

Modular Programming for a Screen Control Language Application

Linda M. Quinn, AcroMed
Allan M. Kominek, Centerior Energy
Tina M. Sluga, AcroMed

ABSTRACT

This paper discusses the development of a system for loading, checking, and correcting the data from a clinical study. The data was collected using a set of five Case Report Forms (CRF). The objective of the project was to develop a system that would allow display and editing of data on screens that resembled the pages of the CRF. Error processing was similar, if not identical, for many of the CRFs. Common modules could be employed for validation routines among screens and programs. Although the system consists of over 20 screens, the actual time of system development was greatly shortened through the use of common modules.

AUDIENCE

This paper is intended for SAS® users who are developing systems that can benefit from the use of program modules, either due to repetitiveness of tasks or similarity of tasks. Specifically, this paper addresses the use of SAS/SCL® in SAS/AF® and SAS/FSP® to define program modules.

PROJECT DEFINITION

The project can be defined as follows: Take the raw data from each of the five CRFs, load it into SAS data sets, check the raw data for errors and inconsistencies, and allow the data to be viewed, corrected, and validated on-line using SAS/AF screens.

The five CRFs are defined as: Form A, a preoperative evaluation; Form B, an intraoperative evaluation; Form C, the postoperative evaluation; Form D, report of any further surgery or complications; and Form E, report of any further injuries or diagnostic studies. Because of the variety of information being recorded by each form, they differ in length. Form A is 6 pages long. Form C is 4 pages long. Form E consists of 3 pages of information. Forms B and D only have 2 pages.

THE DATA

The information on these forms is completed by filling in the appropriate bubbles. The information is then scanned into a PC and transferred to a IBM RISC System/6000®. The data, at this point, is comma delimited and consists of a series of numerics, asterisks, or blanks. Asterisks denote where more than one answer was erroneously supplied to a question that requires only a single answer. Each page of each CRF has its own unique format. One field in each record identifies the CRF and page.

THE ANALYSIS

The first major hurdle was getting the data into SAS. Because each page has its own unique record layout, each page had to be identified first and then the data for that page was read in. The first decision to be made was how to store the data. There were 17 unique pages. The number of variables per page ranges from 7 to 107. If all pages were loaded into a single data set, the number of variables would be in excess of 1,000. Due to CRF format, there was a potential for a large number of blank fields. It was decided to create a separate data set for each page.

The next decision dealt with data validation. The decision was made to pass any errors on to the production data sets, but to generate a report to flag errors so that they could be corrected on-line. The report would be generated from an error data set. The other issue with data validation was to ensure that duplicates were not passed to the production data sets. The potential existed for CRFs to be scanned in multiple times, either due to errors in marking the form, lithograph errors, or scan capture errors. Duplicates would be captured in another data set. Once the data was loaded, an error report could be generated and the data could be corrected on-line.

In addition to displaying the data, there were several functions which the user might wish to perform: adding new observations, deleting

observations, finding particular patients or physicians, updating existing observations, browsing the entire data set, and scrolling up or down. Because these functions applied universally to all pages, a modular approach was taken. The one unique portion of each screen would be the error checking module. Error checking is necessary, because the user was to have the capability of adding or updating the data.

Error checking is conducted in several manners. An error report is generated in the initial load, error checking is used in the code behind each screen, and error reports are produced for the entire database in its current state.

In addition to the actual clinical CRF data, two additional data sets were absolutely essential to the data management and analysis. Both patient and physician were identified on the CRFs with unique identification numbers. A patient and a physician data set were necessary to match names and other crucial data with these identification numbers. This data would be entered through screens rather than scanned and would also be available on-line with the user capable of performing the same functions as on the clinical data.

Finally, it was decided to make the system entirely menu driven in order to eliminate the need for the user to enter any SAS commands, as the user base would include a number of physicians and support staff who have minimal or no SAS background. The wide user base also posed some security issues. Everyone should be allowed view access to the data, but only a select few should be allowed to modify it, e.g., through the add, update, and delete features. A password module was also added. The password module would be shared by all 17 CRF screens. It was decided to password protect all access to patient and physician data as the major reason for entering these screens would be to enter data en masse, and being asked to enter a password every time another patient was added to the table could become rather tiresome.

LOADING THE DATA

As was mentioned earlier, each page has its own unique layout and each record in the file represents a page. The file was read using the following statement:

```
INFILE 'SCAN.DATA' LRECL=467;
```

The maximum record length for the file was 467 bytes. The field identifying the page was read in with a trailing @ sign. A series of IF-THEN-ELSE statements execute the appropriate INPUT and OUTPUT statements to read in the data, perform the necessary error checking, and output it to the SAS data set for that page and to the error file. The error file identifies the form that was in error, and the error that occurred. Each of the data sets for the individual pages has an associated KEEP statement that limits the variables in that particular data set.

