A Journeyman's Reference: The Writing for Reading SAS Style Sheet: Tricks, Traps, Tips, and Templates, from SAS-L's Macro Maven

Ronald Fehd, Centers for Disease Control and Prevention, Atlanta GA

ABSTRACT

A program is a form of communication that occurs in two distinct and vastly different events: the first event is immediate, when the program is submitted to the language processor for execution; the second event takes place at a later time, when the writer or another programmer returns to read that program for maintenance. A good style sheet can facilitate program maintenance. This is especially important when simultaneously writing in two languages: SAS and its macro language. This paper discusses elements of a style sheet for SAS programmers. The intended audiences are intermediate SAS programmers and beginning macro programmers.

INTRODUCTION

Writing is a method of transmitting information. Reading is necessary in order to acquire that information. When programming, once the algorithm is decided upon and written well enough to execute, the next step is to visually format the programming language statements so that a later reader can both understand the algorithm and quickly and easily find program statements to be changed. I have developed the Writing for Reading (W4R) SAS Style Sheet during the decade that I have been writing SAS programs and macros. Baecker and Marcus[2] point out that "[s]everal million individuals are now writing programs . . . They are also reading programs, either those that they themselves have previously written or those that others have written . . . The activity of reading programs has always received far less attention than that of writing programs. We teach students how to write programs, but not how to read them. We build tools to facilitate program composition and editing but not program perusal, browsing, and understanding. Those designing new programming languages have focused on logical syntax and semantics, as well they should, but have typically ignored visual syntax and semantics, i.e., program presentation and appearance . . . Enhanced program presentation produces listings that facilitate the reading, comprehension, and effective use of computer programs . . . We believe . . . [that] Making the interface to a program's source code and documentation intelligible, communicative, and attractive will ultimately lead to significant productivity gains and cost savings."

While writing SAS and macro language together I keep in mind the following thoughts:
* Know what you're doing.
* Know where you're at.
* Know where you're going.
* Facilitate later reading.

Tip: Know what you're doing: Number-crunching or string-processing?
SAS is a number-crunching language, while the SAS macro language is an extension of SAS that facilitates both extension and encapsulation. Programmers know the paradigm of number-crunching; becoming familiar with the string-processing capabilities of both SAS and the macro language can be a challenge because of the different set of functions used, the different paradigm, and last, but not least, because the macro processor is a preprocessor for the SAS language.

SAS crunches numbers; macros generate strings. Strings can be tokens, statement phrases, complete statements, or paragraphs consisting of many related statements. Writing macro statements is about writing correct SAS statements, which is why I recommend that you be an intermediate SAS programmer before starting to write macros. Know SAS well before attempting to write macros, which write SAS for you. Two years or 10,000 statements, whichever comes first!

Review SAS character functions in chapter 11 of SAS Language Reference[4]: compress, index, indexc, input, left, put, repeat, reverse, right, scan, substr, translate, trim, verify, call label, call symput, call vname. Do your own testing and become familiar with SAS string-processing. Then go through the various macro manuals – SAS Guide to Macro Processing[3], SAS Macro Facility Tips and Techniques [5] SAS Macro Language Reference[6], and recognize which SAS functions are in the macro language. The main ones: %compress, %index, %scan, %substr. Do some more testing. Comprehend the

difference between SAS function int and macro functions %eval, and %sysevalf.

As you write SAS statements and macros which produce SAS statements, name and remember what you expect:
* paragraph: many statements between step boundaries
* block: keyword + statements + closure, e.g., do; ... end;
* statement: keyword + tokens + closure
* phrase: part of a statement, may be several tokens
* syllable: part of a token, e.g., prefix, infix, or a suffx

Tip: Know where you're at: SAS or macro?
Switching gears – and paradigms – while thinking and then writing is some days an art and other days a science. There are a number of visual aids which have helped me in my career that I recommend. Baecker and Marcus[2] discuss their research on effective layout of computer language manuals. I draw many ideas from them, but am necessarily constrained by having to work with a simple text processor, thus the style sheet below.

