Paper 2120-2016

More Hidden Base SAS® Features to Impress Your Colleagues

Peter Crawford, Crawford Software Consultancy Limited.

ABSTRACT

Across the languages of SAS® are many golden nuggets—functions, formats, and programming features just waiting to impress your friends and colleagues. While learning SAS for over 30 years, I have collected a few of these nuggets, and I offer a dozen more of them to you in this presentation. I presented the first dozen in a similar paper at SAS Global Forum 2015.

INTRODUCTION

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

Make these gems part of your early learning - to accelerate your path to professional competency – 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 difficult coding problem, only to find that I am searching with the wrong words. As new problems become harder I find base SAS still has the answer, as if by magic[1].

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 beginners' programming courses[2], despite the simplicity of the gems.

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 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 SAS Programmers and SAS Developers at all levels of expertise.

[1] Crawford 2003; "Paper 086-28, More _Infile_ Magic"; Proceedings of the Twenty Eighth Annual SAS Users Group International Conference, www2.sas.com/proceedings/sugi28/086-28.pdf

[2] Link to SAS Training
SAS Programming 1: Essentials https://support.sas.com/edu/schedules.html?ctry=gb&id=2588
SAS Programming 2: manipulation https://support.sas.com/edu/schedules.html?ctry=us&id=2618

FIRST MAGIC GEMS

THE FIRST DOZEN AT SAS GLOBAL FORUM 2015

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

To expand on these Gems, see Crawford (2015)

MORE MAGIC GEMS

ANOTHER DOZEN FOR 2016

1. _DATA_ protect your interests (and others')
2. NLITERAL() – a neat name function
3. Use "global characters" like * in INFILE path (great feature - SAS reading files on UNIX)
4. easily avoid having /*.csv filename start a comment block
5. SASTRACE option – checking what your database will see
6. INFORMAT $ANYFDT – what the ANYDTxxx informats would choose
7. VFORMAT table – what formats are available
8. %pattern (just a small macro missing from the SASAUTOS)
9. Read macro source code and reveal hidden macros
10. SAS editor macros - lead to key definitions and unexpected editor functions
11. Integrate Perl Regular Expressions with the best of SAS programming
12. Profile a data step to review the data flow through statements

1. _DATA_

protect your interests (and others data)

This feature appears and develops out of a default that we are all encouraged to avoid - a DATA step which defines no output dataset name :

```
data ;
  ...
run;
```

. . . Represent regular statements, not related to this feature

SAS will still compile this DATA statement assuming a default dataset name. If the step runs OK the new dataset will be created and named with the next number in the DATAn convention.

This tip → use _DATA_ to enjoy the DATAn convention

Benefit: new intermediate outputs won't overwrite existing datasets.

Here is how to use this tip and still have everything else as normal.

We will still need a handle (dataset name) for the new table. For example, when we use the dataset in subsequent steps (after the next step).

The name of the newly generated dataset becomes available in the automatic macro variable &SYSLAST only once the DATA step completes. Use &SYSLAST to collect the name and assign it to a handle – in this snippet our handle will be the macro variable named &new_load_data:

```
Data ;
  ...
run ;
%let new_load_data= &syslast ;
```

The feature, _DATA_, can be used anywhere Base SAS would write a new dataset :

```
proc sort data= &new_load_data out= _data_ ;
   by something ;
run ;
%let sorted_input_data= &syslast ;
```

The feature ensures new datasets won't overwrite existing data .

This feature is most helpful when we write packages or macros that need to create intermediate datasets but we cannot know at design time, what dataset names to avoid in the run-time environment.

We should not assume that adding more underscores to a dataset name like this OUT= in:

```
    proc sort data= &in_param out= __&in_param ;
```

will provide safety, or protection for our customers' pre-existing data – this "protection method" might already be in use.

The _DATA_ convention already exists as part of the Base SAS foundation. Let's all use it.

2. NLITERAL()

– a neat name function

Among the many-many functions in the SAS System (more than 900 last time I checked v9.4), we can find this function. It checks and protects a string to ensure it will be valid if used as a SAS NAME.

```
var = NLITERAL( txt ) ;
```

If variable TXT contained the string '`Hello World!`', then variable VAR becomes the value '`Hello World!` '`n`

This is most helpful when we write packages or macros for use by "others". For example, this function helps when we ~~hope to~~ make column names from macro parameters, or from headers in a CSV file.

