DEVELOPING USER APPLICATIONS WITH SAS/AF SOFTWARE FOR PCS: THE SAFARI EXTRACT GENERATOR EXPERIENCE

Michelle Fine, Aetna Life & Casualty

Abstract

In today's changing DP environment, greater emphasis is being placed on providing users with tools allowing them to perform functions traditionally handled in a systems department. As a result, there is a need for software products that facilitate the development of these end-user applications. The screen design capabilities of the SAS/AF software for PCs along with its error handling facilities and dataset manipulation functions make it a strong candidate for systems of this nature.

This paper addresses a real-life business problem that was solved with SAS/AF software for PCs. It begins by presenting a background description of the business environment and the existing problem. It then moves on to outline the stages of system development, highlighting features of SAS/AF that were helpful. The final portion of the paper provides "tips and techniques" for using SAS/AF and offers recommendations for future development efforts.

Background

The Personal Financial Security Division of Aetna Life and Casualty maintains a large database as part of its SAFARI (System by Aetna for Fast Access to Records and Information) system. Developed in the late 1960's, this file contains all Personal Automobile and Homeowners insurance policy master records - a number currently in excess of 4 million records. SAFARI policy information is used by different customer areas to follow business trends, report on claim activity, and make underwriting decisions. Since this file is the only central source for data of its type, extracts are commonly required.

The SAFARI masterfile is a BDAM file that, for purposes of extracting data, can only be accessed via a series of Assembler programs. A "homegrown" front-end to these programs also exists to aid in the extract process. The Assembler front-end is composed of two parts: An ASI-ST dictionary of field names that correspond to several SAFARI masterfile fields; and "homegrown" code used to select and sort records, format printed reports, and update the ASI-ST dictionary. Files of "homegrown" code containing the desired data selection criteria are linked up with the Assembler software as well as the SAFARI masterfile to yield the requested subgroups of data. Preparation of this code, however, is a tedious and time consuming process. The logic involved can be complex and there are numerous coding details to consider. In addition, syntax validation is only available for certain portions of the code. As a result, coding errors are easy to make.

When extract jobs abend as the result of coding errors, they need to be fixed and rerun. SAFARI processing time is expensive; hence, reruns are costly. To help alleviate this cost and to reduce the time needed to prepare code files, an Application Generator was developed to produce SAFARI extract and dictionary update files. The Application Generator displays a series of menu screens and input panels that prompt the user for necessary information (e.g. extract criteria, field names). It then uses these parameters to generate the code needed to run the extract and/or dictionary update.

The Application Generator was created on an IBM PC AT using SAS/AF version 6.03. This product was chosen due to its reputation for allowing ease and flexibility in screen design as well as for providing strong help facility functions. A personal computer environment was selected instead of a mainframe environment due to better response times and the superior screen design capabilities offered by the PC version of the product. The system was designed to enable both technical and non-technical users to generate extract files with the anticipation that the responsibility for preparing these files would eventually be transferred from the systems department to a customer area.

Development Experiences

I. Learning the Language

Once SAS/AF had been installed, the first task in the development process was to learn SCL (Screen Control Language) - the language that controls the operation of all SAS/AF

PROGRAM screens. Since no formal training courses were available, the User's Manual became the primary educational resource. For an experienced SAS programmer, working with SCL required some adjustment since both its structure and composition are notably different from those of the base SAS product. In contrast to base SAS programs which are relatively unstructured and are processed on an interpretive basis, SCL programs have a definite structure and need to be compiled prior to execution. While SCL does use many of the base SAS statements, several new statements and dataset manipulation functions unique to SCL have also been included.

II. Developing the Prototype

After an initial period of experimentation with SCL, development of the prototype for the Application Generator began. SAFARI extract logic can range from a simple value check for a single field to more complex processing of multiple occurrence fields. The goal of the prototype was to produce a small-scale workable system that would have the same basic format as the projected final result, but would only address the simpler cases. The system could later be expanded to include more involved situations. The prototype would also include routines to handle dictionary update code generation and file uploads to the mainframe since these functions are straightforward in nature as well.

SAS/AF proved useful in this effort. The product provides eight types of screens, two of which were the most beneficial for the Generator: MENU screens and PROGRAM screens. MENU screens allow the user to choose any one of a variety of options by entering a corresponding number on the command line. These screens have behind-the-scenes "attribute" files associated with them which establish relationships between an option number and the path of programs to be followed upon selection of that option. PROGRAM screens display informational messages and allow the user to input data values to the system. Each PROGRAM screen has an SCL program associated with it that controls validation and processing of the input parameters. This type of structure was very accommodating to the original vision for the system.

