&&&, ;;, and Other Hieroglyphics

Advanced Macro Topics

Chris Yindra, C. Y. Training Associates

INTRODUCTION

SAS® macros are powerful tools that can create reusable code modules to do a variety of programming tasks. A good understanding of macro variables and how macros can generate SAS statements can aid in developing better and more useful macros. This paper will address some more advanced macro features such as the use of multiple ampersands, double semi-colons and required RUN statements.

REVIEW

The macro language allows the SAS user to:

1. Pass system variables to SAS code (i.e. put SYSDATE in a report TITLE).
2. Introduce data values or variable names into SAS code at run time.
3. Use macro variable values to selectively execute SAS code.
4. Pass data values from one SAS step to another (i.e. pass the grand total computed in one step to the next step to calculate the percent to total of each detail item).

Assigning Macro Variables

Values can be assigned to user defined macro variables in several ways. %LET allows for direct assignments. Values can also be directly assigned through user interfaces created with Macro windows. The CALL SYMPUT function allows values to be assigned dynamically within a DATA step or SCL program.

Example 1 (CALL SYMPUT):

Pass the total salary to the global symbol table.

```
DATA _NULL_;
  SET IN.PAYROLL END=FINAL;
  TOTSAL + SALARY
  IF FINAL THEN
   CALL SYMPUT('TOTSAL',LEFT(TOTSAL));
PROC PRINT DATA=IN.PAYROLL;
 TITLE "PAYROLL REPORT";
 TITLE2 "TOTAL SALARY : &TOTSAL";
 VAR SSN SALARY;
RUN;
```

Debugging tools

There are several options which can aid in tracing macros and macro variable values.

OPTIONS:

SOURCE2 - Writes SAS code generated by %INCLUDE to the log.

SYMBOLGEN - Writes the resolved value of macro variables to the log.

MPRINT - Writes SAS statements generated by macro execution to the log.

MLOGIC - Traces macro execution and writes the trace information to the log.

STATEMENTS:

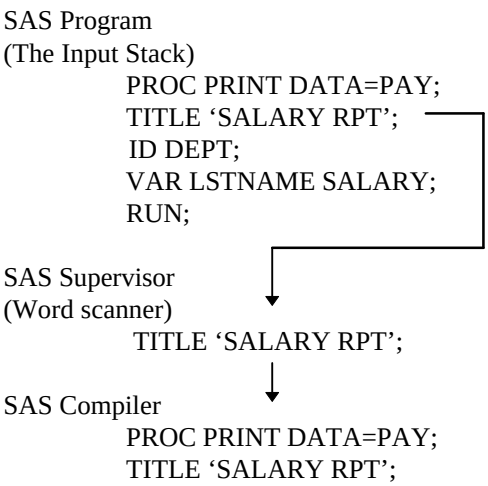
%PUT &variable - Allows the programmer to write macro variable values to the log.

%PUT <_GLOBAL_> <_LOCAL_> <_USER_> writes user defined macro variables to the log.

%PUT <_ALL_> <_AUTOMATIC_> writes both user and system defined macro variables to the log.

Resolving Macro Variables

How the SAS Language Works Without Macros:
The SAS Supervisor takes one word (token) at a time and passes it to the SAS compiler. When a semicolon is reached, the process pauses and the compiler checks the statement for syntax. When the end of the step is reached the SAS step is compiled and executed. The token may be a literal, number, name or special character, usually separated by a blank. Strings in double quotes are analyzed for embedded tokens (macro variables).

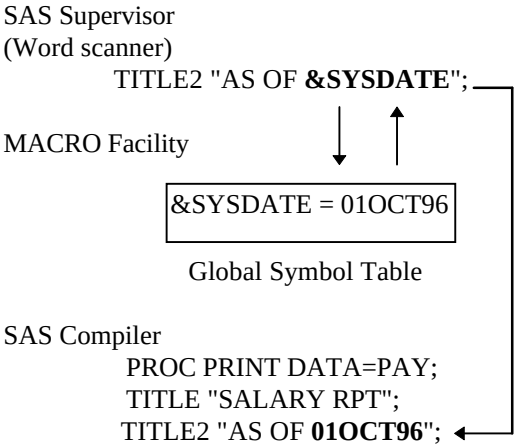


When a step boundary (basically a RUN statement or a new step) is encountered, the previous step is compiled and executed (if no syntax errors were found).

