The Use of the Macro Facility in a System Development Environment
Jeff Phillips, ORI, Inc.

Introduction

There is no question that the Macro facility adds a new degree of power to the SAS(r) system. Now that Macro has been available for a few years, users have been able to explore the various ways in which it enhances SAS. At ORI, the major thrust of SAS programming since about 1981 has been in the development of large-scale software systems. In effect, ORI has been using SAS to do the job that might be done by, for example, PL/I in a similar situation.

The single largest SAS-based contract for ORI has been that with the U.S. Department of Labor, Bureau of Labor Statistics (BLS). BLS has been engaged in what is known as the "CPI Revision" — a major overhaul of all aspects of the Consumer Price Index. Revisions of the CPI occur every 10 years, corresponding to the release of new decennial U.S. Census data, and involve not only changes in sampling, data collection and statistical methodology, but also changes in the computer software used to produce the CPI. One of the features of the current CPI Revision is that SAS is being used as the major mainframe language for CPI processing. (For a complete discussion of the CPI revision and the development of SAS software in support of it, see "The Use of the SAS System in the Development of the Production System for the Consumer Price Index (CPI)" by James Johnstone, Merry Rabb, and Joshua Shari, in these Proceedings.)

The switch to SAS for the CPI means that multiple large-scale production systems are being rewritten in SAS, and the Macro facility has played an important role. In using SAS/Macro, ORI has gone through the learning process as other Macro users, and we have learned that Macro is invaluable for the work we are doing, but not without its hazards. The purpose of this paper is to discuss the role Macro has played, and to identify some of these so-called hazards of using Macro in a systems development environment.

The BLS/CPI Systems Environment

Before discussing the role of the Macro facility, it is necessary to define the arena in which it is used at BLS, and it is especially necessary to give a specific definition to what may be one of the most general terms in the computer software lexicon — "system". At BLS, a software system may be defined as having the following characteristics:

- It is run on a regular or at least a recurring basis
- It consists of multiple functional units, each of which may stand alone
- It runs in batch mode, using interactive processes only to set up the batch jobs and pass in operational parameters
- It is large, often processing hundreds of thousands or even millions of records in a single job

An important implication of the characteristics of a CPI system is that a key use of Macro is for the dynamic control of the system's functioning without user intervention. This fact influences the kinds of Macro features and techniques that are used.

One other aspect of software systems at BLS should be described before proceeding to a specific discussion of Macro's role, and that is how the systems are developed. The scale and complexity of each system combine to require a team development approach. Briefly, these are the characteristics of the system development approach taken at BLS:

- Development tasks are divided among staff
- Production (use of the system) is separated from development — the system users are not the system developers
- Maintenance of the system may be separate from both development and production
- Resources (time, personnel, and money) are highly constrained

The immediate implication of the two major characteristics of system development — separation of tasks and resource constraint — for SAS/Macro (or any code development, for that matter) is that techniques must be both effective and communicable. In this environment, unmaintainable code is just as bad as code that doesn't work at all.

The Role of Macro

The challenge of writing software for the CPI is enormous one, and the Macro facility has been relied upon increasingly to provide the power and flexibility needed to write that software in SAS. Macro gives SAS capabilities that it never had before, or had only to a very limited extent:

- Procedural structure at the system level
- Conditional step compilation and execution
- Repetitive and/or iterative compilation and execution of steps or parts of steps
- Storage of standardized system code modules
- Tailoring of code modules

These capabilities are critical for system design, and the Macro techniques used for each merit a brief explanation. Procedural structure (see Figures 1 and 2) is not so much a technique as the natural result of Macro's ability to package code and to conditionally compile and execute it. Normal SAS
processing is a rigid sequence of DATA and PROC steps, one step following another. Macro allows a main routine/subroutine type of structure, in which code can be called upon as needed.

Conditional step compilation and execution, already referred to in the last paragraph, is an obvious advantage of Macro-based systems; however, an important point about this feature is often overlooked: Macro does not just allow conditional execution, it allows conditional compilation and execution. It is the conditional compilation of code that allows code to be changed by data processed previously, and is a feature that may well be unique to the SAS system. The difference between compilation and execution, which is so critical to understanding SAS processing in general, is even more necessary to an adequate understanding of where, when, and how to use the Macro facility properly.

