Paper 1408-2015

Learn Hidden Ideas in Base SAS® to Impress Colleagues

Peter Crawford, Crawford Software Consultancy

ABSTRACT

Across the languages of SAS® are many golden nuggets—functions, formats, and programming features just waiting to impress your friends and colleagues. Learning SAS over 30+ years, I have collected a few, and I offer them to you in this presentation.

INTRODUCTION

The programming language of the base SAS System™ has many capabilities that are like hidden gems, waiting to be discovered and making the SAS System extremely flexible.

Make these gems part of your early learning to accelerate your path to professional competency – and it is never too late to teach an old "code dog" new SAS tricks – I'm still learning (*after* a career) - I call it continuing professional development.

Often, I would search for the solution to a problem, only to find that I am searching with the wrong words. As problems got harder I found base SAS still has the answer, as if by magic.

Most of these gems and features are fully documented parts of SAS but do not appear among typical learning as they are not part of the beginners training courses, despite their simplicity.

Perhaps course developers found there was too much content in the Base SAS language to introduce all the gems at foundation and essential levels. Even with a simplified introduction, most students feel fully empowered.

This paper provides a list of some of the "hidden" gems and goes on to explain their "magic" in detail and demonstrate with simple code.

While the paper is intended for those new to SAS programming, there will be new ideas for all expertise levels of SAS Programmer and SAS Developer.

MORE MAGIC GEMS

A QUICK DOZEN

1. ! - use environment variables by name (prefixed with !)
2. SAS Libraries don't always appear on the on-screen lists
3. Functions, functions and functions- so many (even one to validate any kind of SAS name)
4. SAMPSIO a collection of sample datasets available on your SAS platform
5. Library list – list the definition(s)
6. Align the text formatted by PUT statement and function
7. Put your name in SAS highlights (well, on the title of your batch SAS logs)
8. Cut the clutter (repeated lists) of variable name throughout your code, using SAS variable lists
9. Aggregate storage references help code become flexible, reuse able and shareable
10. Make cross-platform data reading simpler with a familiar INFILE option
11. More easily, add your personal (or team-) libraries to the lists provided (for macros, formats, user-functions) – and avoid the trouble that occurs when you replace the relevant system option value
12. Enjoy adding, to simplify your SAS language programs with more (and simpler) macros

1 !NAME

Although not well known, the convenience that this feature provides becomes apparent when seen in action.... in the next few tips and here like.....:

```
Filename tempfile '!temp/x.x';
```

or

```
File '!temp/x.x' ;
```

or

```
data _null_ ;
inFile '!sasroot/?*.cfg'   ;
input ;
list ;
run;
```

Testing on windows platforms, not always are we allowed to write to C:\temp but the location referred by the environment variable !TEMP can probably be trusted.

Another common location, !USERPROFILE is the parent of “My documents”.

The third example above, lists the location, name and contents of the files of type “,cfg” in the folder where sas.exe is installed.

2 SAS LIBRARIES DON’T ALWAYS APPEAR

We are able to use “hidden” libraries of programs or data sets. With this feature, we use a LIBREF without pre-defining it in a LIBNAME statement. The assignment is made automatically when we use it.

For this “trick” to work we must have an environment variable pointing to a folder of SAS datasets. Then we can use the name of that environment variable as a LIBREF – as long as the name of the environment variable is valid as a SAS version 6 “name”.

Another (minor) constraint – the engine for the library must be able to be discovered during assignment, from the type of objects found in the folder. That excludes libraries you might like to connect to external databases through SAS/ACCESS®. An empty folder would default to the Base Library Engine.

Of course, once an environment variable is assigned as a LIBREF, it will no longer be hidden, and will appear among the libraries assigned in more-normal way.

Until they “appear” these hidden libraries cause no overheads - adding no metadata among the dictionary tables of column, table, member and other object information. Often there is a lot of metadata for assigned libraries.

As well as data, the feature can work for program code, as a later Gem will show.

3 FUNCTIONS, FUNCTIONS AND FUNCTIONS

There are many functions available in Base SAS. The challenge to know them all is practically impossible. There is a dictionary table of the functions. Using this dictionary table, I can say that at time of writing this paper, there are 929 functions listed in my base SAS v9.4 system. This table provides a convenient list of names to avoid when you want to add your own user defined functions.

