AN EASY-TO-USE DOUBLE-ENTRY ROUTINE

Trent L. McDonald
University of Washington

ABSTRACT

The presence of entry errors in a data set can effect both the accuracy and precision of data. To combat errors made during the entry process, double-entry routines have been developed. A general, easy-to-use, double-entry routine implemented in SAS® and SAS/FSP® software is presented. SAS/FSP is seen to provide an efficient interface for defining, entering, and updating SAS data sets. Enhancements to provide for data validation are discussed, as well as modifications when SAS/FSP is not available. In one implementation of the routine, an average of 53 out of every 10,000 values did not agree between first and second entry.

INTRODUCTION

Large amounts of data are keyed into computers via computer keyboards every day. Under usual circumstances, mistakes will be made during the entry process at some unknown, hopefully small, rate. In an effort to reduce data entry errors, various data entry schemes have been devised. One such data entry scheme is *double-entry*.

Double data entry involves two *entry occasions*; a *first* and a *second*. Values from the first entry occasion are compared with values from the second entry occasion and action is taken depending on the results of this comparison. If values entered on the first occasion agree with values entered during the second occasion, the values are stored. If values entered on tbe first occasion do not agree with values entered during the second, a process of reconciliation is started in which the user is usually prompted to reenter the mismatched value. Several double data entry schemes have been implemented and are commercially available, but all follow this general scheme. Provided that the same mistake is not made twice, a double-entry routine will catch errors made during entry and allow the user another chance to enter the correct value. However, even with the use of a double-entry routine, both entry errors and measurement error may exist in the final data set.

Presented in this paper is a double-entry scheme implemented in SAS and SAS/FSP software. By taking advantage of the built-in features of SAS and SAS/FSP, the double-entry scheme presented here requires a minimum of effort to create and enter data for a wide variety of data sets. It is hoped that the routine will offer an inexpensive alternative to dedicated data entry programs for SAS and SAS/FSP users.

In the *Program Description* section of this paper, the logic of the program is presented and major macros are described. In the *Program Implementation* section, an implementation of the double-entry routine is described. Following this, tbe *Results Using Double-Entry* section describes the number of discrepancies found during entry of three separate data sets. In the *Discussion* section, valid-value and valid-range checks are discussed, as well as how to proceed if SAS/FSP is not available. Source code for some of the more complicated macros in the double-entry scheme routine is given in the *Appendix*.

PROGRAM DESCRIPTION

The double-entry program was written and implemented using PC SAS and PC SAS/FSP version 6.04. The program assumed cooperative, non-malicious, and intelligent use. However, rudimentary modifications are possible when these assumptions do not hold.

The double-entry routine used a combination of macro windows and calls to Proc FSEDIT to construct and fill two separate data sets (see Figure 1 for a flow diagram of the routine). One data set represented the first entry occasion while a second data set represented the second entry occasion. The opening macro window, named TASK in Figure 1, allowed the user to specify two things; a number corresponding to one of three actions (the *entry number*), and a data set name. If a "1" or "2" corresponding to the first and second entry occasions was entered for the *entry number*, macro %ENTER was called, otherwise both macro window ID and macro %VERIFY were called.

During the first entry occasion, if the named data set existed, Proc FSEDIT was invoked from within macro %ENTER with the DATA= option to allow both appending and updating of records (see %ENTER in File 3 of the Appendix). If the named data set did not exist, Proc FSEDIT was invoked with the NEW= option to allow the user to define variables on a new data set. During data set definition, any number of variables could be defined, as well as any type, length, label, and format. However, during verification discrepant variables were not formatted and thus date variables were entered as strings and converted later. Refer to the *SAS/FSP Software* guide for more details on defining data sets with

