Paper OO-02-2013

Macro Design Ideas: Theory, Template, Practice

Ronald J. Fehd, Stakana Analytics, Atlanta, GA, USA

Abstract

Description : This paper provides a set of ideas about design elements of SAS(R) macros.

Purpose : This paper is a checklist for programmers who write or test macros.

Audience : advanced users or intermediate programmers polishing, testing, or writing macros.

Programs : in this paper are available on:

`http://www.sascommunity.org/wiki/Macro_Design_Ideas`

Keywords : macro design

Quote : Quality is free, but no one is ever going to know it if there isn't some sort of agreed-upon system of measurement.

— Philip Crosby: Reflections on Quality

Contents

Introduction

Overview

This section review the context in which macros are written.

Who? : Who writes macros? Either users or programmers write macros.

What? : What is a macro? A macro is a program which can replicate statements around values supplied in parameters.

Why? : Why write macros? These are some reasons to create macros.

1. reuse
2. encapsulate complexity
3. to use either of the macro statements `%do` or `%if`
4. to use either of the macro functions `%sysevalf` or `%sysfunc`

When? : When are macros written, or polished? Macros are written after ad hoc programming produces several examples of similar processing that can be simplified into a macro. Polishing is best accomplished before peer review.

Where? : Where are macros? Macros occur in these places:

- within a program
- in a program in a project
- in a site folder available to all projects

Summary

These ideas on macro design are for programmers writing or polishing macros for use in either a project or site.

Theory

Overview

Bricolage is the art of creative tinkering. Closely related is the idea of kludge: a workaround. Many macros start as kludges discovered whilst in bricolage.

The following topics are areas for further study and ideas for consideration during bricolage.

- Logic or Philosophy
- Quality
- Testing
- Vocabulary

Logic or Philosophy

Overview

One of the keys in design of a good macro is conditional processing. This section covers the historical development of logical reasoning from the 19th to the 20th century

- Boole: not, and, or, xor
- De Morgan: nand, nor
- Venn diagrams for SQL joins

Boole's Rules

George Boole is the author of rules now recognized as Boolean Logic, which are here reduced to three elements.

- unary operation
- binary operators
- precedence rules

unary : The unary operator is *not*. It flips or reverses its argument.

P	not
T	F
F	T

binary : In Boolean Logic there are two binary operators: *and*, and *or*. Each can be used to identify a particular combination of two values. In natural language we use the term *or* instead of the formal term *xor* meaning *exclusive or*. This operator is included in this definition for clarity in later discussions of Venn diagrams and sql (database) joins.

Continued on next page.

This is the truth table of the operators with De Morgan's extensions.

				derived:	De Morgan	
P	Q	and	or	xor	nand	nor
T	T	T	T			
T	F		T	T	T	
F	T		T	T	T	
F	F				T	T

Blanks here are False.

Notes:

- Notice that the *or* operator includes the rows identified by *and*
- Exclusive or (*xor*) is true when either one or the other, but not both, are true. Joins in sql identify each of the `xor` rows: xor(T,F) is `outer left`, xor(F,T) is `outer right`. De Morgan's `nand` includes both of the `xor` rows.

precedence : Parentheses have been added to the precendence list in acknowledgment of the recursive nature of logical resolution.

1. parentheses
2. negation
3. and
4. or

See also: SAS Operators in Expressions; and the SAS OnLine Documentation: operators, Boolean, page-title: SAS Operators in Expressions.

! $\rightarrow$ The importance of parentheses in evaluations is addressed by De Morgan's definitions of *nand* and *nor*.

De Morgan's Laws

Augustus De Morgan was a contemporary of Boole. De Morgan's Laws are stated in formal logic. *Conjunction* means *and*; *disjunction* means *or*.

nand : The negation of a conjunction is the disjunction of the negations.
nor : The negation of a disjunction is the conjunction of the negations.

operator	with parentheses				without parens		
nand	not(P	and	Q)	==	not P	or	not Q
nor	not(P	or	Q)	==	not P	and	not Q

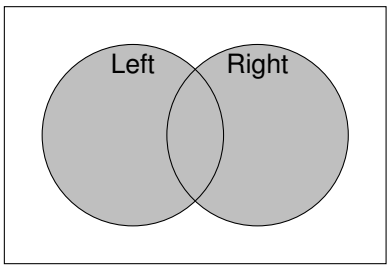
! $\rightarrow$ Notice that the parentheses are necessary!

