List Processing Routine CallXinc: Calling Parameterized Include Programs Using a Data Set as List of Parameters
Ronald J. Fehd, Centers for Disease Control, and Prevention, Atlanta, GA, USA

Abstract

Description: This article reviews the list processing routine CallXinc, a parameterized include program. This routine reads a data set, converts character variables in each row into global macro variable assignment statements, and calls another parameterized include program. Examples are provided which illustrate list processing using this routine.

Purpose: replacement for macro do loop

Audience: intermediate to advanced users and macro programmers

Copyright Notice: This is a Derivative Work of the book, A SAS(R) Companion, a work in progress by R. J. Fehd, copyright 2008, 2009.

Information: This article shows how to convert unknown variable names and values to macro variable assignment statements.

Keywords: call execute global macro variables, includes, list processing, module, nrstr, parameters, routines, source2, subroutines

Contents

Routine CallXinc 3

Algorithm .......................................................... 5
Data Structure ..................................................... 6
Process .............................................................. 9
Program Listing ................................................... 11

Usage Example 12

Usage Development 13

Proc Print .......................................................... 13
Making Lists with Proc Freq ....................................... 15
Making Lists with Proc Sort ..................................... 16

Examples 17

Conclusion 19
Introduction

Overview

This article examines the list processing issues of using data set variable values as parameters when calling a parameterized include program.

Definitions

In this paper the following definitions are used.

Allocate  Choose a name and assign a value to that name; used in referring to global macro variables used as parameters.

Call   The natural language and pseudocode verb call is implemented in the SAS language by the include statement:

\include <file-specification>;

A program is an organized set of SAS statements.

A parameterized include program is any program which uses global macro variables as its parameters; it may be one of three categories.

• modules, call routines or subroutines
• routines, call subroutines
• subroutines, do not call other programs

List Processing is a style of program organization. It depends on a library of subroutines to create data sets — lists — which have unique values. In each row, each variable is a parameter value of a parameterized include program.

Subroutine Library is a collection of general-purpose programs which perform specific tasks. Its advantages are that these programs encapsulate complex tasks and that they are unit tested, which reduces total testing time when used in modules.

CallXinc is a general-purpose list-processing library routine which performs two processes. It reads a data set, converts each variable in a row to a global macro variable, and calls a parameterized include program.

History

SmryEachVar Fehd [3] sgf2008.003 uses two lists, ListMemnames and ListNames. ListMemnames is a list of member names in a libref and is the input for four routines. ListNames is a list of variable names in the data sets and is the input for one routine.

These five routines are consolidated into one routine, CxInclude. CallXinc is a derivative work of CxInclude. This article is a minimal explanation of this list processing routine.

Routine CallXinc

Overview

The program CallXinc is a parameterized include routine.

Process: It performs two processes:

1. read the data set; converts each character variable to a global macro variable
2. call the parameterized include program

Parameters: It has two parameters which support its processes:

1. CxData: a data set name;
2. CxInclude: a file-specification of a parameterized include program

Parameter Values

Each parameter may have options.

**CxData** may have data set options.

```sas
1 %let CxData = sashelp.class(drop = Height Weight);
2 %let CxData = sashelp.class(keep = Name Sex);
3 %let CxData = sashelp.class(rename = (Sex = Value));
```

**CxInclude** may be any valid argument to the include statement.

```sas
1 %let CxInclude = Project(Subroutine);
2 %let CxInclude = SiteIncl(SubroutineA SubroutineB);
3 %let CxInclude = 'MySubroutine.sas';
4 * using a fileref;
5 filename ThisFile 'MySubroutine.sas';
6 %let CxInclude = ThisFile;
```

Tricks

The success of this routine depends on these tricks.

- two references to the data
  - read data structure: if 0 then set &CxData.;
  - read rows: set &CxData. end = EndoFile;
- an array whose upper bound reference excludes local variables
  - array Mvar(*) _character_
  - do _I = 1 to dim(Mvar) -2;
- use of functions vname and catx
  - _Name = vname(Mvar(_I));
  - _Stmtnt= catx(' ','%let',_Name,'=',Mvar(_I),');'
- use of data set subroutine CxStmtnt called by the link statement

Continued on next page.
Example

This an example usage in a module.

