Paper BB-08

Make Macros Safe for Others to Use: Eliminate Unexpected Side Effects

David H. Abbott and Rebecca B. McNeil
Veterans Affairs Health Services Research
Durham, NC

ABSTRACT

The SAS® macro language can, in theory, be used to produce components of SAS software that are safely usable by a broad audience. However, in practice, SAS macros can be problematic for users because they may have unintended and unadvertised side effects. These side effects, e.g. resetting the value of a macro variable already in use, are not just minor nuisances; they can cause the invoking SAS software to fail in ways that are hard to debug. Even worse, they can introduce incorrect behavior that may go undetected until the validity of the results is challenged. Although the potential for expected side effects is usually not realized (e.g., the macro variable written to is usually not previously in use), the potential is neither rare nor isolated. We examined mid-length macros publicized to the SAS community and found the potential for one or more unexpected side effects pervasive.

For macros to achieve their potential as reusable software components, it is necessary to construct them according to practices that *reliably prevent* unintended side effects. Aside from the %LOCAL statement, SAS Base® provides only indirect and obscure support for eliminating unintended side effects in macros. The most difficult problems arise when macros inadvertently redefine symbols already defined in the invoking environment. Inventing names that are thought to be unique does reduce the likelihood of unexpected overwrite but it does not *reliably prevent* such occurrences. This paper shows how to consistently *prevent* this dangerous overwriting in the most important classes of symbols used by macros: global macro variable names, macro names, dataset names, variable names, and format names.

INTRODUCTION

SAS Base V9.2, via its MACRO language component, provides a powerful language for achieving procedural abstraction and enabling users to approach the ideal of creating broadly reusable software components. However, for user-developed macros to achieve this potential, macros must be constructed according to practices that *reliably prevent* unintended side effects. Otherwise, users are exposed to hard-to-diagnose software failures, or worse, apparently successful execution (i.e., without warnings or errors) that produces faulty results.

Several SAS User Group authors have addressed aspects of preventing unexpected side effects in macros (see References). In particular, Redner et al. note “It is important that a macro tread lightly on the SAS session in which it is run.” They also provide techniques pursuant to that goal. This paper builds on these contributions and, in particular, gives a stronger solution to the problem of a macro inadvertently overwriting symbols already defined in the SAS session.

SPECIES OF SIDE EFFECTS

We identified 8 types of unexpected side effects that a macro may cause. These are described briefly in the table below. Some are discussed in previous SESUG/SUGI papers. Five are dealt with in this paper.

Type of Side Effect	Guidance in	Comment
Alteration of titles and footnotes	Redner et al., Bramley	This side effect is easy to spot, but annoying for the user to correct.
Changing a SAS option value	Redner et al., Ivis, Bramley	e.g. setting MERGENOBY to NOWARN, easily prevented with save/restore.

Alteration of ODS state	NA (Not covered previously or in this paper.)	ODS needs to support saving and restoring ODS state, but currently does not.
Resetting value of macro variable present in invoking context	Redner et al., Ivis, DiIorio, *this paper*	Avoidable via use of %LOCAL and SYMPUTX(,,L), unless global scope required for a macro defined macro variable
Overwriting a dataset in invoking context	Redner et al., Ivis, Bramley, *this paper*	Can be dangerous to integrity of results.
Overwriting a dataset variable	*This paper*	Issue for macros that add variables to an input dataset and for macros invoked within a data step.
Overwriting a FORMAT definition	Redner et al., *this paper*	Existing solution needs elaboration.
Redefining a macro name	Redner et al., *this paper*	Full solution is complicated; discussions underway with SAS.

This paper presents two strategies for addressing the last 5 side effects listed above. See the referenced papers for guidance on titles/footnotes and how to save and restore system option values.

PREVALENCE OF MACRO SIDE EFFECTS

To illustrate the prevalence of side effect exposures in macros intended for broad usage, let's take a look at several examples.

The macro *boot* developed by Mayo Research, available from http://mayoresearch.mayo.edu/mayo/research/biostat/sasmacros.cfm, generates bootstrap samples. Potential unexpected side effects that could come from using it include:

- It silently overwrites the global macro variable ERRORFLG.
- It silently overwrites and then deletes datasets WORK._BOOT and WORK._TBOOT.

Neither of these side effects is scandalous – ERRORFLG looks pretty temporary and perhaps any dataset starting with an underscore is fair game? But is this *reliably safe*? We don't think so.

