MANAGING A MULTI-USER SAS® DATABASE WITH VERSION 6 SAS/AF® SOFTWARE: GENERIC TASKS

Todd C. Folsom, Duke Power Company

INTRODUCTION

Tasks generic to database management include providing for data entry and data validation, updating master datasets, making backups, controlling access and keeping quality assurance records. All of these tasks are more complex when multiple users have to be accommodated. However SAS/AF and base SAS provide many tools for accomplishing them.

There are three major areas of activity that have to be managed: 1) data entry and validation by many users, 2) letting the users generate printed requests to update (append) validated data to the appropriate permanent dataset, and 3) having authorized updaters supply information on the source and destination of the data to the AF application. It is good security practice to rotate the duty of updating, so the AF application should be able to determine who is the next person in rotation and assign the job to them. This determination can be integrated with keeping track of update activity by maintaining a dataset of quality assurance records.

Figure 1. The main menu for the zoology database.

The database that I designed runs on a single personal computer and handles environmental monitoring data (Fig. 1). The flow of activity starts with individuals entering and validating their raw data. Then they generate a printed request for updating that is addressed to certain authorized users. Next, the person doing the updating uses the printed request to provide the needed information for appending new data onto existing datasets and making backups. We keep quality assurance (QA) records along the way to meet the requirements of state laboratory certification programs.

There is no need to edit the final datasets unless an error creeps through the validation process. Hence the SAS/AF application provides for individuals to access their personal subdirectory, but, unless they are also one of three authorized updaters, they are kept out of the subdirectories where the permanent datasets reside.

The purpose of this paper is to describe how multiple users can be accommodated, how access can be controlled by identifying users to the system, how to identify source and destination for datasets, how a rotation in updating duties can be achieved, and how QA records can be generated.

LAYOUT OF THE HARD DISK

The major separation to make in laying out the directory structure of the hard disk is between the personal subdirectories of users and the subdirectories where data will be stored (Fig. 2). The \DATA subdirectory contains the AF application catalog, the format library, and certain lookup files needed across the database. Subdirectories under \DATA contain the environmental data separated by major type, and this is a matter of organizational convenience. This keeps the permanent datasets isolated from user work areas. The AF application will keep most users from accessing the subdirectories of \DATA (see below), but it may be wise to protect these areas with security software.

Figure 2. Layout of the hard disk directory structure, keeping data and users separated.

HANDLING MULTIPLE USERS

Each user is forced by menuing software, such as PROTEC®, to start the SAS system and the AF application from his personal subdirectory. Each user was given a customized copy of Autoexec.sas, and the file sets up the appropriate parameters (Fig. 3). If the user is one of the authorized updaters, then libname statements for the subdirectories containing permanent datasets are listed. If not, only libnames for USER and the subdirectory \DATA are specified.
Since the various users have personal subdirectories under \USER (Fig. 2), the default SAS libref of USER is defined here (Fig. 3). This means that the AF application can create and retrieve datasets from USER without the developer having to provide two-level names in all programs that access the users’ subdirectories.

A macro variable is defined with a %LET statement and assigned the user’s initials (Fig. 3). The macro variable &WHO will be resolved later to find out the identity of the user as a way of verifying access authority, and also used to generate Libname or Filename statements.

I used the CALL FILENAME() function in the SCL code to define files that will be read using the INFILE statement in a submit block. The value of &WHO is resolved to indicate the path to the data without hard-coding filenames either in the SCL program or the Autoexec.sas file. For example, if the user whose initials are JJH starts the database application and wants to read new data files into a SAS dataset, the SCL program will start like this:

```
INIT: /* 'who' is macrovar */
TEST=SYMGET('WHO'); /* of initials */
IF TEST='TCF' OR TEST='TJW' OR TEST='ABL'
THEN DO;
  CONTROL ENTER;
END;
ELSE CALL DISPLAY('RETURN.PROGRAM');
RETURN;
```

The call to Return.program runs a short SCL program that issues the cancel command and sets the _STATUS variable to ‘halt’. That returns the unauthorized user immediately to the prior menu or program entry.

