

Fun with %SYSFUNC()

Pete Lund
WA State Office of Financial Management
Forecasting Division

Abstract

The SAS® macro language has always been a powerful tool in the SAS programmers tool kit. One major limitation of the macro language has always been the relatively small number of macro functions.

SAS Version 6.12¹ introduces a new macro function, %SYSFUNC, which allows the use of almost all datastep, screen control language (SCL) and user-written² functions in the macro environment. It also allows formatting of macro variable values using SAS or user-written formats. This paper discusses the general syntax of %SYSFUNC as well as some examples of its use.³

It should also be noted that the SCL functions available to %SYSFUNC are now also available in the data step. The syntax and use of a number of these functions will also be discussed.

Introduction - The Syntax

The general syntax of %SYSFUNC is:

`%SYSFUNC(function(argument(s))<, format>)`

where *function* is the datastep or SCL function to execute, followed by its argument list. A SAS System or user-defined format can be applied to the resultant value.

Functions Available

¹ The %SYSFUNC function is available as an unsupported feature in SAS 6.11 TS040.

² Discussion of user-written functions are outside the scope of this paper. However, they are available to %SYSFUNC.

³ There is a related function %QSYSFUNC which should be used if any of the arguments to the embedded function or the returned value contain special characters which the macro processor will attempt to resolve. %QSYSFUNC masks the same characters as the %NRBQUOTE function. See Online Help or the SAS Guide to Macro Processing for more information.

Almost all of the base SAS datastep functions, as well as a number of SCL functions, are available to %SYSFUNC. There is a complete list of the new SAS 6.12 functions in the Online Help.⁴

There are a handful of functions not available to %SYSFUNC. These are:

DIF	INPUT	PUT
DIM	LAG	RESOLVE
HBOUND	LBOUND	SYMGET

The Basics

To put it simply, the purpose of %SYSFUNC is make functions and formats available to the macro processor. In the following simple example, the macro variable TheDate would be assigned the current system date (in SAS date format):

```
%let TheDate = %sysfunc(Date());
```

You can also add an appropriate format following the function call to change the value to the formatted value. If the above example had set TheDate = 13759, the following:

```
%let TheDate=%sysfunc(Date(),worddate18.);
```

would yield, September 2, 1997.

Changing the Look of a Macro Variable

What if I just want to format my macro variable value, if the PUT and INPUT functions are not available to %SYSFUNC? I'm sure you were asking that question as you read the list of unavailable functions above. Never

⁴ The online help for %SYSFUNC is not listed in the Help...Index. It can be found under Help...Contents...What's New in Release 6.12...What's New in Base SAS...Macro Features. It can also be located under Help...Find, searching on SYSFUNC.

fear, as the SCL functions PUTN, PUTC, INPUTN and INPUTC come to the rescue.

For example, assume a macro variable, &Salary, has a value 12343435. If you wanted to display it in dollar format, the following code, using the PUTN function to apply a numeric format:

```
%put %sysfunc(putn(&Salary,dollar12.));
```

would produce the following in the log:

```
$12,343,435
```

Character formats can also be applied using the PUTC function. Informats can be applied using the INPUTN and INPUTC functions.

There is a time...

Two of the more common automatic macro variables are &sysdate and &systime. These are often used in titles and footnotes to note run date and time. A recurring frustration with those variables is that they contain the date and time that the current SAS session was started, not the current date and time.

This was not as big a problem in the batch processing days of the past when a "session" only lived as long as the job. However, I can't believe I'm the only one who leaves an interactive PC SAS session running for days. It doesn't take long before these variables lose a bit of accuracy.

%SYSFUNC has solved that problem. Using the TIME() function, which returns the current system time, I can now replace the following footnote:

```
footnote "Start: &systime";
```

with

```
footnote "Start: %sysfunc(time(),timeampm.);";
```

Now, not only do I see the actual time that the program was run, but it can be formatted any way I like.

Take note, however, that using this new method in lieu of &sysdate and &systime may damage your reputation. I have a colleague who was praised for his timely work when he created a report which, according to the footnote, was created the day before it was requested. C'est le vie!

