Putting the Pieces Together: Applying Advanced SAS® Macro Techniques to Everyday Problems

Jim C. Knoop, Aetna Life & Casualty, Hartford, CT

Abstract
This tutorial illustrates how SAS Macro features can drastically reduce the amount of maintenance needed by a macro-based reporting system. It also illustrates techniques which help prevent types of everyday problems encountered in a multi-programmer, shared-code environment. More emphasis is placed on the creative use of SAS macros than on their syntax.

Introduction
Over a dozen information specialists and SAS programmers were brought together to implement a reporting system with the following attributes:

- each report program was structured as a self-contained SAS macro, stored in an autocall library.
- code common to more than one report was defined as a macro, and referenced throughout multiple programs.
- explicit references to specific data sets, dates, etc. were eliminated.
- report titles, footnotes and cover pages were standardized, to promote a "common look".

As new ideas surfaced, and enhancements were proposed, we gained considerable experience modifying macros to provide new functionality, without having to convert existing programs which referenced them.

Although only a few examples are presented here, each represents a useful principle, and is well-suited for adaptation.

Overview
This tutorial is organized in the following sections:

- Symbolic Reference to SAS Data Sets
- A Few Macro Quoting Tips
- Undefined Macro Variables
- Minimizing Maintenance
- Upward Compatibility

Each section introduces a few examples and techniques which can be readily adapted to other uses. As you will see, however, none of the sections provide comprehensive or in-depth coverage of the subject matter.

SYMBOLIC REFERENCE TO SAS DATA SETS

SAS Data Sets on Disk

Our data sets are stored in the following SAS libraries on disk: DAILY, WEEKLY, MONTHEND and YEAREND.

Data sets stored in these libraries are named according to the date the data reflect (i.e., YEAREND.DEC92, MONTHEND.APR93, DAILY.MAY93, etc.).

Only one data set is stored in each of these disk libraries, reflecting the most recent year-end, month-end, weekly or daily file.

Macros were created for each library's data set to reflect their current names, and are stored in an autocall library.

For example:

\%
MACRO YEAREND;
DEC92
\%MEND YEAREND;
\%
MACRO MONTHEND;
APR93
\%MEND MONTHEND;

All SAS programs refer to these data sets symbolically:

\%
DATA EXTRACT;
SET MONTHEND %MONTHEND;
\%
DATA TODAY;
SET DAILY %DAILY;
\%
RUN;
\%
RUN;

Instead of manually changing the values of the MONTHEND, WEEKLY and DAILY macros in the autocall library whenever their contents change, these macros are instead updated by the same program that loads raw data into SAS data sets.

This is possible since each of our database extract files contains a field (RunDate) which reflects the "date-of-interest" of the raw data file.

Consider the following macro:
 DATA_NULL;
INFILE RAWDATA;
INPUT @170 SRUNDATESCHARS.;
LEN(YY $ 2 MMM $ 3);
YY = SUBSTR(SRUNDATE,3,2);
MMM = UPCASE(PUT(INPUT(RUNDATE,YYMMDD8.),MONNAME3.).);
CALL SYMPUT("&MACNAME","M" M I I YY);
STOP;
RUN;

DATA_NULL;
FILE AHRlIB(&MACNAME);
PUT @1 '%MACRO' "&MACNAME;" /
@1 "&MACNAME"
@1 '%MEND' "&MACNAME;";
STOP;
RUN;
%MEND _MAC LOAD;

For example, the SAS program which loads raw data into the WEEKLY library contains the following macro call, which updates the contents of macro WEEKLY when the program is run:

%_MACLOAD (WEEKLY)

SAS Data Sets on Tape

All historical month-end data sets are stored on cartridge, using a naming convention of MMMY (i.e., MAR92, OCT91, etc). Since these data sets are catalogued, they can be accessed via the following macro:

