ABSTRACT
As the SAS® System advances to meet user needs and future directions of the software industry, Version 6 of the macro facility makes user-oriented features and enhancements available to users of all operating system environments. Mainframe users will appreciate the new debugging features and macro windows. Microcomputer users will enjoy the autocall facility for establishing source level macro libraries. Minicomputer users will find applications much easier to develop with the full macro facility available to them.

Using specific examples, the paper illustrates the new features and functionality available on all operating systems. Users of the Version 5 System see a preview of what is ahead. Version 6 users see new features and learn new techniques for using the features already available.

INTRODUCTION

The macro facility first became available to SAS software users with Version 8.2. The components of the macro facility at that time are the foundation from which all subsequent versions have evolved. The 8.2.3 macro facility consisted of four components: name style and statement style macros, macro variables, macro functions, and macro statements.

Version 5 of SAS software included enhancements to the Version 82 macro facility. The most significant enhancement added was the autocall facility. Autocall makes it possible to compile macros only on an 'as needed' basis, eliminating the practice that all macros in an application must be compiled before any macro could be executed. The autocall facility also makes invoking macros which emulate SAS functions appear more sophisticated. SAS Institute Inc. began shipping autocall libraries containing utility macros to enhance the current functionality of the macro facility. Unfortunately, the full macro facility is available only to mainframe users under Version 5, only the limited availability of macro variables exists for minicomputer operating systems.

Version 6.03 of the SAS system for personal computers was the beginning of Version 6 macro. The intent of macro for Version 6.03 was that it contain all components of the macro facility available with Version 82.3. In addition to these components, some new features were added. Perhaps the most popular of these features was the improved diagnostic capabilities. MTRACE, a new option for tracing the logic of macro execution was added. With MTRACE in effect, the macro processor displays messages identifying the beginning and the end of macro execution, the values of macro parameters at invocation, the results of %IF conditions, and generally traces the logic a macro follows during execution. The output from the SYMBOLGEN option was also enhanced to display the results of symbolic substitution through all steps of resolution. Macro warnings and errors were enhanced to display more explicit information. Another significant enhancement to this version was the addition of macro windows. These windows can be used to prompt users for information or to display menus.

THE VERSION 6 MACRO FACILITY

With only a few exceptions, the Version 6.06 macro facility offers the full functionality of the Version 5 macro facility in addition to the enhancements of the Version 6.03 macro facility to users of all operating systems.

The Autocall Facility for Source Level Macro Libraries

The autocall facility allows you to store the source code for your macro definitions in a file, associate that file with the SAS session or job, and then execute macros on a 'demand' basis without having to include their source in your session or program. Consider that you may have an application, possibly menu driven, that may execute from one to twenty macros based on user input or other implicit information. Before the autocall facility was available, it was necessary to include the source for all these macros in your program. Using the autocall facility means that only macros that are actually invoked get compiled. This represents a tremendous overhead savings to applications containing macro systems.

Another feature of the autocall facility is that you can actually set up a system of macro libraries. It is common for a site to have its own set of autocall macros, the autocall library that is shipped with the SAS system, and unique macro libraries for individual users. The autocall facility allows you to concatenate these libraries to be seen as a single unit. Macros can be created to emulate functions present in the SAS system. For example, suppose a macro programmer often uses the %GLOBAL statement to initialize a list of variables. Since it is not possible to use a variable list of the form SALES1-SALES10 on the %GLOBAL statement, the user always has to enter the name of each variable. The user can write a macro to generate a list of variables, place this macro in his autocall library, and then execute the macro whenever he needs to generate a variable list. The macro to generate this variable list is shown below:

%Macro varlist(prefix,start,stop);
%Do i = &start to &stop;
&prefix&i
%End;
%Mend varlist;

With the autocall facility in place, this macro can be executed on the %GLOBAL statement as shown:

%Global %varlist(SALES1,10);

The first step in setting up an autocall library is to create a 'member' containing the source for the macro definition. On 'library oriented' systems (OS, CMS, VSE), an autocall library is a PDS or macro containing members. On 'directory based' systems (PC, VMS, AOS, PRIMOS) an autocall library consists of a directory containing files that end in '.sas', and may also contain other files that are irrelevant to the autocall facility.

The 'member' must have the same name as the macro. On 'directory based' machines a 'member' is a file. The macro VARLIST should be put into the file VARLIST.SAS in the directory you are going to use as your autocall library. On 'library oriented' machines, a 'member' is a true member. The macro definition for VARLIST should be put into the member VARLIST of the library that is going to be your autocall library.

The autocall library must then be associated with your SAS session or SAS program. This is done by using the SASAUSOS=
option to specify a single FILEREF, or a quoted string containing one or more FILEREFs. If multiple FILEREFs are specified, they will be searched in the order given. Each FILEREF used must be equated to an autocall library by using a FILENAMES statement or system dependent command. The MAUTOSOURCE option must be in effect in order for the autocall facility to be functional.