Venn Diagrams for SQL Joins

John Venn was an English logician known for the visual representations of set theory known as Venn diagrams. SQL joins are understood more easily using Venn diagrams.

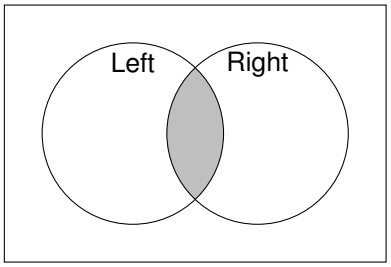
operator	sql	SAS	notes
and	inner join	intersect	compare nand
or	full outer join	union	
xor-left	left join	L except R	
xor-right	right join	R except L	caution!
nand	outer join		xor-left plus xor-right

Or This diagram illustrate Boole's logical operator *or(T,?)* which includes three sets: and(T,T), xor(T,F) and xor(F,T).



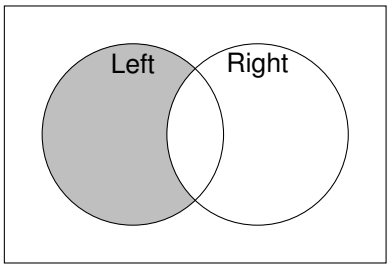
A or B: full outer join, union

And The logical operator *and(T,T)* identifies the row where both values are true.



A and B: inner join, intersect

Xor.Left These diagrams illustrate the extensions of Boole's logical operator Exclusive Or which identifies two combinations: (*xor(T,F)* and *xor(F,T)*);.

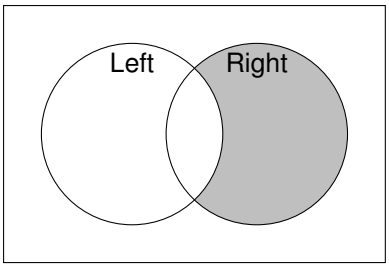


xor(T,F): A xor B: left outer join, except

Continued on next page.

Xor.Right

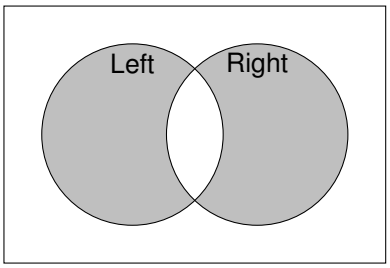
xor(F,T): A xor B: right outer join, except



! → For a right outer join in SAS be sure to swap the Right table to the first position!

Nand: Xor This diagram illustrate De Morgan's logical operator: *nand*, which includes the two rows identified by *xor(T,?)*: xor-left and –right. Visually, this shows *not(and(T,T))*.

A xor B: outer join



Notes:

- See also SAS OnLine Documentation for the following pages:
- outer joins: page-title: Creating and Using Outer Joins
- outer joins: full outer joins, page-title: Selecting Data from More than One Table by Using Joins, contains Venn diagrams for inner (and) and outer (xor, xor-left) and full outer (or) joins
- outer union set operator, page-title: query-expression

  discusses outer union, (inner) union, except and intersect
- Fehd, Evaluating Logical Expressions
- Lafler, Exploring DATA Step Merges and PROC SQL Joins

Summary

Why do we need an understanding of logic? Because the macro `%if` statement is one of the reasons for writing a macro.

Quote

Logic is an organized procedure for going wrong with confidence and certainty — Charles F. Kettering (1876-1958) American inventor, engineer, "Kettering's Law," from address before American Society of Mechanical Engineers (c. 1944)

Quality

Overview

Quality is an elusive concept. We use adjectives such as 'good' and 'high' to modify 'quality'. But quality assurance is simple: does the product conform to the specifications? This section reviews authors who defined our sense of quality, first quality control, and later quality assurance.