Calling Program:

1. %Let CxData = Work.ListMvars;
2. %Let CxInclude = Project(MySubRoutine);
3. %Include SiteIncl(CallXinc);

- module
  - input: parameters of subroutine MySubRoutine are in Work.ListMvars
  - process: calls routine CallXinc, which produces calls to the subroutine
  - output: is results from subroutine for each row of Work.listMvars

- CallXinc
  - input: is Work.ListMvars variables
  - process: read data set variables
  - output: call execute of
    - allocate macro variables
    - subroutine

Log: The module produces this output.

1. NOTE: CALL EXECUTE generated line.
2. 1 + %Let Col1 = Value-Row1-Col1;
3. 2 + %Let Col2 = Value-Row1-Col2;
4. 3 + %Include Project(MySubRoutine);
5. 4 + %Let Col1 = Value-Row2-Col1;
6. 5 + %Let Col2 = Value-Row2-Col2;
7. 6 + %Include Project(MySubRoutine);

Documentation

This is the header of the routine.

1. /*
2.   name: 2\SAS-site\includes\CallXinc.sas
3.   is.a: parameterized include routine
4.   description: using data as parameters, call a parameterized include
5.   purpose : list processing: calling routine or subroutine
6.   parameters : data set name:
   CxData = libref.data(drop | keep | rename | where)
7.   file-specification:
   CxInclude = Project(SubroutineA)
8.   input : &CxData
9.   process : for each character variable make macro variable assignment
10.  call named Include(s): &CxInclude.
11.  output : from (sub)routine(s) : &CxInclude.
This program uses this algorithm.

1. allocate data structure
   (a) read input data structure
   (b) add local variables
   (c) provide index
   (d) enable testing

2. read rows
   (a) read columns
      i. make macro variable assignment statement
      ii. call subroutine
   (b) call list processing routine
      i. assign routine name
      ii. call subroutine

3. housecleaning

4. data step subroutine
   (a) report when testing
   (b) process statement
Data Structure

Overview
This routine reads the data structure of the list processing data set into the Program Data Vector (PDV) before allocating its own character variables, Stmnt and Name. This data structure supports the do I loop upper bound of dim(Mvar) -2.

Allocation
This is the listing of the data structure; items are explained in the following blocks.

23 DATA Work._Null_;  
24 ** read input data structure into PDV;  
25 if 0 then set &CxData.;  
26 ** allocate local processing variables;  
27 attrib _Name length = $ 32  
28 _Stmt length = $128 label = 'Statement';  
29 array Mvar(*) _character_;  
30 retain _Testing %eval(0 or %sysfunc(getoption(Source2)) eq SOURCE2);  

Read input data structure
The ability of this routine to process any data set depends on the variables of the input data set being inserted into the program data vector (PDV) before the local processing variables.

24 ** read input data structure into PDV;  
25 if 0 then set &CxData.;  

using data set options with CxData: p. 3  
second reference to CxData: p. 9  
Continued on next page.
Add local variables

** Principle of Data Step Allocation of Local Variable Names: **
Identify temporary variables with a common prefix; sas programming convention is to use an underline.

Two character variables are used for processing: variable name Name, and statement Stmnt.

```sas
/** allocate local processing variables;
 attrib _Name length = $ 32
 _Stmnt length = $128 label = 'Statement';
```

** Name:** The length of the variable Name is 32, which is the maximum length of a variable name.

** Stmnt:** The variable Stmnt is used to hold two statements.

1. macro variable assignment statement
   ```sas
   %let name = value;
   ```
2. include statement
   ```sas
   %include &CxInclude.;
   ```

** Caution:** The length of Stmnt is frangible. If the length of either the assignment or include statement exceeds the length of Stmnt then the value sent to call execute is truncated. The absence of the terminating semicolon generates an error message.

```sas
1 %Let Test1 = missing semicolon
2 %Let Test2 = has semicolon;
3 ERROR: Open code statement recursion detected.
```

** Solution:** increase the length of Stmnt.

Continued on next page.
All character variables are assigned to the array. This includes the variables in the input data set and the local variables Name and Stmnt.

```plaintext
darray Mvar(*) _character_
```

The local processing variables are excluded from processing by reducing the upper bound of the array by two.

```plaintext
do _I = 1 to dim(Mvar) -2;
```

The variable Testing is used in the CxStmt data step subroutine. The option source2 has two values, true: source2, and false: nosource2. The value of Testing is set to the boolean value of the option source2.

```plaintext
retain _Testing %eval(0 or %sysfunc(getoption(Source2)) eq SOURCE2);
```

