Paper 0872-2017

Learn to Please: Creating SAS® Programs for Others

Peter Crawford, Crawford Software Consultancy Limited

ABSTRACT

Programming for others involves new disciplines not called for when we write to provide results. There are many additional facilities in the languages of SAS® to ensure the processes and programs you provide for others will please your customers. Not all are obvious and some seem hidden.

The never-ending search to please your friends, colleagues, and customers could start in this presentation.

INTRODUCTION

OK we are SAS programmers, we know how to write SAS programs and we start to produce results. While results are the most important output, the program is less important, at least while it is not retained. Once we need results to be audit-able, the importance changes. When we write programs for others to use, to produce results, it is the program and (nearly) only, the program, that is important – well, you do want to be asked back, to write another program.

What will we need and want to learn?

When you create code for others, there are a variety of aspects of Base SAS programming which come into play, ~ some are mandated by the rules of your team, and others offer opportunity to show our own style.

This paper describes features of SAS® software that become more important when you write code for others, rather than just producing results. The paper is organized in these headings:

- code style,
- sharing folders,
- access permissions,
- avoiding contention,
- code review and documentation

CODE STYLE

CREATE CONSISTENT CODE

There are just two guides:

- firstly give priority to “local rules” we have no alternative
- and lastly, “clarity”.

Consistent code makes a process much easier to understand. Often, just by a change in the style, you will recognize where code changes have been made by someone other than the original author.

When setting out your own style rules, pair these with some rationale. For example:

Table 1 sample style rationale:

Style guide	Rationale
Use indentation	Clarifies related blocks and structure
Align ELSE to IF	show to which IF the ELSE belongs
Align END to DO or SELECT (not IF)	Closure is important
Don’t “over-DO it” Don’t use DO-END blocks for single statements	Do not overload code with unnecessary structure support when structure is obvious.

Style guide	Rationale
Indent code within a DO-block	also ensures IF and ELSE in deeper code levels won't be confused with this one
Attach "(" to function name and array name	Clarifies that word (function or array name) is not a variable
Attach = to option names that require a value	Reduce recognition time to a glance
Provide white space around = in an assignment	The corollary of the preceding rule (attach = to option name that needs value)
Make space between each comma and the following entries in a list. for example, a list of function parameters	To ease reading, and in the SASlog the line of code will flow better when too wide for a line
Make space before a semicolon	Important not to miss, so highlight
In a list of assignments, align the =	Clarifies reading
In SQL code, align keywords separately from column names	Ease reading and understanding
In SQL query with many columns to select, place each column on a new line	Easier than managing a large, wide block of code
With SQL columns on separate lines, place comma at beginning, not end of line	Eases adding and moving columns
Align common features where possible	For example aligning length= and format= over several lines eases the checking audit review
Do not use redundant code (e.g. no RUN; after QUIT ;)	If it is there it has to be important
Program header: Keep brief, but make sure there is an author-name-team to contact for support and explanations	Take and share responsibility
More …	More …

Table 1. Code Style Guide Rationale

The list in Table 1 provides a guide that:

- should not be mandatory,
- is incomplete,
- might benefit from additional columns to support cross-referencing and prioritizing,
- can be expected to provoke discussion and
- it highlights how much these styles are personal.

For example when writing code, as I find the space-bar the most convenient key (well, it is the largest key on the keyboard), I'm liberal with "white space" and use it to ease reading of code. I find alignment really helps (even this two fingered typist learns to reach the tab key almost without looking).

ALIGNMENT - USE TABS, BUT LOSE THEM

One (almost hidden) feature for editing code is available and should be selected - please "replace tabs with spaces".

Where this feature is not selected and code containing tab characters is reviewed by another programmer or application, the (probably) different tab setting will ruin our carefully aligned program!

Most applications we come across use eight character boundaries for the tabs, and many old programmers that I know, prefer to set just two or three character boundaries. The SAS editor default is four.