The actual scanning of the data was to take place in stages. In other words, the data would be loaded in a piecemeal fashion. Some provision had to be made for the fact that data that had just been loaded would have to be concatenated to existing data. It was thought to be prudent to separate the actual loading and concatenation into two separate programs. The data could be loaded and the error report reviewed before the data was added to existing data. This would prevent a scanner malfunction from corrupting the existing data.

The actual "load" program was well over 6,600 lines of code, including the error checking. This was far too large to include in a SAS/SCL program within a SAS/AF screen. Remember, it was decided to make the entire system menu driven. There is a memory limit to the amount of code that can be included in an SCL program. The solution was to save the program in an external file and then include it directly to the Preview buffer. This was accomplished with the following program:

```
INIT:
  RETURN;
MAIN:
  RC=FILENAME('LOAD','LOAD.PGM');
  RC=PREVIEW('CLEAR');
  RC=PREVIEW('INCLUDE','LOAD');
  _MSG_='Program submitted';
  SUBMIT CONTINUE;
  ENDSUBMIT;
  CALL EXECCMDI('CLEAR LOG');
  RETURN;
TERM:
  RETURN;
```

The SUBMIT block runs the load program which has been included into the Preview buffer. The log is cleared, because 6,600 lines of code generates quite a few messages and if the user

remains on the system and performs several loads and concatenates back to back, the potential for running out of space exists.

BUILDING MODULES

The analysis phase of the project revealed that a number of functions apply across all screens. These functions included adding new records (ADD), updating existing records (UPD), deleting records (DEL), finding records by patient id, by patient last name, or by physician id (FND), browsing the database (BRS), scrolling to the previous record (PRV), scrolling to the next record (NXT), going to the next CRF page (NPG), returning to the previous CRF page (EXT), and returning to the first page of the CRF (QEXT). These modules are invoked by the three letter commands, which are highlighted. The user invokes them either through a pulldown menu or through the use of PF keys which have been programmed with the commands, circumventing the use of the command line. The WORD function is used to extract the command that has been invoked.

Both the ADD and the FIND functions would require a blank screen. A module to provide a blank screen, the EMPTY module, was written.

THE EMPTY MODULE

The EMPTY module creates a temporary data set with all the necessary variables, initializes the data set variables to missing, and then assigns the values of the corresponding SCL variables to missing.

EMPTY:

```
SUBMIT CONTINUE;
DATA TEMP;
SET CAGE.A1(OBS=0)
    CAGE.SITE_PHY(OBS=0)
    CAGE.PATIENTS(OBS=0);
RUN;
ENDSUBMIT;
TDSID=OPEN('TEMP','U');
TEMPRC=APPEND(TDSID,'NOSET');
CALL SET(TDSID);
FRC=FETCH(TDSID);
TDSID=CLOSE(TDSID);
RETURN;
```

The SUBMIT block creates a data set with all the variables from the CRF, in this case, CRF A, Page 1, the physician data set, and the patient data set. Although this module is copied for each screen, it must be modified to use the correct

data set for the CRF page, i.e., A2, B1, etc. An observation is then appended to the temporary data set with all the values set to missing, the NOSET option for the APPEND function. The CALL SET and the FETCH initializes the values of the corresponding SCL variables to missing.

THE PASSWORD MODULE

Only a limited number of users were permitted to update the system. The ADD, UPDATE, and DELETE functions are password protected. A password screen was created which displays upon entering the system. The user is prompted to enter the password or press enter. This screen created a macro variable called PASSWORD. If the user pressed enter, PASSWORD was set equal to CONTINUE; the user could then go to any screen in the system. However, if the user selected either the ADD, UPDATE, or DELETE functions, the password module would check the value of the macro variable. If the value was CONTINUE, a message was displayed and processing discontinued. This code was copied into the ADD, UPDATE, and DELETE modules.

```
IF PASSWORD='CONTINUE' THEN DO;
    _MSG_='NO PRIVILEGE.';
    WORD='';
    RETURN;
END;
```

THE ADD MODULE

Adding a new record requires the program to perform two functions: provide a clear screen for new data entry, and data verification after new data was entered.