Among the built-in functions there is even one to help validate four types of SAS name. See Base SAS function NVALID():

```
return_code = NVALID( string, type )
```

The first parameter is a string to be tested.

The second parameter signifies

- V7 = 32 character wide name of letters and underscores and (after the first) digits
- ANY = up to 32 wide, allowing any characters, but needs to be quoted as a name literal. When a variable name contains a quote it needs to be handled very carefully as a “name literal”.
- nLiteral = for verifying a quoted string with the trailing N makes a valid variable name (it could fail for having the length too wide)
- V6 = suitable as SAS version 6 name – still relevant for LIBREF and FILEREF (but not documented)

See the example that follows.

This DATA step demonstrates the NVALID function (results appear in the appendix):

```
data namechecks ;
  input string & :$40. ;
  ln = length( string ) ;
  nn = nvalid( string ) ;
  na = nvalid( string, 'any' ) ;
  nl = nvalid( string, 'nliteral' ) ;
  n7 = nvalid( string, 'v7'  ) ;
  n6 = nvalid( string, 'v6'  ) ;
  length lit $40 ;
  format ln -- n6 3. ;
  lit= nliteral( string ) ;
 list; datalines;
  qwer5t6y
 qwer5t6y1
 1wer5t6y
 _2345678
  qwer5t6y qwer5t6y
  qwer5t6y'qwer5t6y
  a234567890123456789012345678901 2
  a2345678901234567890123456789012
 'a2345678901234567890123456789012'n
 'a2345678901234567890123456789012'n
 "qwer5t6y"n
;
```

If you have to generate a name literal from a general string, the following macro-type code should help:

Assuming you have the candidate string in a macro variable named `yourMvar`:

```
%let nameLit = %sysfunc( quote( %qsysfunc( trim(%qsysfunc( substr(
               %superq(yourMvar)%qsysfunc( repeat(%str( ),30)),1,32)) ))))n
;
%put %quote(&nameLit) ;;
```

Or you could use the new(-er) function NLITERAL() like:

```
%let name= %sysfunc( nliteral( %str( 1asdfgh 2#/ uiop 56 )));
```

This generates a name literal but, with the proviso that it doesn’t validate results – see the appendix.

Of course there are so many more functions added in recent releases that it might be difficult to keep up to date. See the section on changes and enhancements in the latest version of documentation. SAS® 9.4 Functions and CALL Routines: Reference, Third Edition: is latest at time of writing.

4 SAMPSIO

SAMPSIO refers to a collection of SAS datasets that support sample programs. These are waiting to be discovered (or, if you are a space administrator, deleted ;). Although this LIBREF is not (normally) on show, being defined as an environment variable, SAMPSIO is easily added to your environment. Just submit the code:

```
libname sampsio list ;
```

After that code runs, all normal listings of libraries will include SAMPSIO, and its objects will be included among dictionary tables and dataset and catalog lists.

Even before making it appear, we can use it:

```
proc sql ;
create table sample_datasets as
      select libname, memname, nobs, nvar
        from dictionary.tables
       where libname = 'SAMPSIO'
    order by 1,2
           ;
quit ;
```

Even if that library is not yet in use, SAS will find it.

Remove the assignment (but not the data) with the code:

```
libname sampsio ;
```

5 LIBRARY LIST

As referred above, the LIST option of a LIBNAME statement provides a report in the SASlog of the engine and path(s) assigned to the LIBREF.

. On a system with just base SAS and SAS/GRAPH® the log shows:

Figure 1

```
23         libname sampsio list ;
NOTE: Libref=   SAMPSIO
      Scope=    IOM ROOT COMP ENV
      Levels=   2
        -Level 1-
      Engine=   V9
      Physical Name= C:\Program Files\SASHome2\SASFoundation\9.4\core\sample
      Filename= C:\Program Files\SASHome2\SASFoundation\9.4\core\sample
        -Level 2-
      Engine=   V9
      Physical Name= C:\Program Files\SASHome2\SASFoundation\9.4\graph\sample
      Filename= C:\Program Files\SASHome2\SASFoundation\9.4\graph\sample
```

Figure 1 Library LIST in the Log

Extra feature: this LIST feature supports _ALL_ for the library name. Then the log will report these details for all assigned libraries.