3. USE GLOBAL CHARACTERS - LIKE * IN INFILE PATH

– (great feature - SAS reading files on UNIX)

In typical installations of the SAS System that serve many users, the servers are protected from all user commands. These include some commands that are popular where the SAS System is installed on a single-user platform rather than a server. Most popular, are commands for retrieving file lists - typically the command to list files would be piped into a DATA Step. On a multi-user server, these commands are blocked. However, if your SAS server runs on UNIX, that platform enables this very neat feature – on a DATA step INFILE statement we can use global characters in the path, as well as in the file name that is defined. For example:

```
infile '~/data/*/*.csv'
       dsd truncover filename= filen  firstobs=1  lrecl= 10000 ;
```

In this example, the "~/data/*/*.csv" directs UNIX to pass into SAS, all .csv files in any folder that matches the path. Paths that match could be:

```
~/data/somepath/*.csv
~/data/anotherpath/*.csv
~/data/s/*.csv
```

Any csv files in these and similar paths would be read by that DATA step.

I have found this a great way to read across a hierarchy of folders.

Imagine this hierarchy of folders for a project constructing some solution

Figure 1 project folders hierarchy

```
Project1
      Import Routines
            Code1
            Logs
            data
      Cleanse
            Code2
            Logs
            data
      Summarize
            Summary_code
            Logs
            data
      Derivation
            Excel models
            Test results
            Test code
      Analysis
             Analytic code
      Present
            Performance code
```

Figure 1 project folders hierarchy

This feature in the UNIX file systems allows a single INFILE path definition to process across this hierarchy enabling a search through all .sas programs, for example, to find the code where a specific string, or column name, appears :

```
infile '/project*/*/*?ode*/*.sas' filename= filen truncover lrecl= 1000 ;
```

This INFILE statement has asterisks (*) in the physical path as well as in the file name.

This asterisk is referred to as a “global” character. It represents any number of characters (even none). As the file name is “*.sas”, only these will be read - and only if they are in folders that have “ode” somewhere in the folder name. Because on UNIX platforms file and path names are case sensitive, for the “C” or “c” in “code”, replace that “C” / “c” with a question mark (?) –a character must be present before “ode”.. Although this would allow folder “Excel models” to be scanned, we can hope that files in this folder will not include any SAS programs.

I include these additional INFILE options because they are necessary or very important -

FILENAME= names a variable that will hold the path and name of the file from which the current input buffer has come (and ensure it is wide enough with a prior LENGTH statement).

LRECL= Enterprise Guide® software can generate lines wider than the default 256

TRUNCOVER almost out of habit – (born of accidents happening when I omit this)

Being able to read across the hierarchy with global characters in the paths is a great facility. It replaces –

- having separate INFILE statements for each separate path, or
- providing a recursive read down the tree from /Project1 to pick out the *.sas files and supply these through the FILEVAR= option, or
- loading an macro variable array of the target files and looping through a complex macro.

I think this way – using global characters in the path – results in a simpler looking program.

Unfortunately “asterisks in the path” is a feature not available when the SAS server is running on Microsoft Windows. However fortunately, at least one version of the SAS University Edition runs SAS foundation on Linux within the virtual machine. So “asterisks in the path” works there (Linux), but not in my “fat client” SAS Windowing Environment, on which the virtual machine executes.

4. EASILY AVOID HAVING /*.CSV FILENAME START A COMMENT BLOCK

Often we wish to read some or all .csv files in a folder. When we pass the “name to be read” as a parameter, it might be used in a macro variable like:

```
%let read_names = *.csv ;
```

If the code becomes anything like :

```
%let read_names = /*.csv ;
```

there is a risk that the SAS language token analyzer will assume a comment block is starting.

While the classic macro language solution to this problem uses %STR() like :

```
%let read_names = %str(/)*.csv ;
```

Sometimes this might still resolve and cause problems. Fortunately, there is an even simpler solution. We can separate the slash(/) from the asterisk(*) with a question mark (?) as in:

```
%let filenames = /?*.csv ;
```

That question mark (?) is just another of the “global characters” permissible in a file name on Microsoft Windows platforms and in both file name and path name on UNIX platforms.

5. SASTRACE OPTION

– checking what your database will see

Although very much part of “Foundation SAS” technically, this option is not part of base SAS – it is part of SAS/ACCESS® :

```
option sastrace = ',,d' ;
```

This will report to the SAS log, the syntax that SAS/ACCESS will pass to the database platform.

The simple way to use database tables in a SAS Procedure requires only a database connection established in a LIBNAME statement. Then just refer to the database table like a base SAS dataset.

With the SASTRACE option above we can see what code is passed to the database engine. This becomes most helpful when the results are not what we wanted - we can diagnose the problem. Sometimes a “query” is too complex for the “implicit pass through”. Then we prepare and EXECUTE an “explicit pass through query”. This needs to be in exactly the style of syntax that the database server expects. Examples of the syntax generated by SASTRACE will help with the style required.