Go Ahead and Use Your Own

You are not limited to using SAS formats in %SYSFUNC. Any user-defined formats, created in PROC FORMAT, are supported as well. Let's suppose I created the following format to group income:

```
proc format;
  value income
    0-500000    = 'a minor leaguer'
    500001-2500000 = 'a utility infielder'
    2500001-10000000 = 'maybe a starter'
    10000001-high = 'probably Ken Griffey';
run;
```

Assuming the macro variable Salary has a value, 343435, the statements:

```
%put With a salary of
      %sysfunc(putn(&salary,dollar12.));
%put You are
      %sysfunc(putn(&salary,income.));
```

would produce the following log:

```
With a salary of      $343,435
You are a minor leaguer
```

Or, using the value from our earlier example (12343435):

```
With a salary of $12,343,435
You are probably Ken Griffey
```

Nesting %SYSFUNC Calls

In base SAS programming there are often times when it is necessary to nest function calls. Functions cannot be nested in %SYSFUNC, but %SYSFUNC calls can be nested. For example, two functions which are commonly nested are TRIM and LEFT. This forces a character value to be left justified and removes any trailing spaces. The following would result in an error:

```
%let trimmed = %sysfunc(trim(left(&myvar)));
```

However, nesting each function in a separate %SYSFUNC call is valid:

```
%let trimmed =
  %sysfunc(trim(%sysfunc(left(&myvar))));
```

There are numerous situations where nested functions can streamline your code. Let's look at another example, one which I use in a number of programs.

I have a macro variable, &ThisMnth, in yymm format (i.e., 9711). I need to create another variable, &LastMnth, which has a value of one month earlier (i.e., 9710). Obviously, you can't simply subtract 1, using the %eval function, because when I get to January (i.e., 9801) my last

month value will be slightly invalid (i.e., 9800). Now that I have all the datestep functions available I can simply convert the macro variable to a SAS date, with the INPUTN function, and decrement it one month with the INTNX function, and return the new value in YYMM format.

Let's look at each of those pieces separately. First, I want to convert &ThisMnth to a SAS date. By adding a day value to the end of &ThisMnth I can use the yymmdd6. informat to convert 971101 to a SAS date.

```
%let SD=%sysfunc(inputn(&Mnth.01,yymmdd6.))
```

&SD now equals 13819, the SAS date value for November 1, 1997. Now, I can use the INTNX function to back this date up one month. Also, remember that I can add a format to the %SYSFUNC call, so in one call the new value can be formatted in yymm format.⁵

```
%let LM=%sysfunc(intnx(month,&sd,-1),yymmdd4.);
```

&LM now has a value of 9710. The two %SYSFUNC calls can be nested to create the new variable with one statement:

```
%let LastMnth = %sysfunc(intnx(month,
    %sysfunc(inputn(&ThisMnth.01,yymmdd6.)),
    -1),yymmdd4.);
```

Put Your SAS to Sleep

There is one SAS data step function that is almost always coded by itself in a _NULL_ dataset. That is the SLEEP function.

```
data _null_;
    x = sleep(30);
run;
```

Code like the above, which would cause SAS to pause (sleep) for 30 seconds, is often used to delay execution while something external to your SAS program is initiated, such as starting Excel before establishing a DDE connection. There is nothing wrong with this code, it just adds a little clutter to the log. It can now be replaced with:

```
%let rc = %sysfunc(sleep(30));
```

⁵ Notice that the time interval argument (month) to INTNX is not quoted as it would be if used in a data step. I have found a few places where arguments are not passed exactly in %SYSFUNC as in data step code. As far as I know, they are not documented and it is a trial and error process to work around them.

The function will return the argument value, 30 in this case, so you need to use a macro assignment statement. Actually, try this at home for fun:

```
title "We Sat for %sysfunc(sleep(5)) Seconds";
```

What do you think will happen?

Are You There? Checking the Existence of a File

It is very often necessary to know whether an external file, dataset or catalog exists while your program is running. SAS provides many tools to obtain this information: PROC DATASETS, PROC CONTENTS, DICTIONARY tables and SASHELP views. All provide useful information but, like the SLEEP function in the _NULL_ dataset, require running a data step or PROC.

