Macro Programming Best Practices: Styles, Guidelines and Conventions Including the Rationale Behind Them

From sasCommunity

Contents

- 1 Presentation Plans
- 2 Abstract
- 3 Insert Your OpEd Here
  - 3.1 Ron.Fehd.macro.maven short list
  - 3.2 Named vs Positional Parameters
  - 3.3 Local vs Global Macro Variables
- 4 Best Practices
  - 4.1 Writing Testing-Aware
  - 4.2 Always suffixing a macro variable reference with a period/dot (.)
  - 4.3 Macro *booleans*
  - 4.4 Defining Macros Inside other Macros? Knot!
  - 4.5 Macro Code Length
  - 4.6 Coding Style
- 5 Macro Storage
  - 5.1 Pointing to Where Macros are Stored
- 6 Sharing Macros with your Friends and Neighbors
- 7 Do we really need macros?
  - 7.1 Replacing macro %do loops
  - 7.2 Replacing macro %if
  - 7.3 Optimization: Saving Compile Time
- 8 Parameterized %Includes: Do we really need macros?

Presentation Plans

First, thanks to everyone for their insights/contributions. At SAS Global Forum 2012 Art Carpenter and Don Henderson led a discussion, with audience participation, on the topics presented here. After the conference all the arguments pro/con will be collected and hopefully the discussion will continue.

Another discussion opportunity will take place when this topic is again presented at PharmaSUG 2015.

Abstract

Coding in the SAS Macro Language can be a rich and rewarding experience. The complexity of the language allows for multiple solutions to many programming situations. But which solution is best, is there even a ***best*** solution, and how do you decide? Should you choose a technique based on ease of programming or because of the efficiency of the solution? What is the best practice for laying out your code, selecting your statements, and organizing the way you use the symbol tables?

Learn about macro language programming styles, guidelines and conventions and why they are important. Since not everyone always agrees on programming practices, this article focuses on the rationales behind the techniques so you can make an informed choice based on your environment.

This article is intended to be living paper (the PDF in the proceedings will point here).

- You can Sugest a Best Practice Topic for Discussion (https://web.archive.org/web/20191017191057/http://www.sascommunity.org/mwiki/index.php?title=Macro_Programming_Best_Practices:_Styles,_Guidelines_and_Conventions_Including_the_Rationale_Behind_Them&action=edit&section=new). Make sure to fill in the *Subject/Headline* field with your topic and optionally provide some detail in the text box that follows.
- Comment on an existing topic by clicking the *edit* link next to.

And don't forget to to use the signature button to *sign* your comments.

The presenters/authors, Don Henderson and Art Carpenter look forward to a lively discussion on this important topic.

Insert Your OpEd Here

Ron.Fehd.macro.maven short list

- overview: maximum line length = 72 for MVS
- macro name limited to 8 characters
  - for MVS
  - lowcase for Linux/Unix

- Use appropriate technology
- Use technology appropriately

Yes, use global macro variables and use allocation statements to create them and call symputx, etc to create global macro variables in open code.

YeahBut: use data sets to pass values from within a macro to the open code environment. This idea addresses the point of having a macro modify a global macro variable.

Named vs Positional Parameters

How should a macro developer decide on the use of named vs positional parameters? --Don Henderson 13:07, 8 January 2012 (EST)

Here's my humble opinion: If a parameter will almost always default to a specific value, make it a named parameter, otherwise make it positional. That way one can take advantage of the ability to assign that parameter to a default value when designing the macro. --Otterm1 18:10, 10 January 2012 (EST)

Personally, I always use Keyword Parameters. Keyword Parameters allow for a true default value for parameter used by the macro, can be used in any order and finally allow for tracing back what was enetered as values for what parameter (think debugging). I have seen way to many people in way too many shops try to use positional parameters, what happens is a big boone doggle as the macro calls become unreadable and people forget what parameter should go where and what value was passed to what parameter. What should have taken them 5 minutes to debug takes them half a day or longer. I do beleive that Ian Whitlock once showed an example where one had to use positional parameters on SAS-L, however, I would caveat this with it was an extreme case and one that 99.9999% of the SAS community would ever need to do. --User:TobyDunn 18:43, 10 January 2012 (EST)

In general I prefer keyword parameters. They make the macro call easier to understand and if there are many parameters then keyword parameters will avoid a lot of confusion. But if there are only one or two parameters, I'll often switch to positional parameters. Less typing, easier to understand.--John Hendrickx 08:38, 11 January 2012 (EST)

1. named is The Way To Go for the following reasons:

- one positional parameter is ok, in an in-program macro.
  - Two? name them. ... because sooner any good-enough macro gets moved out of the MyProgram into MyProject folder for use by other project programs, then TheWord gets out and the macro is on its way to the site sasautos folder, and Documentation had better be good, by then.
  - following this idea, the macro name has to be explanatory of the value passed to it, as well as the result returned.
  - %macro Do_This(zq);
  - %macro Attribute_X_of(entity);
- named parameters == documentation, no need to find the macro definition.