6. INFORMAT $ANYFDT –

what the ANYDTxxx informat could choose

This feature is sitting quietly (undocumented) but waiting worthy praise.

When assessing the challenge of importing a new data supply, we would start with a quick look. When all columns appear consistently of the same format/style, we have confidence it should load (import?) reasonably – but what if, and when, it is not consistent? This will be the time when we most appreciate the ANYDTxxx informats for handling dates and times.

Dynamically – on every row, these informats decide what date style or time style appears in your raw data and they will uses the most appropriate informat to return date, time or datetime value.

As an old programmer (perhaps because I am one), I expect most data suppliers to have more consistency than that. I can (expect) then to refer data in the wrong style, back to the supplier for explanation, apologies and remediation.

For data supplied on a “planned arrangement”, instead of using an ANYxxx informat to load the value, I expect to use the specific informat of the “agreed” input layout.

To investigate the probable structures, of undocumented data, we can use informat $ANYFDT.

The informat $ANYFDT returns, not the input value but which informat, would be chosen by the ANYDTXXX informats as the most appropriate for that “cell”. (every row - not just each column). With a quick run of the SUMMARY Procedure, I can report which date-styles appear most frequently. The following code was a preliminary test with CARDS input:

```
data any_testing ;
infile cards truncover ;
informat dtm  anydtdtm50. dte anydtdte50.
         tme  anydttme50.
         anyfmt  any2 $anydtif20.;
format   dtm  datetime.   dte date11.   tme time12.1 ;
input
    @1 cardconts $char50.
    @1 dtm &
    @1 dte &
    @1 TME &
    @1 anyfmt &
    @1 any2 ;
cards;
12/13/2012 15:54:45
12/13/2012
```

```
12/13/2012 15:54:45.123
22/12/2012 15:54:45
22/12/2012
;
```

This code returned the ANY_TESTING table

```
FSVIEW:  WORK.ANY_TESTING (B)-----------------------------------------------------------
 cardconts                          dtm         dte          tme  anyfmt     any2

 12/13/2012 15:54:45         13DEC12:15:54:45  13-DEC-2012   15:54:45.0  ANYDTDTM   MMDDYY
 12/13/2012                  13DEC12:00:00:00  13-DEC-2012    0:00:00.0  MMDDYY     MMDDYY
 12/13/2012 15:54:45.123     13DEC12:15:54:45  13-DEC-2012   15:54:45.1  ANYDTDTM   MMDDYY
 22/12/2012 15:54:45         22DEC12:15:54:45  22-DEC-2012   15:54:45.0  ANYDTDTM   DDMMYY
 22/12/2012                  22DEC12:00:00:00  22-DEC-2012    0:00:00.0  DDMMYY     DDMMYY
```

Table 1 preliminary investigation of $ANYDTIF.

The trailing “&” in the INPUT statement ensures that text after a single embedded blank will be included in the string passed to the informat – compare the values of the last two variables in that table – without using the trailing “&” only the date-part is considered..

Of course, I’ve not shown all the investigation code, here – see appendix 1

7. VFORMAT TABLE

– what formats are available?

The dictionary table FORMATS also known as SASHELP.VFORMAT lists all built-in and user-defined formats known to the current SAS environment. At the SAS Global Forum in 2014[3] I presented a paper on using this dictionary table to discover which formats would deliver required style. There are other benefits. For example - only %INCLUDE the required code, if a user-format is not already loaded :

```
%let myuserformat= XLDATE ;
%let myfmtcode    = d:\peter\documents\my sas files\xldate.sas ;

proc sql noprint ;
  %let                _fmtName = ;
  Select fmtName into :_fmtname from SASHELP.VFORMAT
   where fmttype= "F" & fmtName = "&myuserformat" ;
quit ;
%scan( %nrstr(%INCLUDE "&myfmtcode"/source2)#*comment, %eval(1+&sqlobs),#);
```

When the format is already loaded, *comment is submitted, otherwise the %INCLUDE is submitted. In this example syntax, the condition is test without using a SAS macro.