The macro *unipvals* developed by Roland Rashleigh-Berry, available from http://www.datasavantconsulting.com/roland/Spectre/maclist2.html , calculates statistics values and p-values for the unistats macro (same source). Potential unexpected side effects that could come from using it include:

- It silently overwrites WORK datasets _uniquadupd, _dummypval, _pvaldsin, _whichtest _notest, _dofisher, _dochisq, _donotest, _temp, etc.
- Without checking for their being already in use, it redefines the macros named uniquad, unisept, notest, cmh, chisq, and fisher.

Again, the dataset overwrites are somewhat mitigated by the leading underscore. The redefinition of 6 fairly typical macro names is, however, not mitigated in this way. These overwrites/redefinitions are not, in our opinion, reliably safe.

The macro *sir*, developed by Gustaf Edgren, available from http://gustafedgren.se/sas/sir.sas , calculates standardized incidence rates for Poisson processes. It is different from the two preceding macros in that it is designed to execute within the user's data step rather than being a full or multiple data steps. Potential unexpected side effects that could come from using it include:

- It first overwrites and then DROPs the variables cl and cu without checking for these two variables being otherwise employed.
- It lacks a %LOCAL statement and hence writes to the macro variable p that is potentially global and in use.

This is just a tiny non-random sample, but it suggests to us that medium-length macros frequently expose users to unexpected side effects. Note that the sample macros chosen are publically available, and were composed by highly experienced SAS users. The "average" macro in use is apt to have as many or more unexpected side effects.

Below, we present two strategies for eliminating unexpected side effects.

LOOK BEFORE YOU LEAP

The primary strategy a macro writer can use to prevent the most frequently occurring unexpected side effects is "look before you leap" – i.e. check for the existence of the symbol in the session before overwriting it with a new value. The details of doing that are different for each applicable species of side effect:

Type of Side Effect	Side Effect Prevention Code
Resetting value of macro variable present in invoking context	%IF %sysfunc(symexist(&macVar))=1 %THEN /*macVar exists, notify user and exit gracefully*/
Overwriting a dataset in invoking context	%IF %sysfunc(exist(&DSN)) = 1 %THEN /*dataset exists*/
Overwriting a dataset variable	%LET dsid=%sysfunc(open(&ds)); %IF %sysfunc(varnum(&dsid,&var)) > 0 %THEN /* variable exists */
Overwriting a FORMAT definition	%if %sysfunc(cexist(work.formats.&name..format, u)) %THEN /* format name already in use*/
Macro name	%if %sysfunc(cexist(work.sasmacr.&name..macro, u)) %THEN /* macro name already in use*/

Of course, when a name conflict is detected, some kind of graceful exit is called for. One possibility would be to notify the user and then invoke %RETURN.

For a macro name already in use, the check shown above is good as far as it goes, but it does not go far enough. Why? Let's call the macro being made safer the "author's macro". Let's call the macro that shares its name with the author's macro the "conflicting macro". When the check is made for a conflicting macro name, the conflicting macro may not yet have been loaded (i.e. compiled) yet and so is not in catalog SASMACR. However, subsequent portions of the user's application may expect to invoke the conflicting macro rather than the author's macro, and so fail.

Given that the conflicting macro isn't in SASMACR yet, wouldn't that mean that it will be subsequently defined in the code of the application and simply replace the author's macro (a different problem)? Unfortunately, there is no guarantee. This is true because SAS macro facility supports access to numerous macros via the SASAUTOS option and associated directory list. If the conflicting macro is normally accessed via SASAUTOS, it does not occur in the application's code per se, and is loaded "as needed" (when a call to the macro name is made and the macro name does not occur in SASMACR). So, the author's macro resides in SASMACR and the conflicting macro is never loaded or used and subsequent code fails – a nasty and unexpected side effect.

What is the solution? It would be ideal if SAS provided a function that attempted to find a macro name in either SASMACR or the SASAUTOS list and returned true or false. If the author's macro name is found with this function then the conflict has been detected and, as usual, user notification and graceful exit can follow. We are working with SAS to realize the function mentioned above; it does not currently exist.

GET HELP FROM THE USER

An alternative strategy is to involve the user in helping the macro avoid symbol overwrite/redefinition. The user is invited to supply a prefix to be used with all macro generated symbols, e.g. t__. This is similar to the common practice in use which is for the macro writer to invent a prefix, but we think the technique presented here has important advantages:

- Users know the overall naming scheme being used in *their* applications. They know if, for example, t__ is already in use by another part of the application and so can choose to go with s__. In contrast, the macro writer has no way to know the overall naming scheme and no way to choose a scheme guaranteed to work properly and conveniently for all users.
- When the user gets to specify this prefix, call it the scratch prefix, the macro writer is obliged to code the macro so that it works with whatever scratch prefix a user provides. So, changing the prefix being used requires no change to the macro per se (just to its scratch prefix argument) and hence it is easy for the user to change the value of the scratch prefix as needed to prevent the scratch symbols used by two or more macros from interfering with each other..