As anticipated, the facilities for creating screens were easy to use and allowed a fair amount of flexibility in terms of the format, color, and size of the screen. SAS/AF provides a free-form, full-screen editor for use in designing panels. Cursor keys are used to maneuver and position various portions of the text. The COLOR command is available to diversify the colors used for different aspects of the screen. The Background, Text, Command Line, and Message areas, among others, can be modified from the defaults to a combination of up to twelve different colors. In addition, the WSHRINK and WGROW commands allow both the width and length of screens to be altered. Once into WSHRINK or WGROW "mode", cursor keys are used to reduce and increase the size of the panel. This is helpful in situations where multiple panels need to be simultaneously displayed.

Another nice feature of SCL is that it contains a series of error control statements and variables which enabled a PROGRAM screen to mimic a MENU screen. Coding SAFARI extracts requires that five main functions be performed. Since some of these functions are composed of several sub-functions, keeping track of steps completed can be confusing. The method chosen to ease this process was to initially highlight each of the five menu options using reverse video. Each time a task is completed, the highlighting is removed from that option. SAS/AF MENU screens, however, do not allow menu options to be highlighted. Consequently, a PROGRAM screen was designed to look like the desired menu and the background SCL program incorporated error statements to control the highlighting.

Exhibit I shows both the PROGRAM screen designed to handle menu functions and the SCL code that controls processing. In the INIT section of the program, five screen variables are initialized to have values that correspond to the five functions for coding SAFARI extracts. Since, by default, SCL highlights (with reverse video) all screen variables that are flagged with the ERRORON statement as having errors,

this statement is used to signal SAS that each of these five variables is "in error". The effect of this is that when the screen is initially displayed, all five menu options are highlighted. Later, in the MAIN section, each time an option is selected, the appropriate PROGRAM or MENU screen is called and the ERROROFF statement is used to indicate that the screen variable for that option is no longer in error. Thus, the highlighting is removed.

The ERRORON and ERROROFF statements are also helpful for trapping data values that are input incorrectly. Screen variables can be edited for appropriate values and, when an incorrect entry is found, the ERRORON statement is used to signal that the field is in error. The _MSG_ variable is provided to communicate messages indicating the nature of the error. These messages are displayed in the message area of the screen. By default, SCL programs cannot be exited if any screen fields are in error. Therefore, the user is required to reenter correct values for the specified fields before processing can continue. Once this is accomplished, the ERROROFF statement signals that the field is no longer in error and system processing resumes.

Two additional features of SCL that proved useful are its datset manipulation functions and its incorporation of macro variables. SCL provides a series of functions that are used to - among other things - open, close, sort, retrieve values from, and update values to SAS datasets. These functions make simultaneously accessing multiple datasets a much easier task than in the base SAS product. This can be helpful for processing data located in several different sources and for transferring values between screens. The Application Generator used this technique to store values entered on all input panel screens so that, in the event a screen was redisplayed prior to completion of coding an extract, it would reflect values previously entered. SCL also provides two macro variable functions - SYMPUT and SYMGET - that are used to assign and retrieve values to and from user-defined macro variables. The Application Generator took advantage of these functions, in addition to those previously mentioned, for transferring values between screens.

III. Creating an Expanded Version

The SAS/AF software for PCs accommodated the Application Generator prototype with few difficulties. Screens created were user-friendly, response times were fast, and the SCL programs compiled in relatively short periods of time. When work began on the expanded version, however, some of the drawbacks of SAS/AF began to surface. Discussion of these drawbacks is in no way intended to discourage development of SAS/AF systems. However, these are important facts to consider when planning potential applications.

The Application Generator prototype was created on an IBM PC AT that had 640K RAM. For purposes of the prototype, this amount of memory was sufficient. As the system grew larger, however, several memory problems were experienced. In order for the system to continue to run, it was necessary to purchase memory boards with additional expanded memory. 1M of memory was needed for the expanded version to run and an additional .5M was also purchased to allow room for future enhancements. This delayed system development for six to eight weeks - the time needed to receive and install the memory boards.

Problems were also encountered with several aspects of SCL. Programs written in SCL have code size limitations of 32K. Since the programs in the prototype were only designed to handle the simpler extracts, this limit was never reached. When the programs were expanded to handle more complex extracts, however, several of them exceeded the size boundaries. To solve this, the programs were broken down into multiple, smaller sections and linked together thru use of the CALL functions.

In addition, compiling and debugging SCL programs presented some difficulties. The smaller programs of the prototype compiled within timeframes of five to eight minutes. Some of the larger programs of the expanded version, however, required as long as twenty minutes to compile. This

factor significantly impeded system development progress. Debugging SCL programs can also be difficult. When a SAS/AF system abends, the message log does not indicate which program caused the error. Instead, it only specifies the error message and the values of all variables in the errant program at the time of the abend. As a result, if program variable names are not unique or easily recognizable, determining the abending program can involve some creative detective work.

Despite these drawbacks, development work on the expanded version of the Application Generator was completed in June 1989 - approximately one year after the project had originally begun. At this writing, the system is in a pilot phase and reaction has been positive. Screens continue to be user-friendly, response times reamin sufficiently fast, and the system has improved productivity by promoting increased efficiency and speed in preparing extract files. The Application Generator is currently scheduled for installation at the first customer site by year-end 1989.