8. %PATTERN

- (just a small macro missing from the SASAUTOS)>

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

Often I would need to generate a list of numbers in my program – usually a list xxxx1 – xxxx99 in places where that type of variable name range definition is not supported – one example – syntax for the SQL Procedure, where I might want something like

```
     alias.amount2000, alias.amount2001, alias.amount2002, alias.amount2003,
```

There is no built-in macro function that does this in the SAS System. There is no AUTOCALL macro supplied that does this. So this kind of functionality must be a practice ground for all budding Base SAS Macro Developers – well I think it could be. Here are my two efforts- first the simplest:

```
%macro gen(base,n);
%local i;
%do i=&from %to &n;
&base&i
%end;
%mend;
```

For simple numbered lists, line name1-name5 this works, but for my "alias amounts" the nearest I can come is:

```
%put alias.amount2000 %gen( %str(, alias.amount200), 3);
```

As this shows, more work is needed to make this simple but limited macro, deliver what is needed.

The second, more substantial %pattern effort, adds parameters for-

- "pattern"   to replace "base" - so the number can go anywhere in the pattern),
- "replace"   (defaulting to ###) to indicate where the "number substitution" should occur
- "start"   (defaulting to 1),
- "by"   which can be fractional or negative, and
- "format"   for the "number-suffix" - to support leading zeroes and dates

Adding these parameters, some "parameter validation", and protecting both parameters and generated patterns, has added more than a few lines to the macro – so it can be found in appendix 2. The macro is still less than 50 lines.

The expansion of functionality often happens to my solutions. However, the simple list of "alias amounts", needs no more than :

```
%genpatN( %str(alias.amount###,), from= 2000, upto=2003 )
```

I accept that this version of the macro has no parameter to deliver a delimiter (where I want a comma), but I rationalize this (current) usage because, using %STR() when the macro is invoked, supports far more complexity than a delimiter – patterns protected with %STR() can contain whole statements or even whole steps.

9. READ MACRO SOURCE CODE AND REVEAL HIDDEN MACROS

Among the installed components of the SAS System, is a collection of base SAS Macros. The paths to these macros are defined by system option SASAUTOS.

Supporting this option the SAS foundation makes available a FILEREF with the same name as the option – SASAUTOS.

Features of the Base SAS Language allow a very simple DATA Step to list the names of these macro program files, and the names of the macros that they contain. When first I explored these macros I was surprised how many macros were defined and waiting to found – they have names different from the files in which they are stored. Here is the simplest version of my search:

```
data all_provided_macros ;
  length filename filen $1000 macName $33 ;
```

```
    infile sasautos("*.sas") filename= filen lrecl= 1000 dlm= ' (/)% ; ' ;
    input @'%macro ' macName ;
    filename= filen ;
  run ;
```

Unfortunately, this did not pull out all the results I expected. The feature which stops scanning input at '%macro ', is case of the letters in the text being read – as in '%Macro'. INPUT @'string' is case sensitive regarding 'string'. Here is revised code – insensitive to the case of %MACRO – we find them all :

```
  data provided_macros2 ;
    length filename filen $1000 macName $33 ;
    infile sasautos("*.sas") filename= filen lrecl= 1000 dlm= ' (/)% ; ' ;
    input @ ;
    pos = find(_infile_, '%macro ', 'i' ) ;
    if pos ;
    input @(pos+6) macName ;
    filename= filen ;
  run ;
```

In that FIND() function call, the third parameter 'i', requests a case-"insensitive" search. This longer version now has to pay attention to every line. Thus we derive the line number where a macro definition starts within each file, more easily – just after that first INPUT, add the lines :

```
      if filen ne filename then line_no = 0 ;
      retain filename ;
      line_no +1 ;
```

The size of the resulting table varies depending on which modules of SAS are installed. Among the results found by the above code, in the "Core" of base SAS over 100 macros are found with a macro name differing from the filename.

Figure 2. Log from the PROC PRINT step indicating "hidden macros"

```
91   proc print data= provided_macros3 ;
92       where lowcase(macname) ne lowcase(scan(filename, -2, './\'))  ;
93   run ;

NOTE: There were 131 observations read from the data set WORK.PROVIDED_MACROS3.
      WHERE LOWCASE(macname) not = LOWCASE(SCAN(filename, -2, './\'));
NOTE: PROCEDURE PRINT used
```

Figure 2. Log from the PROC PRINT step indicating "hidden macros"

When executed in the SAS University Edition, the same code reveals 400 hidden macros. Only a sample of the full list of these macros follows -

Figure 3. Sample Output from the PROC PRINT step indicating "hidden macros"

