A GENERALIZED INTERACTIVE FULL SCREEN EDIT UPDATE SYSTEM USING SAS SOFTWARE

Donald J. Henderson, ORI, Inc.,
Connie G. Welch, ORI, Inc.,
Jesse Gary, ORI, Inc.

1. INTRODUCTION

The editing and updating of data is a very common application. SAS has a variety of tools that facilitate this function. These tools include:

- the UPDATE statement
- the MERGE statement
- PROC EDITOR
- PROC FSEDIT

Edit/Update applications run the range from manual correction of card-image data followed by the creation (or recreation) of SAS data sets through sophisticated full screen applications which update an existing SAS data set.

The use of the above mentioned tools in isolation does not satisfy what we will present as a set of minimum requirements for an update system. In particular, the UPDATE statement and the MERGE statement require user coding of the appropriate data steps to perform the update. Furthermore, the data base is not protected from unauthorized access or even destruction if the user must code or even just run his/her own update programs. Alternatively, the user may use PROC EDITOR or, more recently, PROC FSEDIT to update his/her data files. These procedures are more "user-friendly"; however, they include no built-in capabilities for building or maintaining audit trails. Since these two procedures allow for direct access replacement of data values, the building of audit trails is particularly difficult. As with the UPDATE and MERGE statements, these procedures do not provide for data security or protection.

2. FEATURES OF AN EDIT UPDATE SYSTEM

In this section, we will discuss a list of features that cover the range of requirements for an edit update system. They include:

- selected updates and transaction types
- end-users' accessibility
- comprehensive editing of updates
- maintaining audit trails
- backing out (undoing) updates
- data entry
- interactive vs batch application of updates
- data protection and integrity.

In the following subsections, we will discuss how the above requirements can be addressed by combining features of SAS. The core of our program will be PROC FSEDIT, through which all data changes are entered. The remainder of this section presents each requirement along with sample SAS code which satisfies the requirement.

For purpose of illustration, we assume that we are editing a permanent SAS data set SAMPLE.BUDGET. This data set contains three variables: DEPT, REVENUE and EXPENSES. Our core program is:

```
PROC FSEDIT DATA=SAMPLE.BUDGET;
```

2.1 Selected Updates and Transaction Types

When updating is done on a master data set, it is generally in the form of the replacement of field values. PROC FSEDIT accommodates this type of update, but other types may be desired by the user. For example, the user may want to multiply the existing value by some factor or increment it by a specific amount. In addition, if PROC FSEDIT is used in isolation for updating, all fields in the permanent data set will be rewritten; it is preferable to update only the fields to be changed. Therefore, there is a need for the means to create a traditional transaction data set in the interactive application.

In the example that follows, a pre-FSEDIT DATA step is used to create a local copy of the master data, create copies of the fields that can change, and create a set of change flags. These flags will be used to indicate the type of change to be made to the corresponding fields, and are initialized to 'N', indicating no change. During the FSEDIT step, the user enters values in the desired fields and indicates the type of change in the flag fields. For example, 'R', 'M', or 'A', respectively, can be used to indicate replacement, multiplication or addition. The DATA step that follows the FSEDIT screens the copy data set to detect records with changes; only these records will be kept and used to create the transaction data set. The indicated changes are then applied on the record, with any fields that are not to be updated being set to missing. The master data set is then updated using this transaction data set.

```
DATA COPY;
    ARRAY _VALUES(_I_) REVENUE EXPENSES;
    ARRAY _COPIES(_I_) REVCOPY EXPCOPY;
    ARRAY _FLAGS(_I_) $1 REVCHNG EXPCHNG;
    DROP _I_;
    DO WHILE(NOT LASTREC);
        SET SAMPLE.BUDGET END=LASTREC;
        DO OVER _VALUES;
            _COPIES=_VALUES:
            _FLAGS='N';
        END;
        OUTPUT;
    END;
    STOP;
RUN;
```

```
PROC FSEDIT DATA=COPY SCREEN=S;

DATA TRNSRECS(DROP=REVCOPY EXPCOPY REVCHNG
EXPCHNG);
ARRAY _VALUES(_I_) REVENUE EXPENSES;
ARRAY _COPIES(_I_) REVCOPY EXPCOPY;
ARRAY _FLAGS(_I_) $1 REVCHNG EXPCHNG;
DROP _I_ NOCHANGE;
SET COPY;
NOCHANGE=1;

DO OVER _FLAGS;
   IF _FLAGS NE 'N' THEN NOCHANGE=0;
END;
IF NOCHANGE THEN DELETE;
DO OVER _FLAGS;
 IF _FLAGS NE 'N' THEN
 DO;
   IF _FLAGS='A' THEN _VALUES= _VALUES + _COPIES;
   ELSE
     IF _FLAGS='M' THEN _VALUES= _VALUES*_COPIES;
 END;
 ELSE _VALUES=.;
END;
RUN;

DATA SAMPLE.BUDGET;
    UPDATE COPY (DROP=REVCOPY EXPCOPY REVCHNG
    EXPCHNG)
         TRNSRECS;
    BY DEPT:
RUN;
```