How the SAS Language Works With Macros:
When the SAS Supervisor recognizes a token beginning with a percent sign (%) or an ampersand (&) the token is passed to the Macro Facility instead of the SAS Compiler (quoted strings with embedded tokens must use double quotes). When the Macro Facility receives a token it is checked for validity (the percent sign or ampersand must be followed by a non blank character and able to be interpreted). The Macro Facility will substitute current values of macro variables from the local or global symbol table and pass the token back to the SAS Supervisor for restacking. If the macro variable cannot be resolved, the token will be returned unmodified. When a RUN statement or the beginning of a new step is encountered the SAS statements are executed, after which control is returned to the current processor.

SAS Program
(The Input Stack)

```
PROC PRINT DATA=PAY;
TITLE "SALARY RPT";
TITLE2 "AS OF &SYSDATE";
ID DEPT;
VAR LSTNAME SALARY;
RUN;
```



REQUIRED RUN STATEMENTS

Many times dynamic values from DATA steps or system variables need to be reference within a macro. A run statement can force the execution of a DATA step so that values it passes can be referenced.

Example 2

The following macro can be used to check for the existence of a SAS data set.

```
%MACRO EXISTS;
  %GLOBAL EXISTS;
      DATA _NULL_;
       IF 0 THEN SET &DSN;
      STOP;
      RUN;  <-------------- required!
  %IF &SYSERR=0 %THEN %LET EXISTS =YES;
  %ELSE %LET EXISTS=NO;
%MEND;

%MACRO CHECKIT(DSN);
  %EXISTS;
  %IF &EXISTS=YES %THEN %DO;
      PROC CONTENTS DATA=&DSN;
   %END;
   %ELSE %DO;
     DATA _NULL_;
     FILE PRINT;
     PUT "THE DATASET &DSN DOES NOT
          EXIST";
   %END;
      RUN;
%MEND;

%CHECKIT(CYLIB.XXX);
```

The macro variable &SYSERR is set by the last step processed. A value of 0 means that the previous step was successful. A value greater than 0 means that the previous step generated an error or warning. Because this value is referenced by the next macro statement a RUN statement is required. If the RUN statement had been omitted, the current &SYSERR value would have been determined by step previous to the DATA _NULL_ step. The DATA _NULL_ step would not execute until the next PROC or DATA is encountered by the SAS Supervisor. This is also true if the steps are in separate macros.

Example 3

The following macro will generate a report whether or not there are any observations in a data set.

```
%MACRO TOTOBS(DSNAME);
  %GLOBAL NUMOBS;
     DATA _NULL_;
      IF 0 THEN SET &DSNAME
                  NOBS=HOWMANY;
     CALL SYMPUT
       ('NUMOBS',LEFT(PUT(HOWMANY,8.)));
     STOP;
   RUN;
%MEND TOTOBS;

%MACRO PRINTANY(DSNAME);
  %IF &NUMOBS EQ 0 %THEN %DO;
    DATA _NULL_;
      FILE PRINT;
      PUT //// @20 "DATA SET NAME =
                    &DSNAME"
             / @20 "NO RECORDS TO BE
                    PRINTED";
      RUN;
  %END;
  %ELSE %DO;
      PROC PRINT DATA=&DSNAME;
         VAR LASTNAME SALARY DEPT;
        TITLE "EMPLOYEE LISTING  DATA
               SET: &DSNAME";
      RUN;
   %END;
%MEND PRINTANY;

%TOTOBS(CYLIB.EMPLOY);
%PRINTANY(CYLIB.EMPLOY);
```

The RUN statement in the %TOTOBS macro is required as the next macro %PRINTANY uses the value of NUMOBS passed by the CALL SYMPUT. If the RUN statement had been omitted, the value of NUMOBS would not have been put to the symbol table until after the beginning of execution of %PRINTANY. Note: the %EXISTS macro could be nested within %NUMOBS to initially determine if the data set existed.