The value of Testing is used in the data set subroutine CxStmt to write the values in Stmnt to the log.

```plaintext
CxStmt:
if _Testing then putlog _Stmnt=;
```

Process

Overview

The data step has several processing items:

- Read rows
- Read columns
- Call parameterized include
- Housecleaning
- End processing
- Subroutine CxStmtnt

Read Rows

This loop reads the input data set.

```plaintext
do until(EndoFile);
    set &CxData. end = EndoFile;
    *...;
    end;
stop;
```

first reference to CxData: p. 8

see Do Which Fehd [4] sgl2007.067

Read Columns

This loop through the array creates a macro variable assignment statement.

```plaintext
36    do _I = 1 to dim(Mvar) -2;
37        _Name = vname(Mvar(_I));
38        ** make statement: *let Name = value ;
39        _Stmnt = catx(’’,’%let’,_Name,’=’,Mvar(_I),’;’);
40        link CxStmtnt;
41    end;
```

This upper bound of the array excludes the CallXinc local variables Name and Stmnt.

```plaintext
dim(Mvar) -2
```

vname  An array reference returns the value of the variable; the vname function returns the name of the variable.

catx   The function catx concatenates its arguments and inserts the separator named in the first argument.

link   The link statement is a go-to and return-from a data step subroutine. CxStmtnt: is a label statement.

return: p. 10

Continued on next page.
The parameter CxInclude contains a file-specification of parameterized include programs. These statements call those programs.

\[
\begin{align*}
\text{_Stmt} &= "\%Include \ &\text{_CxInclude};"; \\
\text{link } \text{CxStmtnt};
\end{align*}
\]

reference to definition of CxInclude: p. 3

---

### Housecleaning

**Principle of List Processing:** A program which allocates a global macro variable is responsible for its removal.

**Secondary Processing**

This routine allocates an unknown number of global macro variables for use by the called program. When primary processing is complete, these are no longer needed. They are removed from the global macro variable symbol table with the symdel (Symbol-Delete) statement.

\[
\begin{align*}
\text{** Symbol-Delete global macro variables allocated by this routine;} \\
\text{do } _I = 1 \text{ to dim(MVar) -2}; \\
\text{\quad _Name = vname(MVar(_I));} \\
\text{\quad _Stmtnt = catx(’,’,%symdel,’_Name,’);} \\
\text{link CxStmtnt;} \\
\text{end;}
\end{align*}
\]

---

### End of Processing

The stop statement ends the data set processing. This is the boundary before the data step subroutine definition.

\[
\begin{align*}
\text{stop;}
\end{align*}
\]

---

### Subroutine CxStmtnt

The data step subroutine CxStmtnt encapsulates two processes.

1. when testing, writes the value of Stmnt to the log
2. call execute plus nrstr of the variable Stmnt

\[
\begin{align*}
\text{CxStmtnt:} \\
\text{if } _\text{Testing} \text{ then putlog } _\text{Stmtnt=}; \\
\text{\quad call execute(cats(’\%nrstr,’,_Stmtnt,’));} \\
\text{\quad return;}
\end{align*}
\]

**Label:** CxStmtnt: is a label statement. The statement link CxStmtnt branches to and returns from this data step subroutine.

\[
\begin{align*}
\text{link CxStmtnt;}
\end{align*}
\]

**Testing:** The variable Testing writes messages to the log when included statments are echoed to log with the option source2. It is assigned a value on p. 8.

\[
\begin{align*}
\text{retain } _\text{Testing} \%\text{eval}(0 \text{ or } %\text{sysfunc(getoption(Source2)) eq SOURCE2});
\end{align*}
\]


**Call Execute:** The necessity of using function nrstr with call execute is explained by Fehd and Carpenter [10] sgf2007.113.
List processing routine CallXinc encapsulates the process of converting a data set's variable values into global variables and calling a processing subroutine.

This method replaces the creation of arrays of macro variables and the macro do loop needed to process the elements of the macro array.
Usage Example

Overview
This section shows a testing program and log.