The options are selected in SAS Enterprise Guide® as follows:

Figure 1 Navigating Enterprise Guide Menu for Editor Options

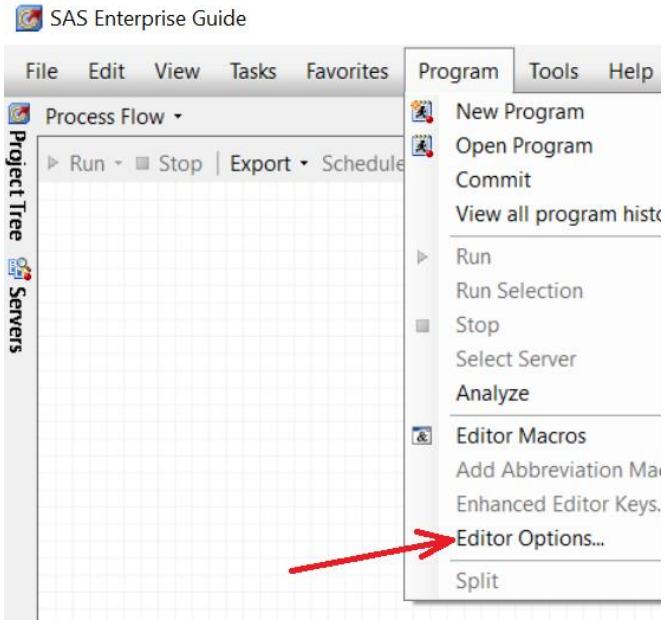

Figure 1 Navigating Enterprise Guide Menu for Editor Options

The Editor Options form provides two checkboxes which should be selected as follows:

Figure 2 Substituting Spaces

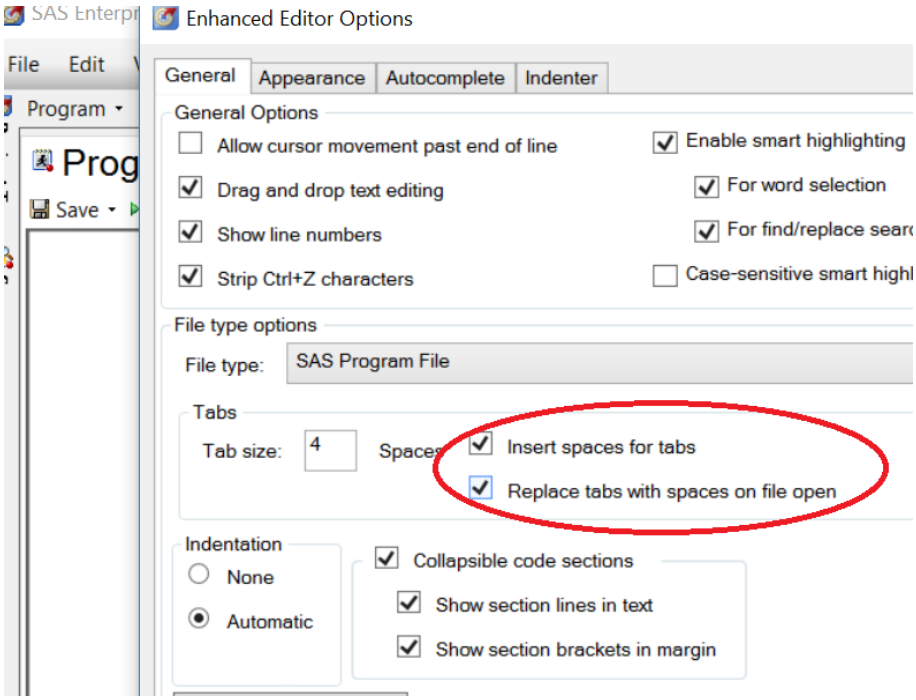

Figure 2 Substituting Spaces for Tabs - Enterprise Guide

In SAS Studio®, "Substitute spaces for tabs":