6 ALIGN THE TEXT FORMATTED BY PUT

Many times I have seen less-experienced programmers use the LEFT() function around a PUT() function in code like:

```
call symput( 'mvar' || compress(left( put( _n_,6.))), value ) ;
```

Of course we all now use

```
call symputX( .............. ) ;
```

but although the newer version of the SYMPUT call routine will convert and strip the second parameter, we still need some support to ensure the concatenation in the first parameter has no blanks embedded from the front of the _N_ value.

The newer concatenation functions which would help here (convert values and trim blanks), will be well known.

Less known is the alignment directive available with PUT() function.

For a long time now, the base SAS language has allowed us to align the results of a PUT with a format modifier. The documentation is at
http://support.sas.com/documentation/cdl/en/lefunctionsref/67398/HTML/default/viewer.htm#n0mlfb88dkhbmun1x08qbh5xbs7e.htm

The default alignment is left for string formats and right for numeric formats. This feature allows you to override, to left, right or center the results – but only within the format width requested. The feature works in PUT() functions as well as PUT statements, like:

```
call symput( 'mvar' || put(_n_,6.-L), value ) ;
```

For this simple case, both COMPRESS() and the LEFT() functions have been eliminated with the format modifier.

In the SQL Procedure this alignment does not work as a modifier of formats assigned to a column in a SELECT query. So this is invalid:

```
select name format= $20.-R from sashelp.class
```

but:

```
select put( name, $20.-R ) as name from sashelp.class
```

Is valid. See the appendix for the results.

This shows that if the selected column value is generated by a PUT() function then the results of format alignment can work in PROC SQL.

7 PUT YOUR NAME IN SAS HIGHLIGHTS

(well on the title of your batch SAS logs)

The LOGNOTE system option allows you to replace “The SAS System” where it appears on the SAS log for batch jobs. In this way administrators can put contact details into batch jobs to link support. When you find it convenient to run jobs in batch, it adds your “chance to star” at the top of every page of that batch SAS log. It is not defined in your program but among options defined as the SAS session that runs your program starts – so it easy for administrators to maintain – and you need not always be “on call” for your program..

8 CUT THE CLUTTER (REPEATED LISTS) OF VARIABLE NAMES THROUGHOUT YOUR CODE

- use SAS variable lists

There are many of these “variable lists” - not only the lists we create with PROC SQL, like:

```
proc sql noprint ;
   select name into :oldest_boys separated by ' '
     from sashelp.class
    where sex='M'
```

```
    having age= max(age)
          ;
    select name into :oldest_girls separated by ' '
      from sashelp.class
     where sex='F'
    having age= max(age)
          ;
    select name into :employ_nums separated by ' '
      from sashelp.vcolumn
     where libname='SAMPSIO' and memname='EMPINFO'
       and type= 'char'
  order by varnum
          ;
quit ;
```

Other lists exist without needing to be programmed:

In a DATA STEP:

1. _ALL_ the list includes just the columns known to the compiler at that line of the data step
2. _NUMERIC_ similar, but restricted to only numeric columns (already known to the compiler)
3. _CHARACTER_ just string variables are implied
4. Column1-column99 columns with a numeric suffix – and all names must be present
5. columnA—columnZ in the order defined to the compiler, lists columns from the first to the last
6. columnA-character-columnZ just character columns in this name range
7. columnB-numeric-columnY just numeric columns – columnA and columnZ need not be numeric

In some procedures variable lists are accepted. For example :

```
proc means data= sashelp.class ;
    var _numeric_  ;
  class name-character-height ;
 output out= stats max= ;
run ;
```

On some statements (like SET) a list of dataset names can be abbreviated to use a prefix as in:
SET AC2014: ;
This will SET all tables in the work library with names beginning AC2014.
Tables can be listed in numbered ranges

9 FILEREF - AGGREGATE STORAGE

Less obviously helpful, this FILEREF is a logical reference pointing to a folder (referred to in sas documentation as aggregate storage location) or more than one folder. This enables a shortcut like

fileref(progname)

to be used by %INCLUDE and FILE or INFILE statements. For example:

```
%include sasautos( sysrc ) /source2 ;
```

This provides very interesting reading.

Try also :

```
%inc sampsrc( formatex) /source2 ;
```

These examples demonstrate using environment variables as an aggregate storage location.

Replacing the physical definition of the “aggregate storage location” with a logical reference (the FILEREF) allows our programs to be reused with new physical storage locations without altering our code. Then processes written for one project can be reused in another which uses different physical paths, but a common FILEREF.

