Paper 054-2009

Using Functions SYSFUNC and IFC to Conditionally Execute Statements in Open Code

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

ABSTRACT

Description : This paper explains how to conditionally execute statements in open code without having to wrap the code in a macro. This is accomplished by combining the macro function `sysfunc` with the data step function `ifc`. The result is that parameterized include programs gain the power of conditional processing — `%if ... %then` — for which macros are usually used.

Purpose : replacement for macro if statement

Audience : advanced users and macro programmers.

Keywords : assertions, function, ifc, nrstr, parameterized include program, sysfunc

Information : assertions, conditional processing, parameterized include programs

In this paper This paper contains the following topics.

Introduction

Overview

Program development proceeds through these stages:

- create ad hoc programs
- identify programs with similar algorithms
- consolidate into parameterized include program
- where necessary, convert to macro for conditional processing

The techniques shown here eliminate the need for macros and replace conditional processing with inline statements.

Example: %if statement used to conditionally execute blocks of code.

```
%macro DoThis;
%if   <condition> %then %do;
      * statements for true;
      %end;
%else %do;
      * statements for false;
      %end;
run;%mend;
%DoThis
```

can be replaced with this:

```
%sysfunc(ifc(<condition>
            ,* statements for true;
            ,* statements for false;
            ))
```

Function IFC

Data Step Function IFC

The ifc function has four parameters:

1. a logical expression
2. character value returned when true
3. value returned when false
4. value returned when missing, which is optional

ifc-syntax.txt

```
IFC( logical-expression
   , value-returned-when-true
   , value-returned-when-false
  <, value-returned-when-missing>
   )
```

This example shows the function in a data step.

ifc-data.sas

```
DATA   Work.TestIfc;
attrib N     length = 4
       Value length = $ %length(missing);
do N = ., 0, 1, -1;
   Value = ifc(N
               ,'true'
               ,'false'
               ,'missing'
               );
   output;
   end;
stop;
run;

Proc Print data = Work.TestIfc;
```

ifc-data.lst

```
Obs    N    Value
---   --    -------
 1     .    missing
 2     0    false
 3     1    true
 4    -1    true
```

Note that minus one is true (or not(false)).

Function SysFunc

Macro Function SysFunc

The macro function `sysfunc` provides a method to access certain data step functions and the values of options.

SysFunc and GetOption

This example shows the combination of `sysfunc` and `getoption` writing a note to the log with the value of the option `source2`.

put-sysfunc-getoption.log

```
1  options nosource2;
2  %Put option source2 is %sysfunc(getoption(source2));
option source2 is NOSOURCE2
3  options source2;
4  %Put option source2 is %sysfunc(getoption(source2));
option source2 is SOURCE2
```

SysFunc and IFC

Macro Statement PUT with IFC

This example contains the previous usage of `sysfunc` and `getoption`. Its purpose is to show that the evaluation of the logical expression can produce conditional text.

put-sysfunc-ifc.log

```
1    options nosource2;
2    %Put option source2 is
3    %sysfunc(ifc(%sysfunc(getoption(source2)) eq SOURCE2
4                ,True
5                ,False
6                ,Missing
7            )   );
option source2 is False
```

Note that the `sysfunc(ifc(...))` is within the `%put` statement whose ending semicolon is on log line 7.

IFC producing PUT

The promise of `sysfunc(ifc(...))` is to generate statements, based on the evaluation. This example shows the first test: a complete `%put` statement as the result of the evaluation.

The Error

sysfunc-ifc-error.log

```
1       options nosource2;
2       %sysfunc(ifc(%sysfunc(getoption(source2)) eq SOURCE2
3               ,%Put True;
True
4               ,%Put False;
False
5               ,%Put Missing;
Missing
6               ))
```

What happened here? All the macro `%put` statements were executed!

Nrstr of PUT

This problem is the same as with `call execute`, which is explained by Fehd and Carpenter [4], sgf2007.113.The statements must be wrapped in macro function `nrstr` (No Rescan String) which delays their execution until after the `ifc` condition has been evaluated.

sysfunc-ifc-nrstr.log

```
1  options nosource2;
2  %sysfunc(ifc(%sysfunc(getoption(source2)) eq SOURCE2
3               ,%nrstr(%Put True;)
4               ,%nrstr(%Put False;)
5               ,%nrstr(%Put Missing;)
6           )   )
7  False
```

Note that the `sysfunc(ifc(...))` is not a statement; the function calls are completed by the two closing parentheses on log line 6.