**ACCESS CONTROL**

Access to sensitive areas can be controlled both within and outside of an AF application. Ignorance of how to write and submit SAS statements using the Display Manager can keep users from altering or deleting important datasets or the AF application itself. Relying on user ignorance is not the safest course, so other software, like PROTEC, that can lock access to specified directories may be a wise choice. However, access must be granted to the directory holding the AF catalog, and there is no facility within AF that I know of to prevent unauthorized modification by someone familiar with PROC BUILD.

Access can be controlled within an AF application by checking the value of the macro variable containing the user’s initials against a list of users authorized to proceed with tasks like updating. This example contains a trivial logic loop, but it has the merit of clarity:

```
INIT: /* 'who' is macrovar */
TEST=SYMGET('WHO'); /* of initials */
IF TEST='TCF' OR TEST='TJW' OR TEST='ABL'
THEN DO;
  CONTROL ENTER;
END;
ELSE CALL DISPLAY('RETURN.PROGRAM');
RETURN;
```

The call to Return.program runs a short SCL program that issues the cancel command and sets the _STATUS variable to ‘halt’. That returns the unauthorized user immediately to the prior menu or program entry.

**UPDATING DUTIES**

The duty of updating datasets should rotate among several people to insure that some good backups exist if one person consistently does it incorrectly, either accidently or deliberately (Allan 1985). A rotation also insures that several people are trained on these procedures. The SAS system can determine who is next in rotation if a dataset of update activity is kept. Such a dataset is also a good way to keep vital quality assurance records of who did what, when.

We maintain a dataset, called QARECORD, that identifies the initials of the person doing the updating, the initials of the requestor (the source of the new data), which permanent dataset was updated, the date/time from the SAS datetime function, and several other variables of specific interest to us. The QARECORD dataset is itself updated by appending a new observation each time the update program is run. This will be illustrated in a later section, but the important point here is that QARECORD can be queried to determine who should be next in rotation.

A user who has validated and corrected his new data with custom error-checking programs and PROC FSEDIT is ready to have it appended (updated) to one of the permanent datasets. Next he selects the menu option for generating an update request. The purpose of this option is to print a request to the next updater in rotation to do
an update and to supply on that request the information the updater needs (Fig. 4). The user enters on the screen the data type and project code that are needed to fully specify the destination subdirectory and dataset name. The extended table on the lower half of the screen helps the user select valid project codes.

The SAS program below runs in the TERM section of the SCL program for this screen (Fig. 4). It creates a temporary dataset and sets QARECORD with the POINT= and NOBS= options. The variable Number equals the total number of observations (NOBS) minus 2, yielding an output dataset with one observation containing the initials of the next person in rotation (when there are three people on the list).

SUBMIT IMMEDIATE;
DATA WORK.UP (KEEP=updater);
/* Updater contains the initials */
  number=total-2;
  SET XXX.QARECORD POINT=number NOBS=total;
  OUTPUT; QUIT;

A FILE PRINT operation uses the initials to address a memo requesting an update and provides the information from the user on what to do (Fig. 5). Again, the macro variable &WHO is used to supply the user's initials on the memo. A PUT_PAGE statement causes a form feed on the printer, then more PUT statements create a requestor's copy of the memo for his own record of what was requested.

Figure 4. Program screen for generating an update request.

Figure 5. Sample update request. Uppercase words come from screen variables.

Update Request-Zoology Database System
Updater's Copy

To: TJW

12MAR90;14:07:46

Please perform an update within 2 days to the dataset identified by the following information.

Data type: BEN
Benthic type: QUAL
Project code: BIOA

Thank you,
JJH
Requestor

Figure 6. Program screen for performing updates.