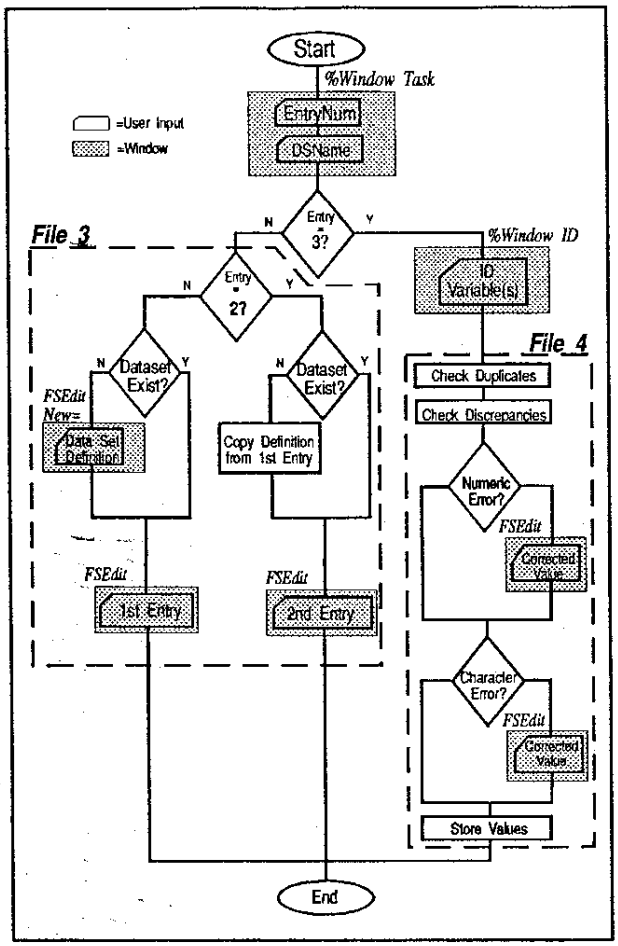

Figure 1: Flow diagram for the double-entry routine. Shaded boxes represent macro %Window and FSP interfaces.

the NEW= option in FSEDIT.

During the second entry occasion, if the second entry data set did not already exist, the structure of the first data set was copied to an empty data set. An error occurred if the first entry data set did not exist before an attempt to begin the second entry occasion. During second entry, Proc FSEDIT was then invoked from macro %ENTER with the DATA= option on either the newly created empty data set or on the existing second entry data set.

During verification (*entry number* = 3) and provided that both the first and second entry data sets existed, the macro window ID was displayed and macro %VERIFY was called. Macro window ID displayed the variables on the data sets and allowed the user to enter a variable or set of variables which uniquely identify an observation. By adding, and later deleting, an observation counter variable, a modified routine could potentially verify data sets without explicit ID variables.

With the ID variable(s) entered, macro %VERIFY first checked for duplicate ID values in the first and second entry data sets (see %FINDDUPS in File 4 of the Appendix). If any duplicate IDs were found, a note was written to the screen, and all duplicate observations, except the last one, were deleted and written to a permanent data set named IDDUPS in a permanent SAS library. Thus, only the last observation of a duplicate ID group was retained for verification and stored in the final double-entry data set.

After checking for duplicates, %VERIFY called Proc COMPARE to compare the first entry of every value in every observation with the second entry of every value in every observation. If a value in the first entry data set did not agree with the analogous value in the second entry data set, %VERFIY called Proc FSEDIT to allow the user to either correct the value or accept the second entry. Since numeric and character variables can not both exist in the same data array, numeric and character variables were treated separately during the verification process. Any numeric discrepancies were corrected first, followed by any character discrepancies. Once discrepancies were corrected, verified values were appended to the final or verified data set. Verified observations were also removed from the first and second entry data sets so that subsequent runs did not verify the same observations.

PROGRAM IMPLEMENTATION

The actual implementation of the double-entry routine involved four files. The first file, File 1, was the main routine which controlled execution. The second file, File 2, contained %Window definitions and other macros necessary for user input. The third file (enclosed in box *File 3* in Figure 1) contained the macro %ENTER. The fourth file (enclosed in box *File 4* in Figure 1) contained macro %VERIFY and other code for verification. The third file containing macro %ENTER as well as the fourth file containing macro %VERIFY are presented in the Appendix. Also presented in the Appendix is macro %GETVARS which, given a data set name, stores its variable names both as macro variables (&VARSN and &VARSC) and character values in a data set with a single observation. The special data set containing variable names is called VNAMES in the Appendix and is used in macro %CORRECT. The other two files (Files 1 and 2) primary purpose was to set appropriate macro variables. Table 1 is a list of the macro variables required by macros %ENTER and %VERIFY and set in Files 1 and 2. The macro variables listed in Table 1 can be set through %LET statements or in macro windows at run time. Using macro windows to set parameters at run

Required by %ENTER and %VERIFY:

Macro Variable Name	Description	Example
&LIBNAME*	Permanent SAS library to store data sets	SAVE
&ENTRYNUM	Action to take 1=1st Entry, 2=2nd Entry, 3=Verify	1, 2, or 3
&DSNAME	Name of final data set (used as root name)	MYDATA
&DATEXIST	Logical: 1 if &DSNAME exists, 0 otherwise	0 or 1

Required by %VERIFY only:

Macro Variable Name	Description	Example
&VARSN	List of numeric variables on data set	AGE WEIGHT INCOME
&VARSC	List of character variables on data set	NAME PHONE
&NVARSN	Number of numeric variables on data set	3
&NVARSC	Number of character variables on data set	2
&MAXNLN	Length of longest numeric variable	8
&MAXCLN	Length of longest character variable	30
&IDVAR	List of ID variables (any type)	NAME AGE
&LASTID	Last variable in &IDVAR	AGE
&NIDS	Number of ID variables listed in &IDVAR	2

*Example statements: Libname SAVE "c:\mylib"; %Let LIBNAME=SAVE;

Table 1: Global macro parameters set via macro %Windows before calls to macros %ENTER and %VERIFY.

time is more general since no %LET statements require editing each run.