Obs	Filename	macName	line_no	pos
2	/opt/sasinside/SASHome/SASFoundation/9.4/sasautos/aalasr.sas	aa_isGrid	6	1
9	/opt/sasinside/SASHome/SASFoundation/9.4/sasautos/aamodel.sas	aa_hpds2_genruncode	29	1
81	/opt/sasinside/SASHome/SASFoundation/9.4/sasautos/armini2.sas	Armnoq	1	1
85	/opt/sasinside/SASHome/SASFoundation/9.4/sasautos/armproc.sas	_ARMread	1	1
94	/opt/sasinside/SASHome/SASFoundation/9.4/sasautos/boxcoxar.sas	NAME	13	4
95	/opt/sasinside/SASHome/SASFoundation/9.4/sasautos/boxcoxar.sas	DSN	37	4
96	/opt/sasinside/SASHome/SASFoundation/9.4/sasautos/boxcoxar.sas	INTEGER	60	4
101	/opt/sasinside/SASHome/SASFoundation/9.4/sasautos/choiceff.sas	Res	1	1
103	/opt/sasinside/SASHome/SASFoundation/9.4/sasautos/choiceff.sas	_vlist	1193	4
104	/opt/sasinside/SASHome/SASFoundation/9.4/sasautos/choiceff.sas	_zlist	1199	4

Figure 3. Sample Output from the PROC PRINT step indicating "hidden macros"

I need to uncover the names of these macros to avoid breaking any of these processes by overriding a hidden macro with a new macro that I make available to other SAS users.

This “overriding” rule applies also to formats and to functions we create for others – but this kind of ”hiding” appears more difficult for those.

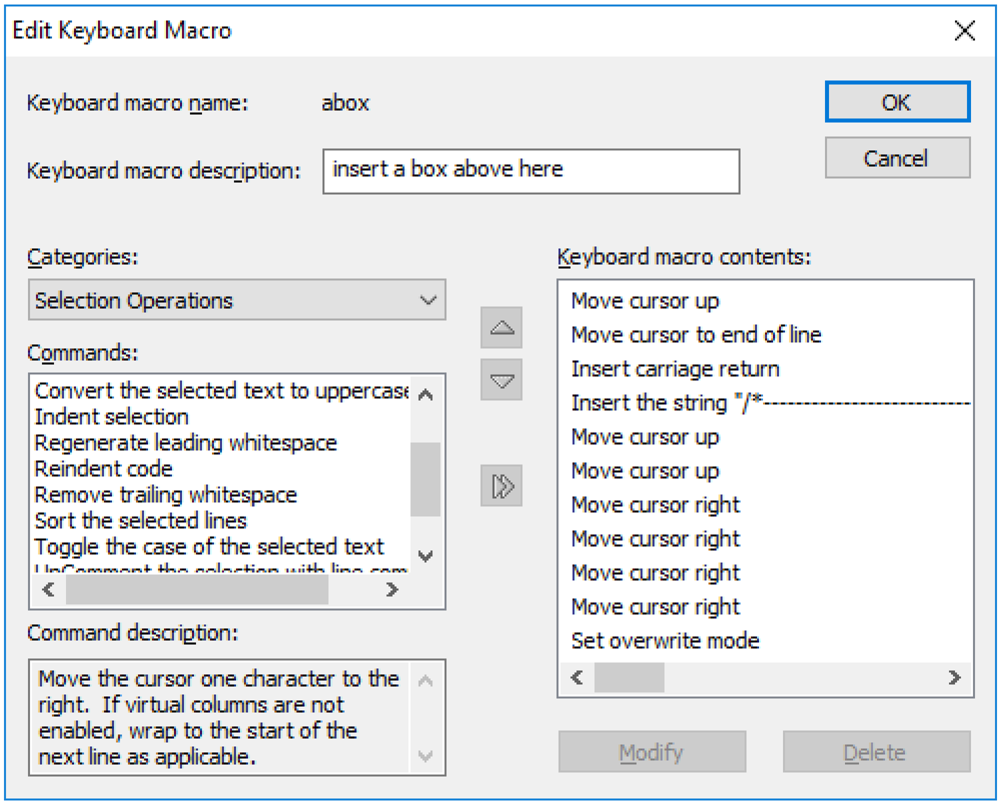
10. SAS EDITOR MACROS - KEY DEFINITIONS AND UNEXPECTED EDITOR FUNCTIONS

First a constraint: this feature applies where SAS Enterprise Guide code node is being used or where the SAS Windowing system (formally known as SAS Display Manager) is being used. It is neither a feature of SAS University not SAS Studio – a constraint that also excludes the SAS Windowing system when running in a z/OS terminal.

This feature has partly been covered many times before (for example, Carpenter 2003 to Borst 2015), except I remain to see the additional actions presented here.

Edit an abbreviation after creating it. It can be found among the keyboard macros. We can edit the macro to position the cursor at the point where text would be added (like a name in a program header) or comment in a box. At the same time, the mode can be set to “overwrite”.

