

More Fun with %SYSFUNC()

Paul Hamilton
Fred Hutchinson Cancer Research Center
HIVNET/SC

Abstract

As the SAS® System has evolved, new functionality has been added to many components of the system. While new statements have been added to the DATA step, the introduction and evolution of Screen Control Language has far exceeded the design assumptions of the original language. For the last several years, developers of interactive applications have had access to numerous functions difficult to implement in the Base SAS.

Starting with version 6.09E and 6.11, a new set of macro functions were introduced which allow Base SAS to tap some of this rich array of functions. The tutorial will examine some new macro debugging tools, and examine some examples of how the new %SYSFUNC macros might be used in the Data Step. SAS application developers who have not yet utilized any features of Screen Control Language might be particularly interested in this tutorial.

Scope

This paper is a companion piece with another presented at this conference: "Fun with %SYSFUNC()", by Pete Lund. As the previous paper presents the central syntax and usage of %SYSFUNC macros, this paper will focus on the new functionality similar to that enjoyed by Screen Control Language (SCL) developers.

The DATA step has been the backbone of the SAS System for decades. Myriad papers regarding its structure, functionality, and optimization have been presented at local, regional, and international conferences. SAS Institute developers have continually added new statements, functions, and formats to the DATA step. However, when the Application Facility introduced Screen Control Language with version 6.03, a new paradigm and set of functions became available to developers of interactive applications. Containing many of the DATA step functions, SCL also extended into exciting new areas. However, those functions were unavailable to the developer working only with Base SAS.

Opening a Data Set

A fundamental concept in SCL is that before a program can read from, add to, or manipulate an object such as a SAS data set, directory, or file, it first must be identified. This is performed by opening the object with a function

call. In the case of a SAS data set, this is accomplished in %SYSFUNC macros by using the corresponding *open* and *close* functions. The *open* function call returns a numeric value, which is captured and utilized by the programmer in many subsequent function calls. The following simple example executes in open macro code:

```
%let dsid = %sysfunc(open(pnwsvg.example1,i));  
%let dsname = %sysfunc(dsname(&dsid));  
%put The data set opened was &dsname;  
%let rc = %sysfunc(close(&dsid));
```

which results in the following string being written to the SASLOG:

```
The data set opened was PNWSUG.EXAMPLE1.DATA
```

Note the importance of always closing open data sets after processing has completed. Also, it is highly desirable to perform error checking¹ before assuming the *open* function was successful. This necessitates the creation of a macro, since open macro code cannot handle %IF...%THEN logic.

Using the technique started above, the macro ODS (Open Data Set) attempts to open a SAS data set; if it exists, the macro extracts the name of the data set using the *DSNAME* function, writes the name to the SASLOG, and closes the data set. If the data set is not found, the macro retrieves the resulting error number and error message through the use of the *SysRc* and *SysMsg* functions.

```
%macro ods(dsname);  
%let dsid = %sysfunc(open(&dsname,i));  
%if &dsid le 0 %then %put %sysfunc(sysrc()) --  
%sysfunc(sysmsg());  
%else %do;  
%let dsname = %sysfunc(dsname(&dsid));  
%put The data set opened was &dsname;  
%let rc = %sysfunc(close(&dsid));  
%end;  
%mend ods;  
  
%ods(pnwsvg.example); * Note incorrect name;
```

Invoking the macro above with the data set name misspelled results in the following message in the SASLOG:

```
70009 -- ERROR: File PNWSUG.EXAMPLE1.DATA does not  
exist
```

Coding one to many extra statements in order to verify the results of an attempted action is a fundamental characteristic of SCL (and now %SYSFUNC) coding. Many

¹ For further information, see "Fun with %SYSFUNC", Peter Lund, *Proceedings* this conference.

interactive SAS applications spend as much time on checking data and user responses as performing actual project work.

Note the use, definition, and repeated reference to `&dsid`, the data set id. This numeric value will be used through SCL and %SYSFUNC code to refer to previously opened data sets. `Dsid` is commonly used as the data set identifier throughout SAS documentation and examples, but any valid SAS name may be utilized.

