**%SYSFUNC: A Macro Variable Can't Function Without It**

By Chris Moriak, AstraZeneca, Wilmington, DE

**Abstract**
For many SAS programmers