Double Data Entry in FSEDIT Using Point-of-Entry Verification

Derek Morgan, Washington University Medical School, St. Louis, MO
Michael Province, Washington University Medical School, St. Louis, MO

Abstract

Double data entry is a proven, heavily used tool for data quality control. Although there are many functions available in SAS/AF®, SAS/FSP® and methods for FRAME entries to reduce entry errors, double entry adds to the overall quality control of the data. This paper describes a method for adding transparent, automatic double data entry verification to an FSEDIT screen using SCL. With this method, discrepancy resolution occurs at the point of the second entry on a field-by-field basis. Discrepancies are investigated and resolved during the second entry, eliminating both the production of a discrepancy report and a third pass through the dataset for correction after both entries have been concluded and compared.

Introduction

Traditionally, double data entry has involved comparison of a historical dataset with an active dataset. Any discrepancies are noted and resolved after entry. Using the SAS System®, methodologies have been developed to increase the ease of this resolution (e.g., PROC COMPARE), or as part of an integrated data entry application using SAS/AF and SCL. However, the resolution of discrepancies is still performed after both entries have been completed.

This method uses an alternative approach: the resolution is performed at the point of the second entry. In essence, each field is checked as it is entered and the entry operator is alerted to take some action if a discrepancy is noted. This eliminates the need for a historical dataset and removes a step from the traditional double data entry process.

Background

The method was developed for a distributed, remotely supported custom data entry and management application using SAS/AF, SAS/FSP and SCL, along with some system functions handled by OS/2® REXX commands. The target users are, in general, not experienced data entry operators, but research and clinic staff with varying degrees of computer (and keyboard) experience. Point-of-entry verification was chosen in part because of the diversity of this user group and to reduce the time burden on them. They do not have to generate (or wait for from a central organization) a discrepancy report, nor do they have to make a third pass through the dataset.

This was originally developed with the SAS System, version 6.08, running under OS/2 and was ported to version 6.11, also under OS/2. This method can be used with FRAME entries as well, with some modifications that will greatly improve its efficiency. The overhead of the method in terms of real time is negligible with respect to the entry of fields. Depending on the size of the form to be developed, it can, however, be a large programming task under SAS/FSP. The current application is still in SAS/FSP, using SCL to achieve the comparison and resolution. The screens in this application are relatively small, therefore, it was a small task to label each field and repeat the coding template. With larger screens, this may not be an option. Then the DATAFORMS subclass in a FRAME entry comes in handy; there is a process that will allow you to get the value of a widget without explicitly assigning the value in a labeled code statement. Point-of-entry verification can then be easily implemented using the general method presented here.

Specifics

The requirements for the method were as follows: foremost, it had to be transparent to the users. Any noticeable lag would add to the perceived burden and would reduce the acceptance level of double data entry. Secondly, it needed to be fully automated. The user does not have to do anything for the double entry process to function. It keeps track of which entry number is the active one, prevents a third entry from being started and performs its checking and discrepancy reporting without user intervention. Error tracking was also required. It automatically generates an error dataset that contains the entry person's ID, which form was being entered, which field on the form was detected as being in error and which corrective action was taken. Finally, the capacity to selectively activate double data entry at the field level was desired.

For the initial criterion, the additional overhead of the method did not cause a delay in system response time where no discrepancy was found. This was not tested quantitatively, but rather piloted with some of the entry operators using a slower platform than the production system (486/33 versus a Pentium-120, both with 16 MB RAM). In production, there is no detectable change in the data entry unless the second entry differs from the first. The method as implemented works very well for relatively small datasets (150 or fewer variables, less than 1000 observations.) A noticeable lag will occur as the dataset becomes larger; if the dataset is indexed, this lag is reduced or disappears entirely (depending on the size of the dataset) and the checking is again transparent to the entry person. The drawbacks are in the extra coding and in the

acceptance from the entry people. This method may not be useful with experienced data entry operators since the <ENTER> key must be pressed after each field is entered. It may then be counter-productive for two reasons: it would slow an experienced operator and possibly even increase the error rate.

The method uses a single dataset to keep both entries and is keyed by entry number, which is automatically assigned by the SCL program after a quick check of the selected dataset. For analysis, this requires that WHERE clauses be utilized so that the verified data from the second entry is the only data used. It was felt that this would improve the system stability and help insure the transparency of the application. When a particular screen is chosen for entry (from the integrated application), the dataset is opened and the number of entries is determined. This value is then passed to the FSEDIT screen via a macro variable. It is also possible to find the number of entries for a given ID within the FSEDIT screen itself. The double data entry method can easily be adapted for two datasets, where the first and second entries reside in separate datasets, especially if the screen is a part of a larger application and the correct dataset name can be passed from a higher-level PROGRAM or FRAME entry.