Tip: Know where you're going: List processing: Object-Oriented Programming (OOP)
The OOP paradigm has helped me immensely in my programming as I have become accustomed to it in the last several years. When I graduated from college, I knew both number-crunching languages and list-processing languages. When I met SAS, I remembered procedures and functions, and whined with the rest of SAS-L about not having any way to write functions in SAS.

Procedures make way for OOP's methods. You'll need to know your data and, more importantly, your meta-data, i.e., it's structure, in order to work in OOP. Read the SAS Procedures Guide (1990) and familiarize yourself with proc CONTENTS and its output data set. This knowledge is key for the meta- programming that I do in the macro language. While you're at it, do a CONTENTS on the output data sets from other procedures, like FREQ, MEANS, and UNIVARIATE and whatever others you regularly use. Data sets? In OOP these are objects; prepare to juggle them and write methods for them.

Tip: KIS: Keep It Simple!
I do 95% of my work with these dozen macros, listed below. The number of lines is approximate, it includes SAS and macro statements, and excludes comments. The last note is year written and last maintenance date.

name	action	lines	-dates-
	utility		
Array	returns macro array	40	1994:97
MemNames	returns macro array	30	1997:98
Nobs	returns number of obs	10	1991:99
	data review		
ComparWS	list differences	140	1992:98
FreqoSSD	freq all variables	130	1990:98
Invalid	lists invalid values	300	99:2002
	summary		
CheckAll	returns FREQ object	140	1992:99
Freq1Var	returns FREQ object	110	1998:99
FreqXTab	returns FREQ object	130	1999:00
ShowComb	returns FREQ object	320	1992:99
SmryPrnt	lists FREQ objects	120	1996:99
Univari8	returns UNIVARIATE	50	1998:99

My point here is that good utilities are constantly being improved. Most are small, and are built of smaller routines.

Tip: Facilitate later reading. Get a style sheet. Use it, consistently.

The Writing for Reading SAS Style Sheet

Tip: Know what you're doing: SAS or macro
To differentiate SAS language from macro language, type SAS statements in lowercase and macro statements in UPPERCASE. Macro variables are global variables, that is, they exist across SAS step boundaries, just like SAS titles and options. Use of different case reminds you what you're doing. Use all caps, mixed case, and lowercase to visually communicate. * ALL CAPS: global constants: TITLE, OPTIONS, etc. macro language and macro variables data set names, filerefs, librefs. * Upper and Lowercase: variable names. * lowercase: SAS language keywords.

Tip: Know what you're doing: control, conditional or closure
Consider these three categories of statements:
1. control statements, unconditionally executed
2. conditionally executed
3. closure

As illustrated below, the purpose of this style sheet is to facilitate two visual actions that occur

in maintenance: first, scanning, then reading.

What is the first thing I know about a program that I want to improve? It works! The first thing I don't care about is closure: end statements and some semicolons. Place these at the right margin, out of the way. The next thing I know is that I want to change either a control statement or a conditionally executed statement. Thus, separate columns for these two different categories of statements.

Tip: Show what you're doing by placement on the page:

1. left: control
2. center: conditionally executed
3. right: closure

Use indentation of three space. See the data step example below. Larger indentations push the control statements across the page.

Documenting closure. For innermost block, none; for each preceding level: a copy of control statement's keyword.

Avoid Traps with these Tips:

W4R: Know what you're doing: string-processing.

feature: macro language is simple: NULL is empty
factoid 1: list of mnemonics for comparison operators
`AND OR NOT EQ NE LE LT GE GT`
factoid 2: two-letter state abbreviations: OR:Oregon, NE:Nebraska

Trap: OREGON will bite you!