Note that macro variable &DSNAME is the name of the *final* data set. In this application, a "1" or "2" wasappended to &DSNAME and the result used for the names of the first and second entry data sets. For example, if the name of the final data set is MYDATA, the first entry data set was named MYDATA1, while the second entry data set was named MYDATA2. This convention constrained &DSNAME to be at most seven characters long in this application.

Table 2 lists the essential PC keys and their function during interaction with Proc FSEDIT for either entering or correcting values. Key definitions will be different for different systems and keyboards. Users can and should assign commonly used functions and text to keys in the SAS KEYS window (type KEYS on the COMMAND line in FSEDIT to see the current key definitions). Refer to the *SAS/FSP Software* guide for a complete listing of commands available in Proc FSEDIT. Refer to the *SAS Language Guide* for more information on assigning both functions and text to keys.

Key	Function	Action
Tab		Next Variable/Field
Shift-Tab		Prev Variable/Field
PgDn	Forward	Next Observation
PgUp	Backward	Prev Observation
F3	Add	Append Observation
F8	Save	Save Data to Disk
F10	End	Save and Quit

Table 2: A subset of the default SAS/PC keys useful in Proc FSEdit.

During reconciliation of discrepant values in Proc FSEDIT (called from macro %VERIFY), users can either enter the correct value or, if the second entry is correct, leave the correct field blank. Again, the essential keys are listed in Table 2.

SOME RESULTS USING DOUBLE-ENTRY

The double-entry routine described here was used and monitored for a period of approximately three months during which time a single individual entered and verified a total of approximately 59,800 numeric and character values. During the entry and verification process, the individual performing the entry did not know that the number of discrepancies were being counted. In each of three separate data files, the proportion of values which did not agree between the first and second entry was 0.583%, 0.584%, and 0.426%. If one makes the assumption that one of the values was correct and one of the values was wrong when a value did not agree between entries, then on average half the number of discrepancies would have gone into the data set as errors had there not been a double-entry routine. In the data sets above, this represents 0.292%, 0.292%, and 0.213% or roughly 26 out of every 10,000 values. In a data set with 20 variables, that is roughly 5 entry errors every 100

observations. In our case, the number of errors remaining in the final verified data sets could not be determined, and thus single-entry error rate could not be compared with double-entry error rates.

DISCUSSION

Double data entry routines can be implemented using other SAS Institute products such as SAS/AF®, or utilizing Screen Control Language within SAS/FSP. It was the desire of this program to require a minimal amount of effort to construct and enter data for new data sets. All too often data entry and database entry routines are difficult to construct and users opt to enter data via a free format text editor or spreadsheet which is be prone to data entry errors.

In addition to double-entry routines, another common device used to combat data entry mistakes is the use of valid-values and valid-ranges. The program presented here is easily altered to perform such validity checks on the data entered. The alteration involves an additional data step in macro %ENTER to perform the validity checks with a possible additional call to FSEDIT to correct invalid values. However, performing validity checks decreases the routine's generality by requiring new valid-values and value-ranges when new data sets are constructed. In some situations, this loss of generality may be trivial compared with the possibility of invalid variable values entering the data set. Another option for performing validity checks is to use custom FSP screens which are permanently stored in SAS catalogs and contain valid-value ranges (i.e., maximum and minimum values specified in the permanent FSP screen).

A modified double-entry scheme can be implemented in cases where SAS/FSP is not available. The modification involves replacing all the calls to Proc FSEDIT with ordinary data steps. One data step could define the data set, while two other data steps could read separate ASCII data files representing the two entry occasions. The ASCII data files would be created using an outside program such as a text editor, spreadsheet, or database program. The procedure would then perform its usual comparison of the two entries and a report of the discrepancies could be generated. Discrepancies could then be corrected outside the SAS system and the routine reapplied until all values agree.