Usage Examples

Overview

These are common tests:

- boolean: zero, one or greater than one
- comparison of two values
- existence

Boolean

Many functions return either the set of boolean values — (0,1) — or zero and a negative or positive integer which evaluates as true. Remember that true is not false, see p. 3.

0,1 : `%sysfunc(cexist(&Catalog.))`

0, ge 1 : `&Nobs.`

-1,0, ge 1 : `%sysfunc(attrn(&dsid.,nobs))`

Comparison

Any logical expression can be used for the first argument of the `ifc` function.

- `&NobsData1. eq &NobsData2.`
- `&Nobs. gt 100`
- `%sysfunc(fileref(&FileRef.)) eq 0`
- `%upcase(%substr(&Type.,1,1)) eq C or &Type. eq 2`

There are pitfalls when comparing user-supplied values; refer to Fehd [2], sas-wiki.Cond-Exec-Global-Stmnts for examples and fixes.

Test Suite

Each of the following programs has been tested as a parameterized include file, illustrated by this example:

assert-cexist-catalog-Test.sas

```
1  options source2;
2  Proc Format library = Work; value Test 0='zero';run;
3  %Let Catalog = Work.Formats;
4  %Include Project(assert-cexist-catalog);
5  %Let Catalog = Library.Formats;
6  %Include Project(assert-cexist-catalog);
```

Both the program `*.sas` and its test `*-Test.sas` are available in a .zip at Fehd [2], sas-wiki.Cond-Exec-Global-Stmnts

Calling Routines

Branching with Includes

Evaluation of the logical expression can be used to branch to the execution of either of two programs.

cond-inc-which.sas

```
* cond-inc-which;
options source2;
%Let Data = sashelp.Class;
%*Let Data = sashelp.ClassX;* test not exist;
%let dsid = %sysfunc(open(&Data. ));
%let Nobs =
%sysfunc(ifc(&DsId.
            ,%nrstr(            %sysfunc(attrn(&dsid.,nobs));
                    %let rc   = %sysfunc(close(&dsid.)); )
            ,%nrstr(0) ));
%sysfunc(ifc(&Nobs.
            ,%nrstr(%Include Project(cond-inc-1);)
            ,%nrstr(%Include Project(cond-inc-0);) ))
```

cond-inc-1.sas

```
%Put Note2: file is cond-inc-1;
```

cond-inc-0.sas

```
%Put Note2: file is cond-inc-0;
```

Procedure By Type

Evaluation of the logical expression can be used to branch to the execution of either of two sets of statements.

The elaborate test of the value of Type ensures that this code can be called with a list processing data set created by either Proc Contents whose Type are in (1,2) and Proc SQL whose Type are in ('char','num').

proc-type.sas

```
title3 "proc-type of &Libname..&Memname.;";
%sysfunc(ifc(   %upcase(%substr(&Type.,1,1)) eq C
             or                  &Type        eq 2
            ,%nrstr(Proc Freq  data   = &Libname..&Memname.;
                               tables   &Name.;)
            ,%nrstr(Proc Means data   = &Libname..&Memname.;
                               var      &Name.;) ))
run;
```

Calling Proc-Type

This code is similar to the conditional processing in the SmryEachVar data review suite Fehd [3], sgf2008.003.

