PharmaSUG 2016 – Paper AD11

Best Practice Programming Techniques for SAS® Users

Kirk Paul Lafler, Software Intelligence Corporation, Spring Valley, California
Mary Rosenbloom, Lake Forest, California

Abstract

It's essential that SAS® users possess the necessary skills to implement "best practice" programming techniques when using the Base-SAS software. This presentation illustrates core concepts with examples to ensure that code is readable, clearly written, understandable, structured, portable, and maintainable. Attendees learn how to apply good programming techniques including implementing naming conventions for datasets, variables, programs and libraries; code appearance and structure using modular design, logic scenarios, controlled loops, subroutines and embedded control flow; code compatibility and portability across applications and operating platforms; developing readable code and program documentation; applying statements, options and definitions to achieve the greatest advantage in the program environment; and implementing program generality into code to enable its continued operation with little or no modifications.

Introduction

Code is an intellectual property and should be treated as a tangible asset by all organizations. Best practice programming techniques help to clarify the sequence of instructions in code, permit others to read code as well as understand it, assist in the maintainability of code, permit greater opportunity to reuse code, achieve measurable results, reduce costs in developing and supporting code, and assist in performance improvements (e.g., CPU, I/O, Elapsed time, DASD, Memory).

Best Practice Concepts

A best practice programming technique is a particular approach or method that has achieved some level of approval or acceptance by a professional association, authoritative entity, and/or by published research results. Successful best practice programming techniques translate into greater code readability, maintainability and longevity while ensuring code reusability. Best practice programming techniques achieve status by being:

- ✓ Measurable with quantifiable results
- ✓ Successful in accomplishing stated goals and objectives
- ✓ Reproducible or adaptable to specific needs

Naming Conventions

The adherence of naming conventions serves the purpose of making program code, SAS libraries, catalogs, catalog entries, datasets (or tables), variables, virtual tables (or views), macro definitions, macro variables, arrays, indexes, hash objects, subroutines, labels, user-defined formats, SAS/ACCESS access descriptors and view descriptors, external files, and SAS name literals more readable. Good naming conventions can also satisfy self-documentation for individual application components by permitting others to better understand the application and code details.

"Good" naming conventions are a must and should always be adhered to whenever possible. To inforce the use of "good" naming conventions in the SAS environment, specify:

```
OPTIONS DATASTMTCHK = COREKEYWORDS ;
```

Naming Conventions for SAS Data Sets and Variables

The rules associated with good naming conventions for SAS data sets and variables include:

- A name consisting of 1-32 positions in length.
- 1st position can only be a letter (A-Z, a-z) or underscore (_).
- Remaining positions can be letters (A-Z, a-z), underscore (_), or numbers (0-9).
- Example:

```
DATA WORK.PG_Rated_Movies ;
  SET libref.Movies ;
  WHERE Rating = "PG" ;
RUN ;
```

In the next example, below, the use of not-so-good naming conventions is illustrated. In this example, you'll notice valid, but confusing naming conventions, rendering the code difficult to read and maintain.

Code Containing Poor Use of Naming Conventions

```
OPTIONS DATASTMTCHK = NONE ;
%LET WHERE = SAN DIEGO ;
DATA WORK.DATA ;
  SET WORK.SET ;
  WHERE UPCASE(VARIABLE4) = "&WHERE" ;
RUN ;
```

In the next example, below, valid and less confusing naming conventions are utilized resulting in program code that is easier to read, maintain and support.

Code Containing Good Use of Naming Conventions

```
OPTIONS DATASTMTCHK = COREKEYWORDS ;
%LET WHERE = SAN DIEGO ;
DATA WORK.SanDiego_Employees ;
  SET WORK.Employees ;
  WHERE UPCASE(City) = "&WHERE" ;
RUN ;
```

Naming Conventions for Librefs, Filerefs and Subroutines

The rules associated with good naming conventions for librefs, filerefs and subroutines include:

1. Assigned name can be 1-8 positions in length.
2. 1st position can be a letter (A-Z, a-z) or underscore (_) only.
3. Remaining positions can be letters (A-Z, a-z), underscore (_), or numbers (0-9).
4. Example:

```
LIBNAME  MYDATA  'C:\DataLibrary' ;
DATA  MYDATA.PG_Rated_Movies ;
  SET  MYDATA.Movies ;
  WHERE  Rating = "PG" ;
RUN ;
```