Module for Testing
CallXinc-Test.sas

```sas
* name: CallXinc-Test;
options source2 obs = 2;

%Let CxData = sashelp.class;
%Let CxInclude = SiteIncl(PutGlobal);
%Include SiteIncl(CallXInc);
```

Log when Testing
Options source2 turns on reporting in the data step subroutine CxStmtnt.

```sas
Stmtn=%let Name = Alfred;
Stmtn=%let Sex = M;
Stmtn=%Include SiteIncl(PutGlobal);
Stmtn=%let Name = Alice;
Stmtn=%let Sex = F;
Stmtn=%Include SiteIncl(PutGlobal);
Stmtn=%symdel Name;
Stmtn=%symdel Sex;
NOTE: There were 2 observations read from the data set SASHHELP.CLASS.
```

Log of Call Execute
Statements submitted from the routine.

```sas
NOTE: CALL EXECUTE generated line.
1 + %let Name = Alfred;
2 + %let Sex = M;
3 + %Include SiteIncl(PutGlobal);
NOTE: %INCLUDE (level 2) file SITEINCL(PutGlobal)
  is file C:\SAS-site\includes\putglobal.sas.
65 +%Put _global_
GLOBAL CXDATA sashelp.class
GLOBAL CXINCLUDE SiteIncl(PutGlobal)
GLOBAL NAME Alfred
GLOBAL SEX M
NOTE: %INCLUDE (level 2) ending.
NOTE: %INCLUDE (level 1) resuming.
4 + %let Name = Alice;
5 + %let Sex = F;
6 + %Include SiteIncl(PutGlobal);
```

End of Job
Delete the unknown number of global macro variables allocated by routine CallXinc.

```sas
7 + %symdel Name;
8 + %symdel Sex;
67 +%Put Note: CallXinc of &CxData. &CxInclude. ending;
Note: CallXinc of sashelp.class SiteIncl(PutGlobal) ending
NOTE: %INCLUDE (level 1) ending.
```
Usage Development

Overview
This section walks through the steps in converting several ad hoc programs into a subroutine and calling routine.

Development of a parameterized include program is an intermediate step in the development of reusable macros.

The process consists of these steps:
1. identify ad hoc programs with similar processing
2. compare programs and identify parameters
3. build model program with parameters
4. split into subroutine and calling program

Proc Print

Ad Hoc Programs
An over-simplified definition of an ad hoc program is one which processes a data set subset, where a subset is defined as either a set of columns, of rows, or the intersection of both columns and rows.

```sas
* name: adhoc-print-sashelp-class-sex-F.sas;
PROC Print data = SAShelp.Class
(where = (Sex eq 'F'));
title3 'SAShelp.Class.Sex eq F';

* name: adhoc-print-sashelp-class-sex-M.sas;
PROC Print data = SAShelp.Class
(where = (Sex eq "M"));
title3 "SAShelp.Class.Sex eq M";

* name: adhoc-print-sashelp-shoes-region-Africa.sas;
PROC Print data = SAShelp.Shoes
(where = (Region eq 'Africa'));
title3 'SAShelp.Shoes.Region eq Africa';
```

Identifying Parameters
The parameters of a reusable model are:

- data set name
- variable name
- value

Continued on next page.
In this intermediate step in the research and development of a subroutine, it is important to write a program that contains both the parameter allocations — macro variable assignment statements — and the references.

```sas
* name: RnD-print-0.sas;
PROC Print data = SAShelp.Class
   (where = (Sex eq 'F'));
run;
```

The intermediate program is split into two programs: the calling routine and the called subroutine.

**routine** : contains the parameter allocations and the first test case.

```sas
* name: print-data-value-Test-class;
options source2;* Testing;
%Let Data = SAShelp.Class;
%Let Var = Sex;
%Let Value = F;
%Include SiteIncl(print-data-value);
%Let Value = M;
%Include SiteIncl(print-data-value);
```

**subroutine** : contains the parameter references

```sas
* name: print-data-value;
PROC Print data = &Data.
   (where = (&Var. eq "&Value.");
   title3 "&Data..&Var. eq &Value." ;
run;
```

The next test case is another data set, variable and set of values.

```sas
* name: print-data-value-Test-shoes;
options source2;* Testing;
%Let Data = SAShelp.Shoes;
%Let Var = Region;
%Let Value = Africa;
%Include SiteIncl(print-data-value);
%Let Value = Asia;
%Include SiteIncl(print-data-value);
```
Making Lists with Proc Freq