- Pareto
- Shewart
- Deming
- Juran
- Crosby

Pareto

Vilfredo Pareto, an Italian economist, wrote in 1906 that 80% of the land in Italy was owned by just 20% of the population. This idea was named a rule in the 1940s by quality managment consultant J. M. Juran who popularized it with the slogan: "the vital few and trivial many". The essence of the idea is that focusing on a few key elements produces the greatest return. In most real-world examples the numbers are not exactly 80/20 but may be 70/30 or 90/10; and their sum does not have to equal 100%.

See Juran's reframe of 'trivial many' to 'useful many' below.

Shewart

Walter A. Shewhart is often referred to as the father of statistical quality control based on a memo he wrote while working at Western Electric Company in 1924 which separated assignable-cause (signal) from chance-cause (noise).

Here are his data presentation rules:

- Data have no meaning apart from their context.
- Data contain both signal and noise. To be able to extract information, one must separate the signal from the noise within the data.

Notes:

- Shewart is the author of the acronym PDCA: Plan-Do-Check-Act.
- Deming noticed Shewhart's publication in 1938 and reframed his own vocabulary about measurement error to the terms used by Shewhart.
- See also: SAS documentation for the Shewart and Ishikawa procedures.

quote:

- Information is *the* difference
  that makes *a* difference

  — Gregory Bateson, 1904–1980, Steps to an Ecology of Mind, 1972

Continued on next page.

Deming

W. Edwards Deming and Juran were contemporaries in the 1940s and -50s; both consulted with Japanese companies during the post-war period. Deming is considered more practical for his focus on statistical process control. Deming's famous quotation is:

1. When people and organizations focus primarily on quality, defined by the following ratio,

   $$Quality = \frac{Results\ of\ work\ efforts}{Total\ costs}$$

   quality tends to increase and costs fall over time.
2. However, when people and organizations focus primarily on costs, costs tend to rise and quality declines over time.

Deming reframed Shewart's acronym PDCA: Plan-Do-Check-Act to PDSA: Plan-Do-Study-Act.

Juran

Joseph M. Juran is famous for reframing Pareto's phrase "the vital few and the *trivial* many" to: "the vital few and the *useful* many". His emphasis during his consulting work in Japan in the 1950s was on educating managers about their cultural concepts of quality. This is Juran's trilogy for managers:

- quality planning
- quality control
- quality improvement

Juran brought Kaoru Ishikawa's concept of quality circles to the U.S.

Crosby

Philip B. Crosby is the author of the sound bytes:

- Zero Defects
- Quality is Free
- Do it Right the First Time

Here are his four major principles:

1. The definition of quality is conformance to requirements (requirements meaning both the product and the customer's requirements).
2. The system of quality is prevention.
3. The performance standard is zero defects (relative to requirements).
4. The measurement of quality (the cost) is the price of nonconformance.

Summary

Why do we need an understanding of quality assurance? Because managerial support for documentation and specifications are necessary when writing macros.

Testing

Overview

Fred Brooks in his book, The Mythical Man Month, provides the following table which describes how much time was spent on testing during development of the operating system for the IBM OS/360 during the 1960s.

Phase	Time	Action	Time
Design	1/2	Understand Problem:	1/3
		Education and Research	
		Development Coding	1/6
Testing	1/2	Component or Unit Test	1/4
		Systems or Integration Test	1/4

! → Note the difference between debugging, which occurs during development, and testing which occurs afterwards.

Basics

Two decisions are needed in order to write testing statements in a macro.

- name of macro variable, recommended: testing, default = 0==false
- information useful during testing: data structure

```
%macro Demo_Testing
       (testing=0);
%*reset testing: add value of global testing option(s);
%let testing = %eval(   &testing
                     or %sysfunc(getoption(mprint)) eq MPRINT);
%*...;
%if &Testing %then %do;
    PROC SQL; describe table &syslast;
              quit;
    %end;
```

The `describe table` statement writes the data structure to the log.

The `contents` and `datasets` procedures can be used as well; their output is written to the output window.

```
PROC Contents data = &syslast;
PROC Datasets library = work;
              contents data = &syslast;
              quit;
run;
```

Varieties

The varieties of testing experience are described by

Fehd [11, nesug2007.cc12]; see also: Writing Testing-Aware Programs.