Label-specific processing was used for all screens, since double data entry needed to be switched on or off at the field level for this application. Assigning the value of the field to a single variable can be achieved through two methods. One uses arrays, but the chosen technique for this application is direct assignment of values in a labeled section of code. The former is probably best if you have a small number of fields, while the latter allows for quick execution of medium-sized screens.

Sample Code

Below is an example of the code for some fields in a double-entry screen. ID is assigned from a higher-level SAS/AF application and DATE is not to be verified. Note the difference in the coding (particularly the variable names) for the character field (LASTNAME) as opposed to the coding for the numeric field (HEIGHT). The checking routine itself begins at the label DDECHK.

```
FSEINIT:
control label enter error allcmds;
/* chkc, valc, withfmt, var and nxtvar are used by
the double data entry method */
length cmd withfmt $ 80 dsn chkc valc $ 200 var
      nxtvar $ 8;
/* STAY, MODIFY and RETURN are all necessary */
call setcr('STAY','RETURN','MODIFY');
valc = ' ';
call execcmd('zoom on');
protect id flg;
return;

INIT:
/* If this is a new observation */
if id eq ' ' then do;
   id = symget('id'); /* From calling app */
   entryid = symget('userid');
/* Entry number is calculated in the calling
application and passed through a macro. */
   entryx = symgetn('entry');
   protect id;
   if entryx eq 2 then do;
/* Load data for non-double-entered fields
   Note: the SCL code for WEIGHT and DOB is not
   In the sample code */
      dsn = "hgdata.anthrop (drop=height weight
      lastname dob " || "where=(id eq '" || id || "' and
      entry eq 1))";
      oldstuff = open(dsn,'i');
      call set(oldstuff);
      rc = fetch(oldstuff);
      rc = close(oldstuff);
   end;
   entry = entryx;
end;
if entry eq 2 then do;
/* Signal user which entry is being updated */
   flg = 'ENTRY 2';
/* This turns off DDE checking */
   val = .I;
end;
nxtvar = 'date';
form = "ANT";
/* Cursor placement when screen is opened */
rc = field('cursor',nxtvar);
return;

MAIN:
cmd = upcase(lastcmd());
/* Only check if field has been modified */
if field('modified',var) eq 0 then
   val = .i;
if entry eq 2 and val ne .i and cmd ne 'END' then
   link ddechk;
/* Turn on checking for next  field */
val = .;
return;

/* Date is a non-DDE field */
DATE:
val = .I;
if error(date) then
   cursor date;
else do;
   nxtvar = "height";
   cursor height;
end;
return;

HEIGHT:
/* Minimum and maximum are set, skip DDE
checking if field is obviously in error */
if error(height) then do;
   val = .I;
   cursor height;
end;
else do;
/* Assign value of field to be checked */
```

```
   val = height;
/* Move cursor to next field */
   cursor lastname;
/* let DDE subroutine know what next field is */
   nxtvar = 'lastname';
end;
return;

LASTNAME:
valc = lastname;
cursor lastname;  /* Last Field in Form */
nxtvar = 'lastname';
alarm;
_msg_ = "END OF FORM";
return;

TERM:
touchdat = datetime();
entryid = symget('userid');
return;

DDECHK:
entryid = symget('userid');
/* Get name of variable to be checked */
var = curfld();
if var eq ' ' then
   return;
/* Open first entry, get value */
dsn = "hgdata.anthrop (where=(id eq '" || id || "' and
        entry eq 1))";
chkfile = open(dsn,'i');
rc = fetch(chkfile);
vnum = varnum(chkfile,var);
/* Determine if numeric or character */
if vartype(chkfile,vnum) eq 'N' then do;
   chk = getvarn(chkfile,vnum);
/* Determine if values match */
   if chk ne val then do;
      withfmt = varfmt(chkfile,vnum);
/* Determine whether to display error message
with or without associated format */
      if withfmt ne ' ' then
         errmsg = "Error in " || trim(var) || "! Original
         value is " || left(putn(chk,withfmt));
      else
         errmsg = "Error in " || trim(var) || "! Original
         value is " || left(putn(chk,'best.'));
/* Open action list dataset */
      menuds = open('hgx.ddemenu','i');
       if menuds le 0 then do;
         _msg_ = 'ERROR in opening Menu Data. Call
         DCC ASAP.';
         return;
      end;
/* Open error tracking dataset */
      errds = "hgdata.errors (read=" ||
      symget('rdpswd') || ")";
      bugs = open(errds,'u');
      call set(bugs);
/* Sound cue to alert entry operator that the field
is in error */
      call sound(1000,100);
      call sound( 857,100);
      call sound( 615,100);
/* Open action list window */
      call wregion(5,3,6,64,' ');
      errortyp = datalistn(menuds,'errortyp
      item',errmsg);
      rc = close(menuds);
/* Take corrective action */
      select(errortyp);
/* Default is to correct the field; return cursor to
field and change color */
         when(1) do;
            rc = field('color orange reverse',var);
            rc = field('cursor',var);
         end;
/* Data correct as typed; proceed to next field */
         otherwise
            rc = field('cursor',nxtvar);
      end;
/* Add error to error tacking dataset */
      rc = append(bugs);
      rc = close(bugs);
   end;
end;
else do;
/* same procedure, except for character values */
   chkc = trim(getvarc(chkfile,vnum));
   if chkc ne trim(valc) then do;
      errmsg = "Error in " || trim(var) ||
      "! Original value is " || left(chkc);
      menuds = open('hgx.ddemenu','i');
      if menuds le 0 then do;
         _msg_ = 'ERROR in opening Menu Data. Call
         DCC ASAP.';
         return;
      end;
      errds = "hgdata.errors (read=" ||
      symget('rdpswd') || ")";
      bugs = open(errds,'u');
      call set(bugs);
      call sound(1000,100);
      call sound( 857,100);
      call sound( 615,100);
      call wregion(5,3,6,64,' ');
      errortyp = datalistn(menuds,'errortyp
      item',errmsg);
      rc = close(menuds);
      select(errortyp);
         when(1) do;
            rc = field('color orange reverse',var);
            rc = field('cursor',var);
         end;
         otherwise
            rc = field('cursor',nxtvar);
      end;
      rc = append(bugs);
      rc = close(bugs);
   end;
end;
/* Close lookup dataset */
rc = close(chkfile);
return;
```

