USING SAS® SOFTWARE WINDOWING PRODUCTS TO DEVELOP
A REMOTE DATA ENTRY DISKETTE TRACKING SYSTEM

John C. Bowen, Merck Research Laboratories
Cathy Carfagno, Merck Research Laboratories

Introduction
Merck Research Laboratories has long been involved in
capturing clinical data via Remote Data Entry (ROE) systems.
One such system runs on a PC located at an investigator's
office to allow patient information to be captured and
checked against programmed constraints. The resulting data
is sent to Merck via diskette where a process is initiated to
copy the diskettes, print-off the data, reformat the data to a
central database, and review the data at separate intervals.
The faster this is accomplished, the sooner the data can be
used for a New Drug Application to be submitted to the FDA.
A diskette tracking system was developed as a means to
measure turn-around times, identify bottlenecks and inform
management and data reviewers of their project's data entry
status.

The focus of this application entails capturing and reporting
who has completed their part of the processing and when.
The extra time to use the application was a main concern so
features were added to expedite the entry of tracking data.

Program Structure
At the core of the tracking application is a normalized SAS
database. Normalization is the action to reduce the data's
grouping into datasets or tables to common dependancies. All data in one dataset must be dependant on the same key
fields and only those fields. This is the extent to which we
applied normalization.

The Master File is a SAS dataset that contains the majority
of the data (fig. 1). The other SAS datasets support the
integrity of the Master File. The records of these
supportive files are updated. and their integrity maintained
through a series of windows and programs utilizing
SAS/AF®, SAS/FSEDIT®, and SAS/FSVIEW®. These
support programs are beyond the scope of this
presentation.

Figure 2 displays the flow and calling sequences for a
subset of the application. The process starts with the
selection of a previously defined drug entity or the
definition of a new drug. From the Main Menu, support
files can be either updated or viewed, or the master dataset
can have records added or updated. There are a few
functional differences between the master dataset "ADD"
and the "UPDATE" therefore they are kept separate. The
following two paragraphs reference the area of the diagram
in the box.

The first window (fig. 3) to be discussed was developed in
SAS/AF to aid the user in selecting the records to be
updated. In the next section we will discuss use of the
DATALISTC function and WHERE clause to make the
selections.

The next window called (fig. 4) was developed in
SAS/FSEDIT. It provides the user, the opportunity to go
through each record for updating. The enhanced
functionality of "carrying forward values" will be discussed
in the next section.

Program Specifics

Referential Integrity through the use of the DATALISTC
Function, NODUPKEY Option and WHERE Clause Option.

In the Selection Criteria window (fig. 3), the user can either
provide actual values known to be valid or is given
assistance if they type a "7" in place of the value. The "7"
will trigger the SAS DATALISTC or DATALISTN function.
A resulting pop-up window gives the user valid choices to
select from.

To accomplish this, the SAS dataset is first opened and
returns a dataset identifier (fig. 5, item A.). The identifier
is then used in a sort function (item B.). This is performed
using the arguments -dataset identifier, bfield to sort on,
the NODUPKEY option and a resulting output file. The
NODUPKEY maintains the uniqueness constraint in
accordance to the first level of integrity by eliminating
duplicate key field values. The DATALISTC function (item
C.) is performed on the sorted file after it is opened and
assigned a dataset identifier. The DATALISTC function
uses the arguments - dataset identifier, field to display, and
a message to display for the user's assistance.

Once the subset is defined, the user is able to update the
records via a CALL to FSEDIT with a WHERE clause. The
conditional value part of the WHERE clause is generated by
the user's response and stored in an SCL variable called
TPR (fig. 5, item D.). The values of the condition are either
selected from one of the possible values returned by the
DATALISTC or by simply typing a valid value into the field.
The whole WHERE clause is completed in the ODSNAME
variable (item E.). Then ODSNAME is used in the FSEDIT
CALL (item F.).

In the Master Record window (fig. 4), the user updates the
fields to the records matching the conditions but can not update the fields selected on. The supportive files contributed the values for many of the key fields (MKNUM, GENSTUDY, WRKBOOK) as well as attributes of these fields (PROJECT, WRKNAME, PROTNAME, INVNUM, INVNAME). These cannot be modified at this point, which is indicated by not being highlighted.

Finally, with the user’s addition of values for ALLOC, the combination of these key fields then uniquely identifies the Master records. The resulting records in this application, have been contained within the confines of the required subset constraints (through the previous window’s selection) and mandatory constraints (by forcing the separate entry and maintenance of the supportive, foreign keys).

Carry Values Forward Capability

One major function of the system is the ability to carry values forward from previously entered records during the current session. This functionality is located in the Master Record window (fig. 4). This allows for fewer key strokes and therefore enhanced speed and accuracy. Values can be either rolled forward or backward.

This has been accomplished by integrating SAS, SCL and MACRO coding. In the INIT block, upon accessing a new record, the screen variable’s (Retby) values are assigned to corresponding macro variables (CRetby) (fig. 6, item A.). With FEDIT, each time a record is left the TERM block is executed. Each screen variable (Retby) is compared to a corresponding macro variable named with a ‘C’ (CRetby, item B.). If they don’t equal (screen variable was just modified) then a return code (RC_Retby) will be set to ‘1’ and the value of the modified screen variable will be assigned to another macro variable (MRetby).

In the INIT block, upon accessing a new record, again the screen variable’s (Retby) value is assigned to a corresponding ‘C’ macro variable (CRetby, item C.). If the field had been updated on the previous window (as detected by the return code), then the question is asked whether the user wishes to bring forward the values entered on the previous window. A positive response will assign the ‘M’ macro variable (MRetby) to the window variable (Retby, item D.).