--macro maven == the radical programmer 09:27, 11 January 2012 (EST)

I am also in the camp of *almost* always (I try to avoid *always*/*never*) using keyword parameters for many of the reasons above. The ***one*** exception is when writing a macro that simulates a SAS function. For example, I have a %getvarc (and %getvarn) macro that uses %sysfunc to implement what the getvarc/getvarn functions do. Since I am trying to create a macro analog to an existing SAS function, I use positional parameters. But the vast majority of the time (99.99%) I use keyword parameters. --Don Henderson 19:44, 28 January 2012 (EST)

Local vs Global Macro Variables

A common perception is that creating global macro variables inside of macros should be avoided. But is that true or even practical? --Don Henderson 13:12, 8 January 2012 (EST)

In my own work I like to delete things from the SAS environment as soon as they are no longer needed. So, if a macro variable is not needed outside of the macro, I would try to make it local. --Otterm1 18:19, 10 January 2012 (EST)

Its both true and practical, its called information hiding. A Macro shouldn't be dependent on a Global macro whose value isnt passed in via a parameter nor and part of the rest of the code outside of the macro be dependent on a global macro being created inside of the macro. In short it almost always leads to "speghetti code with macro sauce", and it in my opinion it shows sloppy coding standards. --User:TobyDunn 18:47, 10 January 2012 (EST)

I agree that global macro variables should be used sparingly. But sometimes they're practical so why avoid them? For example, I wrote a macro that will create a set of order statements for a SAS/Graph axis statement to let a set of graphs all have the same range. The output is a set of global macro variables. I could have achieved this by writing a macro that would define the axis statements but that would have meant adding more options corresponding with axis statement options and that would have resulted in spaghetti code. --John Hendrickx 08:45, 11 January 2012 (EST)

Mu! Wrong Question! The Question is: Who, meaning: which routine or subroutine, is responsible for creating, using, and then deleting/removing/Symbol-Deleting (%symdel) a global macro variable?

My answer is: any (sub-)routine that allocates a global macro variable is responsible for its deletion. After which follows: any (sub-)routine called can use, but not change, a global macro variable. Side-Effects Happen! Now, how to find When and Where they happened!

--macro maven == the radical programmer 15:45, 11 January 2012 (EST)

This is an example where a more *nuanced* analysis is needed IMO as it can depend on the environment that the macros run in. The concept of global/local parameters has been around since the beginning of procedural languages and generally global parameters are frowned upon because of the potential of stepping on each other. And along those lines, being explicit about using %local/%global is perhaps a more important issue than which symbol table is being used.

Parameters that are global, should probably be global parameters. And quite often, macros may need to create such parameters and assign their values. Thus, a hard and fast rule that macros should not create or assign values to global parameters isn't appropriate.

Consider also SAS Stored Process and SAS/IntrNet Application Dispatcher programs. In both cases parameter values passed from the client are global macro variables. And so, having such programs (if they are macros) create global parameters is, in some senses, necessary (e.g., sometimes the client does not pass in a value so there is no macro variable - but the code must depend on it). This is entirely reasonable and acceptable since each Stored Process/IntrNet program runs in its own SAS executive and so there is absolutely no chance that other requests can be impacted.--Don Henderson 19:57, 28 January 2012 (EST)

Another approach to passing macro variable values is to allow them to *flow* up to the calling macro by first establishing them in the higher symbol table. This is a variant of using the Global symbol table. Have you used this approach? thoughts?--Art Carpenter 01:27, 6 February 2012 (EST)

I think that the global/local argument might better be addressed as an issue of naming conventions. If your site does not have naming conventions for global macro variables that are allocated in modules == highest level of program, then, yes, I agree, Enthusiastic-Apprentices or -Novices may write (re-)allocation or assignments of global macro variables in routines, which are called by modules.