Naming Conventions for Macro Names and Macro Definitions

The rules associated with good naming conventions for macro names and macro definitions include:

1. Name that is assigned can be 1-32 positions in length.
2. 1st position can be a letter (A-Z, a-z) or underscore (_) only.
3. Remaining positions can be letters (A-Z, a-z), underscore (_), or numbers (0-9).
4. Example:

```
PROC SQL ;
  SELECT TITLE INTO :mtitle SEPARATED BY "*  "
    FROM MYDATA.Movies ;
      WHERE UPCASE(Rating) = "PG" ;
QUIT ;
%PUT &mtitle ;
```

```
Casablanca*  Jaws*  Poltergeist*  Rocky*  Star Wars*  The Hunt for Red October
```

Comments and Documentation

Comments and documentation serve to assist in the understanding of what's in data sets and variables, provide descriptive information about the intricacies of a program, minimize confusion about what a program is doing, save time during support or maintenance of a program and document program run instructions. They also provide important information about the purpose of the program; how to run the program; what the input, processing and output is; and special operations that need additional understanding.

Assignment of Data Set Labels

Data set Labels are typically specified when saving permanent data sets and are displayed with PROC CONTENTS or PROC DATASETS output.

```
DATA libref.PG_RATED_MOVIES
             (LABEL="PG-Rated Movies") ;
  SET libref.MOVIES ;
  WHERE RATING = "PG" ;
RUN ;

PROC CONTENTS DATA=WORK._ALL_ ;
RUN ;
```

Data Set Name	WORK.PG_RATED_MOVIES	Observations	6
Member Type	DATA	Variables	6
Engine	V9	Indexes	0
Created	Friday, October 03, 2014 02:54:23 AM	Observation Length	88
Last Modified	Friday, October 03, 2014 02:54:23 AM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label	PG-Rated Movies		
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

Assignment of Variable Labels

Variable Labels are typically specified with permanent data sets and can be displayed as column headers using PROC PRINT.

```
DATA libref.PG_RATED_MOVIES
              (LABEL="PG-Rated Movies") ;
  SET libref.MOVIES ;
  WHERE RATING = "PG" ;
  NEW_LENGTH = LENGTH + 1 ;
  LABEL NEW_LENGTH = "Length with Trailer" ;
RUN ;

PROC PRINT DATA=libref.PG_RATED_MOVIES  LABEL ;
  TITLE ;
RUN ;
```

Obs	Title	Length	Category	Year	Studio	Rating	Length with Trailer
1	Casablanca	103	Drama	1942	MGM / UA	PG	104
2	Jaws	125	Action Adventure	1975	Universal Studios	PG	126
3	Poltergeist	115	Horror	1982	MGM / UA	PG	116
4	Rocky	120	Action Adventure	1976	MGM / UA	PG	121
5	Star Wars	124	Action Sci-Fi	1977	Lucas Film Ltd	PG	125
6	The Hunt for Red October	135	Action Adventure	1989	Paramount Pictures	PG	136

Assignment of Comments

Comments can be added to program code in the following ways.

*** (Asterisk) (aka Flower-box)**

```
*****************************************;
*****  This program selects PG movies. *****;
*****************************************;
```

COMMENT statement

```
COMMENT This program selects PG movies ;
```

In-stream comments can be added on any line of code

```
DATA PG_RATED_MOVIES ;  /* Create temp dataset */
SET libref.MOVIES ;     /* Read MOVIES dataset */
WHERE RATING = "PG" ;   /*Select PG Movies */
RUN ;
```