There are a number of SCL functions available to %SYSFUNC which can provide file information without having to run a procedure. There are two functions which can be used to check for the existence of an external file: FILEEXIST and FEXIST. The FILEEXIST function accepts as its argument the name of an external file and returns 1 if the file exists and 0 if not.

```
%let exist = %sysfunc(fileexist('c:\delme.dat'));
```

The FEXIST function accepts as its argument a preassigned fileref and also returns 1 if the file exists and 0 if not.

```
filename myfile 'c:\delme.dat';
%let exist = %sysfunc(fexist(myfile));
```

The following macro uses the FILEEXIST function to check for the existence of a directory.⁶ If it does not exist it is created. A fileref is then established for the directory.

```
%macro ChkDir(dname,fref);
    %if not %sysfunc(fileexist("&dname")) %then
        %let rc = %sysfunc(system(md "&dname"));
        %let rc = %sysfunc(filename(fref,&dname));
    %mend;
%ChkDir(c:\JuneData,Prod);
```

In the above example, the macro checks to see if the directory, c:\Production, exists and, if not, creates it. It then assigns the fileref, Prod, to that directory.

The FILENAME function works much like a filename statement. It takes a file reference and a file or directory

⁶ See Lund (1997) for a more comprehensive discussion of creating directories.

name as its arguments. When called in %SYSFUNC the file reference argument must be a macro variable reference without an ampersand.

You can also delete an external file with the FDELETE function. It takes as its argument a fileref and returns a 0 if successful and non-0 if not.

```
filename x 'c:\delme.dat';
%let rc = %sysfunc(fdelete(x));
```

In much the same manner as checking for the existence of an external file, you can check for the existence of a SAS dataset with the EXIST function. It also returns 1 if the dataset exists and 0 if not. One or two level dataset names are allowed.

```
%let exist = %sysfunc(exist(my_ds));
%let exist = %sysfunc(exist(my_lib.my_ds));
```

There is one other function for checking the existence of a "file". The CEXIST function checks for the existence of a catalog or catalog entry. It accepts as its arguments the name of the catalog or entry and an optional flag (U) to test whether the catalog is available for updating.

```
%let rc =
  %sysfunc(cexist(sashelp.flags.canada.image))
```

and

```
%let rc =
  %sysfunc(cexist(sashelp.flags.canada.image,U))
```

both return 1. Note that the update flag, U, checks if the catalog can be opened for update. In the above case, the image entry cannot be opened for update, but the catalog itself can. Also note that a one- or two-level is assumed to be a catalog name and a three- or four-level name is assumed to be a catalog entry name.

Put Your Code to the Test

Many functions called from %SYSFUNC return a value indicating the status of the function. Very often it is simply a 1 if the function was successful and 0 if not. Others, however, have a more extensive set of return codes. You can test and/or print the return code and associated message to control the flow of subsequent processing. Assuming the following file does not exist, submitting the following:

```
filename x 'c:\delme.dat';
%let rc = %sysfunc(fdelete(x));
%put &rc -- %sysfunc(sysmsg());
```

would produce the following in the log:

```
20006 -- ERROR: Physical file does not exist,
c:\delme.dat.
```

A quick note about the SCL functions SYMSG and SYSRC. These functions return the error messages and error codes related to the execution of the most recent SCL function. The above %PUT statement could be replaced with

```
%put %sysfunc(sysrc()) -- %sysfunc(sysmsg());
```

with the same results. The error message returned by SYMSG is reset to blank when the function is called. However, SYSRC is not reset will always contain the error condition of the last SCL function call. Thus, invoking the above statement twice in succession

```
%put %sysfunc(sysrc()) -- %sysfunc(sysmsg());
%put %sysfunc(sysrc()) -- %sysfunc(sysmsg());
```

would generate the following in the log:

```
20006 -- ERROR: Physical file does not exist,
c:\delme.dat.
20006 --
```

The moral of the story: check SYSRC and SYMSG immediately after a function call. That way you're always sure to which function the error values are related.