In Other Words, this isn't an issue of Sloppy Programming Happening, but of lack of (knowledge of) standards.

--macro maven == the radical programmer 15:55, 6 February 2012 (EST)

Best Practices

Writing Testing-Aware

Writing_Testing_Aware_Programs

Always suffixing a macro variable reference with a period/dot (.)

Some macro developers like to always follow a macro variable reference with a dot, e.g., &myMacVar. instead of &myMacVar. But is that a good idea? --Don Henderson 13:20, 8 January 2012 (EST)

I think you are missing a dot at the end of your first sentence, just before " But", since the one you have now will be removed by the tokenizer and you will be left with nothing to end the sentence. I think a case can be made that you would have a less chance of making this mistake if you were in the habit of using the dot all the time. :-) -- chang_y_chung 13:20, 16 October 2012 (EDT)

It's an extra keystrok most of the time. I would also mention that if one only uses a dot to end a macro variable reference when they are only going to immediatly follow it by more text it is easier to find errors when you screw something up. --User:TobyDunn 18:51, 10 January 2012 (EST)

There are three items where it is necessary:

- FileName.ext
  - %let Filename = X;
  - infile &Filename..txt;
- libref.datasetname
  - %let libref = Work;
  - %let libref = Library;
  - DATA &Libref..MyData;
- format.
  - %let format = YN;
  - NewVar = put(OldVar,&Format..);

All other use is to remember, to have a good habit, when you get to any one of those three head-bangers where it is absolutely necessary. --macro maven == the radical programmer 07:24, 11 January 2012 (EST)

Ron, I can think of a fourth item where it is also necessary, when a macro variable is necessarily followed by non-macro text. e.g., I think that the following requires two ending periods:

```
&&column&count..Character  = put(&&column&count, 6.1);
```

--Art T 15:19, 16 October 2012 (EDT)

IMHO this is convention that is incredibly counter-productive. My view is that suffixing with a period all the time is a bad idea. Better to teach the simple scanning rules *(note: I plan to write an article on that and link to it here)* so the programmers understand how macro works. The rules are not that hard and this practice actually encourages ignorance. All-in-all, this is rarely (remember I said above I don't like to say always or never J) an appropriate practice.--Don Henderson 20:03, 28 January 2012 (EST)

Don, from a non-programmer's perspective at least, I have to disagree. I'm sure that I'm not the only non-programmer/SAS user and, while I can't talk for all of us, I have always been able to get SAS to do what I need inspite of the fact that I "try" to always end macro variables with periods. I've never found a case where it hurts to do so and, on the other hand, have always found the SAS learning curve to be steep enough without unnecessarily complicating it further. I don't encourage ignorance either as I think it is quite important to know as much a possible about what is going on under the hood. However, taking the car analogy one step further, I'm glad that I can still drive my car without having to be intricately familiar with all of its underlying mechanisms. --Art T 15:31, 16 October 2012 (EDT)

Macro *booleans*

There are no real booleans in the SAS macro language. I use keyword=yes|no and the first letter of the keyword (case insenstive). For example `debug=NO` together with `%if %substr(%upcase(&debug),1,1) ne N %then %do;`. In my opinion, `debug=no` is more readible than e.g. `debug=1` and easy to program while still resulting in either true or false.--John Hendrickx 09:10, 11 January 2012 (EST)

There are no real booleans in the SAS macro language. Agreed. YeahBut what is Readable? and what supports Readability?

- `%if %substr(%upcase(&debug),1,1) ne N %then %do;.`
  - YeOwww!
  - you have never met someone who cannot stand double negatives?!
- use zero or one:
  - `%Let Testing = 0;.`
  - `%Let Testing = 1; .`
  - `%if &Testing. %then %do;.`

--macro maven == the radical programmer 15:26, 11 January 2012 (EST)

I also don't really like Yes/No - for reasons similar to the above. What I do with macro parameters is to make them toggles so any non-blank value implies yes/true and a null value implies no/false. Inside my macros I will convert them using something like:

```
%let optionName = %length(&optionName);
/* or perhaps clearer */
%if %length(&optionName) = 0 %then %let optionName = 0;
%else %let optionName = 1;
```

and I can then take advantage of the fact that 0 is false and any other value (e.g., 1) is true and use something like:

```
%if &optionName %then
%do;  /* option is turned on */
    ....
%end; /* option is turned on */
%else
%do;  /* option is turned off */
    ....
%end; /* option is turned off */
```

--Don Henderson 21:27, 28 January 2012 (EST)

Defining Macros Inside other Macros? Knot!