Figure 4 Abbreviation with Additional Macro Actions



Keyboard macro name: abox

Keyboard macro description: insert a box above here

Categories: Selection Operations

Commands:
Convert the selected text to uppercase
Indent selection
Regenerate leading whitespace
Reindent code
Remove trailing whitespace
Sort the selected lines
Toggle the case of the selected text

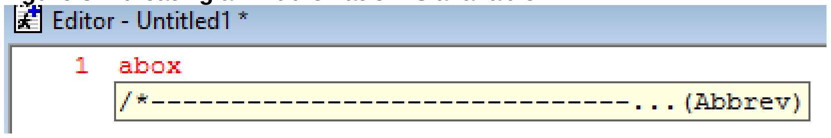
Command description:
Move the cursor one character to the right. If virtual columns are not enabled, wrap to the start of the next line as applicable.

Keyboard macro contents:
Move cursor up
Move cursor to end of line
Insert carriage return
Insert the string "/*--------------------------
Move cursor up
Move cursor up
Move cursor right
Move cursor right
Move cursor right
Move cursor right
Set overwrite mode

Figure 4 Abbreviation with Additional Macro Actions

When, in my code editor I want to write a new comment in a box just above the current line, I type “abox”. When I type the “x” a highlighted string appears, to indicate the abbreviation is available –

Figure 5 Indicating an Abbreviation is available

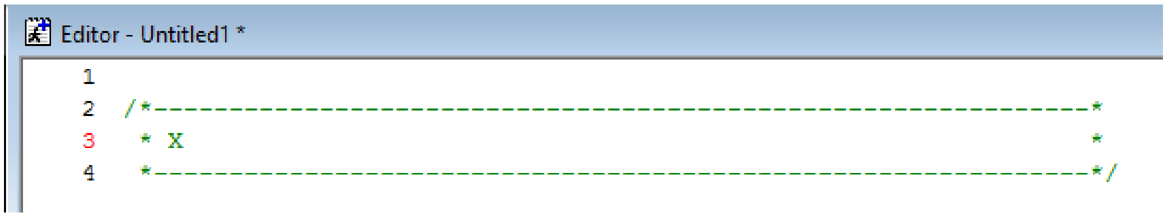


```
abox
/*--------------------------------...(Abbrev)
```

Figure 5 Indicating an Abbreviation is available

I then press the “tab” key. The comment box is inserted just above the current line, insert-mode is switched to overwrite, and the cursor is placed where I like it – in that box.

Figure 6 inserted comment box (X marks the new position of the cursor)



```
/*---------------------------------------------------------------*
 * X                                                             *
 *---------------------------------------------------------------*/
```

Figure 6 inserted comment box (X marks the new position of the cursor)

There are varieties of the current date that can be inserted by the keyboard macro, into a program header.

Another very useful feature in the “keyboard macros” facility is “assign keys”. Any keyboard-macro-command can be assigned a keystroke combination to invoke that macro–command. I assign ctrl+shift+S to sort the current selection. Another useful macro-command is “Toggle case of selected text”.

11. INTEGRATE PERL REGULAR EXPRESSIONS WITH THE BEST OF SAS PROGRAMMING

When version 9 of the SAS System introduced functions to use Perl Regular Expressions (regex), it seemed a great way to standardize text parsing, I found these regex complex to implement. SAS was always the best at parsing anyway because they have so many ways to (parse) with the INPUT statement. The chief facility is the great collections of formats for reading data. With effect from SAS9.3 we can use these regex as an INFORMAT on INPUT statements or in any of the functions where informats appear - INPUT(), INPUTN() and INPUTC(). The functions bring parsing formats into many more environments than DATA Steps – WHERE statements in most procedures and the SELECT statement of the SQL Procedure - all can use these functions.

The best documentation of this feature is found online (Langston 2012).

As well as describing how regex can be implemented as an informat, this paper describes further expansion of the “user-format” capability in the FORMAT Procedure..

12. PROFILE A DATA STEP TO REVIEW THE DATA FLOW THROUGH STATEMENTS

This feature is rarely spotted. To reveal the flow of data through the statements of a DATA Step, it implements monitoring that adds to CPU time, but sometimes that is a small price to pay for the clarification it provides. For each statement it indicates how often it was executed and what proportion of the step CPU time it consumed. An overall summary is provided for compilation, execution. etcetera.

The system option required is :

```
option dsoptions= profile ;
```

The original DATA Step statements are shown with the results- in appendix 3.

These indicate that the SET and OUTPUT statements have the largest impact on CPU time (approx 40% and 25%) and my INDEX() and FIND() function calls were relatively minor.