Figure 3 Navigating To Preferences, For Editor Settings

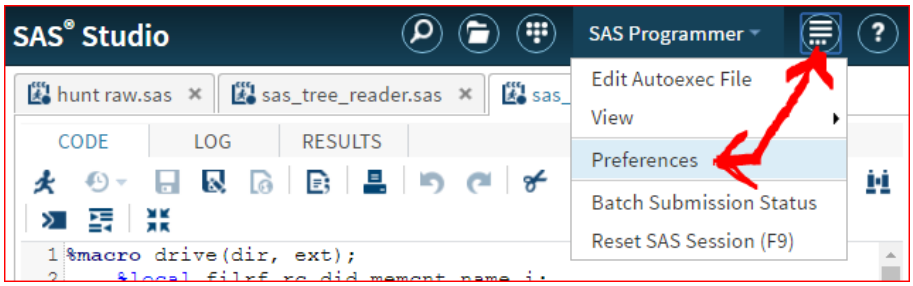

Figure 3 Navigating To Preferences, For Editor Settings

For SAS Studio, only one checkbox needs to be selected, for tabs to be replaced by spaces- as follows:

Figure 4 Substituting Spaces for Tabs – SAS Studio

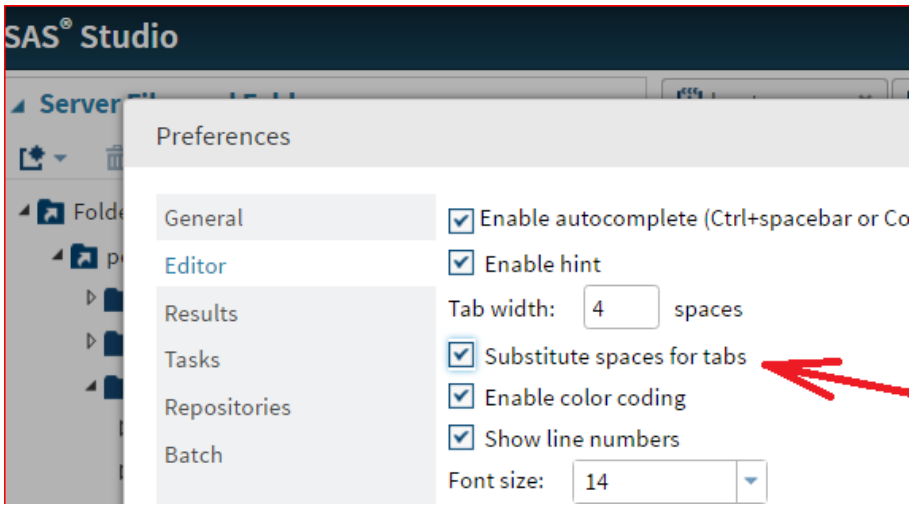

Figure 4 Substituting Spaces for Tabs – SAS Studio

FINDING NEW SAMPLE CODE

When looking for the appropriate code for an unfamiliar procedure, I no longer rely on code output from Enterprise Guide tasks. For me, the Documentation and Samples areas provide far more readable styles.

Most generated code is, intended only for execution by a machine and not for reading by a human being! …. Just review macro-generated code in the MPRINT lines, and code submitted by SAS Enterprise Guide®.

DATA STEP EXAMPLE

To clarify a few of the styles I suggested in Table 1 above, here is a DATA step and MEANS Procedure (SQL Procedure example code, appears later):

```
%let seek = raw ;
Filename hunt '/folders/myshortcuts/peter/Documents/SASGF2016' ;
/*****************************************************
 * hunt all SAS code in my SASGF2016 folder for &seek *
 *****************************************************/
DATA demo ( compress= yes keep= filename line )  ;
   Length filename filen $1000 ;
   Retain filename ;
```

```
   Infile hunt(*.sas ) filename= filen lrecl= 3000 ;
   Input line $char3000. ;
   if filename = filen then delete ;  * already found in this file ;
   pos = find( line, "&seek", 'i' ) ;
   if pos ;                           * subset to code with &seek ;
   filename = filen ;
run ;

proc means data= sashelp.class noprint missing nway ;
   class   age ;
   var     height ;
   output  out= _data_
           min= lightest
           max= heaviest
         ;
run ;
```

STYLE RULES FOR SQL

The style rules for the SQL Procedure are almost totally independent of other procedures and the DATA step. The following two sections of code are intended to demonstrate the convenient adaptability of the style I recommend.

