

On The Macro-Based Information System

Jeff F. Sun, Blue Cross Blue Shield of North Carolina, Chapel Hill, NC

Abstract

Running and maintaining an information system become more complex and more time-consuming when the number of components in a system gets big and the running conditions get more complicated. In order to handle this kind of complex information systems, it is better for us to link all system components together logically and systematically. The SAS macro facility/language is a powerful tool to accomplish this task.

The goal of this paper is to introduce a 3M (i.e. Linkage *Macros*, Operation *Macros*, and Utility *Macros*) systematic structure for the effective development and maintenance of the macro-based information system. This paper also discusses the design and development of the macro-based information system in detail. A case study in the MVS environment is also provided.

Introduction

Running and maintaining an information system become more complex and more time-consuming when the number of components in a system gets big and the number of conditions gets large. In order to deal with this kind of complex information systems, it is more efficient for us to put all system components together logically and systematically. This task can be accomplished by using the SAS macro facility in the SAS world.

This paper explores the overall conceptual design considerations and physical development techniques employed in a macro-based

information system with a 3M structure. It is the intention of this paper to provide an insight into the advanced application of macro facility in the macro-based information system. This paper also discusses the components, structure, design, and development of the macro-based information system with a 3M in detail. A case study on MVS is provided as well.

The body of the paper has six parts: in the first two the definition and structure of a macro-based information system with a 3M structure are discussed. The relations, storage, and installation of this kind of macros are presented and analyzed in the next two parts. A case study is also provided to illustrate the concepts discussed earlier. A general discussion concludes the body of the paper. Note that this paper is not to discuss the macro coding. Instead, the focus of this paper is using SAS macro facility to build a more reliable, maintainable, accurate, and efficient information system.

A 3M Structure - 3 Types of Building Blocks

By a macro-based information system, we mean that an information system is an integrated, systematically linked, and logically organized collection of macro codes. Any tasks that are used repeatedly or in several programs are candidates for a macro. These codes consist of operation macros, linkage macros, and utility macros. As mentioned by Levine (1989), in the macro-based system "all SAS statements are generated from macros. There is no 'loose' SAS code which exists outside of a macro definition."

A macro-based information with a 3M structure is to make an information system more reliable,

maintainable, and efficient. By 3M, we mean that a macro-based information is made up of 3 kinds of macro definitions based on the function they serve. They are Linkage Macros, Operation Macros, and Utility Macros. In order to invoke this macro-based information system we need a DRIVE program to call these macros, which is environment-dependent.

Operation Macros

Operation Macros are the macro definitions which are the mixed macro codes and SAS statements. They are used to perform actual data processing and information reporting tasks. An operation macro may transfer none, one, or more than one parameter or a macro call to an operation, that is, to accomplish a certain task or certain tasks.

The body of an operation macro may include DATA steps or PROC's or both. Figure 1 illustrates a very simple operation macro, which has only one SAS statement.

```
%MACRO PRINTIT(DATAIN=);
  PROC PRINT DATA=&DATAIN;
%MEND PRINTIT;
```

Figure 1. An Example of Operation Macros

Linkage Macros

One of the important uses of SAS macros in developing a user-friendly application is that of controlling code execution (Katsanis, 1991). The macro facility can be used to manipulate the SAS source statements. In other words, we can control the execution of our SAS programs based on different conditions. It is possible to execute certain macro statements or portions of them conditionally or generate SAS code conditionally. The linkage macros are just the application of this feature in macro facility. Linkage Macros are the macro definitions which are the pure macro statements and used to control the conditional execution of the

operation macros. A Linkage macro may transfer none, one, or more than one parameter or a macro call to one or more macro calls.

