Paper 90-26

The Ultimate SAS® Macro
(Make SAS do all the work!)

Mike Tangedal, US Bank, St. Paul, MN

ABSTRACT

With tongue firmly entrenched in cheek, I present the design for the ultimate SAS macro. Actually, this is a compilation of techniques designed to maximize efficiency for both a SAS macro as well as the base SAS code being driven. This includes awareness of the automatic macro variables available as well as techniques for ensuring the best possible parameters passed to the macro. Traversing the Proc Contents of the input data set in order to initialize macro variables to be used later produces valuable results in minimal time. Explanation of a macro variable 'array' from the list of input data set variables is also provided. Cleaning parameters passed to the macro (especially long text stings) is also addressed. Finally, once the user fully tests their modified version of the ultimate SAS macro, it should be placed in an Autocall library.

INTRODUCTION
(Macros keep your brain in shape)

Monotony in the professional programming community breeds contempt. Thankfully we SAS programmers have a great tool available to eradicate needlessly repetitive code. The SAS macro language can be used to implement maximum efficiency of all your SAS programming tasks. Alas, this efficiency does have a price.

The SAS macro language is a cryptic little entity not nearly as inherently easy to understand as the base SAS language. Worse yet, code constructs are sprinkled about amidst the regular SAS code – revealed through only a telltale ampersand or percentage sign.

Through sheer force of will and driven by the desire dwelling within all SAS programmers to develop the most efficient program imaginable using the least amount of effort (i.e. work), you too can learn to love this dirty little facet of SAS known as the macro language. This ultimate SAS macro can help.

PREPARATION
(Insight and Foresight are better than Hindsight)

Before I get to the good stuff, I must endeavor to inform of the volume of work required before implementing any ultimate code. A fully functional data warehouse is not required; however, many of the modular constructs need to be in place. At the very minimum, directories should be in place noting areas for source data, SAS programs, lookup tables, SAS format libraries, and stored data sets to name a few. Also highly useful are data definition tables with summarization codes for any input files to be converted into SAS data sets. A SAS macro does not impart any new processing logic of course, but simplifies and streamlines the existing logic. Therefore, it is imperative that this well-tested logic is in place before attempting to employ an easy-to-use ultimate SAS macro.

DOCUMENTATION
(Every ampersand is sacred)

In light of SAS macro statements appearing enticingly close to base SAS statements, proper documentation for the ultimate macro is essential. Blocks of macro code should be separated and documented. In addition, references to macro variables should be explicitly noted lest you presume those maintaining your macro are as gifted at spotting ampersands as those less versed in the way of the macro. Other than the normal rhetoric on producing well-documented code (fill your macro with clear, concise explanations!), the main pointer to be made here is the type of comment statement which works best.

Within a SAS macro is an additional commenting construct – that being text followed by '%*'. However, just because this is available does not make it your best choice. Common characters found within explanatory written text (i.e. parentheses, single quotes, and commas) wreak havoc when presented as a comment in either the macro comment structure or the base SAS comment structure of '*' preceding text. They key here is to remember you will not get into trouble using the old standby of '/* comment */'.

(MVS users should note the '/*' should not occur in the first two columns.) Any text typed between this structure (other than another '/*' or '*/' of course) is treated as a comment. You have enough to worry about building the ultimate SAS macro without having to scan your comments for abbreviations.

INVOCATION
(Know thy parameters)

As you might imagine, starting up an ultimate SAS macro is not done without a bit of flair. Rest assured this formidable block defining the macro is not for show. Declaration and documentation of keyword parameters sets the tone for the entire macro as well as providing some insurance against improper invocation. Action should be taken against haphazard use of this ultimate SAS macro. By the way, this macro should be invoked outside the confines of a data step.

```
  /************************************/
  /*  - The ULTIMATE Macro -          */
  /*                                  */
  /* Macro Name:  Ultimate            */
  /* Mike Tangedal    September 2000  */
  /*                                  */
  /* This code will ultimately save   */
  /* the SAS programmer a lot of      */
  /* repetitive work with its         */
  /* emphasis on efficiency.          */
  /*                                  */
  /* Example invocation:              */
  /* %ultimate(data=mydata,out=out1)  */
  /************************************/

%macro ultimate(
   data=_LAST_,     /* SAS input data  */
   cntl=default,    /* SAS control file*/
    fldlist=%str(all),
                    /* dset field list */
   out=&sysjobid,   /* SAS output file*/
   debug=N,         /* error checking */
   delete=Y         /* clear work area*/
);
```

Note that default values are set for each parameter so that even the most basic macro invocation of '%ultimate()' could produce valid results. Note the use of the '%str' function with one of the parameters just in case the user enters single quotes, dashes, or slashes into this value. Also note the use of the automatic macro variable 'sysjobid' to define the default value of the output data set. If no value is given, the output data set will be assigned the name of the user ID invoking the macro.