Overview

This block shows examples of proc freq programs and their consolidation into a general subroutine and calling module.

```sas
* name: adhoc-freq-sashelp-class-sex.sas;
PROC Freq data = SAShelp.Class;
  tables Sex
     / list missing
  out = Work.Values;

* name: adhoc-freq-sashelp-shoes-region.sas;
PROC Freq data = SAShelp.Shoes;
  tables Region
     / list missing
  out = Work.Regions;

* name: freq-data-out.sas;
PROC Freq data = &Data;
  tables &Var.
     / list missing noprint
  out = Work.Values;
run;

* name: freq-data-out-Test-class;
options source2;* Testing;
%Let Data = SAShelp.Class;
%Let Var = Sex;
%Include SiteIncl(freq-data-out);
Proc Print data = &SysLast.;
title &SysLast.;
```
Making Lists with Proc Sort

Overview

This block shows examples of proc sort programs and their consolidation into a general subroutine and calling module.

```sas
* name: adhoc-sort-sashelp-class-sex.sas;
PROC Sort data = SASHelp.Class
   (keep = Sex)
   nodupkeys
   out = Work.Sort;
   by Sex;

* name: adhoc-sort-sashelp-shoes-region.sas;
PROC Sort data = SASHelp.Shoes
   (keep = Region)
   nodupkeys
   out = Work.Sort;
   by Region;

* name: sort-data-out.sas;
PROC Sort data = &Data.
   (keep = &Var.)
   nodupkeys
   out = Work.Values;
   by &Var.;
run;

* name: sort-data-out-Test-shoes;
options source2;* Testing;
%Let Data = SASHelp.Shoes;
%Let Var = Region;
%Include SiteIncl(sort-data-out);
Proc Print data = &SysLast.;
title3 &SysLast.;
```
Examples

Overview
This section review modules that use the subroutines shown above.

The process consists of these steps:

1. identify subroutine for processing
2. identify subroutine for making list of values
3. build model program with parameters

Processing

1 * name: print-data-value;
2 PROC Print data = &Data.
3 (where = (&Var. eq "&Value.");
4 title3 "&Data..&Var. eq &Value." ;
5 run;

Making Lists

Use either of proc freq or sort.

1 * name: freq-data-out.sas;
2 PROC Freq data = &Data;
3 tables &Var.
4 / list missing noprint
5 out = Work.Values;
6 run;

1 * name: sort-data-out.sas;
2 PROC Sort data = &Data.
3 (keep = &Var.)
4 nodupkeys
5 out = Work.Values;
6 by &Var.;
7 run;

Continued on next page.
This module uses proc freq to make the list.

* name: print-data-class-sexs;
  options source2;* Testing;

* set parameters for subroutines;
%Let Data = SAShelp.Class;
%Let Var = Sex;

* make list;
%Include SiteIncl(freq-data-out);

* process list;
%Let CxData = Work.Values(rename = (&Var. = Value));
%Let CxInclude = Project(print-data-value);
%Include SiteIncl(CallXinc);

* delete global macro variables;
%symdel Data Var Freq_Out CxData CxInclude;

This module uses proc sort to make the list.

* name: print-data-class-sexs;
  options source2;* Testing;

* set parameters for subroutines;
%Let Data = SAShelp.Shoes;
%Let Var = Region;

* make list;
%Include SiteIncl(sort-data-out);

* process list;
%Let CxData = Work.Values(rename = (&Var. = Value));
%Let CxInclude = Project(print-data-value);
%Include SiteIncl(CallXinc);

* delete global macro variables;
%symdel Data Var Freq_Out CxData CxInclude;
Conclusion

Summary

Call Execute This article shows that call execute is a powerful method for list processing.

%Includes Doing list processing with call execute of %includes can eliminate the use of macros for processing arrays of macro variables. This yields clearer code.

Suggested Reading


Call Execute and %nrstr List Processing Basics, Fehd and Carpenter [10] sgf2007.113 demonstrate the timing of the error of using call execute of macros without the macro function %nrstr.

SmryEachVar A Data Review Suite for Each Data Set in a Libref, Fehd [9] sgf2008.003


References


To get the code examples shown here
search : www.sascommunity.org
for : Call Execute Parameterized Include

Author: Ronald J. Fehd
Centers for Disease Control
4770 Buford Hwy NE
Atlanta GA 30341-3724

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: 20+ years
data manager at CDC, using SAS: 18+ years
author: 20+ SUG papers

SAS-L: author: 5,000+ messages to SAS-L since 1997

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. In the USA and other countries ® indicates USA registration.