proc-type-Test.sas

```
options source2;
%Let Libname = sashelp;
%Let Memname = Class;
%Let Name    = Sex;
%Let Type    = C;
%Include Project(proc-type);

%Let Name    = Height;
%Let Type    = 1;
%Include Project(proc-type);
```

proc-type-caller-SQL.sas

```
options source2;
%Let Libname = sashelp;
%Let Memname = Class;
Proc SQL; select catt('%let Name =',Name,';'
                ,      '%let Type =',Type,';'
                ,'%Include Project(proc-type)')
          into :List separated by ';'
```

```
          from Dictionary.Columns
          where Libname eq "%upcase(&Libname.)"
            and Memname eq "%upcase(&Memname.)"
            and Memtype eq 'DATA';
          quit;
&List.;
```

proc-type-caller-Contents.sas

```
options source2;
%Let Libname = sashelp;
%Let Memname = Class;
Proc Contents data = &Libname..&Memname.
                     noprint
               out  = Work.Contents;
DATA   _Null_;
attrib Stmnt length = $72;
do until(EndoFile);
   set Work.Contents end = EndoFile;
   Stmnt = catt('%Let Name =',Name,';');
   link ExecStmt;
   Stmnt = catt('%Let Type =',Type,';');
   link ExecStmt;
   Stmnt = '%Include Project(proc-type);';
   link ExecStmt;
   end;
return;
ExecStmt: call execute(catt('%nrstr(',Stmnt,')'));
return;
stop;
run;
```

Number of Rows

The `attrn` function can be used to get the number of observations of a data set. The data set must be opened before the function can return the value. If the data set exists then the function can be applied; otherwise when the open of the data set fails, it is not necessary to close the data set.

The values returned by `attrn(...,nobs)` are in:

-1 : for a view

0 : empty

ge 1 : number of rows

assert-nobs.sas

```
%let dsid = %sysfunc(open(&Data. ));
%let Nobs =
%sysfunc(ifc(&DsId.
            ,%nrstr(          %sysfunc(attrn(&dsid.,nobs));
                    %let rc = %sysfunc(close(&dsid.)); )
            ,%nrstr(0) ));
%sysfunc(ifc(&Nobs.
            ,%nrstr(%Put Note2: Data &Data. Nobs: &Nobs.;)
            ,%nrstr(%Put %sysfunc(sysmsg());
                    endSAS;) ))
```

As shown above on p. 3 minus one evaluates as true.

```
21          DATA Work.ClassView / view = Work.ClassView;
22          set sashelp.class(obs=1);
23          run;

NOTE: DATA STEP view saved on file WORK.CLASSVIEW.
```

```
24          %Let Data = Work.ClassView;
25          %Include Project(assert-nobs);
Note2: Data Work.ClassView available Nobs: -1
```

Assertions

Concept

Assertions are tests at the beginning of a program that determine whether the program input parameters are valid. The assertion of existence follows the rules of simple boolean evaulation: the functions return zero or one.

Example of Usage

```
* name: MyProgram;
* confirm input exists;
%let Object = ObjectName;
%Include Asserts(ExistObject);* false==endSAS;
*processing;
```

Exist Catalog

The `cexist` function can be used to test the existence of a catalog.

assert-cexist-catalog.sas

```
1 %sysfunc(ifc(%sysfunc(cexist(&Catalog.))
2             ,%nrstr(%Put Note2: Catalog &Catalog. exists;)
3             ,%nrstr(%Put Note3: not exist &Catalog.;
4                     endSAS;) ))
```

Exist Data

The `exist` function can be used to test whether a data set exists.

assert-exist-data.sas

```
1 %sysfunc(ifc(%sysfunc(exist(&Data.))
2             ,%nrstr(%Put Note2: Data &Data. exists;)
3             ,%nrstr(%Put Note3: not exist &Data.;
4                     endSAS;) ))
```

Exist Fileref

The `fexist` function can be used to test the existence of a fileref which is a directory-specification or folder.

assert-fexist-fileref.sas

```
1 %sysfunc(ifc(%sysfunc(fexist(&FileRef.))
2             ,%nrstr(%Put Note2: FileRef &FileRef. exists;)
3             ,%nrstr(%Put Note3: not exist &FileRef.;
4                     endSAS;) ))
```

Exist File

The `fileexist` function can be used to test the existence of a file-specification.

assert-fileexist-filename.sas

```
1 %sysfunc(ifc(%sysfunc(fileexist(&FileNameExt.))
2             ,%nrstr(%Put Note2: FileNameExt &FileNameExt. exists;)
3             ,%nrstr(%Put Note3: not exist &FileNameExt.;
4                     endSAS;) ))
```

