Paper A11

Writing Testing-Aware Programs that Self-Report when Testing Options are True

Ronald J. Fehd, Centers for Disease Control and Prevention, Atlanta, GA, USA

ABSTRACT

Program development proceeds through four phases: investigation, development, unit testing, and integration testing. Testing typically consumes 50% of project resources. Any programming effort spent in support of testing may significantly reduce project costs and help ensure the delivery of the project's product: a set of tested and integrated programs.

This paper examines the architectural issues of writing SAS® programs that are testing-aware: that have a variable Testing whose default value is false, which can be reset to true based on the values of options and which write information to the log when testing is true.

Topics covered include options used for testing, both command-line and used-anywhere; example code for data step, includes and macro programs used as any of module, routine, or subroutine.

Audience architects, designers, software engineers, project managers, intermediate to advanced programmers, advanced users.

Keywords call execute, echoauto, macro, module, mprint, options, parameterized include file, routine, source2, subroutine, testing, verbose

Information programs using call execute to: 1. call parametized include files and 2. macros.

In this paper This paper contains the following topics.

INTRODUCTION

Overview

The Introduction contains a discussion of these topics:

- History
- What is Information
- Options Used while Testing
- Information Needed while Testing

History

Brooks in his book, Mythical Man-Month, Brooks Jr. [1, AW.Brooks.1995] provides the following chart of time spent in program development:

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

From the table we see that testing occurs in the following phases:

- development
- unit testing
- integration testing

What is Information

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

Programs contain statements that implement an algorithm on a data structure. The most common information wanted during testing is a report of metadata: the data structure of data sets created or read during processing. When testing data steps that do calculations or manipulate character variables notes written to the log are often helpful. Finally, an example listing may show problems.

Options Used while Testing

The following options are used in the examples. Option **mprint** is used to expand text from macros; option **source2** is used to echo statements in included files to the log; options **echoauto** and **verbose** are command-line options and only write to the log during start-up. They are therefore our prime candidates for differentiating between modules and subroutine testing.

Information Needed while Testing

A developer or tester might need the following information while testing.

Data Listing The Print procedure is useful during testing; using the data step option `obs=` limits the number of rows.

```
PROC Print data   =  &SysLast.
          (obs    =   20);
           title2   "&SysLast.";
           run;
```

Data Step Notes Before v9 the `put` statement wrote to the default destination log, — `file log;` — to the listing, — `file print;` — or to the most recent file named in the `file` statement — `file ’filename.txt’`. The `putlog` statement writes only to the log.

```
DATA   _Null_;
putlog ’note:’ x=;
```

Note: the special list `_all_` cannot be used with putlog.

Macro Variable Values The `%put` statement can be used to show macro variable names and their values.

```
%Put _global_;
%Put _local_; %*in macro;
```

Metadata data structure can be obtained from either of:

Proc Contents The Contents procedure prints a complete set of metadata to the listing. Information includes:

- data set label
- number of observations
- data structure, sorted by variable name

Note: the two-page output is written to listing, compare to `describe table`, which is written to log.

```
PROC Contents data   =  &SysLast.;
              title2   "&SysLast.";
              run;
```

Proc Sql The SQL procedure writes metadata to the log. The data structure is listed in variable number order.

```
PROC SQL; describe table &SysLast.;
          quit;
```

WRITING TESTING-AWARE PROGRAMS ZIP

Topics

This section contains the following topics.

OPTIONS USED WHILE TESTING

Overview

The purpose of this paper is to show that the following options can be used to turn on the production of information during the testing process.

option	group	writes to log statements from
echoauto	logcontrol	autoexec
mprint	macro	macros
source2	logcontrol	include files
verbose	logcontrol	configuration files

EchoAuto

The option `echoauto` lists the statements in the autoexec; otherwise only the notes from the statements in the autoexec are shown in the log. Option `echoauto` may be used only on the command-line or in a configuration file. This option adds the option `source2` to the inclusion of the autoexec file; i.e.: `%include 'autoexec.sas' / source2;`
Example usage:

```
rem name: MyProgram.bat
sas MyProgram -echoauto
```

Note: `echoauto` is in the options group LogControl.