Get to Know Your Stuff

We've probably all used PROC CONTENTS or the DICTIONARY.TABLES in a data step of PROC SQL to get information about a dataset. There is now another option. There are two SCL functions which can tell you almost everything you'd want to know about a dataset. They two functions, ATTRN and ATTRC, differ only in the type of information they return.⁷ ATTRN returns numeric attributes; flags, counts and dates. ATTRC returns character attributes; names, labels and types.

In order to use these functions, we'll need to "open" the dataset, call the ATTRx function(s), and close the dataset. The OPEN function returns a dataset id which is used to identify the dataset to the ATTRx functions. The other argument to the ATTRx functions is the name of the attribute. An important note: an open dataset is locked to other processing and must be closed, using the dataset id,

⁷ A list of options for the ATTRN and ATTRC functions can be found in the Online Help under Screen Control Language...Functions or in "SAS Screen Control Language: Reference, Version 6, First Edition"

prior to further use. Always close the dataset when you have completed your work with it.

The following example opens the CRIMES dataset in the SASUSER library. It then gets the following attributes:

- The number of observations (NOBS)
- The number of variables (NVAR)
- The creation date of the dataset (CRDTE)
- The variables the dataset is sorted by (SORTEDBY)

```
%let dsid = %sysfunc(open(sasuser.crime));
%let NumObs = %sysfunc(attrn(&dsid,nobs));
%let NumVars = %sysfunc(attrn(&dsid,nvars));
%let CreateDt = %sysfunc(attrn(&dsid,crdte));
%let SortedBy = %sysfunc(attrc(&dsid,sortedby));
%let rc = %sysfunc(close(&dsid));
```

Notice that all the calls to ATTRN and ATTRC use the dataset ID (&DSID) which was returned by the OPEN function. This ID is also used in the CLOSE function to release the dataset.

All in the Macro Family

SAS Institute has created a set of macros which correspond to most of the functions supported by %SYSFUNC.⁸ An example is the macro %REVERSE:

```
%MACRO REVERSE(ARG1);
  %SYSFUNC(REVERSE(&ARG1))
%MEND;
```

Both of the following statements:

```
%put %sysfunc(reverse(REVERSE ME));
%put %reverse(REVERSE ME);
```

would produce:

```
EM ESREVER
```

If you find yourself using the same %SYSFUNC statement again and again, enclose it in your own macro. Earlier, we used the EXIST function to check for the existence of a dataset. We can create a macro called EXIST.

```
%macro Exist(DSName)
  %sysfunc(exist(&DSName));
%mend;
```

Invoking this macro as follows:

```
%Exist(sasuser.crime)
```

⁸ See Polzin and O'Connor (1996) for details of these macros and details for obtaining them from the SAS ftp site.

will produce a 1 (assuming the dataset exists).

Remember the embedded %SYSFUNC calls we used to calculate the value of the previous month when passed a date in YYMM format. Enclosing this is a macro greatly simplifies further use.

```
%macro LastMnth(ThisMnth);
  %sysfunc(intnx(month,
    %sysfunc(inputn(&ThisMnth.01,yyymmdd6.)),
    -1),yyymmdd4.);
%mend;
```

Invoking this macro, as follows:

```
%LastMnth(9701)
%LastMnth(9812)
```

produces 9612 and 9811, respectively. Much easier to look at than repeating the function call.

Conclusion

I could write quite a bit more on the many and varied uses of %SYSFUNC, but that would take all the fun out of it. After years of having just a handful of macro functions available (but making them do amazing things) it's been fun to find new ways to streamline code, eliminate steps and/or procs and tighten control of programs.

Acknowledgments

Thanks to Mark Terjeson and Valinda Scheibert for giving me some feedback and to the Olympia SAS Users Group for giving me a forum for presenting an earlier version of this paper.

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

References

Lund, Peter, "Always There When I Need You: Macros to Create Directories", *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.

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

Author Contact Information

Pete Lund
WA State Office of Financial Management
Forecasting Division
PO Box 43113
Olympia, WA 98504-3113
voice: (360) 902-0583
FAX: (360) 664-8194
e-mail: peter.lund@ofm.wa.gov