SAS MACRO TECHNIQUES FOR THE DEVELOPMENT OF INTEGRATED ON-LINE SYSTEMS

Clark M. Roberts, San Diego Gas & Electric
James H. Sorenson, San Diego Gas & Electric
John S. Willard, Mascari & Associates
Roger B. Glaser, San Diego Gas & Electric

ABSTRACT

The "Information Age" is placing ever increasing demands on information processing professionals to produce more and more automated systems. Management not only expects systems to be developed faster, but also requires them to be easy to use, flexible enough to adapt rapidly to change and to execute efficiently. Fortunately, there are more development alternatives available today to meet these challenges than in the past. Personal computers and 4th generation mainframe languages can be used directly by those requesting the system when requirements are not complicated. Very large and complex systems, or systems where the volume of transactions is large, still require the use of traditional data processing approaches.

But what about the vast spectrum between these extremes? The majority of organizations categorize systems in this middle-ground as complex and apply slower traditional methods to develop them, increasing their already large backlog. We offer a compromise. This compromise is the use of traditional techniques by experienced users or programmers within the framework of a 4th generation language, specifically the SAS system. Our approach has been successfully applied in the production of several prototype systems that accomplished the aforementioned objectives: 1) ease of use, 2) efficient execution, and 3) rapid development and maintainability. This paper presents a discussion of these techniques and their benefits in a SAS Macro environment.

INTRODUCTION

The introduction of the SAS Macro facility in 1982 greatly increased the capabilities of the systems one may build using SAS Institute products. Moreover, the previous years of SAS Users Group International (SUGI) Conference Proceedings and presentations have contributed greatly toward a wealth of useful SAS techniques and applications that can be shared by us all. This paper brings together a number of published techniques with our experiences in developing some medium scale SAS-based systems that were made feasible with the SAS Macro facility. Although the contents are directed toward those individuals confronted with the responsibility of developing easy to use on-line systems, users of these systems, as well as management may also find the material informative.

There are generally three types of systems that one might develop with SAS products: open ended, command driven and menu driven systems. Open ended systems are the most flexible, but they require a more extensive programming background for everyday use. With command driven systems, one needs to remember the commands and parameters. Because of this they tend to require more training and documentation than menu-driven systems. Cascading through layers of menus can be tedious for someone who is quite familiar with a system, which makes the command driven approach more suitable for systems frequently used by experienced personnel.

In menu driven systems, one merely "sees and points" (15). Far less information is required for the user to remember. Menu driven systems are ideal for occasionally used systems or systems used by inexperienced personnel (see Martin, 16 and Trimble, 29). Minimal training and documentation is required, as well. The menu driven system approach also provides a good structure for the development process since (1) the data requirements dictate the data base design, (2) the functional requirements of the system dictate the menu structure, and (3) the menu structure, in turn, provides a framework for top-down, modular development (see reference 17 for details).

The focus of this paper is on the menu driven systems approach applied within a SAS Macro framework in an IBM MVS/TSO operating environment. This approach has been adopted by the authors to yield a number of very successful systems. Examples of such systems developed by the authors and others at San Diego Gas & Electric (SDG&E) are: A Productivity Analysis System (see Exhibits 1 and 2), a General Rate Case Data Analysis System (see Exhibit 3), an Information Center Training Tracking System (see Exhibits 4 through 7), an Income Tax Accrual Modeling System, a Nuclear Fuel Accounting System, and a Load Management Analysis System, among others. Examples of similar systems developed elsewhere are provided by Epperson (7).

The overall scope of this paper is to bring together the potpourri of productive concepts and techniques one can use for developing menu-driven systems. These techniques, some of which are new, some of which are borrowed and enhanced could, in some cases, also be applied to command driven systems. A course is currently under development at SDG&E that expands the techniques presented in this paper. The outline of this course conveniently provides the structure of this paper as well. Specifically, the techniques for the development of integrated on-line systems are contained in these three major areas:

- Designing easy-to-use systems
- Improving system efficiency
- Increasing development productivity
In order to meet the space requirements for publication in this proceedings, the presentation of materials related to these major areas has been abbreviated. A complete version of the manuscript, however, is available from the authors.