Allowing the FILEREF to refer to more than one physical area supports organization of code libraries, like:

- Personal development
- Team system testing
- Project
- BAU (business as usual) and production

In this arrangement, code like

```
%include proglib( report2 ) ;
```

would take either the version of the code I am testing, the team version in test, the project version or the BAU production code.

To implement change control systems for code need ways we can avoid making code changes after testing is complete.

This aggregate storage FILEREF facility provides one of the ways to add flexibility to deploy code once it has been validated.

10 EXTEND INFILE STATEMENT DLM= WITH CR ('0D'X)

.....explaining why...

When you work in a Unix plus Windows environment (as many users of Enterprise Guide® software, now do), often data transferred to the Unix server arrives with CRLF line endings. In the normal course of handling the data, SAS on Unix expects only the LF. That CR might get in the way (attached to a string value or making a numeric appear to hold invalid data). We can overcome this by recreating the input file (text transfer instead of binary). This (most sensible) option is not always possible. Of course SAS is adding INFILE options like TERMSTR to help us, but you might find these unavailable until the software release is upgraded.

One feature, which helps solve the problem of finding a trailing CR on a line, has been available since before SAS9. It is the DLM= option. This string value, or variable, lists the delimiter character(s). Notice that – it allows more than one character. This is different from a multi-character delimiting string. These characters are alternative delimiters. Each is enough to separate data values.

Just add that unwanted CR to the delimiters. It is a minor thing to add to the value when the delimiter is pre-defined, except that CR, like TAB, cannot be defined as a plain text. The hexadecimal representation of CR is '0D'x.

Any unwanted character can be ignored in this way, but beware using too many alternate delimiters with the DSD option of the INFILE statement. INFILE has been providing its own magic for more than 10 years (see paper “SUGI 28: More _Infile_ Magic” www2.sas.com/proceedings/sugi28/086-28.pdf).

11 LISTINSERTAPPEND

As a developer of program functionality, we need to test and implement our code without having a bad effect on what is there before and without having to know all about it. That is (I think) why the LIST-INSERT-APPEND feature has been added. Although these (LIST and APPEND) might not appear to be the names of SAS System options, in fact they define the action you want to make to another option.

As a developer I want to ensure that this great new user formats I have created can be found when my report program runs, and without continuously recreating the format in the work library. So the format is stored in a catalog in a permanent SAS library. The relevant option to help SAS find format catalogs is FMTSEARCH. Before this feature was added, the option FMTSEARCH could only be replaced. I cannot assume the “current value” is the default value – I might not be the only programmer adding new format catalogs. There is a way to retrieve the old value (function GETOPTION) but, even when implemented as a macro (see SAS Sample 24841: Executing Your Stored Formats at Start-Up http://support.sas.com/kb/24/841.html) extending the option FMTSEARCH in this way is inconvenient.

The features (system options INSERT and APPEND) enable us easily, to update a system option that is normally defined as a parenthesized list. In the example of a format catalog we use INSERT to place our format catalog at the front of the list (superseding formats defined previously) or with APPEND, put our formats after those already defined – guaranteeing that existing formats will not be "hidden" by our new formats..

Another system option where this feature is really useful is SASAUTOs – to add a folder of SAS macros to the "AUTOCALL environment" without having to replace the "current" option value. If you define your own functions, then you will already need this feature to add your functions to the environment.

Base SAS v9.4 will provide a list of all the options we can update with INSERT and APPEND. Just add option LISTINSERTAPPEND to the OPTIONS procedure – on the PROC OPTIONS statement.

Before these options were introduced for use *during* a SAS session, they could be used only as the SAS session started (called invocation options).

12 ENJOY ADDING, TO SIMPLIFY YOUR SAS LANGUAGE PROGRAMS WITH MORE (AND SIMPLER) MACROS

Want a quick (current) time stamp in your program?

Want to apply code for all items in a list?

Have a minor task you want to streamline?

Want to abbreviate repetitive SAS syntax?

Then you can take advantage of simple SAS macros.