Mprint

The option `mprint` writes the text produced by macros to the log.
Example usage:

command line option name is preceeded by a hyphen

```
rem name: MyProgram.bat
sas MyProgram -mprint
```

in program

```
*name: MyProgram.sas
options mprint;
```

Note: `mprint` is in the options group Macro.

Source2

The option `source2` writes the text contained in included files to the log.
Example usage:

command line option name is preceeded by a hyphen

```
rem name: MyProgram.bat
sas MyProgram -source2
```

in program

```
*name: MyProgram.sas
options source2;
```

Note: `source2` is in the options group LogControl.

Verbose

The option `verbose` writes the settings of options specified in any configuration files to the log. This option, like `echoauto` is a command-line or configuration file only statement.
Example usage, command line:

```
rem name: MyProgram.bat
sas MyProgram -verbose
```

Note: `verbose` is in the options group LogControl.

Summary of Options

Program ProcOptions-define-value.sas lists the full description of each option and the options in each group.

```
*name: ProcOptions-define-value.sas;
PROC Options define value option = echoauto;
PROC Options define value option = mprint  ;
PROC Options define value option = oplist  ;
PROC Options define value option = source2 ;
PROC Options define value option = verbose ;

PROC Options group = LogControl; *echoauto
                                  source2
                                  verbose ;
PROC Options group = Macro     ; *mprint  ;
run;
```

THE VARIABLE TESTING

Overview

In this section I examine a variable `Testing` in each of:

- data step
- macro

Data Step

In a data step the variable `Testing` is allocated as type numeric; as an integer its length can be reduced to 4 bytes since its values are in (0,1).

```
DATA   Routine;
attrib Testing length = 4;%*integer: boolean;
drop   Testing  ;
retain Testing 1;
*...;
if Testing then do;
   putlog 'note: ' x=;
   end;
```

Macro

In a macro the variable `Testing` is added to the parameter list.

```
%Macro RoutineA
(data    =
,Testing = 1
) / des  = 'description of macro'
;
*...;
%if &Testing. %then %do;
    %put _local_;
    %end;
```

TESTING OPTIONS IN DIFFERENT PROGRAM TYPES

Discussion

Programs are different in the hierarchy of processing. They may be, at the highest level, a module, which calls other routines and subroutines. Routines may call subroutines. Subroutines do not call other programs. Testing is conducted according to the type of program: once subroutines are tested, then they are used frequently by other modules and routines, consequently their reporting can be minimized when testing is conducted on higher level programs.

This table shows the the various types of programs and what other program types they call.

type	may call
module	routines and subroutines
routine	other routines and subroutines
subroutine	none

Truth Table

This table shows the logic of use of combinations of options.

program	command-line only				used anywhere			reporting or quiet		
type	EchoAuto	op	Verbose	op	mprint	op	source2	module	routine	subroutine
module	T			and	(T	or	T)	report	report	quiet
routine	T	or	T	or	T	or	T	.	report	quiet
subroutine			T	and	(T	or	T)	.	.	report

USING CALL ROUTINES AND FUNCTIONS

Overview

This sections describes the call routines and functions used in the examples.

- Call Execute
- %nrstr: No Rescan String
- Cat* Functions
- Macro Functions
  - %eval: Evaluate Macro Expressions
  - %sysfunc getoption

Call Execute

The call execute routine has one argument, a character expression, that contains either a character variable, a sas statement or a macro invocation.

```
call execute(VarChar);
call execute(’*sas statement;’);
```

The call execute routine is used in the following examples to conditionally execute both sas statements and macro calls.

%nrstr: No Rescan String

The nrstr function has one argument, a character string, that, in the examples shown here, contains a macro invocation. The nrstr function masks the special characters ampersand (&) and percent-sign (%); this function is used to delay the expansion of macro calls until the next step. The call execute routine is used in the following examples to conditionally execute both sas statements and macro calls.
Fehd and Carpenter [5, sgf2007.113] discusses this problem and provides examples to show the error when not using macro function nrstr.

```
call execute(’%nrstr(%MyMacro(data=testing))’);
```

Cat* Functions

The cat* functions replace the concatenation operator (!!).