SUMMARY

The keying of data onto magnetic media is a critical step in many computer applications. SAS and SAS/FSP software were found to provide excellent facilities for the construction, entry, and possible correction of data sets. The presented routine is general in scope and requires a minimum of effort to enter a wide variety of data sets. SAS and SAS/FSP also provide more advanced features which can be used to enhance double-entry in specific applications. In one application of the routine, an average of 53 out of every 10,000 values did not agree between first and second entry.

REFERENCES

SAS Institute Inc., *SAS Procedures Guide, Version 6, Third Edition*, Cary, NC: SAS Institute Inc., 1990.

SAS Institute Inc., *SAS Language Guide for Personal Computers, Release 6.03 Edition*, Cary, NC: SAS Institute Inc., 1988.

SAS Institute Inc., *SAS Guide to Macro Processing, Version 6, Second Edition*, Cary, NC: SAS Institute Inc., 1990.

SAS Institute Inc., *SAS/FSP Software: Usage and Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc., 1989.

APPENDIX

File 3: Macro %ENTER

```
*-------------------------------------------------------------------------------;
%macro enter(entrynum,dsname,datexist);
%* Macro to do the first and second;
%* entries using Proc FSEDIT;
%*   ENTRYNUM = 1 or 2;
%*   DSNAME = <rootname>1 or <rootname>2;
%*     where <rootname> is name of
%*     final data set;
%*   DATEXIST = 1 if data set dsname exists;

%if (&datexist=0) and (&entrynum=2) %then
  %do;
    data &libname..&dsname;
      set &libname..&dsname.1;
      if _n_ >= 1 then delete;
      stop;
  %end;
proc fsedit label
  %if (&datexist=0) and (&entrynum=1) %then
    new=&libname..&dsname;
  %else
    data=&libname..&dsname;
  ;
run;
%mend;

*===============================;
```