*not good:`;%IF &STATE. = OR %THEN` ...
*consider this;
```
%IF &STATE = <null> OR <condition-2>
\%THEN ...
```

Tip: always quote strings in comparisons
`%IF "&STATE." = "OR" %THEN ...`

W4R: Know where you're at: SAS or macro

Trap: run-on statements: not enough semicolons; can't tell difference between SAS semicolon and macro semicolon?:

expected: `TITLE1 <state name>; TITLE2 <date-stamp>;`

```
TITLE1
%IF "&STATE." = "AL" %THEN Alabama ;
TITLE2 %sysfunc(date(),weekdate17.);
result: TITLE1 "Alabama TITLE2 Mon, Nov
1, 1999";
```

The confusion comes in seeing the semicolon after "Alabama" as the closure of the TITLE1 statement, when it is the closure of the macro

Tip: always use `%DO; <...> %END;`

```
TITLE1
%IF "&STATE." = "AL" %THEN %DO;
Alabama %END;
%*end TITLE1; ;
```

Here it is clear that the macro statement returns only a single token with no closing semicolon.

W4R: Know where you're at: SAS or macro

Trap: one dot is not enough! I can't tell the difference between a SAS dot and macro dot.

Old code : `%LET DATA = DATA A; ... LIBRARY.&DATA`
Improvement: add LIBRARY as a macro variable:
`%LET LIBRARY=LIBRARY;`
New code doesn't work: `&LIBRARY.&DATA` resolves to `LIBRARYDATA A`

Why? Dot changes from a two-level name delimiter to a macvar delimiter.

Tip: always use macro delimiters: ampersand and dot

Use snake-eyes in two-level names, formats, and filenames.

```
!Wrong!.            correct..
&LIBRARY.&DATA.     &LIBRARY..&DATA.
$char&WIDTH.        $char&WIDTH..
&FILENAME.sas       &FILENAME..sas
...........^        ...........^^
```

W4R: Know what you're doing; remember what you did.

Trap: many ad hoc reports

Tip: recognize patterns, write a general solution.

Write SAS twice before writing macro once. recognize patterns, write specific solution first, and again, then write a general solution. Pattern recognition is key.

W4R: Know what you're doing: manage complexity

Trap: large macro, with no complexity

Macro complexity is zero if there are no %IF and no %DO loops. If your SAS statements contain only macro references and no conditional execution nor loops of any statements consider using a parameterized %include file. Instead of:

```
%macro PRNT(DATA); PROC PRINT data =
&DATA.;
TITLE "&DATA."; run; %MEND;
```

Tip: use global macro variables with %include

```
- - - - - PRNT.sas - - - -
proc PRINT data = &DATA.;
TITLE "&DATA.";run;
- - - - end PRNT.sas - - - -
- - - - - callPRNT.sas - - - -
filename PRNT "path to:PRNT.sas";
%LET DATA = DATA1;
*LET DATA = DATA2;
%include PRNT;
- - - - - end callPRNT.sas - - - -
```

To run the program enable the %LET statement with the desired data set name.

Trick: conditional execution of %includes

The %include statement, despite its percent sign, is not a macro statement, and is always executed in SAS; though it can be conditionally executed in a macro. Here is a simple trick that shows a macro variable being used to generate a syllable – the suffix – of a token, in this case a fileref. To run the program and %include FileB, enable the second %LET statement by changing asterisk to percent.

```
filename FILEA "c:\sas\fileA.sas";
filename FILEB "c:\sas\fileB.sas";
%LET WHICH = A;
*LET WHICH = B;
data;
*replace this:
if "&WHICH." = "A" then %include
FILEA;
else if "&WHICH." = "B" then
%include FILEB;
*with this;
%include FILE&WHICH.;
```

Templates: no writing, just cut&paste

Hanging Indent is a paragraph style with the first line flush left to the margin and succeeding lines indented. I use this style to illustrate the concept, which is the basis of my programming templates. The SAS editor, and other well-known text editors, indent succeeding lines to the same level as the previous line.