%MACRO HISTOPEN
 (DSN=, DATASET=, DDNAME=HISTORY, DISP=OLD, ENGINE=TAPE,UNIT=T3480, LABEL=);
 OPTIONS FILEDEV=&UNIT;
%IF &DSN EQ AND &DATASET NE %THEN
 %LET DSN=F4WB.PB.SB.&DATASET;
%IF &DISP NE %THEN %LET DISP = DISP=&DISP;
%IF &LABEL NE %THEN %LET LABEL = LABEL=&LABEL;
%IF &ENGINE EQ TAPE %THEN %LET ENGINE = ;
%IF &UNIT EQ T3480 %THEN %LET UNITX = ;
%ELSE %LET UNITX = UNIT=&UNIT;
LIBNAME &DDNAME &ENGINE &DSN &DISP &UNITX &LABEL;
%MEND HISTOPEN;

Sample Usage:

%HISTOPEN ( DATASET = MAR92 )
DATA EXTRACT;
SET HISTORY,MAR92;
etc.
RUN;

Examples

To access month-end data, as well as data for the same time a year ago:

DATA THISYEAR;
SET MONTHEND.%MONTHEND;
etc.
RUN;

DATA LASTYEAR;
SET HISTORY,&YEARAGO;
etc.
RUN;

To load the formatted date of the last day of the month for data sets THISYEAR and LASTYEAR (for use in titles, variable labels, etc.):

%LET YEARAGO =
%SUBSTR(%MONTHEND,1,3)%EVAL(%SUBSTR( %MONTHEND,4,2)-1);
%HISTOPEN (DATASET=&YEARAGO)

A FEW MACRO QUOTING TIPS

The Implicit %EVAL

SAS automatically invokes the %EVAL function with %DO (iterative), %DO %UNTIL, %DO WHILE, %SUBSTR, %QSUBSTR, %SCAN, %QSCAN and %IF.
%IF can be tricky when used within a macro to test macro arguments for specific values (including null), if the value assigned to the argument at macro invocation might contain SAS operators or other characters which require quoting.

For example, if a macro has a parameter (VARLIST=) which is supplied the value of VAR1-VAR10 when invoked, a statement such as %IF &VARLIST EQ %THEN ... will need to be quoted:

```sas
%IF %QUOTE(&VARLIST) EQ %THEN ... 
or
%IF %QUOTE(&VARLIST) EQ %THEN ... 
```

The %IF statement above (without quoting) may work fine for years without any problem, depending on what values are passed to the VARLIST parameter at macro invocation. Then, seemingly out of nowhere, you might encounter an error (with a rather cryptic message), whose cause is far from obvious.

**Using Macro Variables in Macro Calls**

With 14 different SAS programmers producing thousands of reports each year, we like to include the name, phone number and internal mail code of the developer in a footnote to each report.

If any of this information changes, it is not feasible to change each individual report. Thus, each of our reports contains the following, at the very beginning of each program:

```sas
%LET INITIALS = xxx ;
```

where 'xxx' refers to the programmer's initials.

The following macros are stored in an autocall library for each programmer (we'll use myself as an example):

```sas
%MACRO NAMEJCK;
JIM KNOOP
%MEND NAMEJCK;

%MACRO FONEJCK;
(203) 636-3553
%MEND FONEJCK;

%MACRO MAILJCK;
MC19
%MEND MAILJCK;
```

In a program, all I do is define %LET INITIALS = JCK; at the beginning, and call a macro which (among other things) does the following:

```sas
%LET NAME = %UNQUOTE(%NRSTR(%NAME)&INITIALS);
%LET FONE = %UNQUOTE(%NRSTR(%FONE)&INITIALS);
%LET MAIL = %UNQUOTE(%NRSTR(%MAIL)&INITIALS);
%LET F3A = NOTE: CONTACT &NAME AT &FONE/&MAIL FOR;
%LET F3B = QUESTIONS OR CHANGES TO THIS REPORT;
FOOTNOTE3 "&F3A &F3B";
```

**UNDEFINED MACRO VARIABLES**

Our unit's SAS coding standards require that certain macro variables be defined at the beginning of each SAS program.

