UNLIMITING A LIMITED MACRO ENVIRONMENT

David S. Septoff
Fidia Pharmaceutical Corporation

ABSTRACT

The full Macro facility provides SAS users with an extremely powerful programming tool. It allows for conditional execution of DATA and of PROC steps, creation of parameter drive applications, passing SAS variables across steps and much much more. Unfortunately, the full power of Macro is not available in some operating environments. These programmers need not despair. SAS code can be written to simulate SAS Macros using the Base product available on all operating systems.

This paper presents an example of a "SAS Macro Based System" developed in a VAX environment and, finally draws some conclusions on how SAS, as a programming language, can be just as powerful in a "partial" Macro environment as it is in a "full" Macro environment.

INTRODUCTION

The SAS system has experienced phenomenal growth in power, usage, and availability since its introduction. Originally designed to provide scientists and engineers with a data analysis tool that was powerful yet easier to use than traditional programming languages, the SAS DATA step give users the data manipulation capability of a high level language, while many SAS procedures provide a wide variety of easy to use data handling analysis, and reporting capabilities. Unfortunately, the full power of SAS is not available in all operating environments. Those that use the DEC (Digital Equipment Corporation) version of SAS, and all related environments, do not have the full power of the SAS Macro facility (SAS Institute, at SUGI 13, announced that full Macro will be available with the release of version 6 for the mini computer). These SAS users should not despair. From this paper you will get, if nothing else, a generic approach using the Base SAS product with its Macro capabilities to 'simulate' the capabilities of macro in the full Macro environments.

This paper is designed to show how, in a partial macro environment, a SAS user can have power similar to a full macro environment. To explore this area, SAS processing as a whole must be placed in its proper context. The flow of the archetypical SAS job is shown below:
FIGURE 1

DATA

PROCEDURE

DATA

PROCEDURE

PROCEDURE
Notice that each DATA and PROC step is compiled and executed separately and in sequence. The next figure shows the structure of a typical modular system, in which a main routine controls system execution and calls the appropriate subroutines:

**FIGURE 2**

![Diagram showing modular system structure](image)
SEPTOFF / Unlimiting Macros

The figure below lists the three major requirements of software systems, and allows one to compare SAS without Macro to traditional programming languages as to the ability each has to satisfy the three requirements:

<table>
<thead>
<tr>
<th>System Requirements</th>
<th>Feature of Software Systems</th>
<th>SAS Macro Component of Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modular Structure</td>
<td>A main routine governs execution and calls subroutines when needed. Each subroutine performs a particular function</td>
<td>Macros which package code into main and subroutines.</td>
</tr>
<tr>
<td>Conditional Execution</td>
<td>Procedures are executed only when required</td>
<td>Macro %if logic</td>
</tr>
<tr>
<td>Parameter Control</td>
<td>Parameters governing an individual run of a system are passed to the system</td>
<td>Macro Variables</td>
</tr>
</tbody>
</table>

One may look at the chart above and one may be confused as to how, in a partial macro environment, could a system be developed using the same set of requirements. Well the figure below may begin to answer this question:

<table>
<thead>
<tr>
<th>System Requirements</th>
<th>Solution in A Non Mainframe Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modular Structure</td>
<td>Files which package code into Main routines and to functional pieces of the system</td>
</tr>
<tr>
<td>Conditional Execution</td>
<td>‘DATA <em>NULL</em>’ and ‘PUT’, ‘FILE’, and ‘%INCLUDE’ statements</td>
</tr>
<tr>
<td>Parameter Control</td>
<td>Parameter files and Macro Variables</td>
</tr>
</tbody>
</table>

The requirement of modular structure, where a main routing controls the execution of the system and calls in subroutines, is different from standard SAS structure. It is this modular aspect of systems which is the most difficult for SAS to emulate; unfortunately, the other features of software systems are somewhat derived from this feature. For example,
defining system execution parameters is a natural function of modular structure. In SAS without Macro there is no easy way for a system to read parameter values where they are needed. Conditional execution of blocks of code is likewise a natural function of the modular structure. It is certainly not a natural function of SAS, in which each SAS step is separate and sequentially executed.

Macro, in a full macro environment allows a SAS-based system to have a modular structure in which a main Macro can control execution of the SAS steps, calling in other Macros as needed. In a partial macro environment this does not have to change. Instead of a main macro the system is developed using a main routine which will call subroutines and subroutines which will call other subroutines and so on. While macros allow for the packaging of code into functional units, so too do the subroutines just spoken of.

The only thing that differs is that the %MACRO/%MEND are not used within the partial macro environment. The routines, like macro, can contain anything from a DATA or PROC step to multiple DATA and/or PROC steps, and can be invoked whenever needed.

