**Paper 1325-2017**

# Learn SAS® programming features to step up toward team management

Peter Crawford, Crawford Software Consultancy Limited

## ABSTRACT

Managing your career future involves learning outside the box at all stages. The next step is not always on the path we planned as opportunities develop and must be taken when we are ready. Prepare with this paper, which explains important features of Base SAS® that support teams. In this presentation, you learn about the following: concatenating team shared folders with personal development areas; creating consistent code; guidelines for a team (not standards); knowing where the documentation will provide the basics; thinking of those who follow (a different interface); creating code for use by others; and how code can "learn" about the SAS environment.

## INTRODUCTION

OK once we have established that we are SAS programmers, the next step is preparing for promotion!

What will we need and want to learn?

There are a variety of aspects of Base SAS programming which come into play once you want to direct the work of others ~ the kind of topics that mean "house rules for code", "how to test metadata" and "who to ask"

This paper describes features of SAS® software that will become more important when you take team responsibilities like ~

- code review
- validating code and metadata
- integration testing

.

## INTEGRATION TESTING

### CONCATENATE TEAM SHARED FOLDERS WITH PERSONAL DEVELOPMENT

### SEPARATE THE LOGICAL AND PHYSICAL

Integration Testing needs the cooperation of all developers within the team. It helps when testing to use methods that can scale up from, unit testing for one developer, to full testing of applications. This is not only about administering a process. We can take advantage of features in SAS that help to hide the physical and logical boundaries between the storage areas and the objects used by separate developers. Although the SAS Metadata Server provides all the administrative support required, it is useful to understand what is going on when the "physical layer" is associated with a logical layer.

SASHELP.CLASS is a logical reference to a data set. The "SASHELP" is a library reference. It is associated with the physical location of that data. The SASHELP reference is established when a SAS session starts – from details in a SAS configuration file. The physical to logical association of application data and code can be implemented before, during or after a SAS session starts. The basic SAS feature that achieves this is a LIBNAME statement. For code libraries, a FILENAME statement provides, a similar association. This is not always required, but is very helpful for team-work. This association layer enables a common approach for testing a single unit, or multiple components together. It can scale up to a full system test of applications.

The SAS feature is concatenation of logical references, for example:

```
libname prod1 "production path is in sight but locked" access= readonly ;
```

```
libname test1 "testing path that is updateable" ;
libname DATA ( test1 prod1 ) ;
```

The LIBREF, DATA, concatenates two logical references to provide write access to a testing area, and read access to the PROD1 area for tables not already written to the testing path.

Concatenation of logical references to file folders is a little more complicated.

```
Filename teamcode "authorized changes folder" ;
Filename mycode   "my development folder" ;
```

These application code libraries cannot be concatenated in the same way. Unfortunately,

```
filename code( mycode teamcode ) ;
```

does not work!

The FILENAME concatenation, must be of quoted, physical references. Only slightly more complex than the solution for concatenating LIBNAME – logical references can be converted back to physical references with a small macro – see Appendix 2.

```
filename code( "%path(mycode)" "%path(teamcode)" "%path(prodcode)" ) ;
```

provides the required logical view of code in these areas.

When the application executes, all code should be invoked with references like:

```
%include code(thatprog) ;
```

The concatenation ensures that when the %INCLUDE above does not find the file "thatprog.sas" in the MYCODE code library, then the %INCLUDE will check through the code library TEAMCODE, and if not found there, then through PRODCODE.

In this way, having application code with logical but no physical references, we can provide a transparent way to test from single units, through to a whole system, with no code changes needed to pick up special versions of code. The logical assignments can be made in configuration files for the individual, team or production system. Another benefit – the developers' code folder needs only his own programs under development – not a complete copy of the system.

As libraries SASHELP, SASUSER and WORK must be available for a SAS session to start, these cannot be assigned with LIBNAME statements but are prepared as "environment variables" using SET statements in a SAS configuration file.

For a description of this alternative to using LIBNAME or FILENAME statements to assign libraries and files, see the related documentation "Using Environment Variables as Librefs in UNIX Environments" at: http://support.sas.com/documentation/cdl/en/hostunx/69602/HTML/default/viewer.htm#n08yhte7x1xqirn1hjqdcffztfq2.htm

For documentation indicating the use of environment variables as a LIBREF for SAS running on Microsoft Windows platforms, see the SAS-data-library paragraph in the SAS on-line doc at :http://support.sas.com/documentation/cdl/en/hostwin/69955/HTML/default/viewer.htm#chloptfmain.htm

## CREATE CONSISTENT CODE

### GUIDELINES FOR A TEAM (NOT STANDARDS)

Whatever my preferences, I must create code acceptable to my team :

> There might be a brief opportunity to enhance team standards by introducing external ideas from prior professional experience – When not my primary responsibility, this opportunity will be very brief.