MACRO VARIABLE INITIALIZATION
(You know more than you think)

What good is invoking the ultimate SAS macro without explicit parameters? The answer to this question depends much on the planning of the system environment where the macro was invoked and an understanding of what other information is available to the macro other than what is provided from the parameters.

Regardless of system planning, a host of automatic macro variables are available in any macro. These can give insight into how and where the macro was invoked. Below is a list of the more useful automatic macro variables.

Variable	Contains
SYSDATE	character string containing current date
SYSTIME	character string containing current time
SYSJOBID	name of current batch job or user ID
SYSFILRC	return code set by FILENAME statement
SYSLIBRC	return code set by LIBNAME statement
SYSLAST	name of most recent SAS data set w/ libname

In addition to these automatic macro variables, even more power is available through the use of the '%sysfunc' macro function. Tasks such as verifying the existence of files and library names, retrieving file information, and returning information about a directory are available. Full documentation of this function can be found in Appendix 3 of the SAS Macro Language Reference, version 6.

The variable initialization section of any SAS macro should begin with statements declaring whether macro variables are to be local just within the context of the macro or to be used outside the macro. This is done using the '%LOCAL' statement and the '%GLOBAL' statement

```
  /* Note all created macro variables
     as local or global               */
%local j;  /* feel free to add */
%global unique numobs;

  /* set default input dataset to last
     dataset used before macro was
     called                           */
%if %upcase(&data)=_LAST_ %then
  %let data=&syslast;
  /* if input dataset is missing,
     then display error message
     and exit                         */
%if %upcase(&data)=_NULL_ %then %do
  %put Dataset &data not found;
```

```
  %goto EXIT;
  %end;

  /* set default or missing output
     dataset parameter to userid
     prefixed by character 'U'    */
%if &out=&sysjobid or &out= %then
  %let out=U&sysjobid;

  /* if debugging macro, turn on
     macro debug system options */
%if &debug=Y %then %do;
  options mprint mlogic symbolgen;
  %end;
```

The input file name passed to the macro can provide more information than simply the link to the source data. With proper planning, the library name from which it came and the name of the data set itself can contain information used to reference lookup tables and execute one of a variety of different analysis procedures. For example, given the macro invocation of '%ultimate(data=ytd00.clientA)' decisions can be made as to which logic constructs to run for this year-to-date 2000 file. Proper lookup tables and reports for 'clientA' can be referenced. Use of the macro function '%substr' can provide just the right nugget of information in order to execute just the right SAS code.

SUMMARIZATION
(Your input file holds hidden treasures)

Proc Contents is a wonderful SAS tool in that it provides so much information at such a cheap price of minimal processing time. It should be your duty to take advantage of all the information automatically stored with your input data set! Here is some code on how to do just that. Note that this block of code executes conditionally based on the value of the 'cntl' parameter. The default value presumes no external data definition table exists for this data set. Proc Contents provides many table definition elements but others may be needed so this option exists. Programmer discretion will determine how an external data definition table is to be utilized.

```
%if &cntl=default %then %do;
  /* get access to the data definition
     table through Proc Contents     */
  proc contents data=&data out=memlist
    nolist; run;

  /* read output of Proc Contents */
  data _null_;
    set memlist
      (keep = crdate libname memname
              modate nobs name)
    end=last;

  /* temp storage for dataset name*/
  length dsname $8;

  /* note number of variables */
  if last then do;
    call symput('numvars', put(j,3.));
    /* note dataset information */
    call symput('crdate',
      put(crdate,date9.));
    call symput('modate',
      put(modate,date9.));
    call symput('libname', libname);
    call symput('memname', memname);
    call symput('numobs',
      put(nobs,comma10.));
    end;

  /* impose dataset list variable
     selection criteria          */
  %if &fldlist ^= or &fldlist ^=all
    %then %do;
    if index(
    "&all",
    name)=0 then delete;

  /* update temp variable name */
  j + 1;
  dsname='ds' || left(put(j,3.));

  /* create macro variable for each
     variable in input data set   */
  call symput(dsname,name);
 run;

  /* list memlist if debug set to Y*/
%if &debug=Y %then %do;
  title2 'Input dataset contents';
  proc print data=memlist; run;
  %end;

  /* dataset clean up */
  proc datasets nolist;
    delete memlist; run;
  %end;  /* &cntl=default */

  /* clean up all work files if
     delete parameter set to Y  */
%if %upcase(&delete)=y %then %do;
  proc datasets nolist;
    delete _all;
  run;
  %end;

%EXIT;  /* error processing */
%mend;  /* end of ultimate macro */
```

What remains from the execution of this block of code are many global macro variables providing all sorts of

useful information about your source data. The 'call symput' command within a data step creates a global macro variable that can be referenced throughout the program. Data set information such as the number of observations and the creation date are available as are the list of selected variables from the input data set.