It should be noted that this use of a pre- and post-FSEDIT DATA step provides not only a great deal of flexibility in the types of updates, but also in the format in which the data is presented to the user. The 'observation' displayed on the screen does not have to reflect the structure of the master data. For example, by transposing the master data copy in the preliminary DATA step, it is possible to display data from several records on one FSEDIT screen. Consider a data set with one record per employee. It may be desirable to display all employees in a department on a single screen.

2.2 End-Users Accessibility

In any system, a desirable feature is that any function can be invoked by a user without requiring users to write programs. This is particularly true of edit-update applications since such applications are frequently used by personnel not experienced in data processing, such as secretaries, data entry clerks, financial analysts, etc.

One method which can be used to make the update application easily accessible is the use of menus. For example, upon logging onto the computer system, an SPF menu is displayed to the screen. The user can then invoke any application, including the update, by making the appropriate selection.

Menus can also be built using SAS/FSP. Alternatively, menus can be built using the MACRO language. An example of using MACRO follows:

```
%MACRO BUDGET;
    %LET DONE = 0;
    %DO %WHILE (&DONE = 0);
         %PUT SELECTION MENU;
         %PUT %STR (   ) 1 Edit Update;
         %PUT %STR (   ) 2 Report;
         %PUT %STR (   ) 0 Quit;
         %PUT %STR (   );
         %PUT ENTER CHOICE:;
         %INPUT ANS;
         %IF &ANS=1 %THEN %EDUPD;
         %ELSE %IF &ANS=2 %THEN %REPORT;
         %ELSE %IF &ANS=0 %THEN %LET DONE=1;

         %ELSE
             %PUT INVALID - SELECT FROM MENU;
    %END;
%MEND BUDGET;
```

The MACRO can be executed automatically by a SAS CLIST or EXEC, or the user can enter the system by keying "%BUDGET". The edit update module can be executed by keying "%EDUPD" into interactive SAS. Through the use of statement style MACROS, such approaches are even easier for end-users.

2.3 Comprehensive Editing of Updates

Generally, the updating of values in a data set is done in a single step: i.e., transactions or updating code are applied to the master data at one time. If FSEDIT is used, as in our sample program, the update will immediately be applied to the master data set, and any erroneous values must be detected later and additional updating done to correct these errors.

An update application can be developed that allows spontaneous editing and correcting of updates. The code and text that follow illustrate this concept.

First, a local copy is made of the master data, SAMPLE.BUDGET. The updates will be applied by FSEDIT to this copy, eliminating the problem of updating the master data in place and destroying the original values if an error is made. This step creates copies of the fields to be updated; the copy fields hold the original values; the updates will be applied to the original fields. The next 3 steps are the core PROC FSEDIT, an editing DATA step, and a PROC APPEND; these are executed iteratively. In the PROC FSEDIT, the user enters new values in the REVENUE and EXPENSES fields; the screen should be set up so that the copy fields are not displayed to the user. The DATA step edits the modified fields for bad values. All good records are output to the data set GOOD. If there are any bad records, a flag is set, the field values are restored from the copy values, and the records are output to the data set BAD. The value of the macro variable &DATASET, which holds the name of the input data set for the PROC FSEDIT and the DATA step, is set to 'BAD' or 'NOMORE', depending on whether there were errors or not. The good records are then appended to the data set CUMGOOD, which after all editing and correcting will hold the complete