A few examples include:

```sas
%LET INITIALS=JCK; /* programmer initials */
%LET PROGRAM=RPT6029; /* report name */
%LET RNAME=J. SMITH; /* requestor name */
%LET RMAIL=MB65; /* requestor address */
%LET MNAME=SUSAN WRIGHT; /* mail-to name */
%LET MMAIL=&RMAIL; /* mail-to address */
```

A variety of macros which produce consistent titles, footnotes and cover pages for our reports assume that these macro variables have been previously defined.

Therefore, these macros call a utility macro (stored in an autocall library) to test this assumption:

```sas
%MACRO _EXIST_ (NAME);
%IF %NRQUOTE(&&NAME) EQ %NRSTR(&)&NAME %THEN 0;
%ELSE 1;
%MEND _EXIST_; 
```

Note that the argument to macro _EXIST_ is a macro variable's name, and not its value. So, to test if macro variable INITIALS exists, use %EXIST(INITIALS), and not %EXIST(&INITIALS).

Unless, of course, &INITIALS resolves to the name of a macro variable (in this case, JCK) whose existence you wanted to verify.

At first glance, Macro _EXIST_ might appear to be bewildering.

However, given the statement %IF %EXIST(INITIALS) produces the following:

```sas
%LET INITIALS=JCK;
%IF %EXIST(INITIALS) %THEN;
```
INITIALS %IF Condifion %EXIST_
Defined (JCK) JCK EQ &INITIALS 1
Undefined &INITIALS EQ &INITIALS 0

To avoid a "Warning: Apparent Symbolic Reference Not Resolved" message, use OPTIONS NOSERROR before calling %EXIST_:

%MACRO COMPUTE:
OPTIONS NOSERROR;
%IF %EXIST_(TOTAL) %THEN etc.;
%ELSE etc.;
OPTIONS SERROR;
%MEND COMPUTE;

PARMBUFF Option
Use of the PARMBUFF option on the %MACRO statement can be useful when you need to know if a macro was called without any parameters defined, or if a certain parameter was assigned a value at macro invocation:

%MACRO BANNER (COPIES=1,CENTRAL=YES)
PARMBUFF:
We have two types of reports (ad-hoc and scheduled/production). The type of report can be determined at runtime by a macro variable called _MODE_.

For scheduled/production reports (_MODE_=PROD), we want the name and phone number of a single person who is responsible for maintaining and distributing reports to print in the footnote at the bottom of each report page.

For ad-hoc reports (_MODE_=ADHOC), we want the programmer's name and phone number to appear, unless the default value of CENTRAL is explicitly overridden.

By itself, the value &CENTRAL can't be used, since a YES value could either reflect the default that was assigned when the macro was created, or an actual parameter value specified when the macro was called.

%IF &_MODE_ EQ ADHOC %THEN %DO;
%IF %LENGTH(&SYSBPBUFF) EQ 0
%THEN %LET CENTRAL=NO;
%ELSE %IF %INDEX(&SYSBPBUFF,CENTRAL=) EQ 0
%THEN %LET CENTRAL=NO;
%END;

This way, changes to any group can usually be applied to a single source (in an autocall library) in order to take effect in a number of programs.

In the examples above, the entire contents of each macro have been enclosed in parentheses, to prevent any ambiguity in the order of evaluation of logical operators in statements such as:

IF &SALES AND NOT &NEWHIRE;

KEEP= Data Set Option
Our unit's coding standards also require that programmers use a KEEP= data set option on their SET statements, to restrict the number of variables that are read by a Data step. This reflects the fact that our SAS data sets are extremely large, whereas a typical program will only need a very small subset of variables.