The first function is performed by the following code:

```
IF WORD='ADD' THEN DO;
    UNPROTECT _ALL_;
    CURSOR PAT_ID;
    IF DSID NE 0 THEN DSID=CLOSE(DSID);
    WORD='ADDMOD';
    LINK EMPTY;
    PROTECT SITEPHYS SITE LASTNAME
        PAT_LAST PAT_FRST STATUS;
    RETURN;
END;
```

Data that will be supplied from other data sets is protected. For example, the patient id is entered, and the first and last name is supplied from the patient data set. All SCL variables are set to missing by the link to EMPTY. The cursor is

positioned on the patient id. You will also note that WORD is set to ADDMOD. This tells the program to execute the second part of the ADD function when the user presses Enter.

The second portion of the ADD function checks the patient id by creating a temporary data set with a where clause to subset the data. If an invalid id was entered, i.e., there are no observations in the temporary data set, a warning is issued and no more code is executed. If valid data is entered for patient id, the patient's first and last name, the patient's status, and the sitephys are updated from the patient data set. Similar checks for sitephys against the site/physician data set are done. Data is checked for errors by linking to the check subroutine which is tailored specifically to each screen. If the data passes verification, the new observation is appended to the permanent data set. A submit block is executed to update the statistics for the data set. This is done so the pointer can be positioned on this new observation. The observation is fetched, the screen variables are updated, WORD is set to missing, the cursor is positioned on the menu, the data set is closed, and all fields are protected.

THE UPDATE MODULE

The Update Module also consists of a two step process. First, a particular observation is retrieved and the appropriate fields are unprotected so that the user can modify them.

The first part of the Update Module checks to see if the screen data set is open. If it is open, the current observation is used, otherwise the first observation in the data set is used. The specific observation is fetched and the appropriate variables are protected. Word is set to UPD2; the appropriate code can be executed after the user updates the screen.

Data entered is verified through the CHECK subroutine. If the data is acceptable, the observation is updated, the data set closed, WORD is set to missing, all screen variables are protected, and the cursor is positioned on the menu.

THE DELETE MODULE

The Delete Module is the first one step module. The Delete Module assumes that the observation to be deleted has been located using a Find, Previous, or Next command, i.e., the data set is open. If the data set is open, the current

observation number is saved. Note that the data set is opened in utility mode, 'V'. After the current observation is fetched, the user is asked to confirm the delete via the areasure program. The user's answer, Y or N is stored in a macro variable. If the answer is Y, the observation is deleted, otherwise the process is ended.

To actually remove the observation from the data set, a submit block creates a temporary data set and the observation with a missing patient id is deleted. The temporary data set is then set over the permanent data set. The total number of observations in the data set is checked against the observation number of the record deleted. If the last observation in the data set was deleted, the last observation becomes the current observation, otherwise, the new observation number is retrieved. WORD is set to missing and the cursor is returned to the menu.

THE FIND MODULE

The Find Module is another two part function. The first part creates a clear screen and the second part retrieves the appropriate observation based on the data that was entered by the user.

```
IF WORD='FND' THEN DO;
  UNPROTECT PAT_ID PAT_LAST;
  LINK EMPTY;
  CURSOR PAT_ID;
  WORD='FND2';
  RETURN;
END;
```

The user is permitted to locate an observation based on the patient id or last name. The screen is cleared via the EMPTY subroutine and the cursor positioned on PAT_ID and word is set equal to FND2;

After the user has entered either the patient id or last name, the second part of the FIND function executes. The second part of the FIND function first checks to see whether the search is being performed by patient id or by patient last name. The difference being that patient id is unique, whereas, there may be multiple occurrences of patient last name. The search by patient id is relatively simple. If the patient id is valid, the observations is located and fetched. The search by last name is somewhat more complicated.

If the name is contained in the patient data set, a temporary data set is created with only those observations where the last name is equal to the

search name. If the last name is unique, the patient information is retrieved. This information includes the patient id. A temporary data set containing the CRF data is created using a WHERE clause. Another temporary data set contains the site/physician data. The screen is refreshed with this data. Finally, the actual record in the CRF data set is located and the current observation is updated. If the last name is not unique, the pull down menu is turned off temporarily, and an fsview displays a list of all patients with that particular last name. The user can then identify the exact patient and obtain the patient id. The user can then perform another FIND using the patient id. As always, pat_id and pat_last, are changed to protected, WORD is set to missing, and the cursor returns to the menu.

THE BROWSE MODULE

The browse module provides a tabular list of all the data for a particular CRF page.

```
IF WORD='BRS' THEN DO;
  IF DSID>0 THEN DSID=CLOSE(DSID);
  CALL EXECCMDI('PMENU OFF');
  CALL FSVIEW('CAGE.A1','BROWSE');
  CALL EXECCMDI('PMENU OFF');
  WORD='';
  RETURN;
END;
```

THE NEXT MODULE

This module allows the user to scroll forward through the data. There were two potential problems. The user has entered the next command and the data set has not been opened, e.g., the user has just entered the screen and has not executed a FIND. In this instance, the first observation is displayed. The other problem is that the user is positioned on the last record of the data set and attempts to go to the next observation. In this case, a message is issued.