MULTIPLE AMPERSANDS

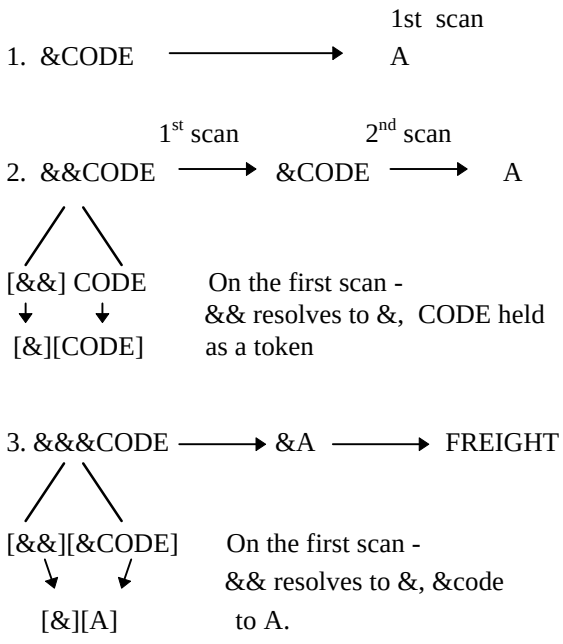
Multiple ampersands can be used to allow the value of a macro variable to become another macro variable reference. The macro variable reference will be rescanned until the macro variable is resolved.

The following demonstrates how macro variables with multiple ampersands are resolved.

Symbol Table

Macro Variable Name	Macro Variable Value
A	FREIGHT
B	PASSENGER
C	SPECIAL
CODE	A

Resolving a macro variable:



Example 4 (&&&):

We would like to run a variety of reports for each department on the data set and include the department name and the department manager in the title. DEPT is on the data set and the department manager is accessed through a format.

```
PROC FORMAT;
   VALUE $DEPTHD
     'GIO' = 'JANE ANDERSON'
     'GPO' = 'STAN JOHNSON'
     'IIO' = 'CAROL SMITH'
     'RIO' = 'STEVE CONARD';
```

We can create macro variables that resolve to the department head values with CALL SYMPUT;

```
DATA _NULL_;
  SET MEANDAT;
   BY DEPT;
  IF FIRST.DEPT THEN
  CALL SYMPUT
      (DEPT,PUT(DEPT,$DEPTHD.));
```

Notice that the CALL SYMPUT macro variable name (DEPT) is not in quotes. The name of the macro variable created will be the **value** of the SAS variable DEPT. Hence the macro variable GIO will resolve to JANE ANDERSON. Now we can refer to the department for the particular report with a direct assignment.

```
%LET DPT=GIO;

PROC PRINT DATA=MEANDAT NOOBS;
  WHERE DEPT EQ "&DPT";
  VAR LASTNAME SEX DEPT;
  TITLE "DEPARTMENT: &DPT   DEPT HEAD:
         &GIO";
```

The global symbol table resolves to:

Macro Variable Name	Macro Variable Value
GIO	JANE ANDERSON
GPO	STAN JOHNSON
IIO	CAROL SMITH
RIO	STEVE CONARD
DPT	GIO

The log looks like:

```
1403  PROC PRINT DATA=MEANDAT NOOBS;
1404   WHERE DEPT EQ "&DPT";
SYMBOLGEN: Macro variable DPT
           resolves to GIO
1405    VAR LASTNAME SEX DEPT;
SYMBOLGEN: Macro variable DPT
           resolves to GIO
SYMBOLGEN: Macro variable GIO
           resolves to JANE ANDERSON
1406   TITLE "DEPARTMENT: &DPT
              DEPT HEAD: &GIO";
```

However we still had to refer to the macro variable &GIO in the title statement. We could use an indirect macro variable reference on DPT to produce the same result without having to hard code &GIO in the title.

```
PROC PRINT DATA=MEANDAT NOOBS;
  WHERE DEPT EQ "&DPT";
  VAR LASTNAME SEX DEPT;
  TITLE "DEPARTMENT: &DPT   DEPT
         HEAD: &&&DPT";
```

On the first scan &&&DPT resolves to &GIO
On the second scan &GIO resolves to JANE ANDERSON.