Code Details

CALL SETCR is used to insure that the MAIN section executes with each return. Each field is set to NOAUTOSKIP, forcing an <ENTER> key press after each field is entered. In the INIT section, the observation is checked to see if it is new. If so, the entry number (and ID) are passed from the calling SAS/AF application. At the second entry, any non-double-entered fields are loaded to provide a complete observation for analysis using only a WHERE clause. An on-screen flag is activated to inform the user that the second entry is being worked upon. At initialization, the double data checking is turned off to prevent the first field from automatically being flagged when MAIN first executes during the opening of the first observation.

In MAIN, the name of the current field is obtained and the double data entry checking routine is selectively invoked. This is done using a special missing value that does not conflict with those used in the project, so that no screen value will turn off its own checking. At each labeled section, there is code that either disables the double data entry checking ("val=.i;"), or enables it by assigning the value of the field to the comparison variable (val, or valc if the field is a character field.) The cursor is moved and a variable containing the name of the next field is assigned for the use of the subroutine.

The subroutine itself consists of two nearly identical segments: one for numeric fields and the other for character fields. The first section of the subroutine is responsible for opening the dataset to be checked and determining if the field is numeric or character. Then the comparison is performed and an error message is assembled if there is a discrepancy. The exact text of the error message depends on whether there is an associated format for the variable. (SAS System date values are an example of why this feature exists.) Next, the discrepancy resolution option dataset is opened (this could also be done as an SCL list), and the dataset for error tracking is also opened. In this application, a distinctive sound cue is generated to alert the operator that an error has been made. A selection list window containing the discrepancy resolution options appears at the top of the screen page, sized so that it does not cover any of the fields on the screen. The user then selects one of the resolution options and the appropriate action is taken. The default is to correct the field, so that pressing <ENTER> when the error is noted will not just accept the entered value. Lastly, the error tracking dataset is updated and the first entry is closed. This process is exactly the same for the character variables; the only difference is that the variable used for checking is character. In the above code, no format checking is done for character variables, but this is a simple modification should it be necessary.

There are alternative methods using arrays to assign the field values to a variable for checking. The disadvantage of using the array method here is the additional overhead in checking the array to assign the variable. While it will save coding time in development, it will, especially for larger screens, eventually cause a noticeable lag in entry. Implementing this as a FRAME entry will save both in development and in processing time, particularly for larger datasets and forms.