It takes only a few changes between this simple join, reporting members of SASHELP with a DATETIME formatted column

```
proc sql print number _method ;
   select t.libname, t.memname, t.nobs, t.modate
     from sashelp.vcolumn   c
     join sashelp.vtable    t
       on c.libname = t.libname
      and c.memname = t.memname
    where c.libname = 'SASHELP'
      and c.format  EQT 'DATET'
 group by 1, 2
   having nobs   = max(nobs)
      and modate = min(modate)
        ;
quit ;
```

The HAVING clause is not necessary for this query, but serves to demonstrate the style.

It takes only a few keystrokes to change this query which is “about the tables” into one which is “column-centric”. The following query selects all variables of SASHELP tables for which at least one variable has DATE or DT in the column name, or a format beginning DATE.

```
proc sql print number _method ;
    select t.libname
         , t.memname
         , t.nobs
         , t.modate
         , c.varnum
         , c.name
         , c.format
      from sashelp.vcolumn       c
      join sashelp.vtable        t
        on c.libname = t.libname
       and c.memname = t.memname
     where c.libname = 'SASHELP'
       and c.memname in( select distinct memname
                           from dictionary.columns
                          where libname = 'SASHELP'
```

```
  /*                            and memname NET 'V'  */
                                and (format EQT 'DATE'
                                     or upcase(name) ? 'DATE'
                                     or upcase(name) ? 'DT'
                                    )
                           )
    order by t.libname, t.memname, c.varnum
            ;
  quit ;
```

Commenting-out part of the second WHERE clause allows DICTIONARY views to be included among the results.

With so few result columns in the first query, I took the liberty of placing all these on one SELECT line. Visually scanning for the table alias is helped by placing these at the same alignment and out in “open space”. Indentation of the sub-query is intended to make this more-complex query structure still straight forward to understand.

In the queries above, the unusual comparison operators, EQT and NET, are only available in PROC SQL. When you need “truncated comparison”, they offer better performance than applying functions on “input data”. So these are worth reviewing in the online documentation – see the reference section.

SHARING FOLDERS

Although not a programming feature of SAS code, it is appreciated by colleagues when you ensure they are able to read not only the code you deliver to share with a team, but also your code under development. It might not really be necessary, but it encourages an openness that appears more professional. The SAS function, DCREATE(),below, allows me to create folders in my area of a server:

```
%put NOTE: creating folder (%sysfunc( dcreate( newdir, parentDir )) );
```

When the sub-folder is created, the SASlog REPORTS:

```
NOTE: creating folder ( parentDir/newdir )
```

When the folder already exists it cannot be recreated and the message will be:

```
NOTE: creating folder ( )
```

ACCESS PERMISSIONS

Unfortunately, unlike function DCREATE() which creates directories, I can find no SAS function to allow me to set the folder permissions, yet (we can always hope).

Permissions granting my team read access, need to be set through the regular operating system access clients, or perhaps with SAS® Management Console. Another issue of permission appears when our code executes in a server environment. Then we need to ensure that the server will be able to read our code – because the server is unlikely to use our personal access account.

AVOIDING CONTENTION

This issue is becomes important when others might use our code. We cannot expect to control what these “others” might already have in their run-time environment – and will not want disturbed.

The features where we need to take care to avoid contention:

- Writing work data sets
- Changing system options
- Writing external files
- Creating formats, informats and functions
- Creating macros

WRITING WORK DATA SETS WITHOUT CONTENTION

The solution for work datasets is becoming better-known.

