Paper BB-11-2015

Do we need Macros?
An Essay on the Theory of Application Development

Ronald J. Fehd, Theoretical Programmer, Stakana Analytics
SAS-L's Macro Maven

Abstract

Description : This paper examines the theoretical steps of applications development (ApDev) of routines and subroutines. It compares and contrasts the benefits of using the %include statement versus macros. It examines the methods of calling subroutines, e.g., sql, call execute and macro loops.

Purpose : The purpose of this paper is to highlight the benefits of using macros to support unit and integration testing, and searching for and finding issues during maintenance.

Audience : managers and project designers, programmers of all levels

Keywords : compile, execute, step boundaries, macro definitions, macro variables, global symbol table, reuse of compiled statements

In this paper

Introduction

Overview

This section has two topics.

- learning
- definitions
- layers of a program

learning

While learning a new computer proramming language we need to pay attention to these categories of ideas.

- control
- functions
- loops
- variables
- syntax

definitions

This is a list of words used in this article.

function : return a value;
in SAS® software a value is a token, and is less than a statement

module : calls routines and subroutines
to process input and produce output;
often called 'main'

program : a set of statements
subprogram: a subset of a program

routine : a program or subprogram;
performs one or more tasks, calls subroutines

subroutine : performs a single task, called by modules or routines

How SAS works

Overview

This section provides both a practical and theoretical explanation of how SAS and the macro language work together.

- programs have subprograms, steps
- program has layers
- loading Global Symbol Table (GST)
- startup: configuration and autoexec

programs have subprograms, steps

A program consists of pieces, some theoretical, some practical.

programs contain subprograms		HIPO: hierarchical input process output
SAS programs contain steps		data, proc, run
steps have two aspects	compile	data structure
	execute	algorithm produces result

program has layers

A program may be read from top to bottom; but it is also important to visualize the various layers of program assets as they occur.

• Global Symbol Table			
• %include statement			subprogram
• macro language	compile	variables:	`%let mvar=...;`
		definitions:	`%macro ... %mend;`
	execution	variable references:	`&mvar`
		macro calls:	`%do_this(data=...)`
• SAS			

loading Global Symbol Table

In SAS documentation the symbol table is always referred to as either the global or local macro variable symbol table.

In this article the Global Symbol Table (GST) refers to this list of sets of GST variable names available to programs.

- environment variables
- location names: filerefs and librefs used in options
- macro
  - variables: system- or user-defined
  - definitions: location of compiled code
- options location names for reuse
- running text: titles, footnotes

Note: The verb 'load' is used because any name in a set can only be assigned or its value retrieved and the last assignment is the value available.

startup: configuration and autoexec

Loading of the Global Symbol Table occurs in two files: one or more configuration files, and optionally an autoexec file. Default names of these files are `sasv9.cfg` and `autoexec.sas`.

config : configuration files and command-line

- environment variables: macro autocall folders
- startup-only options: one third of options

autoexec : location names, options for %includes, macros

- autocall

```
filename project '.';
filename site_mac '...';
options mautosource
        sasautos = (project site_mac sasautos);
```

- compiled and stored

```
libname libmacro '..\sas7bcat';
options mstored sasmstore=libmacro;
```

Theory and Decisions of Macro Usage in Applications Development

Overview

This section discusses the main reasons to use macros in applications.

- optimization
- strategy
- tactics

optimization

SAS software and its macro language provide extra facilities to improve programs.

- autocall: automatic search for reusable macros
- compiled and stored macro definitions in catalog
- testing: unit and integration
  options for debugging
  remote control during testing

strategy

For the big picture, either macros or %includes can provide answers to these choices.

- large — table-top rule: 10 pages,
  50 lines/page = 500 lines
- reuse: used often, compiled once
- guarantee
- hide complexity
- centralization, standardization

tactics

These are the primary reasons to convert programs to macros.

conditional execution:	`%if`	additional code or branching
loops:	`%do`	
functions:	`%sysevalf`	evaluation of real numbers
	`%sysfunc`	access to data step functions

In many cases macro definitions are a simple way to encapsulate loops and function calls that require elaborate data step code.

How to Develop Routines and Subroutines

Overview

This section provides a quick overview of how to build a simple subroutine from working programs through parameterized %includes to a macro. It shows how to test each type of program.