Music. Programming and music share a common skill set:

1. Pattern recognition.
2. Sequential processing.
3. Symbol manipulation.

Templates. Patterns occur everywhere in our experience. The essence of a pattern is not what is obviously happening but its similarity to some previous event. Templates are reusable instances of patterns previously recognized.

Natural language, such as English, is a spoken as well as written means of communication. We learn a natural language in four steps: listening, speaking, reading, then writing. We practice each of these abilities and perfect them through a continual process of reception and reproduction. Consciously or unconsciously, between reception and reproduction is the hindsight of the pattern recognition process and the foresight of our personal template refinement.

Syntax. The English sentence pattern contains three elements: subject, verb, and object.

Artificial language is used to communicate with a machine. An artificial language can be used to convey a series of instructions, e.g., using a touch tone telephone, or a VCR remote control. SAS, like other computer languages, has no need of a subject in its statements. Every declarative sentence is addressed to the mythical computer, whether on our desktop or in a room somewhere else. The Department of

Redundancy Reduction Dept. has decided to eliminate computer as the subject of artificial language statements.

Keywords. Elimination of subject reduces our artificial language template to verb and object. How many of us are still trying to comprehend the object oriented programming paradigm in SAS? Here's the clue: keywords are verbs, in the sense that a keyword acts upon the following words as objects.

Sequence. Order is important. In rhetoric we state our assumptions first. In programming we describe the global environment, then the local.

Symbol. Each programming step has these processes: declaration, description, and manipulation. In the data step manipulation consists of two interrelated processes: control and conditionally executed statements.

Switch. Which template are you using when you write SAS? Do your statements begin with a capital letter and end with a period? Are statements separated from each other by two spaces? Are collections of statements known as paragraphs separated by a blank line, or a new line and paragraph indent of five spaces? Do your paragraphs begin with topic sentences and end with transition sentences? Consider whether your audience is reading a natural language or an artificial language.

Badness. SAS typed as English. Statements begin with Initial caps and are separated by two spaces. Paragraphs are indented five spaces. SAS is written as deathless prose. Can it get any worse? When we wish to debug or maintain we have to read every word. A good program should be easy to maintain because of the style in which it is written. The Maintenance Programmer should not have to read nor understand the whole program, just the section that needs work. See Aster[1].

Goodness. The Writing for Reading SAS Style Sheet reminds you that you are about the business not of reading the whole program, but of scanning to a section, then focusing on that step in the process. A step is a statement is a verb. We want to find the verbs quickly. Order is important, placement is important, sequence is important. White space enhances order, placement, and sequence.

