Paper 95-28

Taking Control of Macro Variables

Dante diTommaso, Fred Hutchinson Cancer Research Center, Seattle, WA

ABSTRACT

Devised for programmers with some SAS® Macro Language experience, this paper reviews fundamental concepts and discusses essential details of dynamic programming. My goal is to guide you towards developing solid macro programming skills through embracing and leveraging SAS features. Nothing (little?) that SAS does should seem mysterious; strength and dexterity lie in understanding.

INTRODUCTION

Dynamic programming can lead to uncontrolled and disorganized macro symbol generation, complicating code development and future maintenance. Except for the simplest programs, unnecessary clutter of any kind compromises value; dynamic programming is no exception and in fact demands greater diligence in maintaining clarity. SAS version 8.2 provides old and new features for controlling macro symbols. You can customize and extend these tools through macro programs. I will review and demonstrate useful features of the SAS macro language, as well as my favorite macros that facilitate efficient dynamic programming and macro development. The reader should have some experience with SAS macro language.

Macros with examples of use are available electronically; see the **CONTACT INFORMATION**.

MACRO REVIEW

- SAS language processes data: numeric and character variables. SAS ***Macro Language*** processes SAS code: text. Embracing this distinction is crucial to macro dexterity.

Macro programming is essential if you generally know all that a program should be able to do but need flexibility in execution.

Program flexibility is achieved through macro variables ("symbols") which store compile- and run-time details. Symbols can contain values, strings, lists, or even a series of instructions or an entire `DATA` step or `PROC` block. To avoid confusion, I will use "symbols" to refer to values, strings and lists. By comparison a compiled macro program ("macro") refers to a block of SAS code and macro instructions bracketed by `%MACRO` and `%MEND` statements. Macro processing adds a compilation layer to standard SAS language compilation and execution. Once submitted to SAS, macros are poised to generate SAS code which SAS subsequently compiles and executes.

Consider a macro that generates daily reports each morning and additional weekly reports on Fridays. Certain instructions are fixed when the macro is first submitted and compiled: daily versus weekly actions, report names (e.g., "`TODAY.RPT`" and "`THISWEEK.RPT`"). Remaining data-driven behavior is the *raison d'être* of macro programming: What are today's data sets? Is today Friday? What is the directory path for today's report(s)?

- Compile-time details are known up front -- at the time of submitting code to SAS. Run-time details are situation- or data-driven. This is something of an oversimplification given the two compilation stages (macro versus code), but a fairly safe one for developing a basic understanding.

Macro symbols follow SAS variable naming rules and can store from zero up to 32k of text (potential SAS code). Macros store any combination of SAS and macro language expressions.

SYMBOL TABLES AND SCOPE

SAS categorizes symbols along two related dimensions based on declaration and scope.

AUTOMATIC VERSUS USER-DEFINED

Macro symbols are either ***automatic*** or ***user-defined***. SAS creates automatic symbols during initialization (the single exception is `&SYSPBUFF` when including `/PARMBUFF` in a macro definition). Automatic symbols provide information on the current date, user, operating environment, SAS version, *et cetera*. The programmer subsequently creates user-defined symbols, either global or local.

GLOBAL VERSUS LOCAL

SAS stores symbols in ***symbol tables*** ("tables"). ***Global*** symbols are available for the duration of a SAS session; SAS stores them in a single ***global symbol table***. Automatic symbols (again with the single exception noted above) are global.

User-defined symbols are global if created outside any macro program or created inside but first declared a global symbol (see **%Global and %Local** below).

If a programmer first declares or defines a symbol inside a macro, SAS creates a ***local symbol table*** that expires when the macro ends. Local symbols are only available for the duration of the enclosing macro.

If a symbol already exists in a table outside the current macro, SAS does not store a new instance in the local table unless first declared a local symbol (see **%Global and %Local** below). Multiple nested tables are possible when macros are called within macros.

Symbol table is equivalent to symbol ***scope***. Symbols in the global table have ***global scope***; those in a local table have ***local scope***. Understanding and managing symbol scope is essential for developing clean, clear and efficient dynamic programs.

- Symbols in the immediate table are available during macro execution.
- Symbols in outer local tables and the global table *that have unique names* are also available during macro execution.
- Symbols in outer local tables and the global table *that have the same name* as a symbol in the immediate local table are not available; the local table dominates (***name hiding***).