Here's an example macro coded to use a user-supplied scratch prefix (&sp):

```
/* Illustration of use of user supplied prefix for scratch variables.
   Macro computes binomial confidence limits and is used within a data step*/
%macro binomCI( numer=, denom=, alpha=0.05, lowlimit=, uplimit=,
              cleanUp=1, sp=);
  if &denom eq .  or &numer eq . then do;
    &lowlimit = .; &uplimit = .;
  end;
  else do;
    &sp.num = &numer; &sp.den = &denom;
    &sp.zsq = (probit( &alpha/2))**2;
    &sp.phat = &sp.num/&sp.den;
    &sp.aa = &sp.phat + &sp.zsq/(2*&sp.den);
    &sp.bb = sqrt(&sp.zsq* ((&sp.phat*(1-&sp.phat))/&sp.den +
      &sp.zsq/(4*&sp.den*&sp.den)));
    &sp.cc = (1 + &sp.zsq/&sp.den);
    &lowlimit = (&sp.aa - &sp.bb)/&sp.cc;
    &uplimit = (&sp.aa + &sp.bb)/&sp.cc;
  end;
  %if &cleanUp eq 1 %then
      drop &sp.zsq &sp.phat &sp.aa &sp.bb &sp.cc &sp.num &sp.den;
%mend;
```

The large number of temporary variables and the "cleanup=" parameter are essentially a debugging device and designed to keep a trace of all the calculations done in the output dataset if desired. They also nicely illustrate that even "messy" macros can avoid unexpected side effects via the user-supplied scratch variable prefix.

RECOMMENDATION

Assuming that the user chooses a scratch prefix with total awareness of symbol usage, no symbol conflicts should result when the user's application is run. But that is expecting a lot from users and, well, accidents happen. So, a hybrid approach is probably best: get the user to provide a scratch prefix, and also check for symbol existence before changing the value of the symbol. When it is not possible or practical to prevent or detect/handle a side effect, providing a clear and attention-getting warning of the side-effect to the user is a strategy of last resort.

CONCLUSION

Some closing remarks regarding this topic are:

- Many SAS macros are capable of producing unexpected and injurious side effects when the macro overwrites/redefines symbols already in use in the user's SAS session.

- It is straight-forward to check for symbols being already in use, although the process of checking is different for each type of symbol.
- The user is better positioned than the macro author to know a suitable prefix to use to distinguish macro specific symbols from all other symbols in use by the application. It is relatively easy to receive this prefix from the user via a macro parameter and apply it to all new symbols introduced by the macro.
- SAS macros can be written to be free of unexpected side effects, and authors who do this save users from nasty bugs and further the effective use of SAS macros.

REFERENCES

Bramley, Michael P. 2004. "Better Clay Builds Better Bricks: Some Simple Suggestions To Writing Professional Macros" Proceedings of the 29th Annual SAS Users Group International Conference, http://www2.sas.com/proceedings/sugi29/055-29.pdf

Dilorio, Frank 2006. "What Happens in the Macro, Stays in the Macro" Proceedings of Southeast SAS Users Group Conference, http://analytics.ncsu.edu/sesug/2006/SC14_06.PDF.

Dilorio, Frank 2010. "Building the Better Macro: Best Practices for the Design of. Reliable, Effective Tools" Proceedings of Southeast SAS Users Group Conference, http://analytics.ncsu.edu/sesug/2010/FF02.Diioriopdf.pdf.

Ivis, Frank 2004. "Guidelines on Writing SAS® Macros for Public Use." Proceedings of the 29th Annual SAS Users Group International Conference, http://www2.sas.com/proceedings/sugi29/047-29.pdf

Redner, Ginger; Zhang, Liping; Herremans, Carl 2008. "Techniques for Developing Quality SAS ® Macros" Proceedings of Southeast SAS Users Group Conference, http://analytics.ncsu.edu/sesug/2008/SBC121.pdf

ACKNOWLEDGMENTS

The views expressed in this paper are those of the authors and do not necessarily reflect the position or policy of the Department of Veterans Affairs or the United States government.

Without the leadership and encouragement of Dr. Dawn Provenzale, director of the Durham Epidemiologic Research and Information Center at the Durham VA Medical Center, this work could not have occurred. She takes a strong interest in fostering many dimensions of excellence in her employees.

CONTACT INFORMATION

Name	David H. Abbott
Enterprise	Center for Health Services Research in Primary Care
Address	Durham Veterans Affairs Medical Center
	HSR&D Service (152)
	508 Fulton St.
City, State ZIP	Durham, NC 27705
Work Phone:	919-286-0411
E-mail:	david.abbott@va.gov