Many programmers with a background in procedural programming (e.g., using routines and subroutines), define macro systems by defining macros inside other macros. Is this a good idea? --Don Henderson 13:23, 8 January 2012 (EST)

This is never a good idea, it forced the inner macros to always get redefined with each call of the outer macro. Aside from the inefficiencies it also makes it reading, understanding, and debugging code a nightmare. Suppose Macro B is defines inside of Macro A, anyone reading and/or maintaining the code has to know that Macro B's definition is defines in Macro A. Further more if a problem arises they have to change the definition for macro A as well Macro B just to change how Macro B functions. Anyone doing this has no concept of cohesion and coupling in the programming world. --User:TobyDunn 18:47, 10 January 2012 (EST)

Agreed, strongly: **NEVER** a good idea. It is a reeeally excellent example of Terminal Cuteness. At a recent RUG, where I was sitting as Code Doctor, I had a user bring me a huge program -- nicely indented, btw -- with macros nested 3 levels deep. Whatever editor he used expanded tabs to 8 spaces, so the right margin was filled with white space and we had to horizontally scroll to the right to see the code. His request: How to optimize these macro calls? They run fine in testing, but begin to slow down in production. solution: removing all the nested macros definitions outside, had a dramatic effect.

How about commenting on this structure:

- %macro InnerA; ...;%mend;
- %macro InnerB; ...;%mend;
- ...;
- %macro InnerZ; ...;%mend;
- %macro TheBigOne;
  - %InnerA
  - %InnerB
  - %InnerC
  - %mend TheBigOne;
  - %TheBigOne

What I am pointing out above, is that none of the macros named Inner? are used more than once.

--macro maven == the radical programmer 15:58, 11 January 2012 (EST)

I pretty much agree, but given that I try to avoid Never/Always, permit me to say that I find it hard to imagine a case where this is a good idea. Though about 20 years ago someone almost convinced me for a particular instance of a recursive algorithm.

This is related IMO to the issue of storing macros - see below. --Don Henderson 22:00, 28 January 2012 (EST)

Macro Code Length

It has been proposed that a macro should be modular in nature, and therefore it should be short. If it is not short it should be broken up into its modular components. Is this approach beneficial? What is a proper target length? Is one page (about 60 lines) a good target length? --Art Carpenter 01:33, 6 February 2012 (EST)

Coding Style

What coding style elements do you consider especially important?

- End macro variable names with a dot.
- Comment style within a macro /* */ or %* ?
- Always name the macro on the %MEND statement?
- Thoughts on code indentation.
- Prohibition on the use of i, j, k etc. as loop indexes.

When performing list processing which type of list do you find easiest to implement and maintain?

- &&var&i within a %DO of some type
- &varlist parsed with some type of a %SCAN
- call Execute from within a DATA step - thereby avoiding the creation of the list.

Macro quoting techniques.

- Do you select quoting functions on a case by case basis or do you always reach first for a specific function? Which one?
- When using macro functions that return text e.g. %LEFT should one start with the function or its quoting analogue (%QLEFT)?
- Do you have any strictures on passing special characters like: "Never store quotes in a macro variable".?

When Calling a macro are these important considerations?

- Never follow the macro call with a semi-colon, unless the semi-colon is needed in the resolved statement
- Always follow the macro call with parenthesis as in: %ABC() Even when there are no parameters

Macro Storage

Where should macros be stored/saved?

- In a defined autocall directory?
- In a program that uses them?
- Should multiple macros be defined in a single file?
- Autocall vs Stored Compiled libraries
  - advantages
  - disadvantages

There likely isn't one answer that fits every scenario. --Don Henderson 13:18, 8 January 2012 (EST)

I use a AutoCall directory, however I am not adverse to a in the program if there arent too many other wise it clutters up the program. I do not think and have no seen a shop who has successfully implemented a way of holding all their macros in one files. What happens is every program any one writes at the shop always %Includes the one base macro file. FIrst in inefficient to compile all those macros when you only may need one or two of them, secondly its inefficient when you have someone else looking over or maintaining the code and nary a single macro is used but they are included in the program. --User:TobyDunn 18:47, 10 January 2012 (EST)

- multiple macros in a single file?

see C:\Program Files\SASHome\SASFoundation\9.3\core\sasmacro\annomac.sas which contains 40 macro definitions. My opionion is there is no good nor bad about many macros are in a single file; when we write a suite of tools that are provided for a task, simple is better, and specific is good, too.

