Paper 38-25

The Writing for Reading SAS® Style Sheet: Tricks, Traps & Tips 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 it's 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(1990) 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 elements of statements, known as tokens; complete statements; or paragraphs consisting of many related statements. Writing macro statements is all 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 (1990): 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 (1990), SAS® Macro Language Reference (1997), SAS® Macro Facility Tips & Techniques (1994) – 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 suffix

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 on my career that I recommend. Baecker and Marcus (1990) 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 several number-crunching languages and as many more 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.

Anyway, 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 about a 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.

utilities:

Array	returns macro array	40 lines	1994:97
MemNames	returns macro array	30 lines	1997:98
Nobs	returns number of obs of data set	10 lines	1991:99

data review: all use Nobs and Array

ComparWS	list differences of two data sets	140 lines	1992:98
FreqoSSD	lists freq of all variables	130 lines	1990:98
Invalid	lists invalid values in all variables	300 lines	1999

data summary: all use Nobs and Array

CheckAll	returns FREQ object	140 lines	1992:99
Freq1Var	returns FREQ object	110 lines	1998:99
FreqXTab	returns FREQ object	130 lines	1999
ShowComb	returns FREQ object	320 lines	1992:99
SmryPrnt	lists FREQ objects	120 lines	1996:99
Univari8	utility, returns UNIVARIATE object	50 lines	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, librefs.
* Upper and Lowercase: variable names.
* Lowercase: SAS® language.

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 acts that occur in maintenance reading, 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 one space. See the example below. Larger indentations push the control statements across the page. This is unnecessary with the three-column style.

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 %IF statement.

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 difference between a SAS® dot and macro dot.

Old code: %LET DATA = DATA1; ... LIBRARY.&DATA
Improvement: add LIBRARY as a macro variable:
%LET LIBRARY=LIBRARY;
New code doesn't work: &LIBRARY.&DATA resolves to LIBRARYDATA1
Why? Dot changes from a two-level name delimiter to a mac-var delimiter.

Tip: always use macro delimiters: ampersand and dot

Use snake-eyes in two-level names, formats, and filenames.
!Wrong! &LIBRARY.&DATA. $char&WIDTH. &FILENAME.sas
correct: &LIBRARY..&DATA. $char&WIDTH.. &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 then 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® - - - -
filename THISFILE "path.filename.sas";
%LET DATA = DATA1;
*LET DATA = DATA2;
%include THISFILE;
```

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.;
```

Tip: Know what you're doing.

Comments? "SAS® code is self documenting." said the programmer, who had made sure he was indispensable.

CONCLUSION

Once a program is written it will have to be read. Ease of reading facilitates maintenance. 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

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

SAS® Institute Inc. (1990), *SAS® Guide to Macro Processing, Version 6, Second Edition,* Cary, NC: SAS® Institute Inc.

SAS® Institute Inc. (1990), *SAS® Language Reference, Version 6, First Edition,* Cary, NC: SAS® Institute Inc.

SAS® Institute Inc. (1994), *SAS® Macro Facility Tips & Techniques, Version 6, First Edition,* Cary, NC: SAS® Institute Inc.

SAS® Institute Inc. (1997), *SAS® Macro Language Reference, First Edition,* Cary, NC: SAS® Institute Inc. catalog # 55501

SAS® Institute Inc. (1990), *SAS® Procedures Guide, Version 6 Third Edition,* Cary, NC: SAS® Institute Inc.

Author: Ronald Fehd e-mail: RJF2@cdc.gov
Centers for Disease Control MS-G25
4770 Buford Hwy NE
Atlanta GA 30341-3724 voice: 770/488-8102

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.

Examples of programming styles: Worst, Students' Block,
and The Writing for Reading SAS® Style Sheet.

Worst: paragraph, run-on statements;
Who can read this? Who would want to try? You have to read every line.

```
do I=1 to 10;if <condition.1> then do;<statement1.1>;<statement1.2>;<sta
tement1.3>;end;else if <condition.2> then do;<statement2.1>;<statement2.
2>;<statement2.3>;end;else<statement3.0>;end;
```

Students' Block style, with 4-character indent
What we all learned in college. Are we still students? Notice that you still have to read every line, beginning at the left margin.

```
do I = 1 to 10;
    if <condition.1> then
        do;
            <statement1.1>;
            <statement1.2>;
            <statement1.3>;
        end;
    else
    if <condition.2> then
        do;
            <statement2.1>;
            <statement2.2>;
            <statement2.3>;
        end;
    else
            <statement3.0>;
end;
```

The Writing for Reading Style: one-char indent, 3 columns
Notice that you don't have to read every line, because you know which type of statement you are scanning for -- control or conditionally executed -- and the column where they are located.

```
*---control statements:--;        *---conditionally---;     *-closure:-;
*unconditionally executed;        *executed statements;     *end,  semi;

do I = 1 to 10;
 if      <condition.1> then do;   <statement1.1>;
                                  <statement1.2>;
                                  <statement1.3>;                   end;
 else if <condition.2> then do;   <statement2.1>;
                                  <statement2.2>;
                                  <statement2.3>;                   end;
 else                             <statemend3.0>                       ;
                                                                %*do I; end;
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012
....+....1....+....2....+....3....+....4....+....5....+....6....+....7
```