Macro variables still are the controlling parameters; they are generated by the system and can be set by the user. Conditional execution is still a natural feature in a system developed under this environment of Macro. The packaging of code and the flexibility allowed by Macro is still retained.

Macro Based System Within The Partial Macro Environment:

1. Main Routines - The important constant of any software system is its main routine - the routine or code that functions as the 'supervisor' of the system. It is the logical core of the system. A main routine in this environment is a set of code that initiates and directs the execution of the subroutines within the system. Compare this to a main routine with an environment with full macro capabilities and you can replace the word "routine" with the word "MACRO." The figure below shows a main routine from a full macro environment:

%MACRO ADD (OLDDAT=, NEWDAT=, TRANS=, BYVARS=, SORT=NO);
%NOBS (DATA=&TRANS)
%IF&NOBS NE'0'%THEN
  %DO;/*PERFORM UPDATE */
  %IF&SORT = YES%THEN
    %DO;/*PERFORM SORT */
The figure below is a main routine within a partial Macro environment. Note there are a few ways to read the parameters:

1. From a file
2. From a SAS data set.
3. From a SAS/AF Screen.

This figure assumes that there is raw data file with the parameters:

FILENAME PARM ‘ PARM.DAT’;
FILENAME OUT ‘RUNFILE.SAS’;
Options DQUOTE;
Data _Null_
Infile parm;
Input olddat $ 1-8
        newdat $ 9-16
        trans $17-24
        Byvars $25-64
        Sort $65-68

Call symput ('olddat',olddat);
Call symput ('Newdat',Newdat);
Call symput ('Trans',Trans);
Call symput ('Byvars',Byvars);
Call symput ('Sort',Sort);
Run;

%INC NOBS; /*utility Routine*/
SEPTOFF / Unlimiting Macros

Data _Null_;  
File OUT;  
If &NOBS NE 0 then  
    DO; /*OK to update*/  
    If &SORT = "Yes" then put '%Include SORT;';  
    If &Newdat = then  
        Call symput ('Newdat', "&olddat");  
    Put '%Include UPDATE;';  
END;  
    Else Put 'Note: *** NO Transactions stop skipped';  
RUN;  
%Include OUT;  

The code is packaged within one file, using parameters provided by the user to govern execution. Inside the main routine skeletal code consisting of DATA and PROC steps generated through the processing; code such as the ‘Data _Null_’ Steps and the ‘PUT’ and ‘File’ statements control what files lines of code are generated into and what files will ultimately be called in for execution.

This routine allows the end-user the ability to run systems by supplying the required parameters. Once provided with the parameters the routine will produce the exact run desired. The end-user need not be concerned with anything else. The developer has the flexibility afforded through the full macro environment because all the code inside the main routine is generic and conditional. The generic nature of the code means that the macro variables need only be specified within the parm file (or what ever medium chosen to enter the run parameters).

2. Subroutines - Looking at the figure above, we can see how a main routine may call other routines. This allows standard sets of code to be stored as subroutines and brought in whenever necessary. Storing code in this manner allows for execution of functional parts of a system and contributes to the modularity of the code. The figure below illustrates a subroutine:

Data &Newdat;  
Update &Master  
    &Trans;  
By       &Byvars;  
    .  
    .  
    .  
RUN;
One can certainly see that there really is not much difference between this figure and a macro created in a full macro environment. This reflects the desire to keep code modular as well as functional.

3. Parameter Control - Parameters that control this system can, as stated earlier, be referenced in several different ways. Global parameters can be set and default parameters can be stated. The important point here is that there is no loss of power when programming in this type of environment rather than a main frame environment.

CONCLUSION

While SAS and the full Macro facility are excellent tools for large scale system development, the mini computer user sometimes feels slighted by the lack of functions it may have on its system. SAS Macro in the mini environment has been shown to have the capabilities of SAS Macro on the Mainframe. It can be truly modular in structure, and can permit conditional execution of code and parameter passing without much problem. At the same time, the finely controllable nature of these systems is such that end-users in batch and interactive modes can have great control over their runs without ever dealing with actual code.

ACKNOWLEDGEMENTS

The author wishes to thank Frank Dorsey for his help with editing and Dianna Delemos for her outstanding typing job.

The author may be contacted at:

Fidia Pharmaceutical Corporation
1775 K Street, N.W.
Washington, D.C. 20006
USA
(202) 466-7066

DEC is a registered trademark of the Digital Equipment Corporation, USA

SAS and SAS/AF are registered trademarks of SAS Institute, Cary, N.C., USA