NOTE: that none of those 40 macros are nested.

--macro maven == the radical programmer 09:32, 11 January 2012 (EST)

Pointing to Where Macros are Stored

How do you make sure your programs can find the macros it uses? Autocall vs %include? --Don Henderson 13:20, 8 January 2012 (EST)

I doubt it makes much a difference either way, my personal preference is with AutoCall. To me it always makes the code look cleaner, which the look and feel of the code goes a long way in peoples ability to understand what you created. --User:TobyDunn 18:53, 10 January 2012 (EST)

Make it easy on yourself and your maintenance: use autocall.

see also: SASautos_Companion_Reusing_Macros

--macro maven == the radical programmer 07:28, 11 January 2012 (EST)

I use autocall libraries, one for general macros, one for project specific macros. --John Hendrickx 08:48, 11 January 2012 (EST)

During development I always use %include and normally leave it that way. If I make a change to the macro I don't need to restart SAS for the changes to be reflected. I also like having the filepath and filename to the specific macro I use explicitly stated in the program. It makes it clear which version I'm using. --Steve 12:09, 11 January 2012 (EST)

I pretty much agree with John and would expand upon that a bit:

- always use autocall
- have different directories for generic macros and separate ones for project/component macros
- use concatenation to define all the appropriate macro directories
  - project specific macro libraries are listed first - that provides an easy way to handle the ***rare*** case where a specific application needs a slightly different macro.
- use the same concatenation approach for Dev/Test/Prod as appropriate
- ***ALWAYS*** use lower case file names so macros can easily be moved from Windows to Linux to Unix, etc.
- Each file should have one, and only one macro.

--Don Henderson 21:45, 28 January 2012 (EST)

Sharing Macros with your Friends and Neighbors

How are people working with macro libraries in a regulated environment where everything needs to be validated?

If I use a macro from a library, and I didn't write it or validate it, how can I be sure that it is working correctly for what I am doing?

How will I know if the macro has been modified since the last time I used it?

If an error is found, how is this communicated to a big team? Can this be handled like a product recall? --Otterm1 12:27, 11 January 2012 (EST)

Do we really need macros?

Here are the reasons one might Really Need macros:

- %do loops
- %if for conditional execution
- optimization: saving compile time

Replacing macro %do loops

This page has code for reading a list, a control data set, and generating calls to a parameterized %include.

Call_Execute_Parameterized_Include

Converting that routine to build a call to a macro, while non-trivial, is left as an exercise for the Alert Reader.

... or ask for CallXMacro.

Replacing macro %if

Conditionally_Executing_Global_Statements

and the first question I had at NESUG when I presented this was: "Do you recommend doing this, All The Time!?"

No, one attends SAS User Group conferences to learn that there are other ways to do what you already know how to do.

Optimization: Saving Compile Time

Saving lots of statements, or steps, in a macro can save compile time for either or both of:

- often-used routines
- large routines

YeahBut, pay attention to the Table-Top Rule: if a program is more than 10 pages ~== 500 lines then find the Murphy's Law that applies to it:

http://www.murphys-laws.com/murphy/murphy-computer.html

--macro maven == the radical programmer 15:47, 6 February 2012 (EST)

Parameterized %Includes: Do we really need macros?

... or can we quit development when we have working code, and use parameterized %include programs?

Parameterized_include_file

I rest my case with this SGF.2008 Best Paper: SmryEachVar_A_Data_Review_Suite

--macro maven == the radical programmer 16:02, 11 January 2012 (EST)

I am not convinced that this is a worthwhile concept. In fact, I go in the opposite direction and use macro with autocall facility even if my macro has no conditional capabilities. I don't see the advantage of using a more obtuse syntax:

- %let for each parameter
- %include to invoke the macro

vs a simple macro call where the parameters are clearly specified as part of the macro call.

And there is also the issue that macros can easily be enhanced with new functionality/parameters where assigning a default preserves current behavior.

I think the approach of using %let with %include is far more error prone and frankly don't see any advantages to it - but this should make for an interesting discussion during the presentation of this topic. --Don Henderson 21:52, 28 January 2012 (EST)

Categories: Presentations | SAS Global Forum 2012 | PharmaSUG 2015 | Macro Language | Best Practices | Donh Papers and Presentations | ArtCarpenter Papers and Presentations | Macro Language Papers

---