MACRO SYMBOL RESOLUTION

The SAS ***Word Scanner*** parses code into recognized pieces or ***tokens***. Tokens are string literals (`'HELLO'`), variable names, numeric values or special characters `(+-/.;&%)`. The scanner typically passes tokens to the ***Data Step Compiler*** or a particular SAS ***Procedure*** which check program syntax. Two special characters (macro ***triggers*** `%` and `&`) followed by nonblank characters prompt the word scanner to first divert code to the ***Macro Processor*** which attempts to resolve symbols into SAS language. For a detailed discussion of how SAS resolves symbol references see "Using Macro Variable" (1, Macro Variables chapter).

As the macro compiler encounters symbols, it attempts to resolve them using the most local table first, then working outward to the global table. If SAS first encounters `%MACRO`, it checks and

compiles macro logic (e.g., matching %IF and %END statements), and simply stores other code as text (e.g., SAS code or symbol references). This compilation of a macro definition ends upon reaching %MEND.

SAS FEATURES

SAS makes it quite easy for programmers to control symbols and tables. Never wonder where a symbol definition will end up, or assume it will end up where you intend. Determine the appropriate approach and specify it in your code.

%GLOBAL AND %LOCAL

Explicit declarations of global and local macro symbols not only make macro programming more intelligible, they eliminate problems that are otherwise difficult to discover. Consider the following code fragment from a common macro that returns the number of observations in a data set:

```
%MACRO NUMOBS (DS);
 %GLOBAL NUMOBS;
... * ACCESS NUMBER OF OBS & *;
    * STORE IN &NUMOBS)       *; ...
%MEND NUMOBS;

%NUMOBS(DATA01)
```

Without the %GLOBAL declaration and assuming &NUMOBS does not exist in the global table the remaining code would create &NUMOBS in the local table, store a value, then erase the local table upon reaching %MEND, trashing the desired information.

Another example demonstrates the importance of using %LOCAL declarations, arguably the most underused macro statement:

```
%MACRO SUBROUTINE (VARS);
 %LOCAL LOOPS;
 %LET LOOPS = 5;      * DECLARATION [2] *;
 ... * LOOP THROUGH DATA *;
%MEND;

%LET LOOPS = 25;      * DECLARATION [1] *;
...
%SUBROUTINE(&VNAMES)
```

Without the explicit %LOCAL declaration inside the definition of %SUBROUTINE(), declaration [2] would update the global symbol &LOOPS that declaration [1] established earlier. This would almost certainly put the program in an illegal state, an egregious bug that would hide until again accessing the global symbol &LOOPS.

- Imagine relying on another programmer's library of macro utilities and having your global symbols mysteriously change due to name conflicts like the one demonstrated above. I recommend rigorous and enthusiastic use of %LOCAL declarations.

%PUT AND %SYMDEL

%PUT and %SYMDEL statements are extremely useful for managing symbols. Combine %PUT with one of the following keywords to dump the corresponding macro variables to the SAS log: _ALL_, _AUTOMATIC_, _GLOBAL_, _LOCAL_, _USER_ (user-defined symbols comprise both global and local symbols).

Use %SYMDEL to specify a list of variables to remove from the global table. %SYMDEL does not accept SAS variable list syntax or macro expressions that resolve to macro variable names.

Statement [1] below prints user-defined symbols to the log, while [2] removes &LOOPS and &VNAMES from the global table:

```
%PUT _USER_;                    /* [1] */
%SYMDEL LOOPS VNAMES;           /* [2] */
```

Both are valuable during program development. %SYMDEL allows the programmer to carefully maintain the global table, minimizing name conflicts and the problems they cause.

FAVORITE DEVELOPMENT MACROS

Macro programming involves careful management of global and local symbols. I use several macros to facilitate common symbol management tasks.

These macros, with examples of use, are available in the appendix or electronically (see the **CONTACT INFORMATION**).

MANAGE THE GLOBAL SYMBOL TABLE

%GLOBVARS;

%PUT _*keyword*_; (described above) writes an unordered and unformatted list of symbols to the log. Initially useful, I find the list difficult to read. While developing programs that include macro language, I am most interested in the state of the global symbols (more so than automatic or even local symbols). In particular, I am interested in when symbols expire, at which point they deserve %SYMDEL. Maintaining a clean global table makes it much easier to later update the middle of a complex program. Again, any unnecessary clutter is a liability.

I use %GLOBVARS to print a formatted and alphabetical report of the symbols from the global table. The report includes a count of symbols, a simple measure of cleanliness.

%CHKGVAR *symbol-name*;

I tend to reuse symbol names for common functions (counters, indices, etc.) between and within programs. %CHKGVAR checks the global symbol table to remind me if a particular symbol is already in use and, if so, reports its current value.