- hard-code
- soft-code
- split in two
- make macro
- making lists
- call execute of %includes
- calling macros
- sql constant text

hard-code

Find two programs that use similar statements.

```
proc freq data = sashelp.class;
          tables age;
```

```
proc freq data = sashelp.shoes;
          tables region;
```

soft-code

Identify the parameters; use SAS keywords as parameter names.

```
%let data = sashelp.shoes;
%let var  = region;
proc freq data = &data;
          tables &var;
```

split in two

Split the soft-coded program into two part: the caller and the subroutine.

sub-program-1-test.sas

```
%let data = sashelp.shoes;
%let var  = region;
%include ’sub-program-1.sas’/source2;
```

sub-program-1.sas

```
%put trace: sub-program-1 beginning;
*leave mvars as reminder of parameters;
*let data = sashelp.shoes;
*let var  = region;
%put echo: &=data &=var;
proc freq data = &data;
          tables &var / noprint
          out    = out_freq;
run;
%put trace: sub-program-1 ending;
```

make macro

Convert the subroutine to a macro and copy the caller program and change from %include to macro call.

sub-program-2-test.sas

```
options mprint source2;
%sub_program_2(data = sashelp.class
              ,var  = sex)
```

sub_program_2.sas

```
%MACRO sub_program_2
      (data     = sashelp.shoes
      ,var      = region
      ,out_data = out_freq
      ,testing  = 0);
%let testing = %eval(not(0 eq &testing)
     or %sysfunc(getoption(source2)) eq SOURCE2);
%put trace: &sysmacroname begining;
PROC freq data = &data;
          tables &var / noprint
          out    = &out_data;
%if &testing %then %do;
    proc sql; describe table &syslast;
              quit;
    %end;
run;
%put trace: &sysmacroname ending;
%mend sub_program_2;
```

making lists

Repetition can be managed, not by manual typing of parameters and calling program names, but by using SAS software to create a control data set, a list, where the values in columns in each row are a set of parameters for a subroutine. The `contents` and `sql` procedures can be used to create lists. This program uses the contents procedure to make a list of variable names.

make-list-vars-contents.sas

```
PROC contents data = &in_data   noprint
              out  = list_variables
             (keep = name type);
run;
```

call execute of %includes

The `call execute` routine can be used to read a list and call parameterized %includes.

proc-freq.sas

```
PROC freq data   = &in_data;
          tables   &name /list;
```

demo-cx-include.sas

```
%let in_data = sashelp.class;
options source2;
%include project(make-list-vars-contents);
%let cx_data = list_variables(keep = name);
%let cx_include = ’proc-freq.sas’/source2;
%include site_inc(cx-inclu)/nosource2;
```

calling macros

The `call-macro` routine can be used to read a list and call macros.

procfreq.sas

```
%macro procfreq(data =
               ,name =
               ,type =);
PROC freq data   = &data;
          tables   &name /list;
run;
%mend;
```

demo-call-macro.sas

```
%let in_data = sashelp.class;
options mprint source2;
%include project(make-list-vars-contents);
%callmacr(data        = list_variables
         ,macro_parms = %nrstr(data=&in_data)
         ,macro_name  = procfreq)
```

sql constant text

The `sql` procedure can be used to read a list and call either parameterized %includes or macros.

See Fehd [6] for example programs.

Summary

Conclusion

The question was ”Do we need macros?”.

The Reframe: Do we need %includes or macros to reuse programs?

- autoexec needed for either, for location names of folders
- use %includes with macro variables as parameters until you need:
  - additional code within subprogram
  - macro functions or loops
- macro language
  - variables — passing values across step boundaries
  - autocall — need *filerefs* for options
  - compiled and stored — need *librefs* for options

Suggested Reading

macro basics : Fehd [10], Autoexec Companion; Carpenter [1], ways to create macro variables; First and Ronk [12], programming with macro variables

testing, tracing : Fehd [3], Writing Testing-Aware Programs;
Fehd [4], Using Sysfunc and Ifc;
Fehd [2], using global macro variables to trace calls

list processing : Fehd and Carpenter [11], List Processing Basics;
Fehd [6], Using Sql for List Processing;
Fehd [5], List Processing Routine Call-Execute-an-Include;
Fehd [9], Macro Call-Macro: using a control data set to call macros

%do loops : Fehd [8], Macro Loops with Dates

opinion : Henderson [13], Macro Programming Best Practices;
Fehd [7], Macro Design Ideas

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
experience:	programmer: 30+ years	
	author: 40+ SUG papers	
	sas.community.org wiki: 400+ pages	
SAS-L:	author: 7,000+ messages to SAS-L since	1997

Trademarks

Bibliography

[1] Arthur L. Carpenter. Five ways to create macro variables: A short introduction to the macro language. In *SESUG*, 2005. URL `http://analytics.ncsu.edu/sesug/2005/HW03_05.PDF`. 12 pp.; call symput, %do, %global, %let, %local, ods, parameters in a macro definition, sql select into, sysparm, %window.