Improvements and Enhancements to Macro Diagnostics

The Version 6 SAS system has been enhanced to give clearer indication of errors and warnings that occur in macro processing. One of the most common warnings encountered by macro programmers is the famous

**WARNING 1321 APPARENT SYMBOLIC REFERENCE NOT RESOLVED**

which occurs whenever the macro facility attempts, unsuccessfully, to resolve a pattern of the form &SASname. This warning has been enhanced so that it now gives the name of the macro variable that could not be resolved. In the following example

```sas
%Let code=1; %Put code is &code1;
```

the macro variable CODE1 has not been defined. The SAS system issues the following

**WARNING: APPARENT SYMBOLIC REFERENCE CODE1 NOT RESOLVED.**

Perhaps the second most common warning encountered is

**WARNING 1353 APPARENT MACRO INVOCATION NOT RESOLVED.**

in Version 6 this warning has also been expanded. In the following example

```sas
%Macro test;
 %Put Beginning of macro Test;
 %Mend;

%Test
```

the following message is generated

**WARNING: APPARENT INVOCATION OF MACRO TEST NOT RESOLVED.**

Errors occurring during the process of evaluation have also been enhanced. In Version 5 these errors were often the most difficult to isolate. The errors that occurred when using the %EVAL function explicitly, or in places where an implied %EVAL was done, were

**ERROR 1550: REQUIRED OPERATOR NOT FOUND**

and

**ERROR 1555: CHARACTER OPERAND FOUND WHERE NUMERIC REQUIRED.**

These errors have been replaced in Version 6 by a new error that reveals the cause of the problem. In the following example

```sas
%Let x=%eval(1+x);
```

the error message generated is

**ERROR: A character operand was found in the %EVAL function.**

or %IF condition where a numeric operand was required.

The condition was &state=NC.

Improvements and Enhancements to Debugging Options

The SYMBOLGEN option has been enhanced in two ways. The first enhancement is that it now prints in a much more readable form. The second feature is that it now shows all steps of substitution encountered when performing resolution. Consider the following macro which writes the values of a list of symbolic variables.

```sas
%Macro chklst(prefix, start, stop);
 %Do i=&start %to &stop;
 &&&=prefix&i
 %End;
 %Mend

%Put %chklist(SALES,1,2);
```

The output generated by SYMBOLGEN for the above example is shown below.

**SYMBOLGEN: Macro variable START resolves to 1**

**SYMBOLGEN: Macro variable STOP resolves to 2**

**SYMBOLGEN: & & resolves to &**

**SYMBOLGEN: Macro variable PREFIX resolves to SALES**

**SYMBOLGEN: Macro variable SALES1 resolves to 1000**

**SYMBOLGEN: & & resolves to &**

**SYMBOLGEN: Macro variable PREFIX resolves to SALES**

**SYMBOLGEN: Macro variable I resolves to 2**

**SYMBOLGEN: Macro variable SALES2 resolves to 6000**

A completely new debugging option for Version 6 is the MTRACE option. MTRACE trace the execution of a macro. Output from MTRACE includes the invocation values of macro parameters, the beginning and ending points of the macro’s execution, the results of %IF conditions, and the value of incrementation variables. Consider the following example.

```sas
%MTrace(DATALST): Parameter START has value 1
%MTrace(DATALST): Parameter PREFIX has value BOOK
%MTrace(DATALST): Beginning execution.
%MTrace(DATALST): Parameter START has value &start
%MTrace(DATALST): Parameter PREFIX has value &prefix
%MTrace(DATALST): Parameter LIB has value work
%MTrace(DATALST): Ending execution.
```

%Macro Datalst(prefix, start, stop, lib = work);
 %If %Upcase(&lib) ne WORK %then
 %Do i=&start %to &stop;
 &lib.&prefix.&i
 %End;
 %Else
 %Do i=&start %to &stop;
 work.&prefix.&i
 %End;
%Mend Datalst;

Data %datalst(Book,1,4);

With the MTRACE option in effect, the following is displayed on the SAS log.

**MTRACE(DATALST): Beginning execution.**

**MTRACE(DATALST): Parameter PREFIX has value Book**

**MTRACE(DATALST): Parameter START has value 1**

**MTRACE(DATALST): Parameter STOP has value 4**

**MTRACE(DATALST): Parameter LIB has value work**

**MTRACE(DATALST): Ending execution.**

Likewise, a problem occurring in the %IF statement would also show the erroneous condition. In the following example

```sas
%If &state=NC %then
 %Let st=North Carolina;
%Mend;

%Check(OR)
```

OR, the abbreviation for Oregon, would be interpreted by the macro facility as an operator. The macro facility would generate the following

**ERROR: A character operand was found in the %EVAL function.**

or %IF condition where a numeric operand was required. The condition was &state=NC.**

The improvement and enhancement to debugging options includes:

**SyambolGen**

- Enhanced in two ways:
  - First enhancement: Prints in a much more readable form.
  - Second feature: Shows all steps of substitution encountered.

**MTrace**

- Outputs invocation values of macro parameters.
- Displays execution points, substitution results, and incrementation variable values.