Summary

Why do we need an understanding of software testing?

Because testing takes up half our budget!

Vocabulary

Overview

A discussion of vocabulary and assumptions is a necessary precursor to effective communication during project development. This section reviews a few items that macro writers ought to agree on.

- program types
- SAS types
- naming conventions
- style guide

! → This story illustrates just how different corporate cultures can be.

> The $125 million Mars Climate Orbiter crashed on the planet because the Jet Propulsion Laboratory used metric units, while engineers at Lockheed Martin used feet and pounds.

Program Types

These are theoretical types and their definitions are exclusive.

1. module: top level, calls routines and subroutines
2. routine: middle level, calls other routines and subroutines
3. subroutine: lowest level, no calls

The author offers the recommendation that SAS programs have three levels, and in extraordinary circumstances a maximum of four. Having more levels in one program increases the complexity of testing and identifying the source of problems.

SAS Types

Program types in SAS are not so definitive as the theoretical types. These definitions are non-exclusive and therefore may overlap.

1. program: has multiple steps, either data or procedure
2. parameterized %include, is.a program with one or more global macro variables in the calling program which act as parameters
3. macro: replicates one or more statements or steps
4. macro function: is.a subroutine, returns one or more tokens within a statement

---

Naming Conventions

Refer to the Style Guide item on capitalization to determine whether to use lowcase with underlines or InitCaps to differentiate global from local macro variables.

Continued on next page.

Style Guide

A Style Guide is a list of preferences for the following items:

- capitalization: lowcase, InitCaps, UPPERCASE
- indentation: tab, 2, 3, 4, 8 spaces
- line width maximum
- naming conventions
- white space

! → This is the author's style guide

- capitalization: see also naming conventions
  - lowcase is the default
  - use InitCaps or under_lines for nouns
  - UPCASE: AVOID! ALL CAPS IS HARD TO READ!
- indentation: data: 3, macros: 4
  replace tabs with spaces to ensure consistent printing
- line width maximum: 72; ensures cross platform compatibility
- one semicolon per line, maximum
- white space between tokens: use to align items for comparison

---

Why Capitalizalization Rules?

This story is from the 1980s when printers were 9-pin dot-matrix.

```
compare capital eye  I
    with lower case el
```

I once programmed a system that came to me with a five-year-old bug. The value of a key data element — shrinkage in the customer's inventory — always came back zero. ... The code logs showed that six programmers before me had failed to fix the bug. ... One day, after some eight weeks of searching, I ... saw the reason for the zero. ... a simplified explanation is that the code read:

`key_data_element=I_value`	(capital *I* (eye),	which had been initialized to zero),
		when it should have read:
`key_data_element=l_value`	(lower-case *L* (el),	holding the real value).

Now this is truly awful programming. No variables should be given such similar names, especially not when their sole differentiator is two letters nearly identical visually. Six programmers before me, looking at code on our white-on-green character screens,
could not tell *eye* from *el*. —Ellen Ulman, Printing, in Wired magazine April 16, 2013.

Summary

The primary purpose of documentation is that it is read and understood by both its creators and users before and after release. A common vocabulary — both stylistic and visual — is necessary for effective communication because readability is the first requirement for reuse.

Templates

Overview

In this section we review the parts of a model macro.

1. documentation
2. macro

Documentation

Overview

The primary audience of documentation is for colleagues before and during peer review. Secondarily, good documentation is the best advertising of the macro for users. Finally, readability is a requirement for reuse.

- identify: name and author
- information:
  - summary
  - contexts
  - specifications
- example
- notes

identify : write a generic *file-specification*

```
/*       Name: <UNC>\SAS-site\macros\callmacr.sas
        Author: Claude Shannon  1963
```

information : This is advertising that the macro has a design.

```
Summary       : description  : does what to which?
                purpose      : used when, where?
Contexts      : program group: data cleaning, ???
                program  type: routine|subroutine
                SAS      type: macro
                uses routines: ...
Specifications: input  : required:
                         optional:
                process:
                output :
```

examples : If at all possible provide an example, with data sets from the libref sashelp, that will work with no initialization.

```
Usage Example:
PROC Freq data = sashelp.Class;
```

Show output from the example: notes in log or listing.