Assignment of Program Headers

Program Headers should be created at the beginning of each program. It serves to provide details about the name of the program, the program purpose, the names of the called and calling programs, who wrote the program, the date it was written, the input and output sources, and the modification history. An example of a program header is illustrated below.

```
*************************************************************************************;
***** Program Name: %SYSFUNC_Welcome_Screen.SAS                                  *****;
***** Purpose.....: Display the Welcome screen for the Building Reusable Tool using  *****;
*****               the SAS Macro Language and %SYSFUNC for accessing functions.     *****;
*****                                                                                *****;
***** Author......: Kirk Paul Lafler, Software Intelligence Corporation             *****;
***** Date Written: 06/13/2010                                                      *****;
***** SAS Version.: SAS 9.2, 9.3, 9.4                                               *****;
***** Input Files.: Workshop Data                                                   *****;
***** Output Files: None                                                            *****;
***** Subroutines.: None                                                            *****;
***** Macro Variables: &sysver, &sysday                                             *****;
***** Includes....: None                                                            *****;
***** Modification History:                                                         *****;
*****  01/20/2015  KPL  Added ODS HTML CLOSE to reflect changes in SAS software.     *****;
*************************************************************************************;

options ls=100 source source2;
ods html close;
ods listing;
libname mydata 'e:\workshops\workshop data';

%window Window_Welcome color=white
   #5 @25 'Welcome to Building Reusable Tools using the SAS Macro Language' attr=highlight color=blue
   #9 @27 "You are executing SAS Release &sysver on &sysday, %sysfunc(date(),worddate18.)"
  #14 @40 "Press the enter key to continue.";
  %display Window_Welcome;
```

Assignment of Section Headers

Section Headers should be created at the beginning of major sections of a program, routines, subroutines and/or complicated sections of code. Generally shorter and less detailed than the Program Header, the section header serves to provide some level of understanding about the purpose of specific program code, along with what the code is doing. An example of a section header appears below.

```
libname mydata 'e:\workshops\workshop data';

/*****************************************************/
/** ROUTINE.....: FIRST-BY-GROUP-ROWS               **/
/** PURPOSE.....: Derive the first (min) row within **/
/**               each by-group using a subquery.   **/
/*****************************************************/
proc sql ;
  create table first_bygroup_rows as
    select rating,
           title,
           'FirstRow' as ByGroup
      from mydata.movies M1
        where title =
              (select min(title)
                 from mydata.movies M2
                   where M1.rating = M2.rating)
          order by rating, title ;

  select * from first_bygroup_rows ;

quit ;
```

Configuration Management

Wikipedia defines configuration management as establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design, and operational information throughout its life. The configuration process includes initialize “key” variables (columns); support structured and/or modular design; construct and use shareable code libraries; enable SAS to systematically handle changes so a program can maintain high integrity over the life of the program; utilize pre-written source code as “callable” routines so that it can be included, as needed, by a calling program.

Determining SAS System Options

SAS Option settings can be examined by the following methods.

Using the PROC OPTIONS statement

```
PROC OPTIONS ;
RUN ;
```

Accessing the contents of DICTIONARY.OPTIONS using SQL

```
PROC SQL ;
SELECT  *  FROM DICTIONARY.OPTIONS ;
QUIT ;
```

Accessing the virtual table SASHELP.VOPTION

```
PROC PRINT DATA=SASHELP.VOPTION ;
RUN ;
```

Using the SOURCE / NOSOURCE System Option

The SOURCE system option tells SAS to write all in-stream SAS source code to the SAS Log. The result is a complete audit trail of executed SAS code.

SOURCE System Option

```
options SOURCE ;
data libref.processed_movies ;
  set libref.movies ;
  <  other SAS statements  > ;
run ;
```

SOURCE2 System Option

The SOURCE2 system option ells SAS to write all included source code to the SAS Log to produce a comprehensive audit trail. This is sometimes referred to as a "Copybook" process.

SOURCE2 System Option

```
options SOURCE2 ;
data libref.processed_movies ;
  set libref.movies ;
  %include 'c:\include-sas-code.sas' ;
  <  other SAS statements  > ;
run ;
```

Coding Conventions

The adherence of coding conventions often produces SAS code that is easier to understand; results in more maintainable code; produces higher quality and more efficient code; results in code with greater integrity over its productive life; enables code to be reused from project to project, department to department, within an organization; results in code with fewer defects providing greater effectiveness and value; and reduces the time spent on debugging.