**DESIGNING EASY-TO-USE SYSTEMS**

The amount of effort required for a user to operate a system can be the determining factor in its successful implementation. It's an old adage that "systems that are difficult to use, won't be!" (Source unknown.) The following techniques have been applied by the authors in the successful implementation of several systems, and will hopefully prove beneficial to others involved in the development of user-oriented systems.

- The effort required to enter a system should be minimal. An inexperienced user should be able to get directly to a systems main menu after logging on.
  1. Modify the installation's standard logon procedure to allocate static application datasets and execute the application CLIST by pointing to it in the EXEC statement PARM field.
  2. Develop a custom application CLIST using the INITSTMT parameter at SAS invocation to load and execute the base level macros that will display the application main menu.

- Selection of available options should be simple.
  1. Develop a well structured hierarchy of option selection menus using %PUT and %INPUT. This approach keeps control within the SAS environment, as opposed to the less efficient method of driving SAS systems with TSO CLISTS (see Exhibits 1, 1A and 4.)
  2. Use SAS/FSP(R) FSEDIT or ISPF dialog to develop screens for multiple option or parameter selection (see Exhibits 2 and 3). Transfer responses to the macro environment using the SYMPUT function in a data step (see references 9, 20 and 28).
  3. Use the "bootstrapping" technique, as discussed in the next section, to bring in the macros required to access the selected options.

- Information appearing, or to be entered, on the screen should not be cluttered or confusing.
  1. Maintain consistency in the use of colors and PP key definition (see Exhibit 2 and Exhibit 7).
  2. Place data entry fields to facilitate minimum cursor movement.

- Understandable documentation should be available to a user through easily accessible menu options.

- A system should provide versatile capabilities for users.
  1. Data manipulation and display capability should be as generalized as possible.
  2. Selection of printing and plotting devices should not be limited (see Exhibits 8 and 8A).

- The availability of the system should not be restricted to one user at a time.
  1. Implement dual databases, one to provide for data entry and maintenance and the other for back-up, query and report generation requirements.
  2. Dynamically allocate and de-allocate the two data bases as required to reduce contention.

- A system should be secure from unauthorized access.
  1. Access to a system or various functions of a system can be limited by user ID and password.
  2. An entire database can be secured by using system level restrictions through ACF or similar software, or by using the dataset security facilities available with the base SAS software.
  3. Access to certain variables in a dataset can be limited by encryption (see Howard, II) or by defining several screens for varying levels of access. In both cases, access is limited by password.

**IMPROVING SYSTEM EFFICIENCY**

Ease of use alone will not guarantee the success of a system. If response time is too slow, there may be some reluctance to use it. Depending on the size and complexity of a system, some or all of the following efficiency improvement techniques may apply:

- There are several SAS options that will affect the execution of macro systems. Options such as LEAVE=, SORT=, MSIZE=, MDELETE=, MSTMSIZE= and MWORK= should all be considered when designing a SAS macro system (see references 24 and 26).

- Since SAS macros must be compiled before they can be executed, it may take an inordinate amount of time to enter large systems. This problem can be solved by using a technique we call "bootstrapping."

1. This technique is started by using the INITSTMT parameter in the main CLIST which loads, compiles and executes a
2. The user now sees the main menu and begins to choose options within the system. It is at this point that the base level LOADMAC and EXECMAC macros (see Exhibits 9 and 10) are used. These macros dynamically load and compile all non-basic macros only once and only when requested (18). The result is increased system efficiency since no unnecessary compiling of macros will ever occur.

3. EXECMAC requires the following syntax:

```
%EXECMAC(macro name,LIB=ddname of macro library,FARM=%QUOTE(the macros related parameter list))
```

- Although internal program documentation is important, we have found that excessive comments in a macro will degrade compilation time due to the way the SAS macro compiler handles comments. We, therefore, recommend that macro comments be minimized and alternative documentation techniques be used.

- Care must be taken to minimize the number of work datasets created within a SAS job as excessive subsetting will reduce the amount of region space available and increase execution time. By taking the time to logically form the conditional and manipulative statements into a few more complex data steps, savings can be made in the work area and execution time used.

- Sorting, especially large datasets, is a prime factor affecting execution time of a SAS program. Basic precautions such as sorting only when needed and using only necessary variables should always be in effect. Other specific precautions which can help fine tune the system include:

1. Try to keep the data set sorted by its main keys as much of the time as possible, if it is predominantly used in a certain sort order. Sort the data into a temporary dataset when the sort order needs to be changed and use a sort directory (in a SAS dataset) to maintain the current sort order of each dataset in the SAS database.