The linkage macros include executable statements that can manipulate information and perform logical control. Figure 2 shows an example of linkage macros. In this macro, three operation macros are conditionally executed based on the different values of the parameter LOCATION.

```
%MACRO SELECT(LOCATION=);
  %IF &LOCATION=STATE OR &LOCATION=%STR()
    %THEN %DO;
      %STATE();
    %END;
  %ELSE %IF &LOCATION=REGION %THEN %DO;
    %DO I=1 %TO 4;
      %REGION(REGCODE=&I)
    %END;
  %END;
  %ELSE %IF &LOCATION=COUNTY %THEN %DO;
    %DO J=1 %TO 20;
      %COUNTY(CTCODE=&J)
    %END;
  %END;
%MEND SELECT;
```

Figure 2. An Example of Linkage Macros

Utility Macros

Utility Macros are the macro definitions which includes the pure macros or DATA steps, for example, DATA _NULL_, or PROC SQL,¹ and employed to generate all kinds of tests and tokens for conditions in the operation macros or linkage macros. They can be used to set default initialization. A utility macro may transfer one or more parameters or a macro call to one or more parameter outputs.

When we run the macro-based information system, it allows the utility macros to update information i.e. macro variables, maintained by

¹ We can use the INTO clause in a SELECT statement in the SQL procedure to specify one or more macro variables to create.

the system. All these macro variables are defined and created in the global environment. Figure 3 gives a simple example of utility macros.

```
%MACRO FLAG(RUNDATE=);
%GLOBAL MFLAG QFLAG YFLAG;
%LET MFLAG=Y;
%LET QFLAG=N;
%LET YFLAG=N;
%IF &RUNDATE=%STR() %THEN %DO;
%LET MONTH=%SUBSTR(&SYSDATE,3,3);
%IF &MONTH=MAR OR &MONTH=JUN OR
&MONTH=SEP %THEN %DO;
%LET QFLAG=Y;
%LET MFLAG=N;
%END;
%IF &MONTH=DEC %THEN %DO;
%LET YFLAG=Y;
%END;
%END;
%ELSE %DO;
%LET MONTH=%SUBSTR(&RUNDATE,3,3);
%IF &MONTH=MAR OR &MONTH=JUN OR
&MONTH=SEP %THEN %DO;
%LET QFLAG=Y;
%LET MFLAG=N;
%END;
%IF &MONTH=DEC %THEN %DO;
%LET YFLAG=Y;
%END;
%END;
%END FLAG;
```

Figure 3. A Sample of Utility Macros

A long program should be written as a collection of linkage macros, operation macros, and utility macros. Each operation macro should capture some small task of the overall problem.

Procedure-like Macros - The Structure of Building Blocks

In order to build a production/reporting program, the quasi-structured programming technique with macros allows us to work with "building blocks" that can be used to assemble SAS programs. Just like structured-programming techniques used in other computer language, quasi-structured programming techniques use subroutines or macros for

repetitive tasks. The use of macros makes it possible that a modular structure can be used in the SAS to reduce the probability of coding error (Howard et. 1995). These programming techniques extend the modular structure of the SAS System, where the DATA and PROC steps define individual processes and many pre-coded subroutines and functions are provided.

All linkage, operation, and utility macros are built as *procedure-like* macros. Johnson (1990) discussed the procedure-like macros in a SUGI paper. He pointed out that the procedure-like macros can make your customized application programs available to other users without their understanding your code. They need only know how to access the macro and how to pass data and parameter values to it.

A Top-Down Design - Relations of Building Blocks

We can decompose a big problem into many subproblems. Each of these subproblems can be coded directly as an operation macro. Then these operation macros can be controlled by linkage macros, which are invoked by the DRIVE program, to solve the overall problem. Note that by designing the code this way, we can add further operation macros analyzing the data or utility macros manipulating the SAS codes without affecting the program structure.

Breaking down the specifications into small, manageable, well-defined tasks or modules makes successful applications development possible. A macro should be used to perform a single, discrete task. This keeps the number of lines of codes in a macro at a minimum. A further benefit of a top-down design is that each program as a whole is more readable and self-documenting.

Breaking a problem into small subproblems that are then coded as operation macros is critical to good programming style. The programmer thinks of the tasks to be performed and codes each task as a macro. If a particular task is complicated, then that task in turn can be subdivided into other tasks, each coded as an operation macro.

