SESUG 2022 Paper 214

Q&A with the Macro Maven: Do we need Macros? An Essay on the Theory of Application Development

Ronald J. Fehd, senior maverick, theoretical programmer,
Fragile-Free Software Institute

Abstract

Description : This paper examines the theoretical steps of applications development (ApDev) of routines and subroutines in SAS® software. 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 : autocall, compile, execute, step boundaries, macro definitions, macro variables, global symbol table, reuse of compiled statements

In this paper

Introduction

overview

This section has these 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.

- variables
- conditions
- loops
- functions
- syntax

---

definitions

This is a list of words used in this article.

function :	returns 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
- assignments to 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.

- assignments to Global Symbol Table:

  filename, libname, options, title

- %include statement — subprogram
- macro language
  - compile
    - variables: `%let mvar=...;`
    - definitions: `%macro ... %mend;`
  - execution
    - variable references: `&mvar`
    - macro calls: `%do_this(data=...)`
- SAS statements

---

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 `sasv9.cfg`
- location names: filerefs and librefs used in options
- macro
  - variables: system- or user-defined
  - definitions: location of compiled code
- options reference 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 saved 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

1	2	3	4
5	6	7	8
9			10

- reuse: used often, compiled once
- centralization
- guarantee
- hide complexity: simplification, standardization

---

tactics

These are the primary reasons to convert programs to macros.

	syntax	
variables :	`%let mvar =`	pass values across steps
conditions:	`%if`	add or skip code
		branch
loops, iterative:	`%do i = 1 %to`	...
functions:	`%eval`	integer arithemetic
	`%sysevalf`	real numbers
	`%sysfunc`	access data step functions

In many cases macro definitions are a simple way to encapsulate loops and function calls that require elaborate data step or scl (SAS Component Language) 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.

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

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

---

soft-code

Identify the parameters; use SAS keywords as parameter names.

```
1  %let data = sashelp.shoes;
2  %let var  = region;
3  proc freq data = &data;
4            tables &var;
```

---

split in two

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

```
*name: sub-program-1-test.sas
%let data = sashelp.class;
%let var  = sex;
*let data = sashelp.shoes;
*let var  = region;
%include 'sub-program-1.sas';
```

```
*name: sub-program-1.sas;
*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.

```
*name: sub-program-2-test.sas;
options mprint source2;
%sub_program_2(data = sashelp.class
              ,var  = sex)
```

```
*name: 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(mprint)) eq MPRINT);
%if  &testing %then %put _local_;
proc freq data   = &data;
          tables   &var     / noprint
          out    = &out_data;
run;
%if &testing %then %do;
    proc sql; describe table &syslast;  quit;
    %end;
%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, an associative array, 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.

```
*name: make-list-names-contents.sas;
proc contents data = &in_data   noprint
              out  = list_names
             (keep = name type varnum);
run;
```

call execute of %includes

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

```
*name: proc-freq.sas;
PROC freq data   = &in_data;
          tables   &name /list;
```

```
*name:   demo-cx-include.sas;
%let     in_data = sashelp.class;
options  source2;
%include project(make-list-names-contents);
%let     cx_data    = list_variables(keep = name);
%let     cx_include = 'proc-freq.sas';
%include site_inc(cx-inclu);
```

calling macros

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

```
*name: proc_freq.sas;
%macro proc_freq
      (data   = sashelp.class
      ,name   = sex
      ,type   = c
      ,varnum = 0);
PROC freq data   = &data;
          tables   &name /list;
run;
%mend proc_freq;
```

```
*name demo-call-macro.sas;
%let     in_data = sashelp.class;
options  mprint source2;
%include project(make-list-names-contents);
%callmacr(data        = list_names
          ,macro_parms = %nrstr(data = &in_data)
          ,macro_name  = proc_freq)
```

sql constant text

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

See R. J. Fehd, "How To Use proc SQL select into for List Processing", 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 : R. J. Fehd, “An Autoexec Companion, Allocating Location Names during Startup”, Autoexec Companion;
A. L. Carpenter, “Five Ways to Create Macro Variables: A Short Introduction to the Macro Language”, ways to create macro variables;
First and Ronk, “SAS(R) Macro Variables and Simple Macro Programs”, programming with macro variables

testing, tracing : R. J. Fehd, “Writing Testing-Aware Programs that Self-Report when Testing Options are True”, Writing Testing-Aware Programs;
R. J. Fehd, “Using Functions SYSFUNC and IFC to Conditionally Execute Statements in Open Code”, Using Sysfunc and Ifc;
R. Fehd, “Journeymen’s Tools: Two Macros — ProgList and PutMvars — to Show Calling Sequence and Parameters of Routines”, using global macro variables to trace calls

list processing : R. J. Fehd and A. Carpenter, “List Processing Basics: Creating and Using Lists of Macro Variables”, List Processing Basics;
R. J. Fehd, “How To Use proc SQL select into for List Processing”, Using Sql for List Processing;
R. J. Fehd, “List Processing Routine CallXinc: Calling Parameterized Include Programs Using a Data Set as List of Parameters”, List Processing Routine Call-Execute-an-Include;
R. J. Fehd, “List Processing Macro Call-Macro”, Macro Call-Macro: using a control data set to call macros