Templates, for data step, procedure, and macro, using the three template verbs: declare, describe, manipulate.

```
               SAS declarative statements
Declare   : 01 DATA DATANAME1 (<options>);                                          01
Describe  : 02 attrib VarA length = $8 format = $formatName.                        02
            03 label = 'var label';                                                 03
            04 set LIBRARY.PREVIOUS                                                 04
            05 by Var;                                                              05

Manipulate:    SAS executable statements
               control statements        conditionally executed    closure
               ------------------------- ------------------------- -------
            10 if condition1             then assignment;                        10
            11 if condition2             then do;                                11
            12    do   I = 1 to 2;                                                 12
            13      do J = 1 to 2;       assignment1;                            13
            14                           assignment2;                            14
            15                           end;                                    15
            16                           %*do I;                   end;          16
            17                           %*if condition2;          end;          17
            18                                                     run;          18

notes: lines 11:13 three space indent,
             12:13 align similar construction to highlight different indexes.
       lines 15:18 15 is closure for do in line 13, no comment since it is inner
                   block
             16:17 close outer blocks 12 and 11, thus they have comments and
                   closure on right.
```

```
              Declare   Describe
              --------- --------------------------------
            01 proc FREQ data   = LIBRARY.WHATEVER                                  01
            02           (where = ( <subset selection> ) )                          02
            03           ;                                                          03
            04           by       var1 ;                                            04
Manipulate: 05           tables   varlist                                           05
            06                    / list missing noprint                            06
            07           out    = WORK.FREQ                                        07
            08           ;                                                          08

declare 1:  01 ;/* macro MAINVERB describe purpose                                  01
describe1:  02 USAGE:    for cut and paste into a calling program                  02
            03 %MAINVERB(SUB_VERB1 = value                                         03
            04          ,SUB_VERB2 = value                                         04
            05          );                                                         05
            06 %MAINVERB(DATA  = DATANAME1                                         06
            07          ,WHERE = VarA eq 2                                         07
            08          ,OUT   = SAVETHIS                                          08
            09          );                                                         09
            10 *** ............... ..................................... */         10
declare 2:  11 %macro MAINVERB                                                     11
describe2:  12 (SUB_VERB1 = /*parameter 1 description NOTE:all parameters are named */ 12
            13 ,SUB_VERB2 = /*parameter 2 description no positional parameters */  13
            14 ,DATA = /*one- or two-level data set name */                        14
            15 ,WHERE = /*subset? */ 15
            16 ,OUT = /*name of output data set, also: returned object */          16
            17 ,TESTING = 0/*want TESTING messages printed? */                     17
            18 );                                                                  18
manipulate: 19 DATA X;                                                             19
            20 set &DATA.(                                                         20
            21     where=(&WHERE.)                                                 21
            22            );                                                       22
            23 %IF &TESTING %THEN %DO; proc PRINT data = X;                        23
            24                         title "data X";                  run;%END;  24

            90 %EXIT: run; %* .......... ............................. .... *; %MEND;  90
            91  /* TEST DATA ********** **********to enable end this line w/slash **  91
            92 %MACRO-NAME(DATA    = X                                             92
            93            ,OUT     = Y                                             93
            94            ,TESTING = 1                                             94
            95            );                                                       95
            96 RUN; /* .................. ............................. .............*/ 96
```

CONCLUSION

Once a program is written it will have to be read. Ease of reading facilitates maintenance. Templates facilitate a uniform program style. A style sheet helps differentiate between SAS and the macro language. Using mnemonics facilitates ease of reading. Get a style sheet; use it, consistently; you'll be glad you did.

REFERENCES

[1] Rick Aster, (1998), Coding for Posterity, Proceedings of the Eleventh Annual NESUG Conference

[2] Ronald M. Baecker, Aaron Marcus, (1990), Human Factors and Typography for More Readable Programs, ACM Press, Addison-Wesley Publishing Company, ISBN 0-201-10745-7

[3] SAS Institute Inc. (1990), SAS Guide to Macro Processing, Version 6, Second Edition, Cary, NC: SAS Institute Inc.

[4] SAS Institute Inc. (1990), SAS Language Reference, Version 6, First Edition, Cary, NC: SAS Institute Inc.

[5] SAS Institute Inc. (1994), SAS Macro Facility Tips and Techniques, Version 6, First Edition, Cary, NC: SAS Institute Inc.

[6] SAS Institute Inc. (1997), SAS Macro Language Reference, First Edition, Cary, NC: SAS Institute Inc. catalog 55501

[7] SAS Institute Inc. (1990), SAS Procedures Guide, Version 6 Third Edition, Cary, NC: SAS Institute Inc.

Author: Ronald Fehd bus: 770/488-8102
Centers for Disease Control MS-G23
4770 Buford Hwy NE
Atlanta GA 30341-3724 e-mail:
RJF2@cdc.gov

ACKNOWLEDGMENTS

This knowledge and wisdom was gained over the last decade while I crunched numbers and read SAS-L. Many thanks to the other SAS Whizards(tm) that contribute to SAS-L. You know who you are. I couldn't have done it without you.

This paper was typeset in LATEX. For further information about using LATEX to write your SUG paper, consult the SAS-L archives:

http://www.listserv.uga.edu/cgi-bin/wa?S1=sas-l
Search for :
The subject is or contains: LaTeX
The author's address : RJF2
Since : 01 June 2003