Retrieving Data Into a Title

If data from a SAS data set needs to be placed into a title, a `DATA _NULL_;` step could retrieve the relevant information, call `SYMPUT` to create a global macro, then reference the macro variable in a `TITLE` statement. %SYSFUNC macros permit this action without the overhead of a `DATA` step. The following macro will open a SAS data set, read through it searching for a particular city name, then retrieve a numeric livability index assigned to that observation. The result is a macro variable which may be used in `TITLE` statements or any programming throughout the job.

1. Begin the macro definition. The macro has one parameter, city (name).
2. Define the macro variable `index` to have global scope, since it will be referenced outside of the current macro environment.
3. Use the %SYSFUNC macro `OPEN` to open the data set in input mode.
4. Use the `ATTRN` function (covered in detail in the previous paper) to retrieve the number of observations in the data set. This allows a macro %DO loop to cycle through the data set, retrieving one observation at a time.

```
%macro ret(city);
  %global index;
  %let dsid=%sysfunc(open(pnwsvg.example4,i));
  %let nobs=%sysfunc(attrn(&dsid,nobs));
```

5. Begin a %DO loop to process all observations. The `FETCH` function retrieves the next observation from the data set, somewhat like the `SET` statement in the `DATA` step. However, (as used here) it does not load any values from the observation into memory (unlike the `SET` statement).
6. Extract the value of `city` from the data set using the `GETVARC` function. The function needs to know which data set and which variable in the data set to retrieve. The data set is identified by `&dsid`, while the variable number is extracted by nesting the `VARNUM` function.

7. If the city just retrieved is the same as the one specified in the macro invocation, proceed as in step 7 to retrieve the variable `index`. The only difference here is the use of `GETVARN`, since `index` is a numeric not a character variable.
8. As always, close data sets once processing has completed.

```
%do i=1 %to &nobs;
  %let rc=%sysfunc(fetch(&dsid));
  %let thiscity=%sysfunc(getvarc(&dsid,
    %sysfunc(varnum(&dsid,city))));
  %if &thiscity = &city %then
    %let index = %sysfunc(getvarn(&dsid,
      %sysfunc(varnum(&dsid,index))));
  %end;
  %let rc = %sysfunc(close(&dsid));
%mend ret;
```

Once the macro has been defined, invoke it with the name of the desired city, and use the resulting value of `&index` as returned by the macro. In production work, additional code would be written to handle the case where the desired city was not found in the input data set.

```
%ret(ISSAQUAH);
proc print data = pnwsvg.example4 nobs;
  title1 "Issaquah enjoys Quality of &index";
run;
```

Issaquah has a livability index of 13.6

CITY	INDEX
LOS MESAS	9.2
LE GRAND	11.6
HOUSTON	1.2
ISSAQUAH	13.6
SEATTLE	12.5
BEDFORD	12.1

Yet Another View of Contents

The SAS developer already has several methods available for obtaining information about the contents of a SAS data set: `PROC CONTENTS`, `PROC DATASETS`, the `ACCESS` window, the `VAR` window, and the `SASHELP.VCOLUMN` view. The %SYSFUNC macros offer yet another way to get at this information. The previous paper showed how to use these functions to obtain data set level information. The next example will show how to obtain variable level information as well.

1. The `STMT` option on the macro invocation means that the macro can be called from the command line of any Display Manager window.
2. The `EXIST` function is then called to determine if the requested data set exists; if not, line 16 reports that condition and the macro ends.
3. If the data set does exist, line 3 uses the `OPEN` function to open the data set in `input` mode.