Figure 4 shows a flowchart for a macro-based information system and its components. The relationship for these components is summarized as follows.

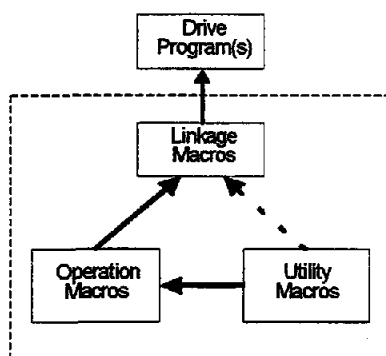


Figure 4. Structure of a Macro-Based Information System

A macro-based information system is made up of one or more subroutines, one of them being a DRIVE program. Program execution always begins with DRIVE program. The main body of the DRIVE program is a macro call. When program control encounters this macro, the linkage macro is called, or invoked. This means that program control passes to the linkage macros. The linkage macro control passes to the operation macros. After the operation macros accomplish their tasks, program control is passed back to the calling environment, which then continues with its work.

For example, a simple DRIVE program on MVS is as follows.

```

//DRIVE JOB accounting-information
// EXEC SAS
//SCLIB DD DSN=$USER.SCLIB,DISP=SHR
//SYSIN DD *

OPTIONS MSTORED SASMSTORE=SCLIB;

%LINKAGE(PROGRAM=TASK1)
  
```

Figure 5. A Sample DRIVE Program

Operation and utility macros are common in the most production systems; however, few linkage macros are used in the production system. In general, running operation macros is based upon different conditions, which in many cases are manually controlled.

Nested macros occur quite often in the 3M structure and are a natural consequence of the top-down design process. Since all operation macros are nested within the linkage macro block, the DRIVE program only needs to invoke the linkage macro for conditional execution of all operation macros.

Macro Storage and Installation

It is desired to utilize the Macro Autocall or Stored Compiled Macro facility to store macro definitions.² The autocall facility or the stored compiled macro facility allows programmers to invoke a SAS macro without having to define the macro in their current job or session. The development of an effective autocall library within a system group allows programmers to quickly perform common processing tasks with consistency and accuracy. It is suggested to create two central libraries. One is to use the stored compiled macro facility to store linkage and utility macros since these two kinds of macros don't need to be changed or modified

² Wilson (1994) discusses using the SAS Autocall Facility for the development of a common set of macro utilities for use by application programmers.

very often. The other one is to use the autocall facility to store the operation macros due to frequent modifications to the operation macros.

A Case Study

In this section an example, which illustrates most of the concepts discussed in this paper, will be presented. Four operation macros and two utility macros are made up for this case study. Two systems will be developed to create different reports. Since we are much interested in developing a macro-based information system rather than coding each macro in detail, the codes for the operation and utility macros will not be shown here.

```
%MACRO STATE(REGION=,DATAIN=);
...
%MEND STATE;
```

Figure 6. The Operation Macro %STATE()

The operation macro %STATE() (see Figure 6) creates a statewide report. Figure 7 shows the operation macro %REGION(), which outputs an annual report for each specified region.

```
%MACRO REGION(REGION=,DATAIN=);
...
%MEND REGION;
```

Figure 7. The Operation Macro %REGION()

```
%MACRO COUNTY(COUNTY=,DATAIN=&DATA);
...
%MEND COUNTY;
```

Figure 8. The Operation Macro %COUNTY()

The operation macro %COUNTY() (shown in Figure 8) gives a total cost report for each specified county. If the cost increase is more than 10%, a cost/benefit analysis report for this

county will be produced by the operation macro %COST() (see Figure 9).

```
%MACRO COST(REGION=,DATAIN=);
...
%MEND COST;
```

Figure 9. The Operation Macro %COST()

The following two figures (Figure 10 and Figure 11) are two utility macros used in the system development. The utility macro %CHECK() reads in a data set, summarizes the data by year, and sets the value for the macro variable FLAG if the increase/decline in total cost or sale from the previous month is greater than a specified percentage. The utility macro %UPDATES() updates all macro variables used in the operation macros, i.e. time periods, file names, titles, initial values, etc.

```
%MACRO CHECK(DATE=&SYSDATE);
...
%MEND CHECK;
```