DATA is a reserved name to be used in any place where SAS code expects to write a dataset or table. The next name in the DATAn convention is allocated. Where you need to know the name of this table, it can be collected from an automatic macro variable, as in this DATA step:

```
data _data_ ;
run ;
%let name_of_table_just_created = &SYSLAST ;
```

The same feature is useful in procedures when you need to write a table:

```
proc sort out= _data_ ;
   by something ;
run ;
%let name_of_sorted_data = &SYSLAST ;
```

This feature was the first tip described in my paper last year. (Crawford 2016).

CHANGING SYSTEM OPTIONS

Before setting a new value for a SAS System option, collect the current value.

The SAS function GETOPTION() will collect the current state of almost any system option or SAS/Graph® option. Conveniently, there is a second parameter, KEYWORD which returns the option name when the value is provided as a NAME=VALUE pair. For example:

```
%let saved_options2 = %sysfunc( getoption( sasmstore, keyword )
                    ) %sysfunc( getoption( mstored            )) ;
%put &saved_options2 ;
```

The %PUT reports the value of macro variable &SAVED_OPTIONS2 to the SASlog.

```
SASMSTORE='' NOMSTORED
```

More usefully, this macro variable can be used to reinstate the options, after your processing completes with the alternative option values.

When development and debugging are complete, your private macro can store the settings for the options for standard reporting of SOURCE and NOTE. Then it would switch off these reports to reduce the “black-box” NOTEs and SOURCE to the minimum considered helpful. When the process is complete, reinstate the saved settings. During development of this “black box” you want as much debugging help as possible. Appendix2 shows how this might be done using GETOPTION().

Another semi-hidden feature that is really helpful – as we cannot presume to replace the value of system options SASAUTOS and FMTSEARCH without preserving what was there to start with. Base SAS provides INSERT and APPEND which appear to be system options, but are actions to perform on the values of system options. This is important for options which act as lists. The following code shows the APPEND acting on the FMTSEARCH option:::

```
option append= fmtsearch mylib ;
```

The effect is to add MYLIB to the FMTSEARCH option, at the end of the option value. INSERT inserts the new item at the beginning of the list. For the OPTIONS Procedure, option LISTINSERTAPPEND will report to the SASlog, the options where this feature is supported.

WRITING EXTERNAL FILES WITHOUT CONTENTION

If you need to write a permanent file, then the consumer of your program must define the path and name. However, if you wish to create an external file that is required only briefly, create it in the work library and automatically, it will be removed when the SAS session closes. Use the PATHNAME() function to retrieve the path to the WORK library. For a file name that is free of contention, include a number at some point in the file name. Run through a loop while the function FILEEXIST() indicates your chosen name already exists, and in this loop, increment the counter. Something like:

```
do ref= 1 by 1 while( FILEEXIST( catt( "%path(work)/f", ref, '.dat' ))) ;
end ;
```

For a DATA step to create and then use this name, the FILE or INFILE option FILEVAR= is needed.

The equivalent as a macro is probably a more versatile solution but might not be as clear. Unfortunately the %DO macro statement cannot combine %BY with %WHILE(). See Appendix 1 for an example.

CREATING FORMATS, INFORMATS AND FUNCTIONS WITHOUT NAME CONTENTION

In SAS9 new dictionary tables SASHELP.VFORMAT and SASHELP.VFUNC list all format and function names including formats defined by the FORMAT Procedure and user defined functions created with the FCMP Procedure.

Dynamically avoiding a format name already in use, you could use an adaptation of the %NEWFILE macro in Appendix1 and look for existing format names in the metadata table DICTIONARY.FORMATS. The iteration becomes more complex. For formats, no function corresponds to the FILEEXIST() function used in Appendix1. To check existence of a format in an iteration, the macro would need to adapt and re-apply a filter to SASHELP.VFORMAT to check if the next name is present. In both PROC SQL and DATA steps, the WHERE clause to filter for a row, is fixed at compile time, so cannot be adapted in this way at run-time. The alternative in Appendix 3 discovers the maximum numeric part present, and derives a new format name with a number above that maximum.