4. Use the **ATTRN** function to determine how many variables **&dsn** contains. Have a **%Do...%END** loop execute once for each variable in the data set.
5. In line 6, the variable name of the current *ith* variable is extracted using **VARNAME**.
6. The *varname* just extracted is then used to determine the variable number (*varnum*) on the data set, using **VARNUM**. SAS developers usually do not concern themselves with the variable number in a data set, but the **%SYSFUNC** functions depend on it heavily.
7. Lines 8-11 then use the *varnum* to extract the variable's label, type (expressed as C or N), format, and informat.
8. Finally, the **%PUT** writes out the information collected in a concise report. Naturally this layout could be altered to suit individual preference.

```
%macro getinfo(dsn) / stmt;
%if %sysfunc(exist(&dsn)) %then %do;
  %let dsid = %sysfunc(open(&dsn, 1));
  %let numvars = %sysfunc(attrn(&dsid,NVARS));
  %do i = 1 %to &numvars;
    %let varname=%sysfunc(varname(&dsid,&i));
    %let varnum=%sysfunc(varnum(&dsid,&varname));
    %let varlabel=%sysfunc(varlabel(&dsid,
                                   &varnum));
    %let vartype=%sysfunc(vartype(&dsid,
                                   &varnum));
    %let varfmt=%sysfunc(varfmt(&dsid,&varnum));
    %let varinfmt=%sysfunc(varinfmt(&dsid,
                                   &varnum));
    %put Varname= &varname (&varlabel)
          &vartype fmt &varfmt infmt &varinfmt;
  %end;
  %let rc = %sysfunc(close(&dsid));
%end;
%else %put &dsn does not exist;
%mend getinfo;
```

Invoking this macro from a command line:

```
%getinfo(pnwsvg.example1)
```

results in the following information being written to the SASLOG:

```
Varname=NAME (Name of person) C fmt $8. infmt $8.
Varname=HIREDATE (Hire Date) N fmt DATE9. infmt
MMDDYY8.
Varname= GENDER (Gender) C fmt $GENDER. infmt $1.
```

Invoking the macro with a nonexistent data set results in:

```
%getinfo(pnwsvg.example)
pnwsvg.example does not exist
```

Directory Information

The **%SYSFUNC** macros permit opening an reading a directory. What information is available to be extracted varies from platform to platform. Because of this fact, these and other, related functions considered below are written so as to be very general in their scope. In general, the following steps are performed:

1. Make the object known to the SAS system. Depending on the nature of the object, this might be performed through a **FILENAME** or **LIBNAME** statement or a corresponding **%SYSFUNC** function call.
2. Open the object of interest (directory, file, data set). Obtain a numeric id used to reference the object in subsequent function calls.
3. Use a function to determine how many options exist under the current platform.
4. Run a **%Do ... %To...** loop over the set of options. Extract the option name and option value with appropriate function calls.

In the example below, the steps outline just above are followed closely:

1. Define the macro name, *d*. There are no macro parameters necessary for this example.
2. Assign a fileref of *thedir* to point at the current directory (abbreviated as '.').
3. Open the directory using the function **DOPEN**. This returns a numeric value assigned macro variable *did* (Directory ID).
4. Using *did*, call the function **DOPTNUM** to determine how many informational items are available.
5. Run a macro **%Do** loop from 1 to the number of options available. All work occurs inside this loop.
6. Use the function **DOPTNAME** to extract the name (title) of the information being retrieved. This is stored in macro variable **&OptName**.
7. Using both **&did** and **&OptName**, call **DINFO** to extract the value of the information.
8. Use **%PUT** to write out the information collected.

```
%macro d;
  %let rc = %sysfunc(filename(thedir, .));
  %let did = %sysfunc(dopen(&thedir));
  %let optnum = %sysfunc(doptnum(&did));
  %do i = 1 %to &optnum;
    %let optname = %sysfunc(doptname(&did,&i));
    %let optvalue = %sysfunc(dinfo
                              (&did,&optname));
    %put Option # &i (&optname) is &optvalue;
  %end;
  %let dnum = %sysfunc(dnum(&did));
  %let rc = %sysfunc(dclose(&did));
%mend d;
&d;
```

Invoking the macro on a UNIX (Sun Solaris) platform results in the following information being written to the SASLOG:

```
Option # 1 (Directory Name) is
/home/hamilton
Option # 2 (Owner Name) is hamilton
Option # 3 (Group Name) is staff
Option # 4 (Access Permission) is rwx-----
```

When executed on the MacOS, the result is

```
Option # 1 (Directory Name) is MPFC:PNWSUG
```

Changing the focus of the macro slightly, we can read the actual list of files from the directory:

```
%macro d;
  %let rc = %sysfunc(filename (thedir,
MPFC:PNWSUG));
  %let did = %sysfunc(dopen(&thedir));
  %let dnum = %sysfunc(dnum(&did));
  %do j = 1 %to &dnum;
    %let fname=%sysfunc(dread(&did, &j));
    %put File # &j is &fname;
  %end;
  %let rc = %sysfunc(dclose(&did));
%mend d;
&d;
```

Partial output from execution on MacOS:

```
File # 1 is 16May schedule
File # 2 is 1997 Conference
File # 3 is 1998 Conference
File # 4 is agendaq.doc
File # 5 is Backup-PNWSUG
File # 6 is Cafe
...
File # 26 is regist.sas
File # 27 is resumes
```

File Information

In a similar set of functions as the directory example above, %SYSFUNC macro functions also permit extracting information about an individual file.