Error Tracking

Error tracking as implemented simply adds observations to an error dataset by using CALL SET and APPEND. Each of these error observations contains: subject ID, form name, field name, entry person and which resolution option (keep or change) was selected. The form name and subject ID are assigned in the INIT section. The entry person's ID is part of the signon procedure for the application and resides in a macro variable. The field name is obtained with the CURFLD function and the resolution option is the value chosen through DATALISTN. If desired, more detailed information on errors can be obtained by recording the first and second entry values for any field. This can be accomplished by adding the variables VAL, VALC, CHK and CHKC to the error dataset. None of the screens would need re-coding.

Determining the Active Entry From Inside The FSEDIT Screen Itself

In the application for which this method was developed, the active entry number for a given ID is usually determined in a calling SAS/AF application, or FSEDIT screen before the screen is opened. This way, the user is restricted to entering data for only that entry for a specific ID. However, there are screens designed for the rapid addition of data without specifying an ID in advance. The code below demonstrates how the active entry number is determined and what actions occur based on that entry number. The same technique is used in any calling applications.

```
ID:
val = .i;
/* Determine number of obs for id selected */
dsn = "hg1.hs3b (where=(id eq '" || left(id) || "'))";
passthru = open(dsn,'i');
rc = varstat(passthru,'touchdat','n nmiss',n1,n2);
lastobs = n1 + n2;
/* If first entry, load last field for completeness
check */
if lastobs eq 1 then do;
  rc = fetch(passthru);
  eid = getvarc(passthru,varnum(passthru,'eligid'));
end;
cmdstr = ' ';
select(lastobs);
/* No observations for this ID → first entry */
  when(0) do;
     protect id;
     entry = 1;
     cursor date;
  end;
/* 1 obs present, check for completeness */
  when(1) do;
```

```
/* Initial entry incomplete, go to existing first entry */
   if eid eq ' ' then do;
      rc = fetch(passthru);
      obx = curobs(passthru);
      cmdstr = "cancel; " || left(put(obx,8.));
      call execcmd(cmdstr);
      cursor momfn;
   end;
/* First obs complete → second entry, proceed
normally */
   else do;
      protect id;
      cursor momfn;
      entry = 2;
      flg = 'ENTRY 2';
   end;
 end;
/* Second entry already exists, go to it for editing */
 otherwise do;
   rc = where(passthru,"entry eq 2");
   rc = fetch(passthru);
   obx = curobs(passthru);
   cmdstr = "cancel; " || left(put(obx,8.));
   call execcmd(cmdstr);
   cursor momfn;
 end;
end;  /* Of SELECT */
rc = close(passthru);
return;
```

Code Details For Checking Entry Status

The process of checking the entry status of an added observation is fairly simple. The VARSTAT function obtains the number of missing and non-missing values of a numeric flag value: that sum is the total number of entries for the entered ID. This application uses a record modification date and time stamp. In addition to the count of entries, completion of the entry must also be taken into account. Without this check, partially filled entries will occur if the entry person does not finish entering the entire form before terminating the screen. In the current application, the last field on the FSEDIT screen (as well as the paper form) is used as a proxy for completeness of an entry. This field should never be missing on the paper form.

If there are no observations in the dataset for a given ID, then the one that is being added is the first entry. If there is one observation already present, it is checked for completeness. If it is complete, then the entry being added is the second entry. If not, the observation number (relative to the entire dataset) of that first entry is determined, the new observation is terminated and the dataset pointer is moved to the observation number of the first entry. If two observations exist for a given ID, then the second entry is the desired observation. To prevent more than two entries for any subject, a safety valve exists to automatically terminate a new observation that would become a third entry and returns the user to the observation number of the second entry.

In the current application, there is also code in the INIT and MAIN sections of the FSEDIT screen that effectively blocks the appearance of a completed first entry; therefore, no alteration of initial entries for subjects is possible once it has been completed. The authors would be happy to distribute the code as implemented for a complete screen, along with a sample dataset for any who are interested. However, a conditionally executed PROTECT _ALL_ would suffice to prevent alteration of the first entry.

Summary

This method works best in relatively small-scale data entry operations, where FSEDIT is the main tool for data entry quality assurance and the entry operators are not experienced or sophisticated. The extra time required in coding provides quality assurance without increasing the learning curve or the task burden for the end-users, given that double entry is required. There are no reports to cross-check, no separate editing of the second entry to resolve discrepancies and no need for centralized verification of the data. There is little or no real-time overhead for small-to-medium sized datasets, especially if the datasets are indexed, making its function transparent and automatic. This method can also be modified for use in FRAME entries, providing for instant double data entry verification in the next generation of SAS System data entry applications.