Example

This is the documentation of Fehd [9, sco.Macro-CallMacro].

callmacr.sas

```
 /*       Name: <UNC>\SAS-site\macros\callmacr.sas
        Author: Ronald J. Fehd  2012, 2013
                ------------------------------------------------------
Summary       : description  : call macro using
                               all values in data set row as parameters
                purpose      : provide generic method to call macro
                               using list::control data set of parms
                ------------------------------------------------------
Contexts      : program group: list processing token generator
                program  type: routine
                SAS      type: macro function
                uses routines: macro named in the parameter macroname
                ------------------------------------------------------
Specifications: input  : required: Data, MacroName
                         optional: MacroParms, Hex16
                process: assemble macro-call, call
                output : from MacroName
                ------------------------------------------------------
Parameters    : Data       = one- or two-level data set name
               ,MacroName  = name of macro to call
               ,MacroName  = put :: default, for testing
               ,MacroParms = additional parameters for called macro
                             *-Constraint-*: must be enclosed in nrstr:
               ,MacroParms = %nrstr(data=sashelp.class,var=Sex)
               ,Hex16      = 1 :: default, convert numerics to hex16
                             used to pass real numbers accurately
                                  across step boundaries
               ,Hex16      = 0 :: pass numerics as decimals
               ,Semicolon  = 0 :: no semicolon after macro call
               ,Semicolon  = 1 :: use when macroname is a statement
                                  note: auto-reset when macroname=put
               ,Testing    = 0 :: default, no extra messages in log
               ,Testing    = 1 :: for testing, note: auto-reset when
                                  options mprint source2;
                ------------------------------------------------------
Bells,Whistles: writes real time used to log
                note: CALLMACR used real time  0:00:00.016
-------------------------------------------------------------------------
Usage Example:
PROC Freq data   = sashelp.Class;
          tables   Sex
                 / noprint
             out =     Work.Freq_Class_Sex;
run;*necessary;
%callmacr(Data       = Work.Freq_Class_Sex
         ,MacroName  = Put note:
         ,MacroParms = %nrstr(data=sashelp.class,var=sex)
         )
log:
note:(data=sashelp.class,var=sex
     ,Sex=F,COUNT=4022000000000000,PERCENT=4047AF286BCA1AE7)
```

Summary

Good documentation is essential to two groups of people:

- colleagues reading before and during peer review, when the majority of errors can be found
- users who want to know how to use the macro

! $\rightarrow$

- remember: readability promotes reuse
- see also: Rhodes [17, sgf2013.Programming-Standards]

A Macro

Overview

A macro is similar to a data step:

both contain compile and execution (run-time) statements.

Data Step

A data step has two sets of statements:

- compiler directives which describe the data structure
- execution statements which implement the algorithm

```
* compiler directives: create input buffer and PDV;
DATA data-set-name
     (label = 'The New stuff');
     * define new variables;
     attrib VarB length = <$> 4
                label  = 'describing Var B';

* execute algorithm: assignment(s);
do until(EndoFile);
   set libref.data-set-old  end = EndoFile;
   * assignments;
   output;
   end;
stop;
run;
```

Continued on next page.

Macro

Macros have a similar set of compile and execute statements:

- compile:
  - name
  - parameters
  - description
  - local
- execute:
  - assignments
  - assertions: `%if`
  - program: `%do`

name : The name of the macro can provide answers to the question:

Does what action on which object?

The simplest idea is to use a verb and an object in the name. The verb names the process; the noun may be either the input or the output.

parameters : Use parameter names that echo their usage in the program; if at all possible provide default values for unit testing.

description : Provide a description, which shows up in the `catalog` procedure output, that indicates where the macro resides: project or site, and what it does, adding more information than the macro name.

local : The macro may require temporary variables.

assignments : In support of testing a macro may standardize parameter values and logically add values from options. In a user-friendly macro a data set name may be one- or two-level; if assertions need two macro variables then recode.

assertions : If parameter values are invalid then exit gracefully.