CONCLUSION

This paper introduces some lesser spotted programming ideas – which I hope you will find useful.

REFERENCES

[1] Crawford 2003; "Paper 086-28, More _Infile_ Magic"; Proceedings of the Twenty Eighth Annual SAS Users Group International Conference, www2.sas.com/proceedings/sugi28/086-28.pdf

[2] Links to SAS Training
SAS Programming 1: Essentials https://support.sas.com/edu/schedules.html?ctry=gb&id=2588
SAS Programming 2: manipulation https://support.sas.com/edu/schedules.html?ctry=us&id=2618

[3] Crawford 2015; "Paper 1408-2015, Learn Hidden Ideas in Base SAS® to Impress Colleagues"; Proceedings of SAS Global Forum 2015; http://support.sas.com/resources/papers/proceedings15/1408-2015.pdf

[4] Crawford 2014; "Paper 1744-2014, VFORMAT Lets SAS® Do the Format Searching"; Proceedings of SAS Global Forum 2014 http://support.sas.com/resources/papers/proceedings14/1744-2014.pdf

[5] Borst 2015 "Paper 3502-2015, Creating Keyboard Macros in SAS® Enterprise Guide"; Proceedings of SAS Global Forum 2015; http://support.sas.com/resources/papers/proceedings15/3502-2015.pdf

[6] Carpenter 2003; "SUGI 28: Creating Display Manager Abbreviations and Keyboard Macros for the Enhanced Editor"; Proceedings of the Twenty Eighth Annual SAS Users Group International Conference, www2.sas.com/proceedings/sugi28/108-28.pdf

[7] Langston 2012; "Paper 245-2012 Using the New Features in PROC FORMAT"; Proceedings of SAS Global Forum 2012; http://support.sas.com/resources/papers/proceedings12/245-2012.pdf

ACKNOWLEDGMENTS

There are too many to mention individually :

- generous posters on SAS-L helped me appreciate that great community of SAS programmers.
- SAS Customer Support, for always being patient with me when I go off on a wild goose chase (when I should just check my code more carefully).

RECOMMENDED READING

- Programmer's Bookshelf at https://support.sas.com/documentation/index.html

CONTACT INFORMATION

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

Peter Crawford
crawfordsoftware@gmail.com

APPENDIX

1. A MACRO TO GENERATE A MORE COMPLEX PATTERN

This macro provides a more “general purpose” pattern generator :

```
/*--------------------------------------------------------------------*
* a macro to generate patterns over a range up or down or fractional  *
*--------------------------------------------------------------------*/
%macro genpatN( pattern      /* e.g. var### ="this variable### label" */
              , replace = ###    /* in the pattern */
              , from    = 1
              , upto    = 1
              , by      = 1      /* can be negative or fractional*/
              , format  = best8. /* z5.  supports leading zeroes */
              ) /des= 'generate a pattern incrementing a number'   ;
%local i j k ;
%let i= &from ;
%if %sysfunc(lengthN(&by)) <1 %then %do ;
  %put MACROERR: &sysMacroName: need a non blank value of by= %QUOTE(&by) ;
%end;
%else
%if %sysfunc( inputn( &by, best12.)) =.  %then %do ;
  %put MACROERR: &sysMacroName: need a numeric value of BY= %QUOTE(&by) ;
%end;
%else
%if %sysevalf( &by *1.0 ) =0  %then %do ;
  %put MACROERR: &sysMacroName: need a non ZERO value of BY= %QUOTE(&by) ;
%end;
%else %do ;
  %do %while( (  %sysevalf( &BY <0 AND %sysevalf( &UPTO <= &I ) )
              OR %sysevalf( &BY >0 AND %sysevalf(    &I <= &UPTO ) )
            ) ) ;
%qsysfunc( tranwrd( %superq(pattern),%superq(replace),%sysfunc(putn(&i,&format)) ))
  %let i = %sysevalf( &i + &by ) ;
  %end ;
%END ;
 /* demos
%put %genpatN( big###nos,upto=3e9,from=0, by=%sysevalf(1e10/9/9),format= best14 ) ;
%* validation ;
%put %genpatN( abc###qw####, FROM=6, upto=3, BY= a) ;
%put %genpatN( abc###qw####, FROM=6, upto=3, BY= .00) ;
%*** showing off features ;
%put %genpatN( abc###qw####, FROM=6, upto=3, BY= -1e00) ;
%put %genpatN( abc###qw####, upto=13, from=7.4, by= 1.3) ;
%put %genpatN( abc###qw####, upto=13, from=7.4, by= 1.3, format= z5.1) ;
%let qq= %sysevalf( 1e10 / 9 / 9 ) ;
%put %genpatN( big###nos, upto=&qq, from=&qq, format= best14.  ) ;
%put %genpatN( ###, UPTO='31AUG2016'D, FROM ='11NOV2015'D,BY=30,FORMAT= MONYY7.) ;
 *************/
%mend  genPatN ;
;
```