- cat: no trim
- catt: remove leading and trailing blanks
- cats: remove trailing blanks
- catx: remove leading and trailing blanks, add separator

Macro Functions

sysfunc getoption The combination of macro functions `sysfunc` and `getoption` return the value of a system option.

```
2    options nosource2;
3    %put text:%sysfunc(getoption(SOURCE2));
text:NOSOURCE2
4    options   source2;
5    %put text:%sysfunc(getoption(SOURCE2));
text:SOURCE2
```

eval The macro function `eval` evaluates its argument and returns an integer.

```
7     %put false:%eval(0 or 0);
false:0
8     %put true :%eval(0 or 1);
true :1
9     options nosource2;
10    %put false:%eval(%sysfunc(getoption(Source2)) eq SOURCE2);
false:0
11    options   source2;
12    %put true :%eval(%sysfunc(getoption(Source2)) eq SOURCE2);
true :1
```

EXAMPLES

In this Section

This section has the following topics.

SITE AND PROJECT

Recommendations for Testing

Folders The programs shown here can be stored in the following folders:

Site example

```
C:\SASsite\
C:\SASsite\includes files: LibWork, Test*
C:\SASsite\macros   files: TestMacro
```

Project example

```
C:\SASproject\
C:\SASproject\ProjectA
C:\SASproject\ProjectA\sas       programs
C:\SASproject\ProjectA\sas7b     sas data sets
C:\SASproject\ProjectA\sas7bWork sas data sets during testing
```

Project autoexec has filename and libname references to above folders

Programs use two-level data set names — e.g.: `LibWork.MyData` — to save temporary output data sets which are input to other programs

Testing reassign libref LibWork to save data sets to permanent folder for review while testing

AutoExec

The include file autoexec.sas is used allocate filenames, libnames, and set global options. Note-1 that the filenames `Project` and `Site` refer to the same folder, which has been used for developing these examples. In practice the `Site` folder would be a site-wide folder accessible by all projects. Note-2 the allocation of a second libname `LibWork` which is used to save output data sets when unit testing of routines and subroutines. In production this libref folder is the same as libref Work.

```
title   ’SUG: Fehd: Writing Testing-aware Programs’;
filename Project ’.’       ;%*here;
filename Site    ’.’       ;%*here;
options  sasautos = (Project Site SASautos);
Libname  Library ’..\sas7b’;%*sibling;
Libname  LibWork "%sysfunc(getoption(Work))";
```

Note: Fehd [4, sugi30.267] discusses the option sasautos.

PROGRAMS FOR TESTING

Overview

This sections shows subroutine programs used for testing.

Program	Page
LibWork	10
TestDescribe	10
TestInclude	10
TestMacro	11
TestPrint-*	11

LibWork

This program clears the libref LibWork and assigns it a value of a permanent folder.

```
* name       : LibWork.sas;
* description: when testing reassign Libref LibWork;
*              to permanent folder                  ;
* purpose    : save temp data sets for unit tests  ;

DATA   _Null_;
attrib Testing length = 4;
retain Testing %eval(0
            or %sysfunc(getoption(EchoAuto))
                               eq ECHOAUTO
            or %sysfunc(getoption(Mprint))
                               eq MPRINT
            or %sysfunc(getoption(Source2))
                               eq SOURCE2
            or %sysfunc(getoption(Verbose))
                               eq VERBOSE  );
if Testing then do;
   call execute("Libname LibWork clear; run;");
   call execute("Libname LibWork '..\sas7bWork';");
   end;
stop;
run;
```

TestDescribe

The include file TestDescribe.sas is used to write the data structure of a dataset to the log when testing with options `mprint` or `source2`.

```
*name: TestDescribe.sas;
*description: for routine                ;
*             call execute describe table;
DATA   _Null_;
attrib Testing length = 4;%*integer: boolean;
drop   Testing ;
retain Testing %eval(0
       or %sysfunc(getoption(MPRINT))
                          eq MPRINT
       or %sysfunc(getoption(SOURCE2))
                          eq SOURCE2  );
if Testing then do;
   call execute('PROC SQL;');
   call execute('describe table &SysLast.;');
   call execute('quit;');
   end;
run;   %*calls executed in this step;
```