See also: Fehd [7, sco.Cond-Exec-Global-Stmnts]

Continued on next page.

Template for a Macro

```
%macro sort_by_category
       (data    = sashelp.class
       ,by      = sex
       ,out     = work.sorted /* may be one_level or two.level */
       ,testing = 0
       )
/ /* ** store source /* */
  des = 'site: this macro does ...';
%***** allocate temp var names;
%local out_lib out_data;

%*** assignments;
%let data = %lowcase(&data);
%let out  = %lowcase(&out);
%let out_lib  = work;
%let out_data = &out;
%** if out is.a two-level name (has dot) then split;
%if %index(&out,.) %then %do;
    %let out_lib  = %scan(&out,1,.);
    %let out_data = %scan(&out,2,.);
    %end;
%** reset: add options info to var testing;
%let testing = %eval(not(0 eq &testing)
                     or %sysfunc(getoption(mprint))
                                 eq %upcase(mprint));
%** assertions;
%if not %sysfunc(exist(&data)) %then %do;
    %put %str(ER)ROR: %sysmacroname &data not exists;
    %return; %* RETURN means jump to mend;
             %* see also %goto exit;
    %end;
%** !NOTE!:LIBREF function returns 0 if libref has been assigned;
%if %sysfunc(libref(&out_lib.)) %then %do;
    %put %str(ER)ROR: %sysmacroname libref of &out not assigned;
    %goto exit;
    %end;
%if %sysfunc(exist(&out.)) %then %do;
    %put Note: %sysmacroname &out was overwritten;
    %end;
PROC Sort data = &data
          out  = &out;
          by     &by;
%if &Testing %then %do;
    PROC Sql; describe table &syslast;
              quit;
    %end;
%exit: %* destination == label of %goto;
run;
%mend <macro-name>;
```

Practice: Tricks or Traps

Overview

This section provides examples of common construction items and provides examples of solutions.

- evaluating Oregon
- reducing choices to boolean
- time-used note

Evaluating ORegon

The two-letter postal code abbreviation of the U.S. state named Oregon is OR, which is a logical operator.

The simple test of the value fails because the value is interpreted as a logical operator.

```
%if &State eq OR ...
%*is expanded to:;
%if OR eq <missing> OR ...
```

The solution is to quote the strings:

```
%if "&State" eq "OR" %then ...
```

See also: Fehd [5, sco.Beginners-Tour].

Reduction to boolean

Many problems arise in logical evaluation of expressions which contain user-supplied values written in natural language. Examples for boolean values include: yes/no, true/false, positive/negative, etc. Further complications come from allowing case variations: lowcase, Propcase, or UPCASE.

The problem can be shown as:

```
%if    &condition eq y    or &condition eq Y
    or &condition eq yes  or &condition eq Yes
    or &condition eq YES %then ...
```

One solution is to use the macro `in` operator

```
%macro chk_this(condition=0)/minoperator;
%if %eval(&condition in y Y yes Yes YES)
    %then ...
```

The author's solution is to standardize the case, reduce to the first letter and recode.

callmacr.sas

```
1  %if not(   &choice eq 0
2          or &choice eq 1) %then
3      %let choice=%eval(y eq %lowcase(%substr(&choice,1,1)));
4
5  %* compare to the radical solution: any response is True=1;
6  %let choice=%eval(not(0 eq &choice));
```

Continued on next page.

Time-used Note

A macro function does not return steps, so there are no timing notes written in the log. For a macro function containing a `%do` loop the following statements can be added to write an elapsed time-used note.

This trick requires local macro variables for the subtraction at the end of the routine. This code shows the initialization at the top of the definition.

```
%local TimeStart TimeEnd;
%let   TimeStart = %sysfunc(datetime(),hex16.);
```

This is the calculation at the bottom of the definition.

```
%let TimeEnd = %sysfunc(datetime(),hex16.);
%Put note: &SysMacroName used real time %sysfunc
           (putn(&TimeEnd.x-&TimeStart.x,time12.3));
%mend;
```

Notes:

- initialize: save system.datetime as hex16

  `TimeStart := '1234567890abcdef'x`

- termination `TimeEnd := '1234567890abcdef'x`

  difference: `TimeUsed := TimeEnd - TimeStart`

  convert real number to user-readable: hh:mm:ss.sss

  `putn(&TimeUsed,time12.3)`

Summary

Conclusion

The primary purpose of a macros is to hide complexity.