2. SUMMARISE DATE AND TIME FORMATS DISCOVERED

What follows remains a work in progress, but is my current effort to analyse frequency of date or time style strings in a CSV file:

```
/*-------------------------------------------------------------------*
 * first, create some random date/time formats in a CSV file     *
 *-------------------------------------------------------------------*/
filename myrawCSV '!TEMP/some raw data.CSV' lrecl= 10000 ;
DATA _NULL_ ;
  FILE myrawcsv DSD ;
  DO row = 1 TO 1000 ;
    PUT row @ ;
    disturb = RANUNI(2)* 5 ;
    DO noise = 1 TO disturb ;
      PUT noise @ ;
    END ;
    real = 1E6/81*RANUNI(1) ;
    PUT real COMMA12.2 real DATETIME. real DATE11. real TIME. real MONYY7. row ;  END ;
  STOP ;
RUN ;

*-------------------------------------------------------------------*
* analyse CSV cells for date/time formats                           *
*-------------------------------------------------------------------* ;
data any_testing( keep= row column col_fmt ) ;
  infile myrawCSV dsd truncover ;
  array    col(    10 ) $32 ;
  informat col1-col10 $anydtif60. ;
  input    col1-col10 ;
  row+1 ;
  do column= 1 to 10 ;
    col_fmt= col(column) ;
    output ;
  end ;
run ;

*-------------------------------------------------------------------*
* SUMMARISE COLUMNS BY FORMAT                                       *
*-------------------------------------------------------------------* ;
proc summary   data= any_testing ;
  class column col_fmt ;
  types column*col_fmt column ;
  output OUT= column_formats ;
run ;

proc sort ;
  by column descending _freq_ ;
run ;

data fmts_freq ;
  retain column %genPatN( inf### frq###, upto= 20 ) ;
  keep column -- frq20 ;
  array inf(20) $32 ;
  array frq(20) ;
  do ex= 1 to 20 until( last.column) ;
    merge column_formats( in= total where=(_type_ = 2) rename= _freq_ = freq_c  )
          column_formats( in= items where=(_type_ = 3)  ) ;
    by column ;
    inf(ex) = col_fmt ;

    if _freq_ then
      frq(ex)= _freq_ / freq_c ;
    format frq: percent6.1 ;
  end ;
run ;
```

3. PROFILE A DATA STEP

DATA Step Profile Statistics			
Count	**CPU Usage**	**CPU %**	**Phase**
1	0.000	0.00	Initialization
1	0.000	0.00	Compilation
1	0.000	0.00	Resolution
1	0.016	0.02	Execution / Init
1	88.815	99.98	Execution
1	0.000	0.00	Execution / Term
Total CPU Time == 88.83			

DATA Step Statement Profile Statistics				
Count	**CPU Usage**	**CPU %**	**Log Line#**	**Source**
			511	data ;
1	0.000	0.00	512	do _n_ = 1 to 1e7 ;
10000000	4.265	4.80	513	poin_row = 1+int(nobs*ranuni(1)) ;
10000000	34.635	39.00	514	set sampsio.empinfo nobs= nobs point= poin_row ;
10000000	3.559	4.01	515	source= poin_row ;
			516	
10000000	4.837	5.45	517	if index(lowcase(addr1), 'avenue') then do ;
777263	0.312	0.35	518	lla = 1 ;
777263	0.393	0.44	519	fav = find(addr1, 'Avenue') ;
777263	0.251	0.28	520	lav = index(addr1, 'Avenue') ;
777263	0.439	0.49	521	fia = find(addr1, 'Avenue', 'i') ;
10000000	3.541	3.99	522	end ;
10000000	4.864	5.48	523	if index(lowcase(addr1), 'street') then do ;
1260859	0.559	0.63	524	lls = 1 ;
1260859	0.675	0.76	525	fst = find(addr1, 'Street') ;
1260859	0.563	0.63	526	lst = index(addr1, 'Street') ;
1260859	0.585	0.66	527	fis = find(addr1, 'Street', 'i') ;
10000000	3.682	4.15	528	end ;
10000000	21.817	24.56	529	output ;
10000000	3.838	4.32	530	end ;
1	0.000	0.00	531	stop ;
0	0.000	0.00	532	run ;
CPU %%'s Relative to Execution Phase				