The log looks like:

```
1445  PROC PRINT DATA=MEANDAT NOOBS;
1446   WHERE DEPT EQ "&DPT";
SYMBOLGEN: Macro variable DPT
           resolves to GIO
1447    VAR LASTNAME SEX DEPT;
SYMBOLGEN: Macro variable DPT
           resolves to GIO
SYMBOLGEN: && resolves to &.
SYMBOLGEN: Macro variable DPT
           resolves to GIO
SYMBOLGEN: Macro variable GIO
           resolves to JANE ANDERSON
1448   TITLE "DEPARTMENT: &DPT
              DEPT HEAD: &&&DPT";
```

We can also create compound macro variable references with numeric endings such as &&VAR&I. Resolving this value will take two scans. For instance we may want to gather an unknown number of variable names into macro

variables with a numeric suffix. Using CALL SYMPUT in a DATA step we can create these macro variables.
i.e.

```
I+1;
IF FIRST.DEPT THEN
  CALL SYMPUT('DPT'||LEFT(I),DEPT);
```

The symbol table might look like:

Macro Variable Name	Macro Variable Value
DPT1	GIO
DPT2	GPO
DPT3	RIO

By concatenating the value of I to the constant part of the macro variable name DPT, each time we encounter a new department on the sorted data set we create a macro variable with a name of DPT1, DPT2 etc. containing the value of the specific department. These macro variables can be referenced later in a macro do loop with &&DPT&I. On the first iteration of the macro loop, the first scan of &&DPT&I will resolve to &DPT1,
and on the second scan &DPT1 will resolve to GIO.

Example 5:
We want to plot all independent variables on a data set with the dependent variable.

```
DATA TEST;
 INPUT YVAR XVARA XVARB XVARC;
CARDS;
1 2 2 4
2 3 2 5
3 3 2 5
4 5 3 7
5 4 3 7
6 6 4 9
7 8 4 10
8 8 5 12
9 8 5 12
;
```

One method would be to hard code each plot:

```
PROC PLOT DATA=TEST;
 PLOT YVAR * XVARA;
PLOT YVAR * XVARB;
RUN;
```

and so on.

We could instead create a macro variable for each independent variable on the data set and a macro variable for the total number of independent variables on the data set. This information can be obtained from PROC CONTENTS and saved to a data set.

```
%MACRO ALLPLOTS(DSNAME,YAXIS);
 PROC CONTENTS DATA = TEST
              MEMTYPE=DATA NOPRINT
              OUT=MYSET(KEEP=NAME
              WHERE=(NAME NE "&YAXIS"));

 DATA _NULL_;
  SET MYSET END=FINAL;
  I+1;
  CALL SYMPUT('VAR'||LEFT(I),NAME);
  IF FINAL THEN
           CALL SYMPUT('NUMVARS',I);
 RUN;
  PROC PLOT DATA = &DSNAME;
%DO I=1 %TO &NUMVARS;
   PLOT YVAR * &&VAR&I;
%END;
RUN;
%MEND;

%ALLPLOTS(TEST,YVAR);
```

On the first iteration of the %DO loop &&VAR&I resolves to &VAR1. On the second scan, &VAR1 resolves to XVARA.

We could use similar methodology to refer to a SAS data set name instead of a SAS variable name.

Example 6:

Write a macro to generate a sample PRINT report for all SAS data sets in a library.

To accomplish this we need to pass the number of data sets and their names to macro variables. We can use PROC DATASETS to get the information.

```
%MACRO SAMPRINT(LIBNAME,OBS);
  PROC DATASETS LIBRARY=&LIBNAME
                        MEMTYPE=DATA;
    CONTENTS
          OUT=WORK.JUNK(KEEP=MEMNAME)
          DATA=_ALL_ NOPRINT;

PROC SORT DATA=JUNK NODUPKEY;
   BY MEMNAME;

DATA _NULL_;
   SET JUNK END=FINAL;
   I+1;
  CALL SYMPUT
        ('SET'||LEFT(I),LEFT(MEMNAME));
 IF FINAL THEN
      CALL SYMPUT('NUM',LEFT(I));
RUN;

%DO J = 1 %TO &NUM;
 PROC PRINT DATA=&LIBNAME..&&SET&J
                (OBS=&OBS);
 TITLE "SAMPLE PRINT FOR DATASET:
         &LIBNAME..&&SET&J";
 RUN;
%END;
%MEND;

%SAMPRINT(WORK,5);
```

WHEN TO USE TWO SEMI COLONS

In many situations we would like to be able to generate a dynamic SAS statement within an macro %DO loop. For instance we may want to create a VAR statement based on macro variable values or create a sub setting WHERE statement based on values in an external file. The %DO loop would look like:

```
PROC PRINT;
 VAR
%DO I = 1 %TO &MAX;
 &&VAR&I;
%END;
RUN;
```

If our symbol table looked like:

Macro Variable Name	Macro Variable Value
VAR1	NAME
VAR2	SSN
VAR3	SALARY
MAX	3

What SAS statements would this macro %DO loop generate?