[2] Ronald Fehd. Journeymen's tools: Two macros — ProgList and PutMvars — to show calling sequence and parameters of routines. In *Proceedings of the 30th Annual SAS® Users Group International Conference*, 2005. URL `http://www2.sas.com/proceedings/sugi30/004-30.pdf`. Applications Development, 8 pp.; debugging, parameterized %includes, testing, tracing.

[3] Ronald J. Fehd. Writing testing-aware programs that self-report when testing options are true. In *NorthEast SAS Users Group Annual Conference Proceedings*, 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.

[4] Ronald J. Fehd. Using functions Sysfunc and Ifc to conditionally execute statements in open code. In *SAS Global Forum Annual Conference Proceedings*, 2009. URL `http://support.sas.com/resources/papers/proceedings09/054-2009.pdf`. Coders Corner, 10 pp.; topics: combining functions ifc, nrstr, sysfunc; assertions for testing: existence of catalog, data, file, or fileref; references.

[5] Ronald J. Fehd. List processing routine CallXinc: Calling parameterized include programs using a data set as list of parameters. In *Proceedings of the Western Users of SAS Software Annual Conference*, 2009. URL `www.lexjansen.com/wuss/2009/app/APP-Fehd2.pdf`. Applications Development, 20 pp.; call execute, data review, data structure, dynamic programming, list processing, parameterized includes, examples.

[6] Ronald J. Fehd. How to use proc SQL select into for list processing. In *South Central SAS Users Group Annual Conference Proceedings*, 2010. URL `http://analytics.ncsu.edu/sesug/2010/HOW06.Fehd.pdf`. Hands On Workshop, 40 pp.; topics: writing constant text, and macro calls, using macro %do loops; references.

[7] Ronald J. Fehd. Macro design ideas: Theory, template, practice. In *MidWest SAS Users Group Annual Conference Proceedings*, 2013. URL `http://www.mwsug.org/proceedings/2013/00/MWSUG-2013-0002.pdf`. 21 pp.; topics: logic, quality assurance, testing, style guide, documentation, bibliography.

[8] Ronald J. Fehd. Writing macro do loops with dates from then to when. In *MidWest SAS Users Group Annual Conference Proceedings*, 2013. URL `http://www.mwsug.org/proceedings/2013/00/MWSUG-2013-S115.pdf`. 20 pp.; topics: dates are integers, formats and functions to convert date references to integers, calculations, do and %do statements; interval incrementing (intnx): intervals and shift-index; month, putn, %sysevalf, %sysfunc, today, day-of-the-week (weekday), year; macro dateloop, bibliography.

[9] Ronald J. Fehd. List processing macro call-macro. In *Proceedings of the Western Users of SAS Software Annual Conference*, 2014. URL `http://www.lexjansen.com/wuss/2014/cc/97.pdf`. Coders Corner, 19 pp.; using %sysfunc with SCL functions to read a list, a control data set, and for each observation, call a macro with variable names and values as named parameters.

[10] Ronald J. Fehd. An autoexec companion, allocating location names during startup. In *MidWest SAS Users Group Annual Conference Proceedings*, 2015. Beyond Basics, 15 pp.

[11] Ronald J. Fehd and Art Carpenter. List processing basics: Creating and using lists of macro variables. In *SAS Global Forum Annual Conference Proceedings*, 2007. URL `http://www2.sas.com/proceedings/forum2007/113-2007.pdf`. Hands On Workshop, 20 pp.; comparison of methods: making and iterating macro arrays, scanning macro variable, writing calls to macro variable, write to file then include, call execute; using macro function nrstr in call execute argument; 11 examples, bibliography.

[12] Steven First and Katie Ronk. SAS(R) macro variables and simple macro programs. In *SUGI-30*, 2005. URL `http://www2.sas.com/proceedings/sugi30/130-30.pdf`. 15 pp.; overview of how macro processor works, use of macro options during debugging and testing, use of conditionals (%if) and loops (%do), example macro application, using macro variables to pass information to later steps.

[13] Editor D. Henderson. Macro programming best practices: Styles, guidelines and conventions. In *sas community.org Wikipedia*, 2012. URL `http://www.sascommunity.org/wiki/Macro_Programming_Best_Practices:_Styles,_Guidelines_and_Conventions_Including_the_Rationale_Behind_Them`. peer discussion.

> Do only what is necessary to convey what is essential. Carefully eliminate elements that distract from the essential whole, elements that obstruct and obscure.. . . Clutter, bulk, and erudition confuse perception and stifle comprehession, whereas simplicity allows clear and direct attention. — Richard Powell