Paper CC132

Bulletproof Macros: Avoiding Macro Name Collisions

David H. Abbott, Veterans Affairs Health Services Research

ABSTRACT

The SAS® macro facility provides an opportunity for a SAS programmer to produce chunks of SAS code that function as reusable components and provide value to a broad audience. However, if users run into unexpected difficulties in using a macro, both they and their colleagues quickly lose interest in taking advantage of such macros. One source of unexpected difficulties is macro name collision – the macro intended for reuse inadvertently overwrites a preexisting macro of the same name. Ensuring that the users of a macro don't get caught in this trap is complicated, but this paper provides a practical solution.

The two keys to the solution are: 1) a macro (written by Rick Langston of SAS) that checks for the prior existence of a given macro, either already compiled or reachable via SAS autocall facility and 2) a simple second macro that allows the macro intended for reuse to be compiled only if a macro of the same name is not already present. This approach is not ideal – users can bypass this convention; it is not enforceable. However, by specifying that the author's macro be compiled using the indicated procedure, the author achieves perhaps the best solution available with SAS 9.

Introduction

The primary concern of this paper is unintended macro name collisions and how to avoid them. Of course, such an accident affects both the author of the offending macro and the user of this macro. The user gets unexpected behavior/wrong answers. The macro author loses the trust of at least one and perhaps many users.

Using abstract terms, the scenario for an instance of macro name collision is simply:

1. The user compiles a macro.
2. The user compiles another macro with the same name.
3. Invocation of the first macro compiled is no longer possible and invocation of a macro with this name runs the second macro body rather than the first.

So, how is this a problem? How is it a source of unexpected behavior? A more concrete example better illustrates the potential for problems:

1. The user's code compiles author A's collection of macros for computing a patient's diabetes risk which includes a macro, %BMI(weight, height).
2. The user's code compiles author B's collection of macros to compute cardiac risk factor which compiles a macro, %BMI(height, weight). Author B's definition of %BMI overwrites author A's (even though the order of the positional parameters is swapped!). SAS provides no warning or error.
3. The user's code invokes the macro for calculating diabetes risk that in turn invokes %BMI with first weight and then height given are arguments (author A's version). However, the version of %BMI invoked (author B's version), expects height to come first and then weight so the BMI value calculated is way off the mark.

The bug in this scenario is not too hard to find since the calculated values are so far off and outside of reasonable ranges (very thin people, indeed). However, suppose two results from the two conflicting versions of the macro were, say, 30% different. The value returned would be wrong but no error message produced and no obviously incorrect values. This might be a dreadfully hard case to debug IF the analyst were lucky enough to even detect it in the first place.

The issue of inadvertent macro name collision was discussed in “Make Macros Safe for Others to Use: Eliminate Unexpected Side Effects” (Abbott, 2012). However, the 2012 paper provided only an incomplete discussion of the issue and SAS had some work to do to ameliorate another aspect of the problem, see “A Macro to Verify a Macro Exists” (Langston, 2013). This paper fills in the picture and shows how to make good use of Langston’s work to achieve a measure of protection from inadvertent macro name collision.

Difficult problem

It is surprisingly hard to systematically prevent macro name collisions. The several difficulties include:

- Once inside an author’s macro the damage is done – the collision has already occurred. So, there is no way for the macro itself to check for a name collision and abort before the collision occurs; it has already occurred. (This key point was missing from (Abbott, SESUG 2012).)
- It is not unusual to intend to overwrite a macro definition with a new definition. So, it is necessary to properly handle both accidental and intentional overwrites appropriately.
- SAS autocall macro facility complicates determining if a macro name is already “claimed”. Without this facility, a macro name is claimed when the definition of the macro is encountered in the code (usually via a string of %includes processed at the top of the SAS program). The sasmacr catalog (visible in the Work library) identifies the macros whose names are claimed in this way. The list of autocall libraries provides another and a less explicit way to establish a claim on a macro name. The full list of autocall libraries must be searched to determine if any given macro name is so claimed.