TestInclude

The include file TestInclude.sas is used to test the routine Call-Execute-Include (CxInclude).

```
*name: TestInclude.sas;
%put _global_;
run;
```

TestMacro

The macro TestMacro.sas is used to test the routine Call-Execute-Macro (CxMacro).

```
%Macro TestMacro(Testing = 1)
/ des = ’site: echo parameter list to log’     parmbuff ;
%If &Testing. %then %put note: &SysMacroName. &SysPbuff.;
run; %Mend;
```

TestPrint

The include files TestPrint* are used to print the first few rows of a dataset when testing.

TestPrint-Module

Checking `echoauto` and either of `mprint` or `source2`

```
*name:  TestPrint-Module.sas;
*description: for module print SysLast;
DATA   _Null_;
attrib Testing length = 4;%*integer: boolean;
drop   Testing ;
retain Testing %eval(0
       or         %sysfunc(getoption(ECHOAUTO))
                                  eq ECHOAUTO
          and (   %sysfunc(getoption(MPRINT))
                                  eq MPRINT
               or %sysfunc(getoption(SOURCE2))
                                  eq SOURCE2  ));
```

TestPrint-Routine

Checking either of `mprint` or `source2`

```
retain Testing %eval(0
       or %sysfunc(getoption(MPRINT))
                          eq MPRINT
       or %sysfunc(getoption(SOURCE2))
                          eq SOURCE2  );
```

TestPrint-Subroutine

Checking either of `mprint` or `source2` and `verbose`

```
retain Testing %eval(0
       or    (    %sysfunc(getoption(MPRINT))
                                  eq MPRINT
               or %sysfunc(getoption(SOURCE2))
                                  eq SOURCE2  )
          and     %sysfunc(getoption(VERBOSE))
                                  eq VERBOSE   );
if Testing then do;
   call execute(’PROC Print data = &SysLast.  ’);
   call execute(’(obs = 20);’);
   call execute(’title2           "&SysLast.";’);
   end;
run;    %*calls executed in this step;
```

USING INCLUDES

Overview

This section describes the programs in the example of includes.

name	type	description
Example-Include	module	main
MakeNameList	subroutine	create data set: List
CxInclude	routine	process each row of List: call subroutine: FreqOfInc
PrintSmry	subroutine	print output

Example-Include

This example of include file usage is a main module which calls two subroutines, MakeNameList and PrintSmry, and one routine, Call-Execute-Include (CxInclude), which calls a subroutine FreqOfInc. The routines and subroutines shown are examples of parameterized include files.

Example-Include Input

```
*name: Example-Include.sas;
*from: FreqAll;

*input  : make data set;
          %Let MnLibName = sashelp;
          %Let MnMemName = Class  ;
          %Let MnOutLib  = LibWork;
          %Let MnOutData = ListNames;
          %Include Project(MakeNameList);
          %Include Site   (TestPrint-Module);
```

Example-Include Process

```
*process: call routine to execute subroutine;
          %Let CxLibName  = LibWork;
          %Let CxMemName  = &MnOutData.;
          %Let CxInclude  = Project(FreqOfInc);
          %Let CxNames    = LibName MemName Name Type;
          %*   subroutine FreqOfInc parameters;
          %Let FoiOutLib  = LibWork ;
          %Let FoiOutData = ListSmry;
          %Include Site   (CxInclude);
          %Include Site   (TestPrint-Module);
```

Example-Include Output

```
*output : print;
          %Let LibName   = LibWork;
          %Let MemName   = &FoiOutData.;
          %Include Project(PrintSmry);
```

Testing Example-Include

Example-Include.bat This program is used to execute the example module as a production job.

```
rem Example-Include.bat
sas Example-Include
```

Example-Include-Test.bat The batch file used to test the example module contains the options `echoauto` and `source2` for includes.

```
rem Example-Include-Test.bat
sas Example-Include-Test -echoauto -source2
rem     integration test module
```

Example-Include-Test This program is used to test the example module. Note: the testing options are contained in the .bat file.

```
*name:           Example-Include-Test.sas;
%Include Site(LibWork);
%Include Project(Example-Include);
```

MakeNameList

MakeNameList is a subroutine which creates the table of parameter values used by the list processing routine Call-Execute-Include (CxInclude).