Repetitive or iterative compilation/executio n is similar to conditional compilation/execution, but its main advantage is that it allows code to be repeated without re-keying or copying using a text editor. Because of the fact that the compile is repeated, the code can be altered for each repetition — all automatically.

Storage and tailoring of code modules may be treated together. Macro allows code to be packaged in just about any way the programmer desires; it also allows the code “package” to be tailored exactly for each application. In this way, it functions much as SPF file tailoring does, but without the necessity of leaving the SAS system. Thus, Macro’s version of file tailoring can be used in a non-interactive environment — a fact critical to the CPI systems applications.

Figure 3 is a table of Macro capabilities and the specific Macro statements or features used to provide these. The last column of the table lists the primary use of the feature, whether it is batch or interactive. The point to this differentiation is that the types of Macro techniques emphasized by the CPI systems environment are different than those required by an interactive, information center type of environment. The difference is one of emphasis, except for those items such as XINPUT, XTSO, and XDAS that are purely interactive in nature. It might be stated that a subset of the Macro facility is used in the CPI environment, but it is more appropriate to state that, because the processing is batch mode, the primary means of control is via the DATA step, as opposed to user intervention. Once the batch job is running, Macro must rely on the DATA step interface (SYNGET and CALL SYNGET) to get the information it needs to conditionally compile/execute a repeat, or tailor a code module. Figure 4 lists those Macro features most frequently used in developing CPI Macro based systems.

Warnings on the Label

Given the above, it is clear that the Macro facility is a godsend to anyone trying to develop a complex software system in SAS. However, like any powerful tool, Macro can be dangerous if misused, and is perhaps somewhat difficult to use properly. There are several factors which must be accounted for when using Macro for system development:

- Using Macro requires a high level of SAS expertise
- Communicating Macro operations in system design diagrams or text can be difficult
- Using Macro requires special attention to code maintainability and clarity

The “high levels of expertise” required of system designers and that required of programmers are different, but both are important to the successful implementation of a system. The designer must be aware of what Macro can do (and what it can’t) when he or she is writing a design document. The programmer’s expertise includes awareness of what Macro can do, but much more importantly, the programmer must fully comprehend how SAS works at compile and execution time, and how the Macro facility interacts with the SAS system. The Macro facility is a sense run contrary to the philosophy of SAS in general: the SAS system allows the user to get results without necessarily being aware of the internals of the computer processing going on, and that is one of its primary strengths and selling points. Macro, on the other hand, demands that the user (or rather, the developer) know exactly what is going on at all times. Macro systems appear very automatic and clean when well designed and implemented, but they are in no way easy to create. There is no “cookbook” approach, either. Using a set of canned Macros and a superficial knowledge of how SAS and Macro work will suffice until the programmer encounters the first Macro error message.

The knowledge of SAS and Macro required for effective programming can perhaps best be illustrated by some negative examples. In other words, where is it that programmers have special trouble because of Macro and SAS? Figure 5 shows the first area of difficulty. In this example, SAS DATA step logic is being used to control the XLET statements highlighted by the arrows. While this would not cause an error message, or even a note of warning, the results would be incorrect — the TITLE statement for the PROC PRINT would always contain ‘HARDWARE’ for the department. Without having taken SAS Institute’s Macro course, or having extensive experience with Macro and SAS, the programmer would not be aware of the fact that Macro operations happen while any DATA or PROC step is still compiling.

Figure 6 illustrates how an imperfect understanding of SAS step boundaries and the compile/execute sequence can lead to frustration. The XNOBS macro contains a NULL DATA step that uses the NOBS option on the SET statement to determine the number of observations contained in the data set represented by BDSN. The result is to be used in a message written to the SAS log by XPUT. Unfortunately, there is no step boundary (i.e., RUN, CARDS, CARDS4, DATA, PROC, or PARMCARDS) statement between the NULL DATA step and the XPUT statement. The macro will work, except that the XPUT will be processed by the Macro facility as the DATA step is compiling, since no step boundary has been encountered to tail the SAS supervisor that compile should finish. When the XPUT is processed, the Macro facility will try to resolve BNM, which will be null because the
CALL SYMPUT function has not yet executed. This sort of problem is almost impossible to diagnose without knowing that (1) Macro operations such as symbolic resolution happen at compile time, but (2) CALL SYMPUT is a DATA step function, and does not do anything until execution time, and (3) SAS must encounter a step boundary in order to stop compilation and begin execution of a step.