Proposed solution

Since the macro itself cannot check for a name conflict (too late) the approach we use is to wrap it in a generic macro that first checks for a macro name being available (i.e., not already in use) and then compiles the macro only if no name conflict is present. For example, rather than using

```
%include "C:\someFolder\someMac.sas";
```

to compile someMac the user would instead use

```
%SafeMacDefine(name=someMac, loc="C:someFolder\someMac.sas", force=0);
```

Key points about the externals of %SafeMacroCompile include:

- For the usual case (force=0), the sas code at the specified location (loc=) is compiled if the macro name given (name=) is unknown to the session (i.e., unclaimed) and is not compiled and a message written to the log if a name conflict is present.
- With force=1, even if a name conflict is present, someMacro is compiled and overwrites the original definition of someMacro (if any) and no message is generated. Including the force= value in the invocation makes it easy to toggle prior use checking on and off if desired.
- %SafeMacDefine is not intended for use with macros provided to users via autocall libraries. Art Carpenter provides a good discussion of avoiding macro name collisions with autocall libraries (Carpenter 2012).

Here’s the code for SafeMacDefine:

```
%*MACRO SafeMacDefine(
  name=, => the name of macro to be defined if not already defined
  loc=, => file location of the macro definition code (no quotes)
  force=0, => replace definition unconditionally if set to 1
)
  Note: requires prior compile of %macro_exists and %macroExists
```

```
;
%macro SafeMacDefine(name=, loc=, force=0);
  %local alreadyDef;
  %if &force %then %do;
    %include "&loc";
  %end;
  %else %do;
    %macroExists(mName=&name, resName=alreadyDef);
    %if not &alreadyDef %then %do;
      %include
    "&loc"; %end;
    %else %do;
      %put SafeMacDefine: macro &name already defined, not
    compiled; %end;
  %end;
%mend SafeMacDefine;
```

Key points about the implementation of this macro include:

- The bulk of the work done within %SafeMacDefine is to check for the name conflict and this is done by %macroExists which is a trivial but useful derivative of Langston's %macro_exists macro. (See Appendix.)
- When force=1, the macro resolves to simply the %include statement. Some users may want to initially use force=0 and then use force=1 when the code and the SAS execution environment are stable in terms of the macros defined.

Take Aways

- Macro name collisions can cause nasty bugs.
- Avoiding these collisions is complicated.
- Authors of macros can instruct users to use %SafeMacDefine instead of a direct %include and thereby obviate the troubles for both users and authors resulting from macro name collisions.

REFERENCES

- Abbott, David, 2012. "Make Macros Safe for Others to Use: Eliminate Unexpected Side Effects". SESUG 2012, available at http://analytics.ncsu.edu/sesug/2012/BB-08.pdf.
- Carpenter, Art. 2012. Carpenter's Guide to Innovative SAS Techniques. Cary, NC: SAS Institute Inc. (pp 427-430).
- Langston, Rick, 2013. "A Macro to Verify a Macro Exists", SAS Global Forum 2013, available at http://support.sas.com/resources/papers/proceedings13/339-2013.pdf.

ACKNOWLEDGMENTS

The views expressed in this paper are those of the author 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 Cooperative Studies Program Epidemiology 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

APPENDIX — SOURCE CODE FOR MACROEXISTS

```
%*MACRO macroExists(
  mName=, => the macro whose existence is to be checked
  resName=, => a previously declared macro variable used to return result
    (can be declared %local in calling context)
  maclib_fileref=maclib, => as stated, change if need be
  sascode_fileref=sascode, => as stated, change if need be
  myautos_fileref=myautos, => as stated, change if need be
  process_sasautos_name=process_sasautos)

This wrapper macro for Langstons macro macro_exists makes same easier to use by
providing defaults for the last four arguments and providing the resName argument
to replace the member_found global employed by macro_exists. The code for
macro_exists is given in Langstons paper (SAS Global Forum 2013).
;
%MACRO macroExists(mName=, resName=, maclib_fileref=maclib,
  Sascode_fileref=sascode, myautos_fileref=myautos,
  Process_sasautos_name=process_sasautos);

  %macro_exists(&mName);
  %let &resName = &member_found;
  * use of global member_found can be avoided by using %macro_exists(&mName,
    &resName)above and replacing
    references to member_found therein with &resultMv;
%MEND;
```