Result:

```
PROC PRINT;
VAR NAME; SSN; SALARY;
RUN;
```

We would get a syntax error as SAS would interpret each of the variable names as a separate statement; If we took the semi-colon off of the &&VAR&I line, this %DO loop would generate the following SAS statements:

```
PROC PRINT;
VAR NAME SSN SALARY
RUN;
```

However this macro would generate an error saying that the variable RUN is not recognized. We need to close the VAR statement with a semi-colon.

```
PROC PRINT;
 VAR
%DO I = 1 %TO &MAX;
 &&VAR&I
%END ; ;
RUN;
```

Closes the VAR statement

Closes %END

Example 7:

Write a macro that subsets a SAS data set with a WHERE statement based on values from an external file.

The external file (ddname of SELFILE) contains the departments that we would like to pull from a SAS data set and looks like:

```
GIO
RIO
GPO
```

Our macro with a dynamic WHERE statement is:

```
%MACRO PRNTRPT;
   DATA _NULL_;
     INFILE SELFILE ;
     INPUT @1 WHRDEPT $3.;
     I+1;
    CALL SYMPUT
          ('DPT'||LEFT(I),""||LEFT(WHRDEPT)||"");
    CALL SYMPUT('TOTDPTS',I);
   RUN;

    PROC PRINT DATA=MYLIB.EMPFILE;
      VAR SSN DEPT;
      WHERE DEPT IN(
    %DO I = 1 %TO &TOTDPTS;
       &&DPT&I
    %IF &I LT &TOTDPTS %THEN , ;
    %ELSE ) ;
    %END;;
     RUN;
%MEND;

%PRNTRPT;
```

The first DATA _NULL_ step creates the following macro variable values:

Macro Variable Name	Macro Variable Value
DPT1	GIO
DPT2	RIO
DPT3	GPO
TOTDPTS	3

Because the department values are character, we must concatenate quotes around the values for the WHERE statement. Each of the values must also be separated by a comma when using the IN operator (the %THEN condition). We must also end the IN list with a closing parenthesis (the %ELSE condition). We close the WHERE statement with double semi-colons on the %END statement.

Example 8:
Write a macro that will generate a flat file from a SAS data set.

A typical program to accomplish this might look like:

```
DATA _NULL_;
   SET MYLIB.MYSET;
   FILE MYFILE;
   PUT @1 NAME $10. @11 AGE Z3.
         @14 SCORE Z3. @17 TESTDT
           YYMMDD8.;
```

The PUT statement defines the start location, variable name, format and length for each of the variables on the data set. Instead of hard coding these values we could gather this information for any data set at run time for any data set and pass the variable names, lengths, formats, and start locations to macro variables. Once this has been done we can use a dynamic PUT statement to write the value out to an output file.

All of the necessary information can be obtained from PROC CONTENTS (the same information can also be obtained from dictionary tables) and saved to a data set. Because we do not know how many variables will be on the data set we can use the &&var&I methodology to create macro variables for each variable, format, length and start position. We should also generate a report describing the output file.

Sample data set:

```
DATA TEST;
  INPUT @1 NAME  $9. @10 AGE 3.
          @14 SCORE  3. @18 TESTDT
              YYMMDD6.;
  FORMAT TESTDT YYMMDD10.
CARDS;
CHRIS     032 087 960101
JILL      029 092 960202
JAN       041 096 960315
BILL      052 095 960704
ROBIN     022 085 960901
;
```