As to the purpose behind creating a range of cryptic global macro variables, that can best be demonstrated by the following block of code.

```
/* The following is analogous
   to a macro array containing the
   list of all variables from the
   input data set               */
%do j = 1 %to &numvars;
  /* present each variable */
  &&ds&j  /* just for show */
  %end;
```

You know I couldn't title a paper "The Ultimate SAS Macro" and not include at least one example of a double ampersand reference. It's within context as well! This easily accessible list of data set variables is potentially useful for a myriad of SAS procedures. For an example of a triple ampersand reference (!), keep reading.

MULTIPLE WORD PARAMETERS
(Why stop at double ampersands?)

The parameter 'fldlist' from the ultimate macro initiation statement is utilized to specify a list of variables to be utilized in your subsequent programming. This common option for data step processing can be made more efficient through a few SAS macro language constructs. The first is the use of the triple ampersand.

Normally a SAS macro completely infected with ampersands indicates nothing more than a grandstanding programmer. However, I did find one valid use of the legendary triple ampersand reference to a macro variable. It is used to prevent duplicates of macro variables containing long text strings.

A macro variable defined outside a macro and then passed as a parameter causes the macro processor to store two values. To avoid this, a triple ampersand reference to the outside macro variable can be used within the macro. The macro processor resolves the parameter to the outside macro variable and the duplicate storage issue is avoided. Here is a quick example.

```
  /* Triple ampersand example */

  /* declare the outside macro var*/
%let outside=%str(This is many words);

  /* macro to reference outside string*/
%macro triple(inside);

  &&&inside
  /* first pass resolves to &outside */
  /* next pass resolves to contents
     of outside macro variable       */
%mend;
```

The result of execution of the following command: '%triple(outside)' would be the contents of the outside macro variable (`this is many words`). For safety sake, the '%quote' function should be used on the results of this macro call to avoid misinterpreting any slashes or dashes in the text string. A typical macro call might be as follows: '%let strng=%quote(%triple(outside));'. Macro variables within the 'triple' macro are referenced only within the confines of the macro.

This is all well and good if you have provided precautions of defining macro variables outside the macro. If you want to clean up a field list within the macro, here is an example showing how to eliminate duplicate words in a list. You cannot always be sure users entering field lists are entering all unique items. This code could be placed within the ultimate macro code since the 'fldlist' parameter is the same here.

```
  /* make string of all unique words */
%macro unique(fldlist);
  %global unique;
  %local j k count cnt word;
  %let cnt = 0;
  %let unique =;
  %let count=0;

  /* count number of separate words in
     a string */
  %if &fldlist ne %then %do;
    %let word= %scan(&fldlist,1);
    %do %while (&word ne);
      %let cnt = %eval(&cnt+1);
      %let word= %scan(&fldlist,&cnt+1);
      %end;
    %end;

  /* build list of unique words */
  %do j=1 %to &cnt;
    %let word=%scan(&fldlist,&j);
    /* first word */
    %if count=0 %then %do;
      %let unique=&word;
      %let count=1;
```

```
      %end;
    /* rest of words in string */
    %else %do;
      %do k=1 %to &count;
        %if &word=%scan(&unique,&k)
          %then %let word=;
        %end;
      %if &word ne %then %do;
        %let unique=&unique &word;
        %let count=%eval(&count+1);
        %end;
      %end; /* rest of words */
    %end; /* traverse whole string */
%mend;
```

MOVE TO PRODUCTION
(Autocall waiting)

Once you have built and tested your own version of the ultimate SAS macro, you can make life even more easy by saving it in an Autocall library. You could even take it one step further by compiling the macros and storing them. I'll leave this to your discretion.

Surely an exclusive location should be set aside for such a grand piece of code. Whichever location this might be, you should store each macro in a separate file in this directory with the file name matching the macro name.

To utilize all your ultimate macros in the production environment, three set up steps should be taken.

1. Use the 'SASAUTOS=' command in the initialization stage of your SAS code to reference the library containing the ultimate macro(s).
2. Make sure the 'MAUTOSOURCE' system option is set. This causes the macro processor to look for your ultimate macro in the directory specified in the 'SASAUTOS=' command.
3. Turn off the system options for macro debugging. These system options are as follows: 'NOMLOGIC, NOMPRINT, NOMRECALL, NOSYMBOLGEN'.

CONCLUSION

Once the above specifications have been set up, any macro reference not explicitly defined in the program is presumed to reside on the 'SASAUTOS' directory. So if you've planned carefully and put as much foresight and insight into the design of your work, when that next project comes along you can simply execute your ultimate SAS macro and make SAS do all the programming work!

REFERENCES

SAS Institute Inc. (1997), *SAS Macro Language:, Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1994), *SAS Macro Facility Tips and Techniques, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), *SAS Guide to Macro Processing, Version 6, Second Edition*, Cary, NC: SAS Institute Inc.

ACKNOWLEDGEMENTS

SAS is a registered trademark or trademark of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.

CONTACT INFORMATION

Mike Tangedal
1204 Harmon Pl #19
Minneapolis MN, 55403
ph: 612-332-4235 e-mail: miamike@mtn.org