Figure 10. The Utility Macro %CHECK()

```
%MACRO UPDATES(DATE=&SYSDATE);
...
%MEND UPDATES;
```

Figure 11. The Utility Macro %UPDATES

System #1

The first system is based on the following information. For each month, the operation macro %STATE() should be run to produce statewide reports. If that month is December, the operation macro %REGION() should be run to produce annual reports for all 5 regions. In the middle of each year, that is, in June of each year, there will be a request for the first 6-month cost summary for each county. So the operation macro %COUNTY() should be run for all 30 counties if it is in June. For each county, if their total cost increase is greater than 10%, a cost/benefit analysis report will be produced for

that county. The first macro-based information system has the following flowchart (see Figure 12).

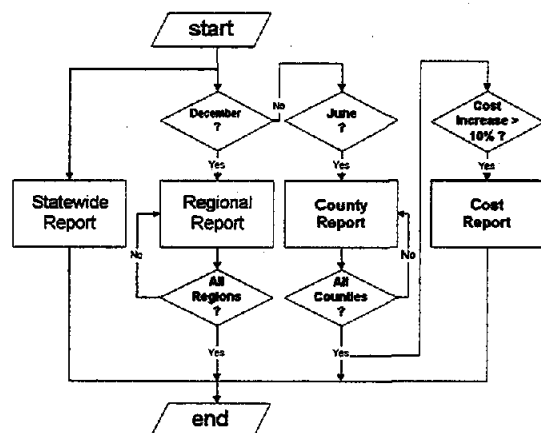


Figure 12. The Flow Chart for System #1

```

//DRIVE JOB accounting-information
// EXEC SAS
//SCLIB DD DSN=$USER.SCLIB,DISP=SHR
//SYSIN DD *

OPTIONS MSTORED SASMSTORE=SCLIB;

%LINKAGE1 (DATE=&SYSDATE)
  
```

Figure 13. The DRIVE Program for System #1

```

%MACRO LINKAGE1 (DATE=&SYSDATE);
  %UPDATES(NEWDATE=&DATE)
  %STATE(DATAINO=&DATA)
  %IF %SUBSTR(&DATE,3,3)=DEC %THEN %DO;
    %DO I=1 %TO 5;
      %REGION(REGION=&I)
    %END;
  %END;
  %ELSE %IF %SUBSTR(&DATE,3,3)=JUN %THEN %DO;
    %DO J=1 %TO 30;
      %COUNTY(COUNTY=&J)
      %CHECK()
      %IF &FLAG=Y %THEN
        %DO;
          %COST()
        %END;
      %END;
    %END;
  %END;
%END;
  
```

```

%END;
%MEND LINKAGE1;
  
```

Figure 14. The Linkage Macro %LINKAGE1()

System #2

The second macro-based information system can be described using the following flow chart (see Figure 3). This system is build on the following conditions. Every month the regional reports should be produced using the operation macro %REGION(). If the total cost increases by more than 10%, the cost/benefit analysis will be performed. The county report for all corresponding counties in that region will be produced as well. If that month is March, or June, or September, or December, the statewide report will be created. If there is a more than 10% decline in sales in any quarter, the cost/benefit analysis will be analyzed for the whole state.

```

//DRIVE JOB accounting-information
// EXEC SAS
//SCLIB DD DSN=$USER.SCLIB,DISP=SHR
//SYSIN DD *

OPTIONS MSTORED SASMSTORE=SCLIB;

%LINKAGE2 (DATE=&SYSDATE)
  