With so many autocall macros referenced by a program, it is difficult to remember which variables are used by the macros called in the program.

Also, when a macro changes (i.e., references a new variable), it can be cumbersome to identify reports which might be impacted.

Our solution is to create macros similar to the following:
%MACRO SALEKEEP;
  ALLOC JOBNO GRADE
%MEND SALEKEEP;

%MACRO STFFKEEP;
  BHOSTATUS STATUS ALLOC ALOCLOC
%MEND STFFKEEP;

These are used as follows:

DATA EXTRACT;
  SET WEEKLY.%WEEKLY
  (KEEP=ID NAME %SALEKEEP %STFFKEEP);
  IF %STFF AND %SALES;
  etc.
RUN;

Future Use Considerations

Our data sets contain 20 levels of organization structure information (OSKEY01-OSKEY20), even though we are only using the first 10 levels at present.

%MACRO OSKEYEQ (VALUE);
  %LET MIN = 1;
  %LET MAX = 10;
  %LET COND = OR;
  %LET ROOT = OSKEYO;
  %DO N = &MIN %TO &MAX;
  %IF &N EQ &MAX %THEN %LET COND =;
  %IF &N GE 10%THEN %LET ROOT = OSKEY;
  &ROOT.&N EQ 8cVALUE &COND
  %END;
%MEND OSKEYEQ;

The macro call IF %OSKEYEQ(04232); produces:

IF (OSKEY01 EQ '04232' OR OSKEY02 EQ '04232' OR OSKEY03 EQ '04232' OR ... OSKEY10 EQ '04232');

%MACRO OSKEEP;
  %LET MIN = 1;
  %LET MAX = 10;
  %DO N = &MIN %TO &MAX;
  %IF &N LT 10 %THEN OSKEY0&N;
  %ELSE OSKEY&N;
  %END;
%MEND OSKEEP;

%OSKEEP produces
(OSKEY01 OSKEY02 OSKEY03 ... OSKEY10).

By changing the value of MAX in OSKEYEQ and OSKEEP from 10 to 20, we can allow for the future use of these variables, without modifying existing reports.

Historical Considerations

We load our SAS data sets from raw data extracts from various databases. Additions, deletions or changes to data elements result in changes to the record layouts of our raw data extracts.

Our SAS data set nomenclature takes the form of MMMYY (i.e., NOV92). Thus, we can use a single macro to construct an INPUT statement that, depending on the date of the raw data file, can locate each element in its proper place:

%MACRO LOADFILE (LIBRARY=.DS=.DDNAME=RAWDATA);
  DATA _NULL -;
  IF "@1&DS-D GE '01JAN92'" THEN CALL SYMPUT(V':3');
  ELSE IF "01 &DS-D GE '01JUL91'" THEN CALL SYMPUT(V:2');
  ELSE IF "01 &DS-D GE '01APR90'" THEN CALL SYMPUT(V:1');
  ELSE CALL SYMPUT(V:0');
  STOP;
RUN;
   DATA &LIBRARY .. &DS;
   INFILE &DDNAME;
   INPUT @1 ID SCHAR7.
   @8 NAME SCHAR20.
   %IF &V LT 3 %THEN
   @45 DEPTUSE SCHAR10.;
   @90 DEPTUSE SCHAR5.;
   @30 YTDPAY PD5.2
   etc.;
RUN;
%MEND LOADFILE;

Sample call:
%LOADFILE (LIBRARY=YEAREND,DS=DEC90)

AUTOEXEC= System Option

We use the AUTOEXEC= system option to point to a file containing SAS statements which will be executed when SAS is invoked.

Some of the statements we include in our SAS autoexec file are:

%LET _MODE_ = TEST;
%LET _WARN_ =
%LET _EMPTY_ = WARNING: NO OBSERVATIONS MET REPORT SELECTION CRITERIA;
LIBNAME XLOG DISK 'F4WB.PB.XLOG' UNIT=SYSDA,
DISP=(MOD,DELETE,DELETE),SPACE=etc.;
etc.
This allows us to initialize and define certain macro variables that might be referenced by any macro invoked in a given SAS session or job.