As long as the macro names you adopt have adequately meaningful names (for example :

```
%macro now( fmt= datetime21.2)/ des='timestamp';
%sysfunc( datetime(),&fmt )
%mend  now ;
```

So any time I want a simple time stamp (data step compile time or macro execution time), the code is :

```
%now;
```

The parameter adds enough flexibility to present a time like the DATE option offers on TITLE lines, just provide a very precise time for calculating duration between stages of a process. There is a short paper describing this at 038-31: The Big Introduction from the Smallest Macro www2.sas.com/proceedings/sugi31/038-31.pdf

Even simple macros can have great utility for example replace:

```
%let workpath= %sysfunc( pathname( work )) ;
File "&workpath?gencode.sas" ;
```

With:

```
file "%path(work)/gencode.sas" ;
```

Anyone new to this code should not have to ask what that macro %path is doing (I hope). The macro code appears in the appendix.

Each of the macros above share a simple feature – they generate code but no semi-colons. This means they can be used within a statement (handy to put a program runtime among other text within a footnote) – or even more than once within a statement. The Common description of this type of macro is "a function macro".

The last simple function macro I'll demonstrate here has a name that is almost meaningless, but when it is used, its context clarifies its role – for example:

```
%let old_opts = %sy(mprint) %sy(mlogic) %sy(symbolgen) ;
option nomprint nomlogic nosymbolgen ;
```

```
...... various statements and steps ......
option &old_opts ;
```

It might be becoming clear that my macro SY returns the value of the system option requested. The macro is in the appendix:

CONCLUSION

There are many golden nuggets to be found among the functionality of the Base SAS language.

Probably there are far more than I could reveal in this short paper. Each of the dozen above has impressed one or more of the many SAS analysts, developers and other users of the languages of SAS that I have been lucky enough to work alongside.

I hope you will find these useful too.

ACKNOWLEDGMENTS

Many contacts established as friends on SAS-L, SAS Forums, SAS Community.org, SEUGI, SUGI, SAS Global Forum and most importantly SAS Customer Support.

RECOMMENDED READING

- *SAS® For Dummies®*

CONTACT INFORMATION

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

Name:	Peter Crawford
Organization:	Crawford Software Consultancy Limited
City, State ZIP:	London, UK, CR0 7HS
Work Phone:	+44(0)7802732254
Email:	Peter.Crawford@blueyonder.co.uk
sascommunity.org:	http://www.sascommunity.org/wiki/User:Peter.Crawford

APPENDIX 1

Results from testing a string as a valid SAS name using function NVALID()

Figure 2

VIEWTABLE: WORK.NAMECHECKS

	string	ln	nn	na	nl	n7	n6	lit
1	qwer5t6y	8	1	1	1	1	1	qwer5t6y
2	qwer5t6y1	9	1	1	1	1	0	qwer5t6y1
3	1wer5t6y	8	0	1	1	0	0	''1wer5t6y''N
4	_2345678	8	1	1	1	1	1	_2345678
5	qwer5t6y qwer5t6y	17	0	1	1	0	0	''qwer5t6y qwer5t6y''N
6	qwer5t6y'qwer5t6y	17	0	1	1	0	0	''qwer5t6y'qwer5t6y''N
7	a234567890123456789012345678901 2	32	1	1	1	1	0	a234567890123456789012345678901 2
8	a2345678901234567890123456789012 3	33	0	0	0	0	0	a2345678901234567890123456789012 3
9	'a2345678901234567890123456789012 3'n	36	0	0	0	0	0	'''a2345678901234567890123456789012 3'n''N
10	'a234567890123456789012345678901 2'n	35	0	0	1	0	0	'''a234567890123456789012345678901 2'n''N
11	''qwer5t6y''n	11	0	1	1	0	0	'''qwer5t6y''n'N

Figure 2 demonstrating NVALID() function

Notice on row 9 and 10 that function NLITERAL() might produce a string that NVALID() would reject!

APPENDIX 2

```
%macro sy( op )/ des='get system option value' ;
%sysfunc( getoption( &op ) )
%mend  sy ;
```

APPENDIX 3

```
%macro path( lref )/ des='path for logical' ;
%sysfunc( pathname(&lref))
%mend  path ;
```

APPENDIX 4

Results show that PUT() with alignment works in PROC SQL

Figure 3

FSVIEW: WORK.RIGHT_NAME

Obs	name
1	Alfred
2	Alice
3	Barbara
4	Carol
5	Henry
6	James
7	Jane
8	Janet
9	Jeffrey
10	John

Figure 3 PUT() function alignment in PROC SQL