Program

```
*name: MakeNameList.sas;
*parameters:       ;
   *input :; *Let MnLibName = sashelp  ;
             *Let MnMemName = class    ;
   *output:; *Let MnOutLib  = work     ;
             *Let MnOutData = ListNames;

PROC SQL noprint;
         create table &MnOutLib..&MnOutData as
         select LibName, MemName, Name, Type
         from   Dictionary.Columns
         where  LibName eq "%upcase(&MnLibName.)"
           and  MemName eq "%upcase(&MnMemName.)";
         quit;
```

Testing MakeNameList

MakeNameList-Test.bat note options for subroutine testing

```
rem MakeNameList-Test.bat
sas MakeNameList-Test -source2 -verbose
rem              Test subroutine
```

MakeNameList-Test compare to usage in Example-Include; note TestPrint

```
*name: MakeList-Test.sas;
*note: test subroutine;
*note: write to LibWork;
*      save for later tests;
%Include Site(LibWork);
%Let MnLibName = sashelp;
%Let MnMemName = class  ;
%Let MnOutLib  = LibWork;
%Let MnOutData = ListNames;
%Include Project(MakeNameList);
%Include Site   (TestDescribe);
%Include Site   (TestPrint-Subroutine);
```

CxInclude

CxInclude is a routine which reads a data set where each row contains the values used by a parameterized include file; the routine makes global macro variables of each named variable and then calls the named include file. Find the program in the Writing Testing-aware Programs zip.

This routine is tested as a stand-alone unit with a dummy include file (TestInclude) and with its production subroutine, FreqOfInc.

CxInclude, Unit Test

CxInclude-Test-unit.bat unit test

```
rem      CxInclude-Test.bat
call sas CxInclude-Test-unit -source2
rem           unit test of routine
```

CxInclude-Test-unit calling a dummy include file; note: no TestPrint

```
*name: CxInclude-Test-unit.sas;
%Include Site(LibWork);
%Let CxLibName = LibWork;
%Let CxMemName = ListNames;
%Let CxInclude = Project(TestInclude);
%Let CxNames   = LibName MemName Name Type;
%Include Project(CxInclude);
```

CxInclude, Integration Test

CxInclude-Test-integrate.bat unit test with production subroutine

```
rem   CxInclude-Test-integrate.bat
sas   CxInclude-Test-integrate -source2
rem integration test of routine
```

CxInclude-Test-integrate calling FreqOfInc; note: TestPrint

```
*name: CxInclude-Test-integrate.sas;
%Include Site(LibWork);
%Let CxLibName  = LibWork;
%Let CxMemName  = ListNames;
%Let CxInclude  = Project(FreqOfInc);
%Let CxNames    = LibName MemName Name Type;
%*subroutine FreqOfInc parms;
%Let FoiOutLib  = LibWork;
%Let FoiOutData = TestReport;
%Include Site(CxInclude);
%Include Site(TestDescribe);
%Include Site(TestPrint-Routine);
```

FreqOfInc

The subroutine FreqOfInc is called by Call-Execute-Include (CxInclude); it does a proc freq and standardizes the output data set. Find the program in the Writing Testing-aware Programs zip.

FreqOfInc-Test.bat note options for subroutine testing

```
rem FreqOfInc-Test.bat
sas FreqOfInc-Test -mprint -verbose
rem           test subroutine
```

FreqOfInc-Test note TestPrint

```
*name: FreqOfInc-Test.sas;
%Include Site(LibWork);
%Let Libname    = sashelp ;
%Let MemName    = Class   ;
%Let Name       = Height  ;
%Let Type       = num     ;
%Let FoiOutLib  = LibWork ;
%Let FoiOutData = TestData;
%Include Project(FreqOfInc);
%Include Site(TestDescribe);
%Include Site(TestPrint-Subroutine);
```