Read Only Fields (Variables) and Data Needed

To reduce the size of the input buffer and SAS data set, read only the fields that are needed from the external file. Any field(s) not listed on the INPUT statement is/are automatically eliminated from being read. An example appears below.

```
filename rawdata
         'e:\workshops\workshop data\movies.dat' ;

data PG_MOVIES ;
  infile rawdata  missover ;
  input  @1 title $30.
         @32 length 3.
         @36 category $20.
         @88 rating $5. ;
  if rating = "PG" or
     rating = "PG-13" ;
run ;
```

Subsetting Data Sets with WHERE=

Specifying a WHERE= data set option on a SET statement or a DATA= parameter is a good way to efficiently subset a SAS data set because it only reads observations into the Program Data Vector (PDV) that satisfy the WHERE-clause expression. An example appears below.

```
data PG_MOVIES ;
  set libref.movies (where=(rating = 'PG')) ;
  < other SAS statements > ;
run ;

           < OR >

proc print data=libref.movies (where=(rating = 'PG'))  noobs ;
run ;
```

Types of IF Statements

- Subsetting IF
- Simple IF-THEN
- Chains of IF-THENs
- Chains of IF-THEN/ELSEs

Subsetting IF Statement

A subsetting IF statement is best used for subsetting in-stream data from non-SAS input files. An example appears below.

```
data PG_Movies ;
  infile carsdata 'e:\workshops\workshop data' ;
  if rating = 'PG' ;
run;
```

Simple IF-THEN Statement

A simple IF-THEN statement can be combined with other SAS statements and operations (e.g., DELETE, DO, OUTPUT, assigning values to variables, etc.). An example appears below.

```
data PG_Movies ;
  set libref.movies ;
  if rating = 'PG'  then output PG_Movies ;
run ;
```

Chains of IF-THEN Statements

Although coding chains of IF-THEN statements is syntactically correct, it forces the SAS System to incur additional processing costs because each IF-THEN statement is executed regardless if the condition was satisfied in an earlier IF-THEN statement. An example appears below.

```
data G_Movies PG_Movies PG13_Movies R_Movies ;
  set libref.movies ;
  if rating = 'G' then output  G_Movies ;
   if rating = 'PG' then output  PG_Movies ;
    if rating = 'PG-13' then output  PG13_Movies ;
     if rating = 'R' then output  R_Movies ;
run ;
```

Chains of IF-THEN/ELSE Statements

A more efficient technique of coding chains of IF-THENs is to insert an ELSE between each IF-THEN, because once a condition is satisfied it branches out of the IF-THEN/ELSE logic construct. An example appears below.

```
data G_Movies PG_Movies PG13_Movies R_Movies ;
  set libref.movies ;
  if rating = 'G' then output  G_Movies ;
  else
   if rating = 'PG' then output  PG_Movies ;
   else
    if rating = 'PG-13' then output  PG13_Movies ;
    else
     if rating = 'R' then output  R_Movies ;
run ;
```

Assigning Many Values

SAS often provides users with many approaches to accomplish the same result. This is also the case with the assignment of values. In the two examples illustrated below, an okay approach is illustrated followed by a better approach.

Okay Approach to Assigning Many Values

Although using a series of IF-THEN/ELSE statements to assign many values is acceptable, it can be harder to maintain and increases CPU costs.

```
data Movies_with_IF_THEN_ELSE ;
  set  libref.Movies ;
  if rating = 'R' then age_grp = '>=17' ;
  else
```

```
   if rating = 'PG-13' then age_grp = '>=13' ;
   else
    if rating = 'PG' then age_grp = '>=13' ;
    else
     if rating = 'G' then age_grp = 'All Ages' ;
run ;
```

Better Approach to Assigning Many Values

A more maintainable and efficient method to assign many values is to use PROC FORMAT with the PUT function.

```
proc format ;
  value $age_grp  "G" = "All Ages"
                 "PG" = ">=13"
              "PG-13" = ">=13"
                 "R"  = ">=17" ;
run ;
data Movies_with_Age_grp ;
  set libref.Movies ;
  age_grp = put (rating, $age_grp.) ;
run ;
```

Processing Data in a Repetitive Way

Data can be processed in a repetitive way in a variety of methods. The two methods that will be illustrated here are DATA step approaches including without arrays and with arrays.