Conclusion

Though speed is often a concern, it is weighed against the necessity of having accurate information. The system is slowed by the number of windows used, but the windows are required for logical separation of the levels of data. The effort made to elaborate the system by accessing dependent data and by carrying values forward has aided in maintaining the integrity of the data and allowed for quick and easy keying. The SAS techniques used for this tracking system could be applied to other applications where data are entered through multiple windows.

The managers of the RDE projects have found the system valuable for informing them where their projects are in the process and whether they are on schedule. It is also expected that the tracking data will provide the means to measure improvements in the overall RDE process.

```plaintext
VAR
Command ->
Libref: MAST
Dataset: MASTER1

Variable Length  Format  Informat  Label
- PROJECT   $24   N
  MNUM           $5   Y
  PRTBY $3   .N
  WRKBOOK $3  3.   Y
  WRKNAME $34  N
  WRKNAME $34  N
  INVNAME $24  N
  INVNUM   $24  N
  INVNAME $24  N
  MTHMTH $1   N
  DNUM     4  4.  4.1 N
  ALLOC    $4   N
  RECDATE 7   DATE7. DATE7. N
  RECDATE 7   DATE7. DATE7. N
  RETBY $3   N
  RETBY $3   N
  RETBY $3   N
  RETBY $3   N
  RETBY $3   N
  RETBY $3   N
  RETBY $3   N
  RETBY $3   N
  RETBY $3   N
  RETBY $3   N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N
  ZDORDATE 7   DATE7. DATE7. N

ZOOM

== FIGURE 1 ==
```
-- FIGURE 2 --
V

+ BUILD: SOURCE LOGIN030.PROGRAM (E) +

| Command = = =>
| 00001
| 00002 INIT:
| ...
| 00008
| 00009 MAIN:
| ...
| 00013 ASKP = 0;
| ...
| 00019 IF (SUBSTR(PR,1,1) = '?') THEN ASKP = 1;
| ...
| 00049 IF ASKP OR ASKG OR ASKW OR ASKD OR ASKA THEN DO;
| 00050 
| 00051 IF PR NE ' ' THEN ASKP = 0 THEN DO;
| 00052 
| 00053 IF COMARE =''
| 00054 THEN COMARE = TPR;
| 00055 ELSE COMARE = COMARE | ' ' AND ' ' | TPR;
| 00056 END;
| ...
| 00081 IF COMARE =''
| 00082 THEN ODSNAME = "MAST.MASTER1";
| 00083 ELSE ODSNAME = "MAST.MASTER1 (WHERE = ( " \ COMARE " ))";
| 00084 A 00085 RCO = OPEN(ODSNAME,'I');
| ...
| 00093 IF ASKP THEN DO;
| 00094 PR = ' ';
| 00095 LINK LEXA1;
| B 00096 RC1 = SORT(RCO,'PROTNUM','/NODUPKEY OUTPUT = WRK');
| 00097 RC2 = OPEN('WRK','I');
| C 00098 PR = DATALISTC(RC2,'PROTNUM PROTNAME',
| 00099 'PROTNUM PROTNAME');
| 00100 RC3 =CLOSE(RCO);
| 00101 RC4 =CLOSE(RC2);
| ...
| 00080 END;
| 00081 END;
| 00180 RETURN;
| ...
| 00204 RETURN;
| 00205 EDTSCRN;
| 00207 COMPARE =''
| 00208 IF PR NE ' ' THEN DO;
| 00209 TPR = 'PROTNUM = '' | PR | ''';
| 00210 IF COMARE =''
| 00211 THEN COMARE = TPR;
| 00212 ELSE COMARE = COMARE | ' ' AND ' ' | TPR;
| 00213 END;
| ...
| 00238 IF COMARE =''
| 00239 THEN ODSNAME = "MAST.MASTER1";
| 00240 ELSE ODSNAME = "MAST.MASTER1 (WHERE = ( " \ COMARE " ))";
| F 00241 CALL FSEDIT (ODSNAME,'LOGSYS.SCREENS.MASTUPD.SCREEN','EDIT');
| ...
| 00249 RETURN;
+-----------------------------------------------------------------------+ 1002
**FSEDIT Program**

Command = =>

/******************** UPDATE AND BROWSE - FSEDIT PROGRAM ********************/

FSEINIT:
RC=PMENU('LOGSYS.AFPGM.MYFSED');
SELMK=SYMGET('SELMK');
A RC_RETBY=0;
...
RETURN;

INIT:
C CALL SYMPUT('CRETBY',RETBY);
...
IF RC_RETBY = 1 OR RC_PRTBY = 1 OR RC_COPYBY = 1 OR RC_FRDBY = 1
...
OR RC_SENTBCK = 1 OR RC_EXCODE = 1 OR RC_RDEREW = 1 THEN
CURSOR BRNGFRWD;
ELSE
CURSOR RECDATE;
RETURN;

MAIN:
IF BRNGFRWD = 'Y' OR BRNGFRWD = 'Y' THEN DO;
IF RC_RETBY THEN RETBY = SYMGET('MRETBY');
...
BRNGFRWD = 'Y';
END;
RETURN;

TERM:
IF RETBY NE SYMGET('CRETBY') THEN DO;
RC_RETBY = 1;
CALL SYMPUT('MRETBY',RETBY);
END;
ELSE
RC_RETBY = 0;
...
CALL EXECMDI('PMENU OFF;');
RETURN;

--- FIGURE 6 ---