%DELGVARS;

The next time you want to quickly start from scratch with the proverbial clean global symbol slate, type just 9 characters (the semi-colon is optional). %DELGVARS removes user-defined global symbols using %SYMDEL.

%FRESHSTART;

I have extended the clean slate concept beyond global symbols with a macro that comes as close as currently possible to re-initialized SAS' interactive development environment: clear LIBNAMEs, global symbols and temporary data sets & formats; restore original system options (which may require site customization); and finish by launching the original session AUTOEXEC file.

Quickly switch between programs during development without shutting down SAS. Now if we could only programmatically clear compiled temporary macros, although doing so from a temporary macro is as recursively inconceivable as most time travel tales! *Contact the author for the code.*

MANAGE DYNAMIC VARIABLE LISTS

The following macros are designed to work in-line – their definitions include only macro language. Invoking these macros generates no SAS code other than inserting the value you ask for right where you specify. The required parameters for each include character-delimited (space-delimited by default) lists of variable names (or similar).

%QUICKCNT(*macro-variable-list*);

Given a list of variable names, %QUICKCNT returns the number of names in the list. For example, if the symbol table contains

&MNTHS (with value JAN FEB MAR) you can use %QUICKCNT to declare a dynamic array used to calculate monthly totals:

```
ARRAY TOTALS[%QUICKCNT(&MNTHS)]
      _TEMPORARY_ (0*%QUICKCNT(&MNTHS));
```

%COMMVAR(*var-list-1, varlist-2*);
%COMMVAR() returns the list of variables common to both *var-list-1* and *var-list-2*. Merging and appending data sets in dynamic programs come to mind.

%DIFFVAR(*var-list-1, varlist-2*);
Complementing %COMMVAR(), %DIFFVAR() returns the list of variables in either list but not the other: the first but not the second, or the second but not the first. I do not need this often, but when I do the convenience is thrilling. *Contact the author for the code.*

POLITE, CONSIDERATE MACROS

Minimizing macro ***side effects*** is somewhat off topic, but in my mind an essential component of designing high quality macros. I consider a side effect any change in the SAS system environment that the macro user does not anticipated, notice, and perhaps most importantly appreciate. For example, a macro may change titles, footnotes or system options such as LINESIZE and PAGESIZE to generate a desired report. Rude macros do so arrogantly, without reporting such changes and without later restoring settings to their original values. I use two pairs of macros to save and restore TITLEs /FOOTNOTEs and system options.

%SAVETTLFN and %RSTRTTLFN
Invoked without parameter, %SAVETTLFN stores SAS system titles and footnotes in a temporary data set. After doing so a macro can freely make necessary changes. At the end of macro processing, simply %RSTRTTLFN to restore original titles and footnotes.

Including these statements in macros allows you to label macro output appropriately without irritating programmers that use your code. *Contact the author for the code.*

%SAVEOPT(*system-options*) and %RSTROPT(*system-options*)
In a parallel fashion %SAVEOPT(*system-options*) sets aside current settings for listed system options. Change page layout, form delimiter, missing values, etc., with absolute impunity. Once macro processing is complete, %RSTROPT(*system-options*) restores all settings or just those listed.

Given human nature, being polite and considerate is rarely so easy. Thus rudeness and the irritation of mysterious environment changes is inexcusable. End of lecture. *Contact the author for the code.*

CONCLUSION

Understanding and managing symbols is essential for developing clear, reliable and maintainable dynamic programs and macro libraries. Making use of SAS features for working with symbol tables and developing generic helper macros saves development time and makes dynamic programming efficient and downright fun. Explore, experiment and enjoy!

REFERENCES

1. SAS Institute Inc., SAS OnlineDoc®, Version 8, Cary, NC: SAS Institute Inc., 1999 (*SAS Macro Language: Reference* section)

2. Art Carpenter, Carpenter's Complete Guide to the SAS Macro Language, Cary, NC: SAS Institute, Inc., 1998, 242pp.

CONTACT INFORMATION

I value and encourage your comments and questions:

Dante diTommaso
Fred Hutchinson Cancer Research Center
1100 Fairview Ave N, MW-500
Seattle, Washington 98109
Phone: (206) 667-6470
Fax: (206) 667-4812
Email: dante@scharp.org
Files: http://dantegd.home.mindspring.com/sas/