%do loops : R. J. Fehd, “Writing Macro Do Loops with Dates from Then to When”, Macro Loops with Dates

opinion : R. J. Fehd, “Macro Design Ideas: Theory, Template, Practice”, Macro Design Ideas

---

About the author:

Ronald J. Fehd Ron.Fehd.macro.maven at gmail dot com

affiliation	senior maverick, theoretical programmer, Fragile-Free Software Institute
also known as	`macro maven` on SAS-L
facebook	`https://www.facebook.com/ron.fehdmacromaven`
linkedin	`https://www.linkedin.com/in/ronald-fehd-5125991/`

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 comprehension, whereas simplicity allows clear and direct attention. — Richard Powell

References

Carpenter, Arthur L. (2005). "Five Ways to Create Macro Variables: A Short Introduction to the Macro Language". In: *SESUG*. 12 pp.; call symput, %do, %global, %let, %local, ods, parameters in a macro definition, sql select into, sysparm, %window. URL: http://analytics.ncsu.edu/sesug/2005/HW03_05.PDF (cit. on p. 9).

Fehd, Ronald (2005). "Journeymen's Tools: Two Macros — ProgList and PutMvars — to Show Calling Sequence and Parameters of Routines". In: *SAS Users Group International Annual Conference Proceedings*. Applications Development, 8 pp.; debugging, testing, tracing, routine and subroutine calls, using parameterized %include files; using options when testing, writing list of macro variables to log. URL: http://www2.sas.com/proceedings/sugi30/004-30.pdf (cit. on p. 9).

Fehd, Ronald J. (2007). "Writing Testing-Aware Programs that Self-Report when Testing Options are True". In: *NorthEast SAS Users Group Conference Proceedings*. Coders' Corner, 20 pp.; topics: options used while testing: echoauto, mprint, source2, verbose; variable testing in data step or macros; call execute; references. URL: http://www.lexjansen.com/nesug/nesug07/cc/cc12.pdf (cit. on p. 9).

— (2009a). "List Processing Routine CallXinc: Calling Parameterized Include Programs Using a Data Set as List of Parameters". In: *Western Users of SAS Software Annual Conference Proceedings*. Applications Development, 20 pp.; call execute, data review, data structure, dynamic programming, list processing, parameterized includes, examples. URL: http://www.lexjansen.com/wuss/2009/app/APP-Fehd2.pdf (cit. on p. 9).

— (2009b). "Using Functions SYSFUNC and IFC to Conditionally Execute Statements in Open Code". In: *SAS Global Forum Annual Conference Proceedings*. Coders Corner, 10 pp.; topics: combining functions ifc, nrstr, sysfunc; assertions for testing: existence of catalog, data, file, or fileref; references. URL: http://support.sas.com/resources/papers/proceedings09/054-2009.pdf (cit. on p. 9).

— (2010). "How To Use proc SQL select into for List Processing". In: *SouthEast SAS Users Group Conference Proceedings*. Hands On Workshop, 40 pp.; topics: writing constant text, and macro calls, using macro %do loops; references. URL: http://analytics.ncsu.edu/sesug/2010/HOW06.Fehd.pdf (cit. on pp. 8, 9).

— (2013a). "Macro Design Ideas: Theory, Template, Practice". In: *MidWest SAS Users Group Annual Conference Proceedings*. 21 pp.; topics: logic, quality assurance, testing, style guide, documentation, bibliography. URL: http://www.mwsug.org/proceedings/2013/00/MWSUG-0002.pdf (cit. on p. 9).

— (2013b). "Writing Macro Do Loops with Dates from Then to When". In: *MidWest SAS Users Group Annual Conference Proceedings*. 20 pp.; topics: dates are integers, formats and functions to convert date references to integers, calculations, bibliography. URL: http://www.mwsug.org/proceedings/2013/00/MWSUG-2013-S115.pdf (cit. on p. 9).

— (2014). "List Processing Macro Call-Macro". In: *Western Users of SAS Software Annual Conference Proceedings*. 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. URL: http://www.lexjansen.com/wuss/2014/cc/97.pdf (cit. on p. 9).

— (2015). "An Autoexec Companion, Allocating Location Names during Startup". In: *MidWest SAS Users Group Annual Conference Proceedings*. Beyond Basics, 15 pp.; autocall macros, global symbol table, filerefs, librefs, cexist catalogs, exist data set, sasautos. URL: http://www.lexjansen.com/mwsug/2015/BB/MWSUG-2015-BB-10.pdf (cit. on p. 9).

Fehd, Ronald J. and Art Carpenter (2009). "List Processing Basics: Creating and Using Lists of Macro Variables". In: *SouthEast SAS Users Group Conference Proceedings*. 8.of.9. URL: http://analytics.ncsu.edu/sesug/2009/HOW008.Fehd.Carpenter.pdf (cit. on p. 9).

First, Steven and Katie Ronk (2005). "SAS(R) Macro Variables and Simple Macro Programs". In: *SUGI-30*. 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. URL: http://www2.sas.com/proceedings/sugi30/130-30.pdf (cit. on p. 9).