In general we prefer all utility macros of this type to achieve general unlimited usage - we want to avoid having the macro submit steps, or even semi-colons. If this becomes too restrictive, and we allow the macro to “cross the step boundary”, then straight forward PROC SQL code, or a DATA step, can offer the simplest solution. To achieve the “general preference” and avoid “going through a statement boundary” the more complex macro must open the view in SASHELP with a WHERE filter using the OPEN() function in a macro. Then the macro passes through just the rows which have the pattern of name preferred. This can be seen in Appendix 3.

We can use the same approach to dynamically derive a name for a function that guarantees no contention with current function names – using the view SASHELP.VFUNC. The macro might be cloned or extended.

CREATING MACROS WITHOUT NAME CONTENTION

When and if you need to create a macro that is not part of the “final deliverable”, you will want to avoid replacing a macro that is already in the customer’s environment. Those macros already invoked will be in the compiled macros catalog. This takes a platform-dependent name (for example WORK.SASMACR or WORK.SASMAC1). In addition, your customer might be using stored compiled macros. All these names can be identified, and so avoided with methods similar to the solution for external files. The function CEXIST() will check the existence of a catalog entry like a macro. Alternatively, the view SASHELP.VCATALG could list all MACRO-type objects in all catalogs in the current SAS session environment. For example:

```
proc sql ;
   create table _data_ as
         select distinct objname as macroName
           from sashelp.vcatalg
          where objtype = 'MACRO'
               ;
quit ;
%let list_of_macros = &SYSLAST ;
```

Unfortunately, this will not predict subsequent action by customer applications.

Among the SAS supplied auto-call macro pool also knows as SASAUTOS, more than 600 macros are defined. Our customer applications might use any of these. If one of our programs creates a macro that “overlaps” the macro in SASAUTOS, the customer won’t use our processes. A SAS Global Forum paper (Crawford 2016) reveals the names of the 600+ macros within the supplied SASAUTOS folders – look at

“9. Read macro source code and reveal hidden macros” . The search could be extended to customer libraries of auto-call macros.

However, sometimes it is necessary to accept the limits of transparency and forewarn the user of possible name contention.

CODE REVIEW

Up until this stage (programming for results) the code has remained “on-screen”, or on a device. For code review around a table, paper becomes more important.

Well produced printed code with the syntax coloring, achieves greater acceptance, in my experience, than a plain text print. I expect we each will have a different approach, from simple through to Microsoft PowerPoint or Google-presentation sophistication. I have adopted and recommend:

1. Use Microsoft Word
2. Where the SAS code includes macro definitions, consider whether it is the macro coloring you want or the base code step coloring you want. For the latter, just insert a blank between the “%” and “macro “. Then the syntax coloring resumes as normal. Easily reinstate by removing that blank later.
3. Paste the SAS editor code, with the syntax color attributes, into Word as “rtf formatted text”
4. Among the page layout options insert Line numbers – and choose continuous
5. Modify the lineNumber style to a small point size (I prefer 6)
6. Insert as header, the path and name of the program file being reviewed
7. Insert in the trailer (right aligned), the timestamp for the latest modified date of the program – which also acts to define the version of the code
8. Check the code line widths because Word handles wide lines differently from the SAS editors and the line number provides referencing to code. Word increases the line number for overflow lines, but the SAS editor sees the line as one line. To have the same line numbering in Word that we have in SAS editor, we must make all the SAS code lines fit in Word (or – unwanted option – change the code to make the code narrower). My strategies :
    9. Reduce the margins (especially on the right) but without losing the line number
    10. Reduce the point size (to no less than 6)
    11. Use Arial for messages and comments that are too wide
    12. Use landscape

As this paper version will hang around (until an updated version gets the same print treatment), and carries your name, it continues to impact on your reputation with colleagues and clients – those whom you might most, wish to please…….

That is why I give documentation and code for review, this extra special treatment.