A good macro is the result of several ideas coming together:

- First is management support for quality assurance and a style guide.
- Next is design, which leads to good documentation and programming. This produces a macro ready for peer review and testing.
- Finally, readability of documentation, and program, promotes reuse.

Further Reading

programs : for this paper are in Macro Design Ideas

peer review : Rhodes [17, sgf2013.Programming-Standards]

predecessor : Fehd [12, sgf2008.003] (SmryEachVar) developed a suite of programs to return a list of the frequencies of each variable in a data set or libref. This suite illustrates the three levels of program complexity: module, routines, and subroutines. Programs for SmryEachVar are here: Fehd [8, sco.SmryEachVar].

quality : Crosby [3, Crosby-Quality-and-Me] and Crosby [2, Crosby-Quality-is-Free]

Deming [4, Deming-Out-of-the-Crisis]

Juran [13, Juran-Architect-of-Quality]

! $\rightarrow$ Quality assurance is different from quality control.

style guides : Celko [1, Celko.2005-SQL-programming-style]

This paper was typeset using the TEX program developed by Donald Knuth. Knuth [14, Knuth-Literate-Programming]

Martin [15, Martin.2009-Clean-Code]

McConnell [16, McConnell.2004-Code-Complete-2e]

examples : One of the best ways to learn to design and write macros is to examine other people's work.

list : Jiantang Hu wrote a blog entry with a list of macro collections.

Berry : Roland Rashleigh-Berry has a list of macros that he has developed for clinical reporting.
He has good notes on avoiding name collisions.

Devenezia : Richard A. DeVenezia has a page with his macros.

Fehd : Many of the author's macros are on sas.community.org:

Fehd [10, sco.Macro-Loops-With-Dates] is in this conference proceedings. Fehd [9, sco.Macro-CallMacro] is the example documentation shown here.

This program may be useful: Fehd [6, sco.Indexing-Programs]

Friendly : Michael Friendly has a suite of macros which have internal markup symbols so that they can be processed and an html page extracted from them.

Schick : Arnold Schick of University of Marburg, Germany,
has a collection of macros.

Acknowledgements

Several Alert Readers contributed to a SAS-L thread whose subject contains: Theory: reducing positive/true/yes, etc to boolean, date-start: June 2013. There was disagreement about whether to recode *any* user-supplied value to true=1 for the macro variable testing. Search for: `%eval(not(0 eq &testing)`.

Bibliography

[1] Joe Celko. *Joe Celko's SQL Programming Style*. Elsevier, San Francisco, CA, USA, 2005. URL http://store.elsevier.com/Joe-Celkos-SQL-Programming-Style/Joe-Celko/isbn-9780120887972/. 217 pp., 10 chap., index: 8 pp.; theory of readability: naming conventions and visual layout.

[2] Philip B. Crosby. *Quality is Free*. Mentor Books, Denver, CO, USA, 1980. URL http://www.philipcrosby.com/25years/read.html. 270 pp., 13 chap., index: 6 pp.

[3] Philip B. Crosby. *Quality and Me*. Jossey-Bass, San Francisco, CA, USA, 1999. URL http://www.josseybass.com/WileyCDA/WileyTitle/productCd-0787947024.html. autobiography; 251 pp., 14 chap., index: 13 pp.

[4] W. Edwards Deming. *Out of the Crisis*. MIT Press, Cambridge, MA, USA, 2000. URL http://mitpress.mit.edu/books/out-crisis. 507 pp., 17 chap., index: 15 pp.

[5] Editor R.J. Fehd. Beginners tour of project using macros. In *sasCommunity.org*, 2001. URL http://www.sascommunity.org/wiki/Beginners_Tour_of_Project_Using_Macros. topics: definition and programs.

[6] Editor: R.J. Fehd. Indexing programs. In *sasCommunity.org*, 2006. URL http://www.sascommunity.org/wiki/Indexing_Programs. program to prepare an index of macro variables used in programs.