We will assume some defaults for this macro. All non formatted numeric variables will use a standard length. All character variables will use there storage length. Formatted variables will use there format lengths.

```
%MACRO
OUTFILE(DSNAME,DDNAME,NUMLEN=8);

/* DSNAME IS THE INPUT SAS DATA SET
   DDNAME IS THE OUTPUT FILEREF
   NUMLEN IS THE DEFAULT LENGTH FOR
   NON FORMATTED NUMERIC VARIABLES  */

/* GET FILE INFORMATION */

  PROC CONTENTS DATA=&DSNAME
   OUT=FILE1 (KEEP=LENGTH VARNUM NAME
          TYPE FORMAT FORMATL)  NOPRINT;

  PROC SORT DATA=FILE1; BY VARNUM;
RUN;

/* PUT VARIABLE NAMES, LENGTHS,
     FORMATS AND START LOCATIONS TO
     MACRO VARIABLES */

  DATA FILE1;
    SET FILE1 END=FINAL NOBS=NUMVARS;

    IF _N_ = 1 THEN  STARTPOS = 1;

  /* DETERMINE FORMAT AND LENGTH */
    IF FORMATL > 0 THEN LENGTH=FORMATL;
    ELSE IF TYPE = 1 THEN
        LENGTH=&NUMLEN;

/* DETERMINE START POSITION */
    POS + LENGTH;
    STARTPOS = POS - LENGTH + 1;

    IF FORMAT = ' ' THEN DO;
      IF TYPE = 1 THEN FORMAT='Z';
      IF TYPE = 2 THEN FORMAT='$';
     END;

 /* CREATE MACRO VARIABLES */
     FMT=LEFT(TRIM(FORMAT))||
           TRIM(LEFT(LENGTH))||'.';
     I+1;
     CALL SYMPUT('VAR'||LEFT(I),NAME);
     CALL SYMPUT('FMT'||LEFT(I),FMT);
     CALL SYMPUT
          ('START'||LEFT(I),STARTPOS);
     IF FINAL THEN
       CALL SYMPUT('NUMVARS',NUMVARS);

    RUN;

 /* CREATE FLAT FILE */

   DATA _NULL_;
    SET &DSNAME;
    FILE &DDNAME;
    PUT
    %DO I = 1 %TO &NUMVARS;
      @&&START&I  &&VAR&I &&FMT&I
    %END;;
    RUN;

   /* PRINT FILE DESCRIPTION */

   PROC FORMAT;
    VALUE TYPEFMT 1= 'NUMERIC'
              2= 'CHARACTER';

   PROC PRINT DATA=FILE1 NOOBS SPLIT='*';
    TITLE 'FILE DESCRIPTION';
    VAR NAME TYPE STARTPOS LENGTH
          FORMAT;
    FORMAT TYPE TYPEFMT.;
    LABEL NAME= 'FIELD*NAME'
        TYPE= 'FIELD*TYPE'
        STARTPOS= 'START*POSITION'
        LENGTH='LENGTH'
        FORMAT='FORMAT';
    RUN;
%MEND;

%OUTFILE(TEST,PRINT,NUMLEN=4);
```

This macro produces the following output file:

```
CHRIS    003200871996-01-01
JILL     002900921996-02-02
JAN      004100961996-03-15
BILL     005200951996-07-04
ROBIN    002200851996-09-01
```

and the following report:

FILE DESCRIPTION

FIELD NAME	FIELD TYPE	START POSITION	LENGTH	FORMAT
NAME	CHARACTER	1	9	$
AGE	NUMERIC	10	4	Z
SCORE	NUMERIC	14	4	Z
TESTDT	NUMERIC	18	10	YYMMDD

For simplicity, this macro does not decimal positions in numeric fields. The macro could be modified put the decimal point out to the flat file or to put the numeric out with implied decimal positions.

SUMMARY

1. RUN statements are required as a step boundary when a macro variable value generated in a DATA step must be referenced prior to encountering the next DATA or PROC step.
2. Multiple ampersand referencing can be used when several scans of a macro variable is required. This is particularly useful in creating macro variables with numeric endings. These variables be like ‘arrays’ when used in a macro %DO loop. These ‘array’ values can be used to reference a series of similar values i.e. variable or data set names.
3. Two semi colons are required to close a SAS statement generated by conditional macro statements or a macro %DO loop.

CONCLUSION

The power of macros is in their flexibility and reusability. An understanding of how SAS statements are generated by macro statements, the impact of step boundaries and the use of multiple ampersand references enable a programmer to take better advantage of the macro language.

I can be reached at:

Chris Yindra
C. Y. Training Associates
80 West Mountain Road
Canton Center, CT 06020
(860) 693 - 4297
yindra@ix.netcom.com

REFERENCES

SAS Institute, Inc, SAS Guide to Macro Processing, Version 6, Second Edition, 1990