**Examples**

- **Error 1550**:
  - Required operator not found.
- **Error 1555**:
  - Character operand found where numeric required.

- **Macro chklst**
  - Writes the values of a list of symbolic variables.
  - Example:
    ```sas
    %Macro chklst(prefix, start, stop);
    %Do i=&start %to &stop;
    &&&=prefix&i
    %End;
    %Mend
    
    %Put %chklist(SALES,1,2);
    ```

- **MTrace**
  - Outputs execution details.
  - Example:
    ```sas
    %Macro Datalst(prefix, start, stop, lib = work);
    %If %Upcase(&lib) ne WORK %then
    %Do i=&start %to &stop;
    &lib.&prefix.&i
    %End;
    %Else
    %Do i=&start %to &stop;
    work.&prefix.&i
    %End;
    %Mend Datalst;
    
    Data %datalst(Book,1,4);
    ```

- Additional examples demonstrating similar enhancements and improvements are included in the documentation.**

---

**221**
MTRACE(OATALST): %IF condition %upcase(&llb) no WORK is FALSE

MTRACE(OATALST): %IF condition %upcase(&llb) no WORK

MTRACE(DATALST): %00 loop index variable I is now 2; loop will iterate again.

MTRACE(DATALST): %00 loop index variable I is now 3; loop will iterate again.

MTRACE(DATALST): %00 loop index variable I is now 4; loop will iterate again.

MTRACE(DATALST): %00 loop index variable I is now 5; loop will not iterate again.

MTRACE(OATALST): Ending execution.

Macro Windows

A new enhancement available with the Version 6 macro facility is the windowing capability. It is now possible to design and display customized windows from which macro variable values can be input. A macro window is created inside or outside a macro definition by using the %WINDOW statement. The syntax of this statement is

%WINDOW windowname| windowoptions | { field...; }

where

windowname

names the window. Windowname is required.

windowoptions

specify characteristics of the entire window. Options include

COLOR=, IROW=, ICOLUMN=, ROWS=, COLUMNS=, and KEYS=.

field

identifies and describes a macro variable or string to be displayed in the window, its position in the window, and its attributes.

The form of a field definition is

{ row column } macrovariable { fieldlength }'text' { options }

where

row

column

controls the position of the text or macro variable. The following pointer controls are used to move the pointer to the desired position.

#expression

specifies the row within the window.

/ expression

moves the pointer to the first column of the next line.

@expression

specifies the column within the window.

t expression

moves the pointer the number of columns given by the expression.

macrovariable

names a macro variable to receive a value when the screen is displayed.

fieldlength

is an integer defining how many positions in the current row are available for accepting input.

'text'

contains text to be displayed.

options

can include ATTR=, AUTOSKIP, COLOR=, DISPLAY=, PROTECT=, and REQUIRED=.

Once the window has been created using the %WINDOW statement, it can be displayed, either inside or outside a macro, by using the %DISPLAY statement. The syntax of this statement is

%DISPLAY window { NOINPUT }{ BELL };

where

window

names the window to be displayed.

NOINPUT

specifies that you cannot input values into fields.

BELL

rings the terminal's bell when the window is displayed.

The Macro Facility Beyond Version 6.06

The next Version 6 release will complete the conversion process and begin the addition of more features and enhancements. Items to be included in the next version will include command style macros, the RESOLVE function, and the CALL EXECUTE subroutine. Options for allowing explicit control over how memory is managed in the macro facility will also be available. In Version 6.06, the macro facility performs optimal memory handling, that is, it decides how memory would be used most appropriately based on the machine size available. The memory handling options would allow users to choose to reduce memory in exchange for higher execution time, or to reduce execution time and increase memory.

Items under consideration for a future version of SAS software include the ability to store compiled macros, a popular request from users over the past few years, and a window to 'pop up' when a %INPUT statement is executed. The window would allow the user to respond to the %INPUT and would then disappear. Others items include the ability to %DO over a list of values, the ability to read and write files, the ability to generate but not execute SAS source statements when a macro is executed, and a full screen debugger.

CONCLUSION

The macro facility did not undergo major changes from Version 5 to Version 6.03. The development goal for Version 6.03 was to provide a macro facility containing all available features from the 92 version. The major development goal for Version 6.06 was to provide a macro facility that would be compatible to Version 5. With few exceptions, this has been accomplished. Macro programmers across all operating system environments will now have the added features and extensions to the SAS system provided by the macro facility.

The improvement in diagnostics and the windowing capabilities will be a major benefit to mainframe users. The addition of auto-call and significant performance improvements over Version 6.03 will assist programmers on personal computer and UNIX* environments in developing more efficient and useful macro applications. Minicomputer users will have all these benefits as they begin to recognize the extended features and capabilities now available to them. A goal for the future is to provide users in all environments with the tools they need in order to design and maintain systems that will be applicable to future demands.

SAS is the registered trademark of SAS Institute Inc, Cary, NC 27512.
UNIX is a registered trademark of AT&T.