[7] Editor R.J. Fehd. Conditionally executing global statements. In *sasCommunity.org*, 2008. URL http://www.sascommunity.org/wiki/Conditionally_Executing_Global_Statements. topics: combining functions sysfunc and ifc; info: example assertions, caveats on logical comparisons.

[8] Editor R.J. Fehd. SmryEachVar: A data-review suite for each variable in all data sets in a libref. In *sasCommunity.org*, 2008. URL http://www.sascommunity.org/wiki/SmryEachVar_A_Data_Review_Suite. list processing using parameterized includes.

[9] Editor R.J. Fehd. Macro Call-Macro. In *sasCommunity.org*, 2012. URL http://www.sascommunity.org/wiki/Macro_CallMacr. using SCL functions to read a data set and call macros.

[10] Editor R.J. Fehd. Macro loops with dates. In *sasCommunity.org*, 2013. URL http://www.sascommunity.org/wiki/Macro_Loops_with_Dates. example macros, programs and updates.

[11] Ronald J. Fehd. Writing testing-aware programs that self-report when testing options are true. In *Proceedings of the NorthEast SAS User Group Conference*, 2007. URL http://www.nesug.org/Proceedings/nesug07/cc/cc12.pdf. Coders' Corner, 20 pp.; topics: options used while testing: echoauto, mprint, source2, verbose; variable testing in data step or macros; call execute; references.

[12] Ronald J. Fehd. SmryEachVar: A data-review routine for all data sets in a libref. In *Proceedings of the SAS® Global Forum Conference*, 2008. URL http://www2.sas.com/proceedings/forum2008/003-2008.pdf. Applications Development, 24 pp.; call execute, data review, data structure, dynamic programming, list processing, parameterized includes, utilities (writattr, writvalu) to repair missing elements in data structure; best contributed paper in ApDev.

[13] Joseph M. Juran. *Architect of Quality*. McGraw-Hill, New York, NY, USA, 2004. URL http://www.baldrige.com/criteria_informationmanagement/the-architect-of-quality-statistical-quality-control/. autobiography, birth of statistical quality control; 392 pp., 30 chap., index: 13 pp.

[14] Donald E. Knuth. *Literate Programming*. CSLI, Stanford, CA, USA, 1992. URL http://en.wikipedia.org/wiki/Literate_programming. 368 pp., 12 chap., index: 10 pp.

[15] Robert C. Martin. *Clean Code*. Pearson Education, Boston, MA, USA, 2009. URL http://www.pearsonhighered.com/educator/product/Clean-Code-A-Handbook-of-Agile-Software-Craftsmanship/9780132350884.page. 431 pp., 17 chap., index: 19 pp.; subtitle: handbook of Agile software craftsmanship.

[16] Steve McConnell. *Code Complete: A Practical Handbook of Software Construction, 2e*. Microsoft Press, Redmond, WA, USA, 2004. URL http://cc2e.com/. 960 pp., 35 chap., index: 30 pp.

[17] Dianne Louise Rhodes. If you have programming standards, please raise your hand: An everymans guide. 2013. URL http://support.sas.com/resources/papers/proceedings13/189-2013.pdf. 8 pp.; topics: programming standards, style sheets, and check lists for peer review and testing.

Closure

Contact Information: **Ronald J. Fehd** mailto:Ron.Fehd.macro.maven@gmail.com
http://www.sascommunity.org/wiki/Ronald_J._Fehd

About the author:

education:	B.S. Computer Science, U/Hawaii,	1986
	SAS User Group conference attendee since	1989
	SAS-L reader since	1994
experience:	programmer: 25+ years	
	data manager using SAS:	17+ years
	statistical software help desk:	7+ years
	author: 30+ SUG papers	
	sasCommunity.org: 300+ pages	
SAS-L:	author: 6,000+ messages to SAS-L since	1997
	Most Valuable SAS-L contributor:	2001, 2003

Trademarks

Close Quote

Technical skill is mastery of complexity
while creativity is mastery of simplicity.

— E. C. Zeeman, British mathematician