Figure 7 uses the same %NOBS macro to illustrate yet another source of confusion. Macro variables can exist in global or local symbol tables, depending on how and when they are set up. By removing the %GLOBAL statement from the %NOBS macro, we have made &NUM a local macro variable. Once %NOBS is finished executing, &NUM will disappear. This will cause a 1351 message to appear on the log. And, once again, the %PUT text will be incorrect. Unlike the problem in Figure 6, which has as much to do with SAS as it does with Macro, the problem in Figure 7 is entirely a Macro-based one.

Figure 8 shows the %NOBS macro properly written. The RUN statement and the %GLOBAL statement are both present, and the %PUT message will issue the proper text to the SAS log. Debugging the %NOBS macro is a relatively easy task for someone familiar with the workings of SAS and Macro; it could be a confidence destroying puzzle for anyone else.

Even if both the designer and the programmer are Macro "experts", there can be a problem in communicating what the designer has in mind where Macro is concerned. ORI uses Yourdoun structured analysis techniques in designing systems, and these techniques work very well when it comes to breaking down a system into its component processes in SAS, because eventually one gets to the DATA or PROC step level where each Yourdoun circle or box becomes equivalent to one SAS step. When Macro enters the picture, however, things begin to branch and loop in a way more reminiscent of traditional flowcharting. Unfortunately, flowcharting is really meant to describe processing within a step, and Macro by its very nature crosses step boundaries, and operates simultaneously with SAS processing. Thus it appears that some hybrid form of design technique is needed to properly communicate Macro to the programmer. Until one is developed, it is up to the designer and the lead programmer of a system to deal with the problem on an individual basis.

The issue of code maintainability in an environment where the developers, users, and maintainers of a system are not the same people is a fairly obvious one, no matter what software is being employed for development. Macro, however, because it works at a level "beneath" the SAS code it affects, and because it has its own independent operations such as symbolic resolution, can be extremely hard to follow when it is built into a system. (Anyone who has stored at code containing such things as multiple emperors, %UNQUOTE, or other Macro monstrosities knows just what this means.) To some degree, the use of Macro must be governed by the maintainability issue more than by the more usual criteria of "does it work?" and "can it get the job done?".

Conclusions

The BLS/ORI systems development environment has been an excellent proving ground for the use of Macro. The special nature of the systems being designed, as well as the development environment itself, have led to a special emphasis on those parts of Macro that enable a batch system to control itself during its run. While Macro has proved to be invaluable in giving SAS the capabilities it needs for this type of system implementation, ORI's experience has shown that the Macro facility is not without its problems when used for large systems. The way Macro is used, and the training and guidance given to those who develop, use, and maintain Macro-based systems must all be given strong consideration during the system development process.

Note: SAS is a registered trademark of SAS Institute, Cary, N.C.

Acknowledgements: The author would like to thank Merry Rabb, Neil Howard, and Craig Ray of ORI, Inc. for their help in preparing this paper and the SUGI presentation that went with it.
Without the Macro Facility, the processing of a SAS job or session is sequential and fixed:

![Diagram: Traditional SAS Processing]

With the Macro Facility, SAS processing can be constructed using a main/subroutine "procedural" structure:

![Diagram: SAS Processing Using the Macro Facility]
### Figure 3: An Overview of Macro Capabilities