CONCLUSION

We want to ensure we take advantage of any aspect of the SAS programming environments that improve our working relationship with colleagues. It should be part of our early learning when we look to progress in our career. There are more features than a 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*, SAS Institute. Available at http://support.sas.com/resources/papers/proceedings16/2120-2016.pdf

Crawford, Peter. 2014; “Paper 1744-2014, VFORMAT Lets SAS® Do the Format Searching”; *Proceedings of SAS Global Forum 2014 Conference*. Available at http://support.sas.com/resources/papers/proceedings14/1744-2014.pdf

SAS Institute, SAS® 9.4 SQL Procedure User’s Guide, SQL Expression link “Truncated String Comparison Operators”, available at http://support.sas.com/documentation/cdl/en/sqlproc/69822/HTML/default/p020urejdmvi7vn1t9avbvazqapu.htm#n0tcl9lmusfteon1entll2ua55r1

RECOMMENDED READING

- Step-by-Step Programming with Base SAS® 9.4
- SAS OnlineDoc® documentation *at http://support.sas.com/en/product-resources.html*
- SAS Support / Samples SAS Notes *by topic at http://support.sas.com/kb/?ct=51000*
- SAS® 9.4 SQL Procedure User’s Guide

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 MACRO NEWFILE TO DERIVE A NEW FILE NAME

The code below defines a macro NEWFILE which derive a new file name that does not already exist.

Parameters:

1. The PATTERN parameter provides the path as well as the pattern of the required file name. Within the pattern, some feature will be replaced by a number – the first that does not already exist.
2. The feature to be replaced is defined by the REPLACE parameter which defaults to ¬¬.

```
%macro newfile( pattern, replace= ¬¬ )
       /des= ' generate a new filename by incrementing and replacing &replace ' ;
 %local limit ref newname ;
 %let   limit = 10000 ;
 %do ref= 1 %to &limit ;
    %let newname = %qsysfunc( tranwrd( %superq(pattern), %superq(replace), &ref ));
    %if NOT %sysfunc( fileexist( %superq(newname) )) %then %do ;
        %let ref= &limit ;
    %end ;
 %end ;
 &newname
%mend  newfile ;
```

With its arbitrary limit of 10000, this macro will fail should there already be that many files of this kind of name in that folder.

The following demonstration of %NEWFILE uses my simple macro %PATH to locate the WORK library.

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

The demonstration DATA step below writes the first row of SASHELP.CLASS into a new file in the work library.

```
data _null_ ;
  set sashelp.class( obs=1) ;
  file "%newfile( %path(work)/my_¬¬_off_file.sgf2017.demo)" dsd ;
  put (_all_)(:) ;
run ;
```

Repeated execution increments the counter in the name, at that ¬¬ point.

There might be occasions when I might need this functionality in paths other than WORK, so the file path is part of the parameter whith no additional complexity in the macro.

APPENDIX 2 DEMONSTRATE BLACK-BOX DEVELOP AND RUN MODES

First a small macro to quietly collect option values.

```
%macro sy( op, kw= ) / des= 'collect sys options kw= k for value options' ;
%sysfunc( getoption( &op %if %superq(kw) ne %then
          %do ; , keyword %end ; ))
%mend  sy ;
```

Since the GETOPTION function now generates a NOTE if the KEYWORD option is used when unnecessary, the %SY code now uses a parameter to help eliminate the NOTE.

The following macro %DEMO uses parameter TEST= prod (the default) to indicate “black box” operation in production. Then, not only should macro diagnostics be suppressed, but also the regular step NOTES: can be hidden along with any SOURCE – my assumption being that these messages in the SASlog are no longer required.

However, the reverse applies when in development (TEST= DEV), then all macro diagnostics should be switched on.