Tips and Techniques

The most important recommendation for others contemplating projects of this type is to keep applications as simple as possible. The less complicated the system, the less likely it will be to require additional memory or to exceed the limitations for program size. Smaller applications also have faster response times and can be developed more quickly since the SCL programs compile more rapidly. In the event that an application does become complex, a utility package such as Norton Utilities' "Speed Disk" will help to decrease system processing time. Speed Disk will organize the SCL program catalogs and SAS datasets in such a way that accessing them requires only minimal activity on the part of the read-write head. Therefore, it takes less time to access files and less time for the system to run.

One other suggestion is to license the PC SAS/FSP product along with SAS/AF. This package will greatly aid system development by affording a means by which dataset observations can be viewed and interactively updated. Given the powerful dataset handling capabilities that SAS/AF makes available, this is desirable.

Conclusion

SAS/AF proved to be a helpful tool in the development of the Application Generator. Despite the drawbacks encountered, the system was able to be completed and to function effectively in the business environment. Consensus is that the decision to use SAS/AF for this project was a good one. Had the scope of the project been less complex, development would certainly have been easier. However, it was still possible for the application to meet all of the originally specified business needs.

Exhibit I

```
INIT:

***** INITIALIZE SCREEN VARIABLES TO
***** HAVE VALUES THAT CORRESPOND TO
***** THE FIVE MAIN FUNCTIONS FOR
***** CODING SAFARI EXTRACTS. ;

INITL = 'INITIALIZATION';
SCANSEL = 'SCAN/SELECTION CRITERIA';
EXTRC = 'EXTRACT CARDS';
SORTC = 'SORT CARDS';
DETHDR = 'DETAIL CARDS';
S = ' _ ';

CONTROL ERROR;
ERRORON INITL SCANSEL EXTRC SORTC
        DETHDR;
_MSG_ = '      ';

RETURN;

MAIN:

***** CHECK TO BE SURE OPTION NUMBER
***** ENTERED IS WITHIN THE CORRECT
***** RANGE;

ERROROFF S;
IF (S < '0') OR (S > '5') THEN DO;
_MSG_ = 'SELECTION MUST BE A NUMBER
         BETWEEN 0 - 5';
ERRORON S;
RETURN;
END;
```

```
***** DEPENDING ON WHICH OPTION
***** NUMBER IS SELECTED, CALL
***** THE APPROPRIATE PROGRAM OR
***** MENU SCREEN AND REMOVE ERROR
***** HIGHLIGHTING FROM THE OPTION
***** DISPLAY;

IF S = '1' THEN DO;
           CALL DISPLAY('INITLA');
           ERROROFF INITL;
           _MSG_ = '   ';
           S = '   ';
           RETURN;
           END;


IF S = '2' THEN DO;
           CALL DISPLAY('SCANA');
           ERROROFF SCANSEL;
           _MSG_ = '    '
           S = '   ';
           RETURN;
           END;


IF S = '3' THEN DO;
           CALL DISPLAY('EXTRA');
           ERROROFF EXTRC;
           _MSG_ = '   ';
           S = '   ';
           RETURN;
           END;


IF S = '4' THEN DO;
           CALL DISPLAY('SRTA');
           ERROROFF SORTC;
           _MSG_ = '    ';
           S = '   ';
           RETURN;
           END;


IF S = '5' THEN DO;
           CALL DISPLAY('DETTA.MENU');
           ERROROFF DETHDR;
           _MSG_ = '    ';
           S = '   ';
           RETURN;
           END;


IF S = '0' THEN DO;
           ERROROFF SORTC DETHDR EXTRC
                    SCANSEL INITL;
           CALL PREVIEW('CLEAR');
           RETURN;
           END;

RETURN;



TERM:

RETURN;
```

```
;BUILD: DISPLAY AUTOEXT.PROGRAM (E)MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM;
;Command ===>                                                                  ;
;                                                                              ;
;                                                                              ;
;                           AUTO                                               ;
;                                                                              ;
;                    1)  &INITL________  (FIRST)                               ;
;                                                                              ;
;                    2)  &SCANSEL_______________                               ;
;                                                                              ;
;                    3)  &EXTRC________                                        ;
;                                                                              ;
;                    4)  &SORTC____                                            ;
;                                                                              ;
;                    5)  &DETHDR_____     (LAST)                               ;
;                                                                              ;
;                    0)  RETURN TO PREVIOUS MENU (PRESS F3 WITH THIS SELECTION) ;
;                                                                              ;
;                    ENTER YOUR SELECTION:  &S                                 ;
;                                                                              ;
;                                                                              ;
;       *****  THE HIGHLIGHTED OPTIONS HAVE NOT YET BEEN VIEWED  *****         ;
;                                                                              ;
;                          PRESS <ENTER> TO CONTINUE                           ;
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMRJCMMMMMM<
```