File 4: Macros %MES, %FINDDUPS, %INTSECT, %CORRECT, and %VERIFY

```
*-------------------------------------------------------------------------------;
%macro mes(message);
%* Macro to put a message on the screen ;
data _null_;
  filename scrn terminal '';
  file scrn;
  put &message;
run;
%mend;
*-------------------------------------------------------------------------------;
%macro finddups(dsn);
%*  Macro to find duplicate values of id variables ;
%*  in data set DSN. Id variables stored in IDVAR;
  %* Sort the entries by the id variables ;
  proc sort data=&dsn;
    by &idvar;
  %* Find duplicates with use of BY statement;
  %let iddup=N;
  data &dsn &libname..iddups;
    set &dsn;
    by &idvar;
    if not(last.&lastid) then
      do;
        call symput('iddup','Y');
        output &libname..iddups;
      end;
    else
      output &dsn;
  run;
  proc print data=&libname..iddups;
    title "DUPLICATES found in &dsn";
  %if &iddup=Y %then
    %do;
      %mes(/ "Duplicate ID found in %upcase(&dsn)"
           / "Last duplicate ID used for verification."
           / "Other duplicates written to
%upcase(&libname).IDDUPS.");
      proc append base=&libname..IDDUPS data=iddups;
    %end;
%mend;
*-------------------------------------------------------------------------------;
%macro intsect;
%*  Macro to find the observations which;
%*  have same ID value between entries;
%*  (ie. intersecting observations);
data final(keep=&idvar);
  set diffs;
  if _type_="DIF" then
    output;
data final;
  merge &libname..&dsname.2 final(in=ovrlap);
  by &idvar;
  if ovrlap then output;
%mend;
*-------------------------------------------------------------------------------;
%macro correct(vtype,nvars,vlist,maxlen);
%* Macro to separate discrepancies, from data set DIFF;
%* and call FSEDIT to correct if needed.;
%*  VTYPE = NUM or CHR;
%*  NVARS = Number of variables in VLIST;
%*  VLIST = List of variables of type VTYPE;
%*  MAXLEN= Maximum length of any variable in VLIST;
%if &vtype=NUM %then
  %do;
    %let vt=N;
  %end;
%else
  %do;
    %let vt=C;
  %end;
%*  Compare first and second entry, data set FIX;
%*  contains the ones that do not match.;
data fix(keep=varname varnum fent sent cent &idvar);
  length fent sent cent fv1-fv&nvars
         sv1-sv&nvars %if &vt=C %then $; &maxlen;
  array frst{*} %if &vt=C %then $; fv1-fv&nvars;
  array sec{*} %if &vt=C %then $; sv1-sv&nvars;
  array varsn{*} %if &vt=C %then $; &vlist;
  array vname{*} $ v&vt.1-v&vt.&nvars;
  retain  fv1-fv&nvars sv1-sv&nvars v&vt.1-v&vt.&nvars;
  set diffs;
  if _n_=1 then
    set vnames;
  if _type_="BASE" THEN
    do;
      do i=1 to &nvars;
        frst{i} = varsn{i};
      end;
    end;
  else if _type_="COMPARE" then
    do;
      do i=1 to &nvars;
        sec{i} = varsn{i};
      end;
    end;
  else if _type_="DIF" then
    do;
      do i=1 to &nvars;
        if not((varsn{i} = 0) or ((frst{i}=.) and (sec{i}=.))) then
          do;
            varname=vname{i};
            varnum=i;
            fent=frst{i};
            sent=sec{i};
            cent=.;
            output;
          end;
        frst{i}=.; sec{i}=.;
      end;
    end;
%* Set MISMTCH to 1 if any values need reentry ;
%let mismtch=0;
data _null_;
  set fix nobs=n;
  if not(n=0) then
    call symput('mismtch','1');
  stop;
run;
%* Invoke FSEDIT to correct/change second entry if needed;
%* Leaving correct field missing will accept second entry;
%if &mismtch = 1 %then
  %do;
```

```
  proc fsedit
    data=fix label;
    var &idvar varname fent sent cent;
    label   fent="First entry: "
            sent="Second entry: "
            varname="Variable with discrepancy: "
            cent="CORRECT value here ->";
  data fixed(keep=&idvar &vlist);
    set fix;
    by &idvar;
    array varsn{*} %if &vt=C %then $;&vlist;
    retain &vlist;
    varsn{varnum} = cent;
    if last.&lastid then
      do;
        output;
        do i=1 to &nvars;
          varsn{i} = %if &vt=C %then ''; %else .;;
        end;
      end;
   data final;
    update final fixed;
    by &idvar;
  %end;
%mend correct;
*-------------------------------------------------------------------------------;
%macro verify(dsname);
%* Macro to verify first and second entries ;
%* DSNAME = <rootname> of data sets;
proc compare data=&libname..&dsname.1
             compare=&libname..&dsname.2
             out=diffs
             outbase
             outcomp
             outdif
             noprint;
  id &idvar;
%* Find matching ID values ;
%intsect;
%* Check numeric variables if any;
%if &nvarn>=1 %then
  %correct(NUM,&nvarn,&varsn,&maxnln);
%* Check character variables if any;
%if &nvarc>=1 %then
  %correct(CHR,&nvarc,&varsc,&maxcln);
%* Update first and second entry data sets by deleting verified
obs ;
data &libname..&dsname.1;
  merge &libname..&dsname.1 final(keep=&idvar in=verified);
  by &idvar;
  if (verified) then delete;
data &libname..&dsname.2;
  merge &libname..&dsname.2 final(keep=&idvar in=verified);
  by &idvar;
  if (verified) then delete;
%* Save verified observations to permanent data set ;
proc append base=&libname..&dsname data=final;
%mend verify;

*=============================================;
```

Macro %GETVARS

```
*-------------------------------------------------------------------------------;
%macro getvars(dsn);
%* Macro to store variable defined on DSN;
%* in global macro variables and temporary;
%* data set;
%global varsn varsc nvarn nvarc maxnln maxcln;
proc contents data=&dsn out=outvars noprint position;
proc sort data=outvars;
  by type varnum;
data _null_;
  length varsn varsc $200;
  retain varsn varsc maxcln maxnln;
  set outvars end=last;
  if type=1 then
    do;
      maxnln = maxnln <> length;
      nn+1;
      varsn = left(trim(varsn)) || " " || left(trim(name));
    end;
  else
    do;
      maxcln = maxcln <> length;
      nc+1;
      varsc = left(trim(varsc)) || " " || left(trim(name));
    end;
  if last then
    do;
      call symput("varsn",left(trim(varsn)));
      call symput("varsc",left(trim(varsc)));
      call symput("nvarn",left(trim(nn)));
      call symput("nvarc",left(trim(nc)));
      call symput("maxnln",left(trim(maxnln)));
      call symput("maxcln",left(trim(maxcln)));
    end;
run;
%* Set up data set VNAMES with one observation containing;
%* the names of variables.;
%* VNAMES is used in macro %CORRECT.;
data outvars;
  set outvars;
  if type=1 then
    do;
      n+1;
      _idlab_="vn" || left(trim(n));
    end;
  else
    do;
      c+1;
      _idlab_="vc" || left(trim(c));
    end;
run;
proc transpose data=outvars out=vnames(drop=_label_);
  id _idlab_;
  var name;
run;
 %mend;
```