Processing Repetitive Data without an Array

Without arrays, some tasks require a series of IF-THEN/ELSE statements to be written, making the code harder to maintain. An example appears below.

```
data IF_THEN_Example ;
  set transposed_movies ;
  if _NAME_ = 'Length' and col1 < 100 then col1 = . ;
  else
   if _NAME_ = 'Length' and col2 < 100 then col2 = . ;
   else
     . . .       . . .       . . .       . . .
    else
     if _NAME_ = 'Length' and col22 < 100 then col22 = . ;
run ;
```

Processing Repetitive Data with an Array

Arrays can be used to reduce the amount of code that is needed to perform a repetitive task. With arrays, actions on variables can be repeated any number of times and the code is typically easier to maintain. An example appears below.

```
data Array_Example;
  set transposed_movies;
  array a1 (22) col1-col22;
  do i = 1 to 22;
     if _NAME_='Length' and a1(i) < 100 then a1(i) = .;
  end;
run;
```

BY-group Processing with CLASS

Specifying a CLASS statement with PROC MEANS is an efficient way to process BY-groups without the need to first perform costly sort operations. An example appears below.

```
proc means data=libref.movies ;
  class category ;
run ;
```

Specifying an OUT= Parameter with PROC SORT

To prevent writing over the original SAS dataset with PROC SORT, always specify an OUT= parameter with a different libref and/or a different dataset name. An example appears below.

```
proc sort data=libref.Movies
          out=WORK.sorted_movies ;
  by rating ;
run ;
```

Conclusion

Code is an intellectual property and should be treated as a tangible asset by the organization. This paper offers numerous best practice programming techniques to clarify the sequence of instructions in code, permit others to read code as well as understand it, assist in the maintainability of code, permit greater opportunity to reuse code, achieve measurable results, reduce costs in developing and supporting code, and assist in performance improvements (e.g., CPU, I/O, Elapsed time, DASD, Memory).

References

Lafler, Kirk Paul (2015), *"Top Ten SAS® Performance Techniques,"* Western Users of SAS Software (WUSS) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2014), *"Top Ten SAS® Performance Techniques,"* South East SAS Users Group (SESUG) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2013), *"Top Ten SAS® Performance Techniques,"* Nebraska SAS Users Group (NebSUG) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2013), *"Top Ten SAS® Performance Techniques,"* Iowa SAS Users Group (IASUG) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2012), *"Top Ten SAS® Performance Techniques,"* Kansas City SAS Users Group (KCSUG) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2012), *"Top Ten SAS® Performance Techniques,"* MidWest SAS Users Group (MWSUG) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2012), *"Top Ten SAS® Performance Techniques,"* North East SAS Users Group (NESUG) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2012), *"Top Ten SAS® Performance Techniques,"* Iowa SAS Users Group (IASUG) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2012), *"Top Ten SAS® Performance Techniques,"* SAS Global Forum (SGF) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Acknowledgments

The authors thank Peter Eberhardt and Gary Moore, Applications Development Section Chairs for accepting our abstract and paper; Eric Larson, PharmaSUG 2016 Academic Chair; Sandra Minjoe, PharmaSUG 2016 Operations Chair; SAS Institute Inc.; and the PharmaSUG Executive Committee for organizing a great conference!

Trademark Citations

About the Author

Kirk Paul Lafler has used SAS since 1979, and is consultant and founder of Software Intelligence Corporation. He is a SAS Certified Professional, provider of IT consulting services, trainer to SAS users around the world, UCSD Extension professor, mentor, and sasCommunity.org emeritus Advisory Board member. As the author of six books including Google® Search Complete! (Odyssey Press. 2014) and PROC SQL: Beyond the Basics Using SAS, Second Edition (SAS Press. 2013); Kirk has written more than five hundred papers and articles; been an Invited speaker and trainer at five hundred-plus SAS International, regional, special-interest, local, and in-house user group conferences and meetings; and is the recipient of 23 "Best" contributed paper, hands-on workshop (HOW), and poster awards.

Mary Rosenbloom is a statistical programmer in Lake Forest, California. She has been using SAS for over 20 years, and is especially interested in using macros to generate data-driven code, documenting best practice techniques, DDE, and program validation methods.

Comments and suggestions can be sent to:

Kirk Paul Lafler
Senior SAS® Consultant, Application Developer, Data Scientist, Educator and Author
Software Intelligence Corporation
E-mail: KirkLafler@cs.com
LinkedIn: http://www.linkedin.com/in/KirkPaulLafler
Twitter: @sasNerd