2. Data that is not required should be eliminated from the dataset prior to sorting by subsetting the desired observations into a temporary dataset and by using a KEEP or DROP statement to eliminate variables that are not required.

3. Use PROC SORT (an undocumented SAS procedure) to sort small datasets in core.

- For systems with large data bases, the time to process data and produce reports on-line may be unacceptable. In these situations, background processing should be considered.

1. Develop a batch processor that will allow a user to specify report options within the on-line system and hold them for future execution in background.

- Systems with large data bases that require access to specific records can benefit from the use of indexing techniques. A directory (index) file is updated and sorted as observations are added to or deleted from a data file. Since this directory contains only a key and the observation number of the corresponding record in the data file, specific records in the data file can be located much faster using the POINT option of the SET statement. Other indexing techniques to consider include:

1. Parrow (21) describes a technique which incorporates the use of ISPF tables for efficient data storage and retrieval.

2. DS/DBtm a proprietary enhancement to SAS, provides indexed access capability to SAS datasets (see reference 12).

3. Ramsey (23) illustrates access to groups of observations based on values of partial keys.

- Certain principles relating to storage of data should always be followed to maintain optimum execution performance. The SAS Applications Guide (5) discusses such things as storing only required data, use of the LENGTH statement for character variables and storing numbers as character variables when they will not be used in arithmetic operations. Other useful storage techniques include:

1. Store categorical data in coded form and maintain a format library that contains the actual values of the variables. Encoding or table look-up can be accomplished by using the SAS PUT and INPUT functions to access actual values when they are required in a data step. (10).

2. Use the relational approach to data base design to eliminate redundant data (see references 6, 27, 30 and 31). Relational structures also will simplify data maintenance by storing non-key items in one location. A major drawback of relational data structures is increased processing time required to manipulate the data. Since relational data bases "break-up" the data into groups of datasets (relations) which are interrelated by keys, they must be put back together for processing. The programming effort required to manipulate relations can be reduced if the language provides a syntax for relational operations (see reference 3, 6, 22, 25).
In rare cases where the above techniques have been applied and there has been insufficient improvement in performance, it may be necessary to develop specialized SAS procedures in assembler or PL/I to replace frequently executed SAS or TSO CLIST code.

1. A possible alternative for increasing efficiency in the CLIST environment has been provided by Fischer Innis Systems. XLRB™, a recently developed CLIST compiler, claims to improve performance in this area by a factor of up to 10 times.

INCORPORATING PRODUCTION PRODUCTIVITY

Making systems efficient and easy to use introduces design and programming complexities that the average SAS programmer normally have to confront. How can these types of systems be developed quickly, without sacrificing maintainability? Not easily! There are some techniques that can reduce the development and maintenance efforts substantially over time. These techniques include prototyping, top-down design and development, modular development, structured coding, and the use of system building blocks.

- Use prototyping to design and build a working model of key components of the proposed system early in the analysis stage (see references 2, 13 and 19).
  1. This allows users to better define their requirements while actually working with their system.
  2. As new requirements emerge, the model can be easily refined.
  3. The conversion of existing manual systems which use forms as a data acquisition and storage media can be easily prototyped and operational in a few days (17).

- Menu-driven systems lend themselves well to top-down design and development.
  1. Position major or primary functions like data entry, database query and report generation as options on the main menu while including subordinate and secondary functions on lower level menus.
  2. Use dummy routines for options that have not yet been implemented to facilitate testing of program control logic and to notify users that the option is not currently available. The appropriately structured modules can then be inserted and tested as they are developed (19).
  3. Top-down implementation provides access to portions of the system before completion and allows for the analysis of user feedback earlier in the development process.

- The SAS macro facility provides an excellent environment for modular development techniques (19).
  1. Develop subroutine like modules that perform one specific task or function. When linked together, the subroutines will form a system that can be implemented in top-down fashion.
  2. Reduce the dependency between modules so that changes can be more easily isolated.

- Structured coding techniques can help to simplify the maintenance effort by bringing conformity to the development process (1).
  1. Develop standards for variable names, dataset names and coding conventions.
  2. Write short, concise modules using simple constructs as much as possible. Provide documentation for complex logic.
  3. Make programs more readable by indenting code and separating logical segments with comments.