For either situation, these options should be reinstated to settings at entry, when the black-box, or development, macro completes.

```
%macro demo( this, that,the,other, test= prod )/ des= 'expose and hide' ;
    %local saved_options ;
      %let saved_options = %sy( mlogic ) %sy( symbolgen ) %sy( mprint
                         ) %sy( source ) %sy( notes ) ;

      %if %superq(test)= prod %then %do ;
        option NOmlogic NOsymbolgen NOmprint NOsource NOnotes  ;
        %put NOTE: BLACK BOX IN ACTION : ;
      %END ;
      %else
      %if %superq(test)= DEV %then %do ;
        option mlogic symbolgen mprint source notes  ;
        %put NOTE: collecting diagnostics for development :  ;
      %END ;
 %*    ~~~~~~~~~appropriate code for process~~~~~~~~~~~~~~~   ;
      %*reinstate options state ;
      OPTION &SAVED_OPTIONS ;
      %if %superq(test)= prod %then %do ;
            %put NOTE: BLACK BOX COMPLETED : ;
      %END ;
%MEND  demo ;

%demo ;
%demo( test= DEV ) ;
```

APPENDIX 3 DEMONSTRATE FINDING A NEW FORMAT NAME

To avoid using a name that already exists and without submitting statements, this macro will collect existing formats that match the requested name pattern, identify the maximum number in the defined position then generate a new format name with next number in sequence. The parameter &REPLACE indicates where a number should appear within the format name – as defined by parameter &PATTERN

```
%macro newFmt( pattern, type= format, replace= ¬¬ )
       /des= ' generate a new (in)format name by replacing &replace ' ;
%local rc dsid ref newname row limit rpos fmtname fmttype numend this_num max_num ;
 %let ref = %qsysfunc( tranwrd( %superq( pattern ), %superq( replace ), %nrstr(%%%%) )) ;
 %let ref = %qupcase( &ref ) ;

 %let rpos = %sysfunc( find( %superq( pattern ),%superq( replace )) ) ;
 %if NOT &rpos %then %do ;
    %put &sysmacroname:FAIL- nothing to replace in pattern ;
    %put &sysmacroname:FAIL- replace= %superq(replace) ;
    %put &sysmacroname:FAIL- pattern= %superq(pattern) ;
    %return ;
 %end ;
 %if &type= format %then %do ; %let fmttyp = F ; %end ;
                   %else %do ; %let fmttyp = I ; %end ;

 /*********************************************************
 * now open VFORMAT with where clause LIKE the pattern    *
 *          and on format type                            *
 *********************************************************/
%let dsid = %sysfunc( open( sashelp.vformat( where=( fmttype= "&fmttyp"   AND
                                                     fmtname like "%superq(ref)"
                                                )       )
                          , is )) ;
%if &dsid < 1 %then %do ;
    %put &sysmacroname:FAIL- %SYSFUNC(sysmsg()) ;
    failed
    %return ;
%end ;

 %let   limit = 10000 ;
 %let max_num = 0 ;
 %do row= 1 %to &limit ;
    %syscall set( dsid ) ;
    %let rc=%sysfunc( fetch( &dsid )) ;
    %if &rc eq -1 %then %do; %let row= &limit ; %end ;
    %else %do ;
        %if &rc ne 0 %then %do ;
             %put &sysmacroname:FAIL- %sysfunc( sysmsg() );
        %end ;
        %else %do;
            %let   numend = %sysfunc( verify( %substr( &fmtname, &rpos )
                                              , 1234567890 ) );
            %let this_num = %substr( &fmtname, &rpos, %eval( &numend -1 ) ) ;
            %let  max_num = %sysfunc( max( &this_num, &max_num ) ) ;
        %end;
    %end ;
 %end ;
 %let rc = %sysfunc( close( &dsid )) ;

 %let newname = %sysfunc( tranwrd(  %superq( pattern )
                                 , %superq( replace )
                                 , %eval( &max_num +1 )
                         )        ) ;
  &newname
 %put NOTE: &sysmacroname version 1 -- result &newname ;
%mend  newfmt ;
```

The following statements demonstrate the facility with a dummy user format.

```
%let        newname= %newFmt( my¬¬pic, type= format, replace= ¬¬ ) ;
proc format ;
     value &newname      1= one ;
run ;
```