copy data set. If there were no errors, the permanent data set SAMPLE.BUDGET is replaced. If there were errors, the bad records (data set BAD) are processed through the FSEDIT, DATA step, and APPEND, as before; the program will iterate through these steps until there are no more bad records. At that point, SAMPLE.BUDGET will be replaced.

```
DATA COPY;
SET SAMPLE.BUDGET;
EXPCOPY=EXPENSES:
REVCOPY=REVENUE;

%MACRO EDIT;

%DO %WHILE (&DATASET NE NDMORE);

    PROC FSEDIT DATA=&DATASET SCREEN=S;

    DATA GOOD (DROP=EXPCOPY REVCOPY)
         BAD;
    SET &DATASET END=LASTREC;
    RETAIN ANYBAD;
    IF EXPENSES GT REVENUE THEN
    DO;
         ANYBAD = 1;
         EXPENSES = EXPCOPY;
         REVENUE = REVCOPY;
         OUTPUT BAD;
    END;
    ELSE OUTPUT GOOD;
    IF LASTREC THEN
     IF ANYBAD THEN
                CALL SYMPUT('DATASET', 'BAD');
    ELSE CALL SYMPUT('DATASET', 'NOMORE');

   PROC APPEND DATA = GOOD BASE = CUMGOOD;
%END;

PROC SORT DATA = CUMGOOD OUT = SAMPLE.BUDGET;
BY DEPT;

%MEND EDIT;

%LET DATASET=COPY;

%EDIT
```

Immediate editing of values entered by the user can be achieved by utilizing one of the properties of PROC FSEDIT. When the observation being updated is displayed on the screen after the user hits 'RETURN' or 'ENTER', the formatted value will be displayed for that field. By using formats to identify valid values, invalid values for fields that are restricted to discrete values or exclusive ranges can be identified. An example is:

```
    PROC FORMAT;
         PICTURE DEPCODE 1,3,4,7='99'
                         9-15   ='99'
                         OTHER  ='INVALID';

    DATA COPY;
         SET SAMPLE.BUDGET;
         FORMAT DEPT DEPCODE.;

    PROC FSEDIT DATA=COPY SCREEN=S;
```

2.4 Maintaining Audit Trails

One very useful component of an edit/update application is the creation and maintenance of an audit trail data set. This feature can provide users and management with two major advantages: it provides documentation of all updates applied to the master data set, and can be used as a 'backup' copy of the original records in case restoration of values is required later.

A DATA step is used prior to the PROC FSEDIT to create the local copy of the master data set and make copies of the fields to be updated. The date and time are obtained in this step, and will be part of the audit trail records. After the PROC FSEDIT, the copy data set is screened to detect updated records, outputting only those that were changed to the transaction and audit data sets. The new set of audit records is then concatenated to the end of the permanent audit trail (SAMPLE.AUDITRL).

```
DATA COPY;
    FORMAT TRNDATE DATETIME18.;
    RETAIN TRNDATE;
    IF _N_=1 THEN TRNDATE=DATETIME(  );
    DO WHILE (NOT LASTREC);
         SET SAMPLE.BUDGET END = LASTREC;
         REVCOPY=REVENUE;
         EXPCOPY=EXPENSES;
         OUTPUT;
    END;
    STOP;
RUN;

PROC FSEDIT DATA=COPY SCREEN=S;

DATA TRNSRECS (DROP=REVCOPY EXPCOPY TRNDATE)
    AUDITRCS;
    SET COPY;
    IF (REVCOPY NE REVENUE) OR
    (EXPCOPY NE EXPENSES) THEN OUTPUT;
RUN;

PROC APPEND DATA=AUDITRCS BASE=SAMPLE.AUDITRL;
```

2.5 Backing Out Updates

The ability to automatically 'back out' or undo updates that were in error is crucial. Depending on when the error is realized, the technique for backing out updates is different.

If the error is realized immediately on the FSEDIT screen, the "CANCEL" command can be used. If the error is discovered within a single FSEDIT session, but the change has been processed, edit flags, as discussed earlier, can be used to restore the original values from "copy" values in a post processor data step. A sample follows:

```
PROC FSEDIT DATA=COPY SCREEN=S;

DATA COPY;
    SET COPY;
    ARRAY _VALUES(I) REVENUE EXPENSES;
    ARRAY _COPIES(I) REVCOPY EXCOPY;
    ARRAY _FLAGS(I) $1 REVCHNG EXPCHNG;
    DO I=1 TO 2;
         IF RSTR_FLG='Y' OR _FLAGS='N'
              THEN _VALUES=_COPIES;
    END;
```