Exist Fileref and File

The `fileref` function is a special case. Instead of returning one as success and zero as failure, it returns this set of values:

-1 : fileref exists but file associated with the fileref does not exist

0 : fileref and external file both exist

+1 : fileref not assigned

Thus success is defined as `eq 0`!

assert-fileref.sas

```
%sysfunc(ifc(%sysfunc(fileref(&FileRef.)) eq 0
            ,%nrstr(%Put Note2: FileRef &FileRef. exists;)
            ,%nrstr(%Put Note3: not exist &FileRef.;
                    endSAS;) ))
```

For an example of usage of function `fileref` see the OnLine Doc for function `finfo`.

Summary of Assertion Usage

Each of these examples show the failure as sufficient to terminate the program — `endSAS;` In practice consider adding other programs which provide information written to a log or data set. Nelson et al. [6], sesug2003.001 discuss a system of event management where subroutines log program completion and e-mail notifications of failure are sent to interested users.

Conclusion

The combination of functions `sysfunc` and `ifc` can be use to eliminate the need for macros where only conditional processing is needed.

Further Reading

Assertions Nelson et al. [6], sesug2003.001 describe assertions in an event management and notification system.

Call Execute and Nrstr Fehd and Carpenter [4], sgf2007.113 demonstrate the problem of using call execute to generate macro calls and how using function nrstr solves that problem.

Conditional Execution of Includes Fehd [1], sugi25.038 provides a trick for conditional execution of included programs.

Macro IFF Henderson [5], saspress.60282 provides a macro to get around the problem of user-supplied values in comparisons.

Using Parameterized Includes Fehd [3], sgf2008.003provides a data review suite of 80 programs with modules, routines and subroutines of parameterized includes.

References

[1] Ronald Fehd. The Writing for Reading SAS Style Sheet: Tricks, traps, tips, and templates, from SAS-L's Macro Maven. In *Proceedings of the 25th SAS User Group International Conference*, 2000. URL `http://www2.sas.com/proceedings/sugi25/25/ad/25p038.pdf`. Coders Corner, 4 pp.; topics: avoiding common mistakes when writing macro statements, recommendations; info: conditional execution of includes.

[2] Ronald J. Fehd. Conditionally executing global statements. In *SAS Community Wikipedia*, 2008. URL `http://www.sascommunity.org/wiki/Conditionally_Executing_Global_Statements`. topics: combining functions sysfunc and ifc; info: example assertions, caveats on logical comparisons.

[3] Ronald J. Fehd. SmryEachVar: A data-review routine for all data sets in a libref. In *Proceedings of the SAS Global Forum Annual Conference*, 2008. URL `http://www2.sas.com/proceedings/forum2008/003-2008.pdf`. Applications Development, 24 pp.; topics: data review; info: utilities to repair missing elements in data structure.

[4] Ronald J. Fehd and Art Carpenter. List processing basics: Creating and using lists of macro variables. In *Proceedings of the SAS Global Forum Annual Conference*, 2007. URL http://www2.sas.com/proceedings/forum2007/113-2007.pdf. Hands On Workshop, 20 pp. including examples; topics: arrays of macro variables; info: using call execute and nrstr to call macros, testing.

[5] Don Henderson. *Building Web Applications with SAS/IntrNet: A Guide to the Application Dispatcher*. SAS Institute, 2006. URL http://www.sas.com/apps/pubscat/bookdetails.jsp?catid=1&pc=60282. ISBN 978-1-59994-189-9.

[6] Greg Barnes Nelson, Danny Grasse, and Deborah Pine. Automated testing and real-time event management. In *Proceedings of the SouthEast SAS User Group Conference*, 2003. URL http://www.thotwave.com/Document/papers/SESUG_03/EventSys/eventmngmt.pdf. 6 pp. topics: testing, definition of events, batch scheduler, program agent, execution monitor, real-time notification.

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

The code examples in this paper are available on the web at Fehd [2], saswiki.Cond-Exec-Global-Stmnts

	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: 12+ SUG papers	
SAS-L:	author: 4,000+ messages to SAS-L since	1997
	Most Valuable SAS-L contributor:	2001, 2003