<table>
<thead>
<tr>
<th>DESIRED CAPABILITY</th>
<th>MACRO TECHNIQUE</th>
<th>APPLICABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Storage</td>
<td>%MACRO ... %END</td>
<td>Both</td>
</tr>
<tr>
<td>File Tailoring</td>
<td>Macro Variables/</td>
<td>Both</td>
</tr>
<tr>
<td></td>
<td>Symbolic Resolution</td>
<td></td>
</tr>
<tr>
<td>Step-to-Step Information</td>
<td>CALL SYMPUT, SYMGET</td>
<td>Both</td>
</tr>
<tr>
<td>Passing</td>
<td>Symbolic Resolution</td>
<td></td>
</tr>
<tr>
<td>Conditional Execution</td>
<td>%IF / %THEN, %ELSE</td>
<td>Both</td>
</tr>
<tr>
<td>Repeated or Iterative</td>
<td>%DO ifdef, %UNTIL</td>
<td>Both</td>
</tr>
<tr>
<td>Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Interaction</td>
<td>%INPUT, %PUT</td>
<td>Interactive</td>
</tr>
<tr>
<td>String Processing</td>
<td>Macro Functions</td>
<td>Both, emphasis on Interactive</td>
</tr>
<tr>
<td>System Commands</td>
<td>%TSO, XCMAS</td>
<td>Interactive</td>
</tr>
</tbody>
</table>

### Figure 4: Macro Features Frequently Used in the CPI Environment

- %MACRO ... %END
- %IF / %THEN, %ELSE
- SYMPUT, CALL SYMPUT
- Symbolic Resolution
- %DO ifdef, %UNTIL
Figure 5: Using DATA Step Logic to Control Macro Operations

```sas
OPTIONS DQUOTE;
DATA SALES;
  INPUT DEPT $ NAME $ SALES $;
  IF DEPT = 'SOFTWARE' THEN
    %LET MACDEPT = SOFTWARE;
  ELSE %LET MACDEPT = HARDWARE;
CARDS;
  [RAW DATA LINES]
PROC PRINT DATA = SALES;
  TITLE "SALES FOR &MACDEPT;";
RUN;
```

Figure 6: The Imperfect Understanding of Step Boundaries

```sas
%MACRO NOBS;
%GLOBAL NUM;
DATA _NULL_;
  IF 0 THEN SET AOSN POINT = _N_ NOBS = TOTAL;
  CALL SYMPUT('NUM',LEFT(PUT(TOTAL,8.))); STOP;
%MEND NOBS;

DATA TEST;
  DO I = 1 TO 5;
    OUTPUT;
  END;
RUN;
%LET DSN = TEST;
%NOBS
  PUT THE DATA SET AOSN HAS &NUM OBSERVATIONS.;
  [%NUM WILL RESOLVE TO NULL] 
NOTE: THE DATA SET WORK.TEST HAS 5 OBSERVATIONS AND 1 VARIABLE
THE DATA SET TEST HAS OBSERVATIONS
```

528
Figure 7: Confusion Over Global and Local Symbol Tables

PROCEDURE NOBS;
 DATA _NULL_;
   IF 0 THEN SET &DSN POINT = _N_ NOBS = TOTAL;
   CALL SYMPUT('NUM',LEFT(PUT(TOTAL,8.)));
   STOP;
RUN;
RPROC NOBS;

DATA TEST;
  DO _I_ = 1 TO 5;
    OUTPUT;
  ENDDO;
RUN;
RLET DSN = TEST;
RPROC NOBS
  XPUT THE DATA SET &DSN HAS &NUM OBSERVATIONS.;
  [&NUM WILL NOT RESOLVE]
NOTE: THE DATA SET WORK.TEST HAS 5 OBSERVATIONS AND 1 VARIABLE
NOTE: 1361 APPARENT SYMBOLIC REFERENCE NOT RESOLVED.
THE DATA SET TEST HAS &NUM OBSERVATIONS

529
Figure 8: The %NOBS Macro Property Written

%MACRO NOBS;
%GLOBAL NUM;  %GLOBAL STATEMENT PRESENT
DATA _NULL_;
IF 0 THEN SET MSON POINT = _N_ NOBS = TOTAL;
CALL SYMPUT('NUM',LEFT(PUT(TOTAL,8.))); STOP;
RUN;  %END NOBS;

DATA TEST;
DO I = 1 TO 5;
  OUTPUT;
END;
RUN;
%LET DSN = TEST;

%NOBS

%PUT THE DATA SET %DSN HAS %NUM OBSERVATIONS.:  %NUM WILL RESOLVE TO 5
%LET DSN = WORK.TEST;

%NOBS

NOTE: THE DATA SET WORK.TEST HAS 5 OBSERVATIONS AND 1 VARIABLE

THE DATA SET TEST HAS 5 OBSERVATIONS

530