PERFORMING AN UPDATE

The updater fills in the entry fields of the update screen (Fig. 6) with information taken off the printed request he was given. The information specifies the source and destination datasets for the APPEND statement of PROC DATASETS. The permanent, or base, datasets are designated through a naming convention by the type of sampling done (first four letters) and the environmental project name (last four letters). Adopt a naming convention that uses consistent lengths for parts of dataset names.

The source data is designated by a two-level name corresponding to the requestor's initials and the sampling type. Remember that the requestor's initials are linked to a hard disk subdirectory through a libname statement:

LIBNAME &request "d:\user\&request";
so now the data can be appended with SAS code like this:

PROC DATASETS LIB=&dtype;
   APPEND BASE=&type&proj DATA=&request...&type;
QUIT;

(The double periods are necessary between two macro names to obtain a single period after resolution.)

The &dtype is the data type from screen variable 2 on Figure 5, and it designates the subdirectory where the base dataset resides. The &type is from screen variable 3 (benthic type). If the data type is not "benthos" in screen variable 2, then logical IF statements in the SCL program assign a value to &type based on the value of &dtype. The reason for that is because I had to allow for several types of benthic data whereas the other kinds of data were of one type. As long as base datasets follow a defined naming convention, any number can be handled, and more added to the database later, without hard-coding difficulties.

Ageing of datasets is a smart idea because it produces backup copies of intact base datasets before new data is appended. This macro shows how to handle the two types (&type) of benthic data, qualitative (QUAL) and quantitative (QUAN), and provide new names for the aged datasets:

%MACRO age;
   %IF &type=QUAL %THEN %DO;
      AGE qual&proj qu1&proj qu12&proj;
   %END
   %ELSE %IF &type=QUAN %THEN %DO;
      AGE quan&proj qun1&proj qun2&proj;
   %END;
%END age;

Macro Age can be invoked to produce two older generations of the dataset on the backup medium (floppy disk, Bernoulli cartridge, tape, etc.) before the newly updated version from hard disk is copied to the backup medium. For example:

PROC DATASETS LIB=a; /* libref "a" means floppy drive A */
   %AGE
      COPY IN=&dtype OUT=a;
   %SELECT &type&proj;
QUIT;

With screen prompts to the updater, this sequence can be repeated for making multiple backups on separate diskettes, to be stored in different areas for the greatest security.

The last step in the updating process is to add an observation to the QARECORD dataset showing who did the update, when, and to which datasets. A work dataset is created and variables are assigned the values of the screen variables and the result of the datetime function. This work dataset is appended to the permanent QARECORD dataset, and then it is printed by PROC PRINT for a paper copy. The paper is put in the quality assurance logbook for the database. QARECORD is now ready to be queried to obtain the next updater in rotation when the next update request is generated.

**SUMMARY**

1. A multi-user database is best handled by having user-specific subdirectories where each user's data can be segregated until it is ready for updating to a permanent dataset.

2. Autoexec.sas files should be customized to set up the necessary librefs and to assign the user's initials to a macro variable. The macro variable can be tested to see if the user is authorized to perform updates and resolved to provide the libref of the data source.

3. All reading of raw data, validating, and editing should be done in the users' subdirectories. Valid datasets are then appended onto permanent base datasets and backed up.

4. A dataset of who updated what, when, forms a quality assurance record and can be queried to provide the initials of the next person in rotation to do an update. Printed update requests from the user tell the next updater what to do, and provide a record of that communication.

5. The SCL variables provided on the update screen designate the subdirectories for source and destination datasets during appending. They are the basis for specifying dataset names, ageing datasets, and creating backup copies.

**REFERENCE CITED**


Author: Todd C. Folsom
Duke Power Co.
Production Support Dept.
13339 Hagers Ferry Rd.
Huntersville, NC 28078.
(704) 875-5426.

SAS and SAS/AF are registered trademarks of SAS Institute, Inc., Cary, NC.

PROTEC is a registered trademark of Sophco, Inc., Boulder, CO.