Since changes can be applied to a single source, future maintenance is simplified considerably.

**UPWARD COMPATIBILITY**

**Avoiding Conversion**

Efforts to improve existing macros (used by many programmers in numerous applications) are usually hampered by conversion considerations.

When a better mousetrap has been acquired, nobody likes to expend time and money fixing the older models—even though it can be argued that they do, in fact, still work.

To stick with the old results in considerable opportunity costs.

Dual maintenance (of old and new) can be tedious, expensive, confusing for programmer/users, and risky (from a quality perspective).

We have developed some useful strategies which, in many cases, allow us to continuously improve our macros, without jeopardizing our investment in their older counterparts.

**Adding Parameters**

Consider the following macro, which produces a standard report for any area of the Company, sorted in a standard order:

```
%MACRO ORGLIST (SELECT=TITLE=);
```

Users supply selection criteria and a title, and the appropriate report is produced. Numerous programs referencing macro ORGLIST are run weekly.

After a recent reorganization, the standard sort order was not useful for a few departments. Rather than create a separate version of the macro for these customers, a new parameter `SORTBY=` was added, and assigned a default value which reflected the original sort order:

```
%MACRO ORGLIST (SELECT=TITLE=SORTBY=LEVEL1 LEVEL2 LEVEL3 LEVEL4 LASTNAME);
```

The PROC SORT statement in the original version of ORGLIST was changed:

```
PROC SORT DATA=EXTRACT;
BY LEVEL1 LEVEL2 LEVEL3 LEVEL4 LASTNAME;
RUN;
```

from:

```
PROC SORT DATA=EXTRACT;
BY LEVEL1 LEVEL2 LEVEL3 LEVEL4 LASTNAME;
RUN;
```

to:

```
PROC SORT DATA=EXTRACT;
BY &SORTBY:
RUN;
```

Existing programs which call macro ORGLIST continue to produce the same results as before. However, new programs can change the sort order of their reports by simply specifying an appropriate value for `SORTBY=`.

**Old Macros Calling New Macros**

Consider the following macro, which joins two tables, dropping observations not found on both tables:

```
%MACRO JOINK (MAINFILE=,TABNO=.DATEVAR=.KEEP=);
```

A more robust version, with a different name, was developed a year later by another unit, which was based on the original JOINK macro:

```
%MACRO JOIN (MAINFILE=,TABNO=.DATEVAR=.KEEP=.DROP=.DDNAME=AHRTAB.SORTBY=.OUTFILE=&MAINFILE):
```

The two units, along with their respective SAS programs, were consolidated into one area.

JOINK was then redefined, as follows:

```
%MACRO JOINK (MAINFILE=,TABNO=.DATEVAR=.KEEP=.DROP=.DDNAME=AHRTAB.SORTBY=.OUTFILE=&MAINFILE) / PARMBUFF;
%UNQUOTE(%NRSTR(%JOIN)&SYSPBUFF)
%MEND JOIN;
```

In other words, every time macro %JOIN is now called in existing programs, the parameter list specified in the %JOIN macro call is actually passed to the %JOIN macro, which subsequently executes.
Conclusion

The preceding examples represent only a small sample of the many and various applications of SAS macro techniques to improve the management and reduce the maintenance requirements of reporting systems. Hopefully, they will stimulate your thinking, and lead to new and useful ideas on your part.

Although not directly addressed here, performance and efficiency concerns need to be factored into your own design criteria.

Comments, suggestions and questions are welcome. The author can be reached at the following address:

Jim C. Knoop
Aetna Life & Casualty
MC19
151 Farmington Avenue
Hartford, CT 06156

Voice: (203) 636-3553
FAX: (203) 636-3573

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