- Maintain a system of central libraries containing general purpose routines or system building blocks. This will reduce duplication of effort during development and minimize maintenance redundancy (19). These libraries should contain:
  1. Skeleton system modules: modules that can be modified for use in different applications, e.g., logon procedures and main CLIST.
  2. Installation dependent routines: modules that perform functions specific to a particular installation and would require modification to be used at another site, e.g., specification of output parameters (see Exhibits 8 and 8A) and output processors.
  3. General system components: macro functions (14) or subroutines that perform specific functions, e.g., dynamic dataset allocation, EXECMAC (Exhibit 10), binary search of a SAS dataset and a routine to determine if a given date is a holiday.
  4. System development aids: a menu driven sub-system that provides developers with access to functions like ISPF and TSO.

CONCLUSIONS

The techniques discussed here can improve the productivity of users by providing them with systems that are easier to use. Developers can benefit through increased productivity and naturally, increased productivity translates to lower costs for management. Based on our successful experiences with these techniques, we feel our productivity has increased...
by as much as 500%. These productivity gains are made possible because of the ideal environment provided by the SAS Institute products. By combining a little creativity and imagination with the SAS System, just about any software development requirement can be fulfilled.

FOR FURTHER INFORMATION

The authors welcome inquiries concerning the subject matter of this paper. Individuals interested in obtaining further information or a copy of the full manuscript with additional references and examples are invited to contact us at the following address:

San Diego Gas & Electric
P. O. Box 1821
San Diego, CA 92112
Telephone: (619) 696-4389

BIBLIOGRAPHY AND REFERENCES


---

SAS(R) and SAS/FSF(R) are registered trademarks of SAS Institute, Inc., Cary, NC, USA.

RAQL™ is a trademark of Michael Gilman, Montreal, Quebec, Canada.

DS/DB™ is a trademark of Infotech Systems, Inc., Orlando, FL, USA.

XLR8™ is a trademark of Fischer Innis Systems Corporation, Naples, FL, USA.

---

**EXHIBIT**

**EXHIBIT 1**

**PRODUCTIVITY ANALYSIS SYSTEM**

**MAIN SELECTION MENU**

**AVAILABLE SELECTIONS:**

1. EDIT OR ENTER PRODUCTIVITY DATA (DEFAULT)

2. BROWSE PRODUCTIVITY DATA

3. GENERATE REPORTS

4. BACK-UP DATABASE

X. EXIT PRODUCTIVITY ANALYSIS SYSTEM

**ENTER SELECTION:**

---

**EXHIBIT 1A**

SAS Macro Code to Display Productivity Analysis System Primary Selection Menu and evaluate the users response.
EXHIBIT 2: Productivity Analysis System Data Entry and Edit Screen

---

Edit SAS data set: SASDB.BOTH

Command ==>

---

Gas Operations Department - Productivity Analysis System

Daily Header Information

Employee Number: 4845 Section: PIPELINE OPERATIONS (Lh, Mh, G1, P, O1, PM, GP)

Date: 12DEC84 (MMDDYY) Organization: T (T=Gas Operations) Shift: 2

Beginning Ending Total
Mileage: 26350 Mileage: 26410 Mileage: 69 (Computed)

Dailv Route Detail

Work Order No.: 18th & Imperial Work District: G0 G/T: N

Arrival Time: 8:30 Departure Time: 8:45 Total Time: 15.00
HH:MM HH:MM Computed

Driving Time: 0 Primary Job Code: TESTING OF J.W. LEAK DETECTORS
Min Number of Times Task Performed: 1
Secondary Job Code: ___ Secondary Job Time: 0 (Minutes)

---

EXHIBIT 3: Rate Case Data Analysis Parameter Selection Screen

---

Edit SAS data set: WORK.PARAMS

Command ==>

---

Note: This application uses 2 screens.