1. Assign a fileref to an individual file using the **FILENAME** function.
2. Obtain a file identifier, *fid*, by using **FOPEN** to open the file.
3. Use the function **FOPTNUM** to retrieve the number of options available for use.
4. Start a **%DO...%TO** loop to process all of the available options (captured in *&optnum* here).
5. Use the **FOPTNAME** function to retrieve the type of information, and **FINFO** to capture the actual information.
6. Write out the information captured using a standard **%PUT** statement.
7. Consistent with good coding practices, always close (**FCLOSE**) what you open.

```
%macro f;
  %let rc = %sysfunc(filename(thefile,
MPFC:PNWSUG:OT112.txt));
  %let fid = %sysfunc(fopen(&thefile));
  %let optnum = %sysfunc(foptnum(&fid));
  %do i = 1 %to &optnum;
    %let optname = %sysfunc(foptname(&fid,
&i));
    %let optvalue = %sysfunc(finfo(&fid,
&optname));
    %put Option # &i (&optname) is
&optvalue;
  %end;
  %let rc = %sysfunc(fclose(&fid));
%mend f;
```

Using techniques similar to those detailed above, one can also open an individual file and read its contents. As one might expect, this does become a little more complicated.

Other Useful Functions

Historically, the macro language has not support non-integer arithmetic expressions. Adding the values 1 + 2 with the **SYSEVAL** function properly returned 3, but 1.0 + 2.0 would return an error. If the macro programmer needs to evaluation floating point numbers, the new **SYSEVALF** supports this ability.

```
%macro addint(num1, num2);
  %let result = %eval(&num1 + &num2);
  %put &num1 + &num2 is &result;
%mend;

%addint(1, 2);
1 + 2 is 3

%addint(1.0, 2.0);
ERROR: A character operand was found in the %EVAL
function or %IF condition
       where a numeric operand is required. The
condition was: 1.0 + 2.0
ERROR: The macro will stop executing.
```

A new function, **SYSEVALF**, enables non-integer expressions to be evaluated correctly during macro execution:

```
%macro addflt(num1, num2);
  %let result = %sysevalf(&num1 + &num2);
  %put &num1 + &num2 is &result;
%mend;

%addflt(1, 2);
1 + 2 is 3

%addflt(1.1, 2.1);
1.1 + 2.1 is 3.2
```

Shortly prior to publication of this paper, Mike Rhoads contributed an interesting illustration of a %SYSFUNC function to the SAS-L:

```
%put Base date for SAS software is
%left(%sysfunc(putn('01jan1960'd, best.)));
```

resulting in

```
Base date for SAS software is 0
```

He further illustrated the computational power of macros:

```
%let begdate = '01Jan1995'd;  
%let enddate = '31Dec1995'd;  
%put Number of days from BEGDATE to ENDDATE  
is %left(%sysfunc(putn(&enddate-&begdate+1,  
best.)));
```

resulting in:

```
Number of days from BEGDATE to ENDDATE is  
365
```

Saving and Restoring System State

When developing interactions, care should be taken to ensure the application is "well behaved". Should the application need to set SAS system options to particular values in order to function properly, it should first record and save the current values, set the options as needed for itself, then restore the system to the original state.

First, create a macro which will store the current value of a system option as a global macro variable. In the example macro *keepset* shown below, two macro parameters are defined:

- *optname* is the name of the system option to be saved. This must be a legal SAS option name.
- *oldval* is the name of the macro variable which will store the current setting of the option. This permits re-setting the value in a later step.

```
%macro keepset(optname, oldval);  
  %global &oldval;  
  %let &oldval=%sysfunc(GetOption(&optname,  
KEYWORD));  
%mend keepset;
```

Line 1 ensures that the macro variable created by this code will have global scope within the SAS environment. Line 2 uses the **GETOPTION** function, along with the macro parameter *&OptName* and the function argument **KEYWORD**, indicating that the option being extracted is a SAS keyword. Other options for this second parameter to the **GETOPTION** function are **IN** and **CM**, appropriate if one is retrieving measurements in support of graphical applications.