If the error is realized after the updates have been processed and loaded, they can be "backed-out" by processing an audit trail to select the record corresponding to the first change made in the indicated time window. Assuming the "copy values" are on the audit trail and correspond to the original values, a MACRO such as the following could be used to reload original values:

```
%MACRO BACKOUT(SDATE,EDATE);
    DATA BACKOUT;
         SET SAMPLE.AUDITRL;
         IF &SDATE LE TRNDATE LE &EDATE;
    PROC SORT;
         BY DEPT TRNDATE;
    DATA BACKOUT;
    SET BACKOUT;
         BY DEPT;
         KEEP DEPT REVCOPY EXPCOPY;
         RENAME REVCOPY=REVENUE
                EXPCOPY=EXPENSES;
         IF FIRST.DEPT;
    DATA SAMPLE.BUDGET;
         UPDATE SAMPLE.BUDGET
                BACKOUT (IN=B);
    BY DEPT;
    IF B THEN TRNDATE=DATETIME();
%MEND BACKOUT;
```

This MACRO can be used to back out updates made in a specified time period. For more complicated situations, a comparable approach could be taken.

2.6 Data Entry

If the application should allow data entry of new records, this can be done directly in FSEDIT. Typically, a post-processor edit and sort will be needed in such cases.

If new records are not allowed, there are a variety of techniques that can be used. One technique is to use a special screen data set with a key field (such as DEPT) defined as both protected and required. This prevents entry of new records from the screen. It does not prevent records from being added through use of the DUP PF key. A better technique is to determine the number of observations before FSEDIT and then to only process those observations in the post-processing data step. Sample code to implement this follows:

```
DATA _NULL_;
    CALL SYMPUT ('NOBS', PUT (TOTOBS, BEST.));
    STOP;
    SET SAMPLE.BUDGET POINT=_N_ NOBS=TOTOBS;

PROC FSEDIT DATA = SAMPLE.BUDGET SCREEN=S;

DATA COPY;
    IF _N_=2 AND NOT LASTREC THEN
    DO/*OBS ADDED*/;
         FILE SCRN;
         PUT 'OBS CANNOT BE ADDED';
    END/*OBS ADDED*/;
    DO I=1 TO &NOBS;
         SET COPY END=LASTREC;
         .
         .
         .
         .
    END;
```

2.7 Interactive vs Batch Application of Updates

Depending upon the circumstances, it may be desirable to load the updates into the master file during the interactive session (foreground) or at some later point in time (background).

Immediate loads can be accomplished by having FSEDIT act on the saved SAS data set (this is not recommended) or by loading the post-processed transactions using the UPDATE statement.

Alternatively, transactions can be bulk-loaded at a later time by having them appended (using PROC APPEND) to a data set which is used to update the master file on a regularly scheduled production basis. Once the records have been loaded, the temporary repository can be initialized to null. Sample program shells follow:

```
*Post transactions;

    DATA TRANS:
         SET COPY;
         .
         .
         .
         .
    PROC APPEND BASE=MYTRANS.BUDGET
                DATA=TRANS;

*Load transactions;
    DATA SAMPLE.BUDGET;
         UPDATE SAMPLE.BUDGET MYTRANS.BUDGET;
         BY DEPT;
    DATA MYTRANS.BUDGET;
         SET MYTRANS.BUDGET;
         STOP;
```

The bulk load approach is particularly useful in addressing the problem of multiple users who need to edit the same Data Library or Data Set. By giving each user a separate "temporary" repository for his/her transactions, no user needs exclusive use of the data set since he/she is not writing to it. Thus, multiple users can "edit" the same data set concurrently. Furthermore, the step that creates the local copy for editing could select only those records which the given user should have access to.

2.8 Data Protection and Integrity

If the data needs protection from unauthorized access, read and write protect passwords can be used on the data sets. The data can also be encrypted (see PROC CRYPT, a SAS 82.4 Supplemental Procedure).

One form of protection that should not be overlooked is structuring the system so that users do not know where the data is and can only access it with system applications. One technique is to set up logonids so that upon logging on, the user is presented with a choice of applications available through a menu and upon exiting the menu, the user is logged off.

None of these techniques is failsafe. We must recognize that no system or set of data is completely safe from a determined and competent programmer.

