and Correction System

This system is built to provide the users with data in a data table, a spreadsheet-like format in which the users can view several records at a time. The data form is designed for the users to view, edit and enter data in a record. The data form has text boxes for each of the fields in the data set. In the data form, Figure 8, we have REC_NUM, MODIFIED_BY, DATE_MODIFIED, STATUS, and TODAY'S_DATE as protected fields; values cannot be changed by the users. The OCC_1990, OCC_WORK, IND_1990, IND_WORK, and DESCRIPTION are edited fields; values can be changed by the users.

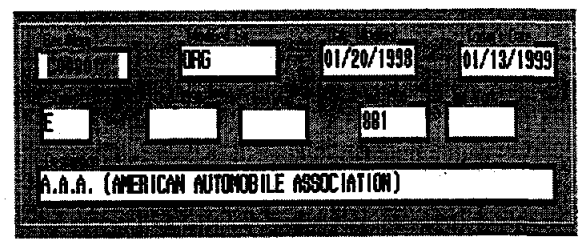

Figure 8. The Data Form with First Obs Displayed

When the users click on the ADD button, a new blank record is created and is ready to be used.

When the users click on the DELETE button, the STATUS field will change to a 'D' for delete, the MODIFIED_BY field will reflect the log-in name of the person who is doing the delete,

and the DATE_MODIFIED field will change to the current date.

Similarly, when the users click on the UPDATE button, the new information entered in the edited fields will be saved. For the protected fields, the STATUS field will change to an 'R' for revise, the MODIFIED_BY will reflect the log-in name of the person who makes the change, and the DATE_MODIFIED will change to the current date.
Listing 7 shows the SCL code which is used to implement the ADD, UPDATE, DELETE buttons.

```
*****************************************
/* implement the UPDATE button */
UPDATE:
/* change STATUS flg to 'R' for Revise,
       DATE_MODIFIED to today's date,
       MODIFIED_BY   to username */
 status   = 'R';
 modified = today();
 username = sysuname;
/* update the current record in WK_DSET*/
 rc=update(dsid);

 _msg_= 'The current record is updated.';
return; /* UPDATE */

/* implement the DELETE button */
DELETE:
/* change STATUS flg to 'D' for delete,
       DATE_MODIFIED to today's date,
       MODIFIED_BY   to username */
 status   = 'D';
 modified = today();
 username = sysuname;
/* update the current record in WK_DSET*/
 rc=update(dsid);

 _msg_= 'The current record is deleted.';
return; /* DELETE */

/* implement the ADD button */
ADD:
/* create a new record and append it to
the WK_DSET */
  num_rec = num_rec+1;

/* identify the person who do the add and
assign init values for the new record */
    username = sysuname;
    status   =  'B'; /* B for blank */
    modified =  today();
    ind_1990 =  '';
    ind_work =  '';
    occ_1990 =  '';
    occ_work =  '';
    descript =  '';

/* append a new record to the WK_DSET */
   rc=append(dsid);

   _msg_= 'A new record is added.';
 return; /* ADD */
*****************************************
```

Listing 7. UPDATE, ADD, DELETE section

Conclusion

This paper demonstrates how to build a Data Entry and Correction System by Synchronizing the Data Table and Data Form.

The system was built with a very generic and generalized approach in order to work with any SAS data sets without requiring a lot of changes to the code.

Acknowledgments

The author wishes to thank Mr. Rick Denby and his staff who have opened doors and provided encouragement at every step. Special appreciation and love go to my wife for help editing this paper.

References

U.S. Bureau of the Census, January, 1992, 1990 Census of Population and Housing, Alphabetical Index of Industries and Occupations.

Destiny Corporation, 1997, Building Frame Entry Applications

SAS Institute Inc.,1994, SAS Screen Control Language Reference, Version 6, Second Edition

The author can be contacted at:

Mr. Hung X Phan
U.S. Bureau of Census
Rm 1483/3 CPAS/HHES
Washington, D.C. 20233
Tel: 301-457-3204
E-mail: hphan@census.gov

SAS, SAS/AF, SAS/EIS, and SAS/FSP are registered trademarks or trademarks of SAS, Institute Inc., Cary, NC, USA

Other brand and product names are registered trademarks or trademarks of their respective companies.