PrintSmry

The subroutine PrintSmry prints the report data set. No testing subroutines are provided.

```
*name: PrintSmry.sas;
*note: from FreqAll;
*parameters:       ;
%*input; *Let LibName = work;
         *Let MemName = ListSmry;

Proc Print data = &LibName..&MemName.
          (drop = MemName);
           title2 "LibName: &Libname.";
           title3 "MemName: &MemName.";
           by     VarName notsorted;
           id     VarName          ;
run;
```

USING MACROS

Overview

Programs:

name	type	description
Example-Macro	module	main:
ExMacro	module	create data set call CxMacro print summary
CxMacro	routine	process each row of List
FreqOf	subroutine	proc freq out=

Example-Macro

The example macro contains a call to the module ExMacro.

```
*name: Example-Macro.sas;
%ExMacro(LibName = sashelp
        ,MemName = class);
```

Example-Macro.bat

```
rem Example-Macro.bat
sas Example-Macro
```

Example-Macro-Test.bat

```
rem Example-Macro-Test.bat
sas Example-Macro-Test -echoauto -mprint
rem               Test module
```

ExMacro

As a module ExMacro contains the subroutines which make the list processing data set and prints the output. It calls the routine Call-Execute-Macro (CxMacro) which generates the calls of the subroutine macro FreqOf.
The code is similar to Example-Include so I show only dissimilar parts.
Find the program in the Writing Testing-aware Programs zip.
This shows the reassignment of the macro variable Testing to the values of the module testing options `echoauto` and `mprint`.

```
*name: ExMacro.sas;
%Macro ExMacro
(LibName = sashelp
,MemName = class
,OutLib  = Work
,Testing = 0
) / des  = ’example macro as module’
;
%Let Testing = %eval(&Testing
     or      %sysfunc(getoption(ECHOAUTO))
                             eq ECHOAUTO
        and  %sysfunc(getoption(MPRINT))
                             eq MPRINT    );
```

ExMacro.sas subroutine: make list This is same code as the subroutine MakeNameList.

```

PROC SQL noprint;
```

ExMacro.sas subroutine: testing This is the same code as TestDescribe and TestPrint.

```

%If &Testing. %then %do;
    Proc SQL;  describe table &SysLast.;
               quit;
    Proc Print data   =       &SysLast.(obs = 20);
               title2         &SysLast.;
```

ExMacro.sas routine: call execute macro subroutine FreqOf This routine is similar to CxInclude.

```

%CxMacro(CxLibName = &OutLib.
        ,CxMemName = ListNames
        ,CxMacro   = FreqOf
        ,CxNames   = LibName MemName Name Type
```

ExMacro.sas last test This is the same code as TestDescribe.

```

%If &Testing. %then %do;
    Proc SQL;  describe table &SysLast.;
               quit;
```

CxMacro

The routine CxMacro is similar to the parameterized include file CxInclude. Find the program in the Writing Testing-aware Programs zip.

CxMacro-Test.bat

```
rem CxMacro-Test.bat
sas CxMacro-Test -mprint
rem         test routine
```

CxMacro-Test.sas

```
*name: CxMacro-Test.sas;
%Include Site(LibWork);
%CxMacro(CxLibName = LibWork
        ,CxMemName = ListNames
        ,CxMacro   = TestMacro
        ,CxNames   = LibName MemName Name Type
        ,Testing   = 1
        );

```

FreqOf

The subroutine FreqOf is similar to the parameterized include file FreqOfInc. Find the program in the Writing Testing-aware Programs zip.

FreqOf-Test.bat

```
rem FreqOf-Test.bat
sas FreqOf-Test -mprint -verbose
rem        test subroutine
```

FreqOf-Test.sas

```
%Include Site(LibWork);
%FreqOf(Libname = sashelp
       ,MemName = Class
       ,Name    = Height
       ,Type    = num
       ,OutLib  = LibWork
       ,Testing = 1
       );
%Include Site(TestPrint-Subroutine);
```

CONCLUSION

Summary

Testing programs may take up to half of time on a project. In this paper I have demonstrated the use of include files and additional code within macros that may be used to provide necessary information when testing.