To test this macro, set the value of **CENTER/NOCENTER** to the value *Center* and invoke the macro to save the value:

```
options center;  
%keepset(center, centset);
```

Now change the value to *NoCenter* and execute a **PROC PRINT** for testing purposes. As the title indicates, the output from this procedure should be left justified.

```
options nocenter;  
proc print data = pnwsug.example1;  
  title 'Should Be Left Justified';  
run;
```

Now restore the system option to the original setting, and execute a new procedure and title to observe the result:

```
options &centset;  
proc print data = pnwsug.example1;  
  title 'Should Be Center Justified';  
run;
```

This simple example could easily be extended to more complex tasks. Instead of storing the desired value in a global macro variable (which is lost when the SAS session terminates), the macro could store the value in a permanent SAS data set for later use at any point in the future.

Odds and Ends

While not a **%SYSFUNC** macro statement per se, a wonderful new macro enhancement is a set of new keywords which are supported on the **%PUT** statement. Aside from the usual set of constant text and macro variable, the following keywords are now supported:

1. **_ALL_**: writes out all macro variable information to the SASLOG. This includes the macro scope (local or global), macro names, and current values.
2. **_AUTOMATIC_**: as above, but provides information only on SAS System variables.
3. **_GLOBAL_**: as # 1, but only macro variables with global scope, and not system supplied macro variables.
4. **_LOCAL_**: only provides information on macro variables inside the current macro. This selection is an superb tool for debugging macros.
5. **_USER_**: combination of options 3 and 4, but not 2.

A really excellent set of advanced string parsing functions are now available both in the **DATA** step and in **%SYSFUNC** macro functions. See Paper # 72 from Coder's Corner in SUGI 22, written by Mike Rhoads.

Conclusion

Hopefully both tutorials from this conference have supplied the SAS developer with an extensive set of new functions. These functions are much more than small, incremental additions to the programmer's tool chest; they represent major advances in how one approaches a problem to be solved.

Some advances in the SAS System are incremental, beneficial but limited in scope. Others types of progress represent dramatic new tools (e.g. **PROC SQL**, Views, the

WHERE statement) which can change the way a programmer approaches and defines, and solves a problem. The new macro functions discussed in both %SYSFUNC tutorials represent such breakthrough advances. The new functions contain enough substance and richness to justify careful investigation and research.

Acknowledgments

SAS is a registered trademark of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.

References

Lund, Peter, "Fun with %SYSFUNC", *Proceedings of the Sixteenth Annual Pacific Northwest SAS Users Group Regional Conference*, 1997.

Polzin, Jeff and Susan O' Connor, "Macro Facility Enhancements for Release 6.09E and Release 6.11", *Proceedings of the Twenty-First Annual SAS Users Group International Conference*, Cary NC: SAS Institute Inc., 1996.

Mike Rhoads, *SAS Tip: Date constants and calculations*. Reprinted from SAS-L with permission from Mike Rhoads, Westat (RhoadsM1@Westat.com).

Mike Rhoads, "Some Practical Ways to Use the New SAS Pattern-Matching Functions", *Proceedings of the Twenty-Second Annual SAS Users Group International Conference*, Cary, NC: SAS Institute, Inc., 1997.

Macro Facility Enhancements for Release 6.09E and Release 6.11, Version 6, First Edition, Cary, NC: SAS Institute, Inc., 1996, 100 pp. Available in PDF format from www.sas.com.

SAS Institute Inc., *SAS Screen Control Language: Reference, Version 6, Second Edition*, Cary NC: SAS Institute Inc., 1995.

Author Contact Information

Paul Hamilton
Freed Hutchinson Cancer Research Center
HIVNET/SC
P.O. Box 19024 MP-900
Seattle, WA 98109
voice: (206) 667-7883
FAX: (206) 667-4812
e-mail: hamilton@hivnet.fhcrc.org