If the data set is not already open the module opens the data set, determines the total number of observations in the data set, NOBS, and compares this to the current observation number, CUROBS. If they are equal, the End of File message is displayed. If the current observation number is 0 or the screen is blank, the first observation is displayed. Otherwise, one is added to the current observation number and that observation is displayed. If the data set is already open, the module checks for the End of File condition and either issues a message or displays the next observation. If either the first or the

next observation is displayed, another subroutine is invoked, the LOCATES subroutine. This subroutine, retrieves the patient's name, etc. from the patient data set, and the site/physician information from the site/physician data set.

THE LOCATES MODULE

This module uses temporary data sets to first determine if the patient id and the site/physician id are valid. If they are valid the appropriate data is retrieved to the screen.

THE PREVIOUS MODULE

This module is similar to the NEXT module. The only difference being that this module scrolls back through the data. The potential problems are being at the first record in the data set and attempting to display the next record and attempting to display the next record without first locating at least one record in the file.

In the first instance a Top of File message is displayed. In the latter, the last observation in the data set is displayed.

THE NEXT PAGE MODULE

Once the desired CRF data has been located on the first page, the user can then go to subsequent pages of the CRF. Because each CRF is identified by a unique form set id, this value is then passed to the next screen and used in a LOCATE function to select the appropriate information. Upon returning from the next page, word is set to missing and the cursor is positioned on the menu.

```
IF WORD='NPG' THEN DO;
  CALL SYMPUT('FORM_SET',FORM_SET);
  CALL DISPLAY('SURVEYA1.PROGRAM');
  WORD='';
  HOME;
  RETURN;
END;
```

THE EXIT MODULE

The Exit Module returns the user to the previous CRF page or to the previous menu, if the user is on the first page of the CRF.

```
IF WORD='EXT' THEN DO;
  CALL EXECCMDI('PMENU OFF');
  IF DSID>0 THEN DSID=CLOSE(DSID);
  _STATUS_='H';
END;
```

Before the CRF page is exited, the data set is closed. By setting the _STATUS_ to 'H', control

is returned to the calling routine. For subsequent pages of the CRF the calling routine is the Call Display contained in the Next Page module. For the first page of the CRF, the calling routine is the previous menu.

THE QUICK EXIT MODULE

This module was developed to allow the user to quickly return to the first page of the CRF in order to begin research on another patient or on another set of data for the same patient.

```
IF WORD = 'QEXT' THEN DO;  
  WORD = 'EXT';  
  CALL SYMPUT('ANS', 'QEXT');  
END;
```

If the user selects to do a Quick Exit, WORD is set to EXT and a macro variable is set to 'QEXT'. There is no RETURN in this module, so that if this code is placed before the Exit Module, control will pass to the calling routine or the Next Page Module. The Next Page Module was modified for every page except the first page. Each page now Exits to the previous page until the first page is reached.

```
IF WORD = 'NPG' THEN DO;  
  CALL SYMPUT('FORM_SET', 'FORM_SET');  
  CALL DISPLAY('SURVEYA3.PROGRAM');  
  ANS = SYMGET('ANS');  
  IF ANS = 'QEXT' THEN WORD = 'QEXT';  
  ELSE DO;  
    WORD = '';  
    HOME;  
    RETURN;  
  END;  
END;
```

Again, there is no RETURN when there is a Quick Exit, so that an Exit will be performed.

BUILDING THE SYSTEM

Now that a library of modules has been created, the system can be built. The bulk of the remaining work was constructing the screen formats and building the individual CHECK subroutines for each screen. The customization needed for each module was minimal. The correct data set names had to be used, but this could be accomplished with a CHANGE command. The Quick Exit code had to be removed from the Next Page module on the first page of the CRF, otherwise the user would Exit all the way back to the menu.

CONCLUSION

The development time for this system was shortened through the use of common modules. Any maintenance that has been necessary has been much easier, because all the screens have common syntax.

AUTHOR

Linda M. Quinn
AcroMed
3303 Carnegie Avenue
Cleveland, OH 44115
Tel: 216-431-9900
Fax: 216-431-5076

SAS, SAS/AF, SAS/FSP, and SAS/SCL are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. IBM and RISC System/6000 are registered trademarks or trademarks of International Business Machines Corporation, Armonk, NY. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.