This work was partially supported by NHLBI grants HL 54473 and HL 47317.

Further inquiries are welcome to:

Derek Morgan
Division of Biostatistics
Washington University Medical School
Box 8067, 660 S. Euclid Ave.
St. Louis, MO 63110
Phone: (314) 362-3685 FAX: (314) 362-2693
E-mail: derek@wubios.wustl.edu

Figures 1 - 4: Sample Screen Dumps of the Double Data Verification

Figure 1: The first entry of the observation has just been concluded.

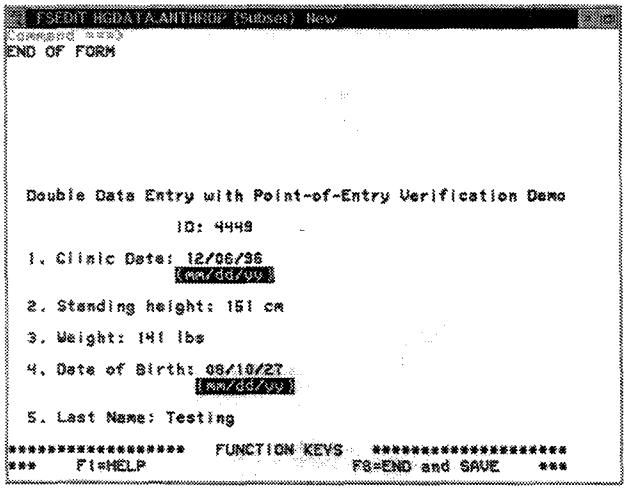

Figure 2: The same observation is opened for the second entry. Note the "ENTRY 2" next to the ID field, and that the clinic date field (which is not a double-entry field) is already filled in. The cursor is automatically positioned on the standing height field when this screen comes up.

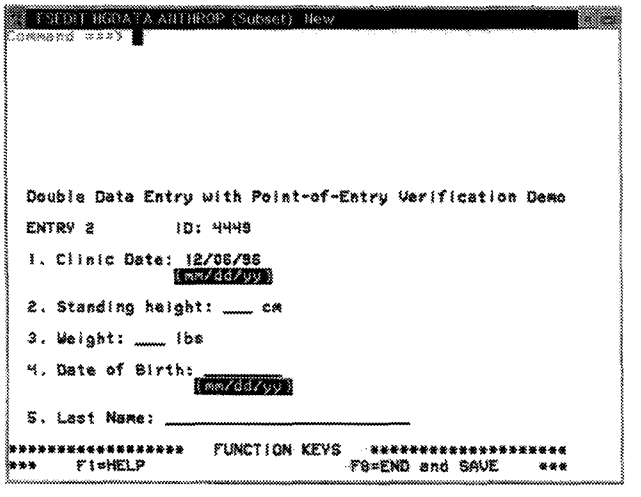

Figure 3: The corrective action window. The value in the date of birth field is not the same as the value for the first entry. The default selection is 1, "Change this entry." Note the correct formatting of the SAS date value in the error notification. The VARLABEL function could be used in the same way to put the variable label in the error string instead of the variable name.

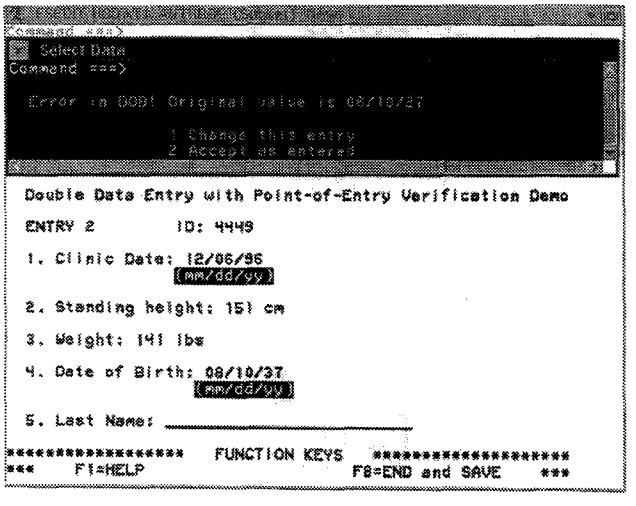

Figure 4: Option 1 has been chosen. The cursor moves to the beginning of the field in question, and the field itself is highlighted. The correct value may now be entered. Note: if the value is entered incorrectly a second time, the routine will flag the field again.