For team cohesion, coding style must accommodate what the team needs. "Snippets" in SAS® Studio and "Abbreviations" in SAS® Enterprise Guide® are not only a time saver. As these can be customized they can provide code layout for SAS Procedures and skeleton DATA steps, in the guideline style for the team. Because Snippets are stored as files, they can be distributed or stored in a shared location.

For on-line documentation of the Enterprise Guide® “Abbreviations”, see http://support.sas.com/rnd/base/topics/abbreviations/. As the article says “You can share abbreviations with others by exporting and importing them as a KMF file”. Place these Enterprise Guide® Editor macro files in a network share for all developers to import and move to standardized style in code nodes.

For SAS® Studio “snippets” see “Working with Code Snippets” in “SAS® Studio 3.5: User's Guide” at: “http://support.sas.com/documentation/cdl/en/webeditorug/68828/HTML/default/viewer.htm#n002bgjjez3z63n16y88kjsobrha.htm

Both Enterprise Guide® and SAS® Studio editors have options to format code. Even if that formatting is not to your preference, it does ensure a consistent style and that is perfect for a code review.

## THINK OF THOSE WHO FOLLOW (A DIFFERENT INTERFACE) – CREATING CODE FOR USE BY OTHERS

A separate paper will go into this issue in more depth. We must accept the compromise that a team needs integration almost as much as testing, so preferred personal style must take second place to acceptance by others in a team.
In addition to needing a clear style (see prior points), when creating code for others, we need to accept that we cannot predict the environment in which the code might need to execute.

For stored processes, and macros, to be executed (safely) by those who are not the creator of the process, a surprising amount of “protection” might be needed. Not only must the macro execute without creating syntax errors (of course, these will only occur with unexpected parameters), the users’ and customers’ environments must be protected from the less obvious changes that a macro might make (for example replacing lib- or file-refs, or deleting tables, files or formats) when these are not the purpose of the macro. One approach might be to ensure the customer knows that these actions will occur. Unfortunately, this is something I would be very unhappy to delegate to third parties so I recommend methods to avoid such effects - as in the paper. Paper 0872-2017 Learn to Please: Creating SAS® Programs for Others.

## WHERE IS THE DOCUMENTATION?

Of course, it is all on-line – SAS documentation can be searched more effectively than we can read through it all. However, as we step forward from “programming to deliver results”, to “programming to deliver a program” we need to provide more. Even when limited to a header in the program there are a surprising number of issues to document. This is my list – your manager’s list will probably be different, but here are some of the questions I have when confronted by another’s code:

- prerequisites for a process (like data, libraries and permissions)
- limitations presumed
- impact – like files and tables, created or updated or deleted
- impact – like a large Cartesian sql join taking significant runtime and utility/work space
- any assumptions about the validity of parameters

When your responsibilities involve development and operational support, you might need to create and share this kind of information in a form more structured than a program header. It becomes as important as the definition of the interdependency of processes in a business application, along with the “touch points” where customers interact with the application. You might want that header standardized so that a process can read through all code to collect this documentation as new metadata.

## HOW CODE CAN "LEARN" ABOUT THE SAS ENVIRONMENT

Metadata of all sorts are always being used in a SAS session. Some of this lies hidden – just waiting to be used. I expect there are metadata constantly in use that are still to be discovered by most users.

There are helpful functions and metadata:

## FUNCTIONS

The function mentioned earlier, PATHNAME(), provides the physical path for a logical reference (for example, library names like WORK, SASUSER and SASHELP). When the physical is concatenated, the function returns the paths within parentheses. Appendix 2 shows a small macro to streamline the use of this function.

The GETOPTION() function returns the value of a SAS system option or a SAS Graph® option.

To capture these values within the macro language environment, this approach works:

```
%macro msgs( msg= default message  surrounded with hyphens ) ;
  %local linesiz ;
  %let   linesiz= %eval( %sysfunc( getoption( ls )) -1 ) ;
  %put NOTE- %sysfunc( repeat(%str(-), &linesiz )
      )NOTE: &msg  %sysfunc(
                       repeat(%str(-), &linesiz )) ;
%mend  msgs ;

option ls= 80 ;
%msgs ;
%msgs( msg= %sysfunc( datetime(), datetime ) time-stamped message ) ;
```

The macro %MSGS captures the LINESIZE option (via documented abbreviation, LS) simply to highlight the message by printing a full line of hyphens before and after the message. A single %PUT statement can be used and the “log line-overflow” routines will format the whole. As the REPEAT() function generates 1 plus the repeat-count of hyphens, the code has to subtract one from the LINESIZE value. Results appear in the log showing in Figure 1.