```

Figure 15. The DRIVE Program for System #2

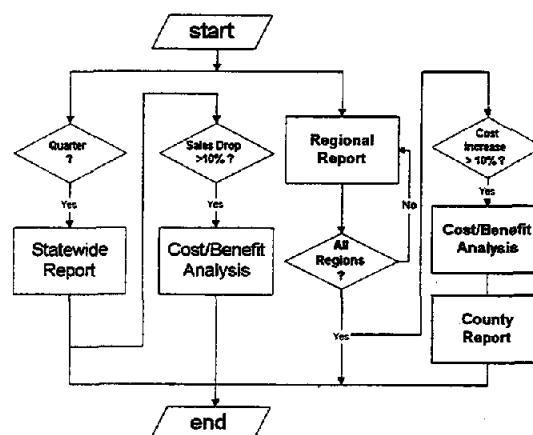


Figure 16. The Flow Chart for System #2

```

%MACRO LINKAGE2 (DATE=&SYSDATE);
  %UPDATES (NEWDATE=&DATE)
  %LET MONTH=%SUBSTR(&DATE,3,3);
  %IF &MONTH=MAR OR &MONTH=JUN OR
    &MONTH=SEP OR &MONTH=DEC %THEN %DO;
    %STATE()
    %CHECK()
    %IF &FLAG=Y %THEN %DO;
      %COST()
    %END;
  %END;
  %DO I=1 %TO 5;
    %REGION (REGION=&I)
    %CHECK()
    %IF &FLAG=Y %THEN %DO;
      %DO J=1 %TO &K;
        %COUNT (COUNT=&J)
        %COST()
      %END;
    %END;
  %END;
%END LINKAGE2;

```

Figure 17. The Linkage Macro %LINKAGE2()

Figure 17 translates the logic shown in Figure 16 into a macro code. From the above example, we can see that using the concepts of the 3M structure and macro building blocks, we can construct a new system easily and maintain this system efficiently.

Conclusion

Considering the growth in the amount of processing data in a production system, running and maintaining a production system can be a difficult task. This paper has presented a 3M structure for the macro-based information system, which may be suitable for efficiently developing, running, and maintaining an information system.

One major advantage of using 3M structure is that reusable codes can help us to develop new system rapidly. Another important reason for using this kind of modular 3M structure is that it

is much easier to test and debug a macro as a separate entity.

Most of the advantages discussed so far were from the programmer's perspective. There are quite a few advantages to this system that pertain to the operator. One of the obvious advantages is that it is very easy for an operator to maintain this kind of systems due to a procedure-like structure for all three kinds of macros.

Trademarks

SAS is a registered trademark of the SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

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

References

- Forest, Christian M., (1994) "SAS Macro Techniques Useful in Development of Flexible Applications," in *Proceedings of the Nineteenth Annual SAS Users Group Conference*, Cary, NC: SAS Institute Inc.
- Howard, Neil, Linda W. Pickle, and James B. Pearson, Jr., (1995) "It's Not a Bug, It's a Feature!" in *Proceedings of the Third Annual SouthEast SAS Users Group Conference*, Cary, NC: SAS Institute Inc.
- Hubbell, Katie A., (1990) "Conquering the Dreaded Macro Error," in *Proceedings of the Fifteenth Annual SAS Users Group Conference*, Cary, NC: SAS Institute Inc.
- Johnson, Robert E., (1990) "The SAS Macro: An Aid to the User," in *Proceedings of the*

Fifteenth Annual SAS Users Group International Conference, Cary, NC: SAS Institute Inc.

Levine, Howard, (1989) "Building Macro-Based Systems," in *Proceedings of the Fourteenth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute Inc.

Katsanis, Gary, (1991) "An Introduction to the SAS Macro Language," in *Proceedings of the Sixteenth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute Inc.

SAS Institute Inc., (1990) *SAS Guide to Macro Processing, Version 6, Second Edition*, Cary, NC: SAS Institute Inc.

Wilson, Steven A., (1994) "Development A SAS System Autocall Macro Library As An Effective Toolkit" in *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute Inc.

Any questions or comments regarding the paper may be directed to the author:

Jeff F. Sun
Health Services Information And Analysis
Blue Cross Blue Shield of North Carolina
ET 600, 3rd Floor
Durham, NC 27705
Phone: (919)-765-7854
Fax: (919)-765-7840
E-mail: jeff_sun@scp11.bcbsnc.com