Paper 038-31

# The Big Introduction from the Smallest Macro

Peter Crawford, Crawford Software Consultancy Limited

## ABSTRACT

The KISS (see later) approach provides a big learning opportunity for those starting to write their own base SAS® macros. The charming simplicity of the smallest macro makes it very memorable, and introduces the macro beginner to far more issues than it's short (three-line) simplicity might lead a macro newcomer to expect.

(KISS= Keep It Short & Simple ).

## INTRODUCTION

Although it is very short, this base SAS macro:

```
%MACRO  now( fmt= DATETIME23.3 ) /DES= 'timestamp';
 %SYSFUNC( DATETIME(), &fmt )
%MEND   now ;
```

carries some valuable lessons. They are the structure of this paper:

1. macros derive information; contrast this with macro variables which just pass values -- text
2. macros deliver only text, even when dealing with real numbers
3. a macro can generate even less than a statement (no semi colon need be generated)
4. because a macro can derive things, what executes can be determined after a job starts

In this paper, I assume that those who wish to start designing and building base SAS macros, are already familiar with base SAS programming. This paper is about concepts, not syntax.

## VALUABLE LESSONS, 1

### MACROS CAN DERIVE INFORMATION

In TITLE and FOOTNOTE statements, automatic macro variables &SYSDATE and &SYSTIME, provide a convenient way to add flexibility in reporting. However, it takes some processing to reformat these until acceptable. We might want the information passed in macro variables to change while a program runs, so the ability to refresh a value like run-time, is important. It is just not available from static source code. Macros can derive text that will be used as syntax.

## LESSON 2

### A MACRO DELIVERS TEXT, EVEN WHEN IT IS DEALING WITH REAL NUMBERS

There needs to be conversion of derived results into a form the macro environment can handle.

Just as %EVAL() doesn't like decimal points and decimal fractions, the results of functions executed within the %SYSFUNC() environment must control their conversion of numbers into a form that the macro environment can tolerate. Like the rest of base SAS, the macro environment uses formats for converting numbers into strings.

In

```
   %SYSFUNC( DATETIME(),&fmt )
```

the optional second parameter of %SYSFUNC (here it is &fmt), provides a format that will be used to convert the results of the function – here, that is DATETIME().

Not only regular number formatting is available. Any format can be used. Dates and times are the normal formatting you would want for reading, but a simple number of seconds might be more convenient for the timestamp in a filename.

I like the flexibility of date-time user defined formats – The Appendix offers a date-time format that most text import conversion wizards will respect as a date-time value. Where the complete precision of real numbers is needed, you can use a $HEX format to release the content of an 8byte real number into the macro environment without any loss of precision. Just beware that on EBCDIC platforms, $HEX is different from $HEX on ASCII platforms.

## LESSON 3

**A MACRO CAN GENERATE EVEN LESS THAN A STATEMENT**

When we need only some trivial processing for a macro parameter or macro variable, we don't need to run a whole step. The text a macro generates does not need to go through the step boundary (RUN, QUIT, DATA or PROC). It can generate statements like TITLE<n> and SYMBOL<n> and BY statements. Further, as in the very small macro, it can simply generate text for use outside the macro. The technique is not available when the macro generates one or more semicolons because these would terminate the statement where the macro is called. Here is a very simple example:

```
   %PUT job %SYSFUNC( GETOPTION( SYSIN )) finished at %now ;
```

When designing macros, the simpler the better. Focus on a single purpose. Those macros which deliver just text are the simplest to understand. That means they are the simplest to maintain. *(A personal strategy)*

## LESSON 4

**BECAUSE A MACRO CAN DERIVE THINGS, WHAT EXECUTES CAN BE DETERMINED AFTER A JOB STARTS.**

Decision can be deferred until made "on-the-fly" rather than needing to be made when a task/process/job is started. Compare a %INCLUDE statement which brings in stored code, with a macros that generate statement. The nature and content can be determined by a macro examining the business data. You might want a report when PROC PRINT produces nothing because there are no observations in the data set to be printed.

## CONCLUSION

Keep It Short and Simple.
The concepts of base SAS macro design are different from base SAS programming.

in implementing this macro, I learned these non-trivial lessons

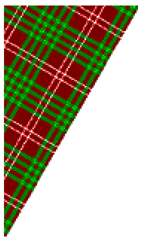
- SAS AUTOCALL environment - makes utilities like %now available most conveniently
- personal preferences versus corporate – for macro support, have different constraints
- don't replace, just extend the environment – and utilities support more widely

Describing these takes longer than a short paper allows.

## CONTACT INFORMATION

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

Peter Crawford
Crawford Software Consultancy Limited
31 Sefton Road,
Croydon, Surrey, CR0 7HS, UK
+44 7802732254
CrawfordSoftware@gmail.com



## APPENDIX

A user-defined format that creates a date-time string that import wizards treat as a value.

```
PROC FORMAT ;
   PICTURE
       u_dt .= ' '
       OTHER =   '%0Y/%0m/%0d %0H:%0M:%0S'( DATATYPE= DATETIME );
RUN ;
```

These %0Y and %0m are defined in online documentation for the PROC FORMAT, PICTURE statement at
http://support.sas.com/onlinedoc/913/getDoc/en/proc.hlp/a002473467.htm#a000530223