Figure 1 Log Output from %msgs Macro Call

```
61         %msgs ;

-------------------------------------------------------------------------------
NOTE: default message  surrounded with hyphens
-------------------------------------------------------------------------------
62         %msgs( msg= %sysfunc( datetime(), datetime ) time-stamped message ) ;

-------------------------------------------------------------------------------
NOTE: 20JAN17:10:12:57 time-stamped message
-------------------------------------------------------------------------------
63
```

**Figure 1 Log Output from %msgs Macro Call**

### METADATA IN SASHELP

A quick hint of the range of metadata tables available can be gleaned by browsing the SASHELP library entries down to, and through, the letter ”V”. Among the 48 I listed, just 3 were not SAS metadata. Although you might be familiar with VTABLE and VCOLUMN, have you used VMACRO, VFORMAT or VFUNC? Try:

```
proc sql ;
   describe view sashelp.vformat ;
   describe table dictionary.formats ;
quit;
```

**OTHER IN-SESSION METADATA**

## Code Audit

If you want to verify the versions of code which execute in a production process, would you know where to start?

Next time, persuade administration colleagues to include system options ECHOAUTO, OPLIST, SOURCE, SOURCE2, MPRINT and MAUTOLOCDISPLAY when the SAS job starts. These options are always worth revising in the on-line documentation, even if you are not involved in auditing SAS processes.

## Avoiding Names Already In Use

When choosing a name for a new SAS data set or macro, not only local standards and guidelines must be respected. We do not want to find our chosen name will be over-ridden by a macro that is already present. We need a list of these macros. The DICTIONARY.MACROS metadata provides no list of macros – only macro variables. We can get a list of the compiled macros in a SAS session from DICTIONARY.CATALOGS:

```
proc sql print number ;
   select * from dictionary.catalogs
   where objType = 'MACRO'
   order by 1,2,4 ;
quit;
```

The results reveal that there are compiled macros in the SASHELP library. Unless you run an application that uses the catalog SASHELP.SASMACR (as a catalog of pre-compiled macros), you do not need to avoid these macro names.

I discovered that the SAS University Edition uses more than a few macros. It is wise to avoid using those names for any of our own macros. Your own environment might involve a different list.

The SAS system provides a simple way to write a new WORK data set, with a dynamically assigned name. _DATA_ avoids overwriting any existing tables. (From SAS Global Forum 2016, see paper 2120-2016 “More Hidden Base SAS® Features to Impress Your Colleagues”).

Unfortunately, there is no convenient equivalent for macros, user-formats and user-functions. The use of the &SYSINDEX automatic macro variable might help, but the name with the generated number might already be in use.

It is possible to create a macro with a name assigned in a dynamic way, so it will not replace a pre-existing macro. For an example see Paper 0872-2017 Learn to Please: Creating SAS® Programs for Others.

However, it is/seems impossible to predict what macros might “open later”. A program described in the paper referenced above 2120-2016, sought to discover all (600+) macro names that could be compiled from the source folders provided when SAS is installed - defined in the SASAUTOS macro code libraries. The searching code was designed to cope with user macro libraries, too. Although fairly comprehensive, it does not allow for the “as yet unknown” - macros that are still to be compiled from in-stream code rather than through auto-call (SASAUTOS) routines.

One catch-all solution was proposed many years ago on SAS-L (Dorfman 2009) – generate a random number-based name. This would not be bound to the limits of one SAS numeric integer, as a name can use 32 characters selected from any of 37 letters digits and the underscore character (ok the first character cannot be a digit):

```
unique_valid_names = 27*( 37 **31 ) ;
```

The resulting value is 1.1107522719738**E50** – expressed in COMMA format style, the value would need 15 or 16 commas.

By the nature of random numbers, the unexpected will happen – but with that many to choose from, it seems unlikely that a failure would repeat. I feel fortunate that I do not need to provide this level of guarantee that my code will not be in contention with yours'.