TRADEMARK CITATION

```
*+-------------------------------------------------------------+*;
*|                      A P P E N D I X                        |*;
*---                                                          ---*;
*| [GLOBVARS]  NO PARAMETERS: REPORT GLOBAL SYMBOLS, THEIR     |*;
*|             VALUES, AND A COUNT OF GLOBAL SYMBOLS           |*;
*| PROGRAMMER: DANTE DITOMMASO (SUGI 28, PAPER 95, MARCH 2003) |*;
*+-------------------------------------------------------------+;
%MACRO GLOBVARS;
  %LOCAL MAXNAME MAXVAL NGLOBVAR;

  DATA GLOBVARS (KEEP=NAME VALUE);
    SET SASHELP.VMACRO (KEEP=SCOPE NAME VALUE
                        WHERE=(UPCASE(SCOPE) EQ 'GLOBAL'))
                       END=NOMORE;
    RETAIN MAXNAME 10 MAXVAL 20;

    IF (LENGTH(NAME) GT MAXNAME) THEN MAXNAME = LENGTH(NAME);
    IF (LENGTH(VALUE) GT MAXVAL) THEN MAXVAL = LENGTH(VALUE);

    IF (NOMORE) THEN DO;
      CALL SYMPUT('MAXNAME', TRIM(LEFT(PUT(MAXNAME, 8.))));
      CALL SYMPUT('MAXVAL', TRIM(LEFT(PUT(MAXVAL, 8.))));
    END;

  PROC SQL NOPRINT;
    SELECT (NOBS - DELOBS) INTO :NGLOBVAR
    FROM DICTIONARY.TABLES
    WHERE UPCASE(LIBNAME) = 'WORK' AND
          UPCASE(MEMNAME) = 'GLOBVARS';

  %IF &NGLOBVAR. > 0 %THEN %DO;
    %LET NGLOBVAR = %TRIM(%LEFT(&NGLOBVAR.));
    PROC REPORT DATA = GLOBVARS SPLIT = "|" CENTER HEADLINE NOWD;
      COLUMN NAME VALUE;
      DEFINE NAME  / ORDER   FORMAT=$&MAXNAME.. WIDTH=&MAXNAME.
                             SPACING=2 LEFT
                             "SYMBOL|NAME|(N=&NGLOBVAR.)";
      DEFINE VALUE / DISPLAY FORMAT=$&MAXVAL.. WIDTH=&MAXVAL.
                             SPACING=2 LEFT
                             'SYMBOL|VALUE| ';
  %END;

  PROC DATASETS LIB=WORK MEMTYPE=DATA NOLIST NODETAILS;
    DELETE GLOBVARS;
  QUIT;
%MEND GLOBVARS;

*-------*; %let ONE = a global symbol;
* USAGE *; %let TWO = another global symbol;
*-------*; %globvars
```

```
*+-------------------------------------------------------------+*;
*|                      A P P E N D I X (continued)            |*;
*---                                                          ---*;
*| [CHKGVARS] CHECK GLOBAL SYMBOL DATA SET AND REPORT WHETHER  |*;
*|            OR NOT A MACRO SYMBOL ALREADY EXISTS.            |*;
*| PARAMETER: <VN> GLOBAL SYMBOL NAME TO CHECK AGAINST DATA SET|*;
*| PROGRAMMER: DANTE DITOMMASO (SUGI 28, PAPER 95, MARCH 2003) |*;
*+-------------------------------------------------------------+;
%MACRO CHKGVARS(VN);
  DATA _NULL_;
    SET SASHELP.VMACRO (KEEP=SCOPE NAME VALUE
                        WHERE=(UPCASE(SCOPE)='GLOBAL')) END=NOMORE;
    RETAIN MATCH 0;
    IF UPCASE(NAME) = UPCASE("&VN.") THEN MATCH = 1;
    IF NOMORE AND MATCH THEN DO;
      PUT '+' 24*'-' '>';
      PUT "+ [CHKGVARS] GLOBAL SYMBOL <%UPCASE(&VN.)> EXISTS: " VALUE;
      PUT '+' 24*'-';
    END;
    ELSE IF NOMORE AND ^MATCH THEN DO;
      PUT '+' 24*'-' '|';
      PUT "+ [CHKGVARS] GLOBAL SYMBOL <%UPCASE(&VN.)> DOES NOT EXIST.";
      PUT '+' 24*'-';
    END;
  RUN;
%MEND CHKGVARS;

*-------*;
* USAGE *; %let PI = 3.1416; %chkgvars(PI)
*-------*; %chkgvars(THREE)



*+-------------------------------------------------------------+*;
*|                      A P P E N D I X (continued)            |*;