---

A A & G Analysis Environment - 2 Year Comparison

Title of Comparison: DESCRIPTION OF REPORT
Years in Comparison: 84 83
Amount: 920.00000 932.99900
See screen 2 for
Month Range: 1 12 additional subsetting
Source Code Range: 0 99
options.
Difference Highlight: 10000 30
Minimum 0
Amount Percent

---

Use the numbers in the Variable Tables for
Sort Order Variables and Extra Variables in
your report. Do not repeat any variable
numbers and observe maximums and minimums.
Exhibit 4: Information Center Training Tracking System Primary Option Menu

1. FILE INFORMATION
2. BACKLOG INFORMATION
3. OUTPUT FACILITY
4. PROGRAM LIBRARY
5. END SESSION

PLEASE ENTER REQUEST NUMBER ==>

Exhibit 5: Information Center Training Tracking System Output Facility Menu

1. COURSE BACKLOG
2. SESSION SUMMARY REPORT
3. GRADUATION LISTING BY DIVISION/COURSE
4. LETTER GENERATION
5. INDIVIDUAL FILE LISTING
6. EMPLOYEE FILE UPDATE (FROM EMP. MASTER)
7. END OUTPUT SESSION

PLEASE ENTER REQUEST NUMBER ==>

Exhibit 6: Information Center Training Tracking System Letter Generation Menu

1. EMPLOYEE NOTIFICATION
2. COORDINATOR NOTIFICATION
3. CREATE/MODIFY LETTER FORMATS
4. END LETTER GENERATION

PLEASE ENTER REQUEST NUMBER ==>}

Exhibit 7: Information Center Training Tracking System Data Entry Screen
SAS data set: WORK.OUTSPTMP

San Diego Gas & Electric - SAS System Output Location Selection menu

Output Location: 1 (Press PF 2/4 to generate output)
1. SCREEN: Color: Y (Yellow, Cyan, White, Red, Pink, Green, Blue)
2. IMPACT PRINTER
4. REMOTE IBM PRINTER: Printer ID: F
5. REMOTE XEROX PRINTER: Printer ID: F Font: 1 (Listed below)

Number of Copies: 1 (99 Maximum)

Valid IBM3800 Character Sets:
1 = IX12 2 = PR10 3 = PR12 4 = Vintage 12-P 2 = XCP 14-L
4 = SJ15 5 = CE12 6 = SR12 3 = Titan 10 ISO-P 4 = Titan 10-L
7 = SJTR 8 = OB10 9 = LB12 5 = Letter Gothic 12-P 6 = Spokesman 10-P 7 = Titan 10B-P

Exhibit 8: Output Parameter Selection Screen

Exhibit 8A: SAS Macro code to display output parameter selection screen (exhibit 8) and Interpret responses

305
%MACRO LOADMAC(MODULE,LIB=PRGLIB);
%* *************************************************************;
%* MACRO TO LOAD AND COMPIL A MACRO *
%* *************************************************************;
%LET MODULE = %UPCASE(&MODULE);
%LET LIB = %UPCASE(&LIB);
%
%* LOAD AND COMPILE MACRO AND UPDATE MACDIR INDEX;
%
%INCLUDE &LIB(&MODULE)/NOSOURCE2;
RUN;
%LET MACDIR = &MACDIR &MODULE;
%MEND LOADMAC;

Exhibit 9: LOADMAC Macro Code

%MACRO EXECMAC(MODULE,LIB=PRGLIB,PARGS=);
%* *************************************************************;
%* ***** MACRO TO LOAD, COMPILE AND EXECUTE A *****
%* ***** MACRO. IF MACRO IS ALREADY LOADED *****
%* ***** AND COMPILED THEN IT WILL JUST *****
%* ***** EXECUTE. *****
%* *************************************************************;
%LET MODULE = %UPCASE(&MODULE);
%LET LIB = %UPCASE(&LIB);
%
%* DETERMINE IF MACRO IS ALREADY LOADED;
%* AND COMPILED. IF NOT THEN LOAD AND ;
%* COMPILE. 
%
%IF %INDEX(&MACDIR,&MODULE%STR( )) = 0 THEN %DO;
%LOADMAC(&MODULE,LIB=&LIB)
%END;
%
%* EXECUTE MACRO 
%
%IF %QUOTE(&PARGS) = %STR() %THEN %DO;
%QUOTE(%%%QUOTE(&MODULE));
%END;
%ELSE %DO;
%QUOTE(%%%QUOTE(&MODULE))(%UNQUOTE(&PARGS))
%END;
%MEND EXECMAC;

Exhibit 10: EXECMAC Macro Code