As a final reflection on this point, I would like to quote from Paul Dorfman's SAS-L posting (Dorfman 2009) where he puts into context the scale that a fully randomized name, conforming to the usual rules of column names defined in documentation of the SAS system option VALIDVARNAME=V7;

> *…… it would be practically impossible for them to collide with anything else.*
>
> *To wit, if you randomly form a 32-byte name using all 37 valid SAS name characters ……. then you are selecting a variable name randomly from the pool of 37**32=1.5*50 names.*
>
> *Since the estimated age of the Universe is about 10**18 seconds, the probability of observing a collision between 2 names in the pool would be about one-quintillionth if 100 trillion names were selected each second during the existence of the Universe.*
>
> *Or, to have a certain chance to encounter a collision, one would have to create 10e+32 names every second and observe the process during 100 trillion years.*

## CONCLUSION

There are many features in Base SAS to support working in teams. It should be part of our early learning when we look to progress in our career. There are more than this short paper can completely cover. Please follow up the references and recommended reading, for clarification and in at least one, your entertainment.

## REFERENCES

Crawford, Peter. 2016. "More Hidden Base SAS® Features to Impress Your Colleagues", *Proceedings of the SAS Global Forum 2016 Conference*>, Las Vegas, NV: SAS Institute. Available at http://support.sas.com/resources/papers/proceedings16/2120-2016.pdf

Dorfman, Paul. 10Aug2009. "Re: Coupling and cohesion in SAS macros", SAS-L Archives, Available at https://listserv.uga.edu/cgi-bin/wa?A2=SAS-L;326d32a1.0908b .

## RECOMMENDED READING

- *Step-by-Step Programming with Base SAS(R) 9.4,*
- *SAS® For Dummies®*

## CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Peter Crawford
Crawford Software Consultancy Limited, UK
CrawfordSoftware@gmail.com

For a list of downloadable papers and presentations, see
http://www.lexjansen.com/search/searchresults.php?q="Peter Crawford"

Appendix 1

Posting copied from SAS-L

**Subject:** Re: Coupling and cohesion in SAS macros
**From:** Paul Dorfman <sashole@BELLSOUTH.NET>
**Reply-To:** Paul Dorfman <sashole@BELLSOUTH.NET>
**Date:** Mon, 10 Aug 2009 16:27:48 -0400
**Content-Type:** text/plain
**Parts/Attachments:** text/plain (155 lines)

Reply

```
Matt,

As you may imagine, in the many years of SAS-L, this is not the first time
the question was raised. I recall a lively discussion on the matter some
11-12 years ago; you can unravel it by tugging on the following threads,
for example:

http://www.listserv.uga.edu/cgi-bin/wa?A2=ind9810A&L=sas-l&P=R782
http://www.listserv.uga.edu/cgi-bin/wa?A2=ind9812C&L=sas-l&P=R6053
http://www.listserv.uga.edu/cgi-bin/wa?A2=ind9912B&L=sas-l&P=R12635
http://www.listserv.uga.edu/cgi-bin/wa?A2=ind0502D&L=sas-l&P=R28725

The basic idea there was that if you need to develop a macro called from a
DATA step and worry about its internally used "local" variables clashing
with other DATA step variables, then first write the routine as you
normally would (with any meaningful variable names you want) and test it,
then at the time of call automatically replace their names with auto-
generated random variable names so long that it would be practically
impossible for them to collide with anything else.

To wit, if you randomly form a 32-byte name using all 37 valid SAS name
characters (and many more if validvarname=any is in effect), then you are
selecting a variable name randomly from the pool of 37**32=1.5*50 names.
Since the estimated age of the Universe is about 10**18 seconds, the
probability of observing a collision between 2 names in the pool would be
about one-quintillionth if 100 trillion names were selected each second
during the existence of the Universe. Or, to have a certain chance to
encounter a collision, one would have to create 10e+32 names every second
and observe the process during 100 trillion years.

A simpler idea, which I have also used in practice, is to use meaningful
internal names, assign a random name to each, and then use their macro
dereferences instead of the names themselves. For instance, if you intend
to use variables names A, B, C, you would pre-form

%local a b c ;
%let a = _%sysfunc(putn(%sysevalf(%sysfunc(ranuni(0))*1e31),z31)) ;
%let b = _%sysfunc(putn(%sysevalf(%sysfunc(ranuni(0))*1e31),z31)) ;
%let c = _%sysfunc(putn(%sysevalf(%sysfunc(ranuni(0))*1e31),z31)) ;
drop &a &b &c ;

and then use &a, &b, &c in your code as if they were a, b, c.

Kind regards
------------
Paul Dorfman
Jax, FL
------------
```

## APPENDIX 2 MACRO %PATH

```
%macro path( logical_ref) /des='return the physical path' ;
%sysfunc( pathname( &logical_ref ))
%mend  path ;
```