*---                                                          ---*;
*| [DELGVARS]  NO PARAMETERS, DELETE !!ALL!! GLOBAL SYMBOLS    |*;
*| PROGRAMMER: DANTE DITOMMASO (SUGI 28, PAPER 95, MARCH 2003) |*;
*+-------------------------------------------------------------+;
%MACRO DELGVARS;
  DATA DELGVARSTEMP;
    SET SASHELP.VMACRO (KEEP=SCOPE NAME
                        WHERE=(SCOPE='GLOBAL'));
  DATA _NULL_;
    SET DELGVARSTEMP;
    CALL EXECUTE('%SYMDEL ' !! TRIM(LEFT(NAME)) !! ';');

  PROC DATASETS LIBRARY=WORK MEMTYPE=DATA NOLIST NODETAILS;
    DELETE DELGVARSTEMP;
  QUIT;
%MEND DELGVARS;

*-------*; %let PI = 3.1416; %globvars
* USAGE *; %delgvars
*-------*; %globvars
```

```
*+-------------------------------------------------------------+*;
*|                      A P P E N D I X (continued)            |*;
*---                                                          ---*;
*| [QUICKCNT]  IN-LINE UTILITY TO COUNT WORDS IN A CHARACTER   |*;
*|             DELIMITED STRING (SPACE-DELIMITED BY DEFAULT)   |*;
*| PARAMETERS: <QSTR> REQUIRED -- DELIMITED STRING OF WORDS    |*;
*|             NB: QUOTE STRING AS NEC ON MACRO CALL, (EG, IF  |*;
*|                 <QSTR> HAS SPECIAL CHARS ( ;, ) USE %STR)   |*;
*|             <QDLM> OPTIONAL LIST OF DELIMS (SPACE DEFAULT)  |*;
*| PROGRAMMER: DANTE DITOMMASO (SUGI 28, PAPER 95, MARCH 2003) |*;
*+-------------------------------------------------------------+;
%MACRO QUICKCNT(QSTR, QDLM=%STR( ));
  %LOCAL QCN;

  %LET QCN = 0;
  %DO %WHILE (%QSCAN(&QSTR., %EVAL(&QCN.+1), &QDLM.) ^= );
    %LET QCN = %EVAL(&QCN. + 1);
  %END;
  &QCN.
%MEND QUICKCNT;

*-------*;
* USAGE *; %let VARS = %str(var1,var2^var3#var4 var5);
*-------*; %put _%QUICKCNT(&VARS., QDLM=%STR( ^#,))_;



*+-------------------------------------------------------------+*;
*|                      A P P E N D I X (continued)            |*;
*---                                                          ---*;
*| [COMVAR]  IN-LINE UTILITY RETURNS LIST OF WORDS COMMON TO   |*;
*|           THE TWO LISTS PASSED IN AS DELIMITED STRINGS      |*;
*|           NB: !!NOT!! CASE SENSITIVE                        |*;
*| PARAMETERS: <CLST1> REQUIRED -- DELIMITED STRING OF WORDS   |*;
*|             <CLST2> REQUIRED -- DELIMITED STRING OF WORDS   |*;
*|             NB: QUOTE STRING AS NEC ON MACRO CALL, (EG, IF  |*;
*|                 <CLST1> HAS SPECIAL CHARS ( ;, ) USE %STR)  |*;
*|             <CDLM> OPTIONAL LIST OF DELIMS (SPACE DEFAULT)  |*;
*| PROGRAMMER: DANTE DITOMMASO (SUGI 28, PAPER 95, MARCH 2003) |*;
*+-------------------------------------------------------------+;
%MACRO COMVAR(CLST1, CLST2, CDLM=%STR( ));
  %LOCAL COMVAR;

  %LET COMVAR =;
  %DO CV1 = 1 %TO %QUICKCNT(&CLST1., QDLM=&CDLM.);
    %DO CV2 = 1 %TO %QUICKCNT(&CLST2., QDLM=&CDLM.);
      %IF %UPCASE(%SYSFUNC( scan(&CLST1., &CV1., &CDLM.) )) EQ
          %UPCASE(%SYSFUNC( scan(&CLST2., &CV2., &CDLM.) )) %THEN
          %LET COMVAR = &COMVAR. %UPCASE(%SYSFUNC(
                        scan(&CLST1., &CV1., &CDLM.) ));
    %END;
  %END;
  &COMVAR.
%MEND COMVAR;

*-------*; %let L1 = V1 VARiable2@V3 V5^V6 v7;
* USAGE *; %let L2 = %STR(VARIABLE2!V5 V6;V7 V8);
*-------*; %put _%COMVAR(&L1., &L2., CDLM=%STR( !@;^))_;
```