3. OVERVIEW OF A SAMPLE SYSTEM

This section presents an edit/update application which has been implemented through a combination of some of the previously described techniques. A diagram illustrating the techniques used is shown in Figure 1.

The application was developed to meet the following requirements:

- must be usable by non-dp personnel
- must be interactive
- must maintain an audit trail
- data base must be protected from access except through approved applications
- must perform multiple types of updates.

and is executed by the user by selecting the appropriate option from an SPF menu.

Special FSEDIT screens greatly simplify the updating process by displaying information in a form that reduces ambiguity and increases the verifiability of transactions. In this application, special screens (see Figure 2) were designed that display information using screen attributes according to each variable's characteristics. For instance, key variables, because they are not user-entered, are defined as both protected and required. This prevents the user from inadvertently adding or changing key variable values.

This application did not include the iterative editing discussed in Section 2.3 because the MACRO Language was not available at the time.

4. CONCLUSION

A flexible, "user-friendly" updating mechanism can be easily built by appropriately combining SAS programming tools. By combining such tools, and not forcing one procedure/technique to do everything, it is possible to build complete update systems which can be adapted to meet most, if not all, of a user's needs.

The author can be contacted at:

ORI, Inc.
Software Applications and Training
Program Office
7910 Woodmont Ave., Suite 1405
Bethesda, MD. 20814
(301) 656-3276

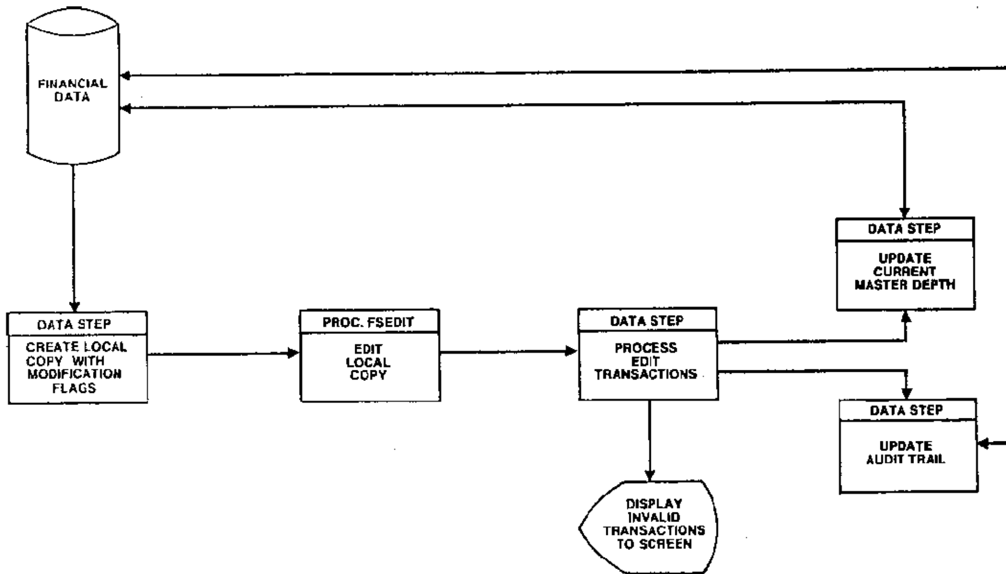

FIGURE 1

FSEDIT SCREEN MODIFICATION

COMMAND ⇨ SCREEN 1 OBS 129

DATA CHANGES OR ENTRY — CEMA BUDGET & PLANNING SYSTEM

	AMOUNT	TYPE CHANGE
CURRENCY	1000 DEM	
PLAN YEAR 5 (1988)	380	N
PLAN YEAR 4 (1987)	340	N
PLAN YEAR 3 (1986)	300	N
PLAN YEAR 2 (1985)	270	N
PLAN YEAR 1 (1984)	240	N
CURRENT YEAR (1983) EST/ACT	211	N
CURRENT YEAR (1983) BUDGET	164	
PRIOR YEAR (1982) ACTUAL	188	

"TYPE CHANGE" KEY

N = NO CHANGE

A = ADD THIS AMOUNT TO ORIG. AMOUNT

S = SUBTRACT THIS AMOUNT

M = MULTIPLY ORIG. AMOUNT BY THIS NUMBER

R = REPLACE ORIG. AMOUNT WITH THIS AMOUNT

FIGURE 2