Suggested Reading

Call Execute and %nrstr Fehd and Carpenter [5, sgf2007.113] demonstrate the timing of the error of using call execute of macros without the macro function %nrstr.

Documentation Fehd [2, sugi30.067] provides a template for a program description.

Proc Freq Fehd [7, sgf2007.028] wrote the proc freq code upon which the example includes and macros in this paper are based.

Project Fehd [6] provides a production project using the testing methods described here.

Using Options Fehd [3, sugi30.004] shows a macro ProgList, which tests options.

REFERENCES

[1] Frederick P. Brooks Jr. *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition, 2/E*. Addison-Wesley, 1995. URL `http://www.aw-bc.com/catalog/academic/product/0,,0201835959,00%2ben-USS_01DBC.html`.

[2] Ronald Fehd. Journeymen's tools: The writing for reading and reuse program header. In *Proceedings of the 30th SAS User Group International Conference*, 2005. URL `http://www2.sas.com/proceedings/sugi30/067-30.pdf`. Coders' Corner, 4 pp.; topics: documentation, program development costs, quality, reuse, theory; info: example program documentation header.

[3] Ronald Fehd. Journeymen's tools: Two macros — proglist and putmvars — to show calling sequence and parameters of routines. In *Proceedings of the 30th SAS User Group International Conference*, 2005. URL `http://www2.sas.com/proceedings/sugi30/004-30.pdf`. Coders' Corner, 8 pp.; topics: tracing included routine and subroutine calls, using parameterized include files; info: using options when testing, writing list of macro variables to log.

[4] Ronald Fehd. A SASautos companion: Reusing macros. In *Proceedings of the 30th SAS User Group International Conference*, 2005. URL `http://www2.sas.com/proceedings/sugi30/267-30.pdf`. Tutorials, 12 pp.; topics: autocall, autoexec, configuration file (sasv9.cfg), compiled and stored macros, masking, options (mautosource, mstored, sasmstore), program reuse, sasautos (environment variable, filename, option); info: autoexec examples, utility program ListMcat: show same-named macros in different catalogs.

[5] Ronald Fehd and Art Carpenter. List processing basics: Creating and using lists of macro variables. In *Proceedings of the SAS Global Forum*, 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 with call execute argument; 11 examples, bibliography.

[6] Ronald J. Fehd. Freqlibname: A data review routine for all memnames in a libname. In *Proceedings of the NorthEast SAS User Group Conference*, 2007. Coders' Corner, 22 pp.; topics: replacing macros with call execute of parameterized include files, saving procs freq and summary output data set; info: complete test suite of modules, routines, and subroutines, getting mode from proc freq.

[7] Ronald J. Fehd. Journeymen's tools: Data review macro freqall – using proc sql list processing with dictionary.columns to eliminate macro do loops. In *Proceedings of the SAS Global Forum*, 2007. URL `http://www2.sas.com/proceedings/forum2007/028-2007.pdf`. Coders' Corner, 10 pp.; topics: designing macros for reporting, creating and using macro arrays, writing text of macro calls into macro variable, executing macro calls in macro variable, bibliography.

To get the code examples in this paper search http://www.sascommunity.org for the Writing Testing-aware Programs zip.

Author: Ronald Fehd `mailto:RJF2@cdc.gov`
Centers for Disease Control
4770 Buford Hwy NE
Atlanta GA 30341-3724

	about the author:	
education:	B.S. Computer Science, U/Hawaii,	1986
	SUGI attendee	since 1989
	SAS-L reader	since 1994
experience:	programmer: 20+ years	
	data manager at CDC, using SAS: 18+ years	
	author: 12+ SUG papers	
SAS-L:	author: 4,000+ messages to SAS-L since1997	
	Most Valuable SAS-L contributor:	2001, 2003

Document Production: This paper was typeset in LaTeX. For further information about using LaTeX to write your SUG paper, consult the SAS-L archives:

```
http://www.listserv.uga.edu/cgi-bin/wa?S1=sas-l
Search for                 :
The subject is or contains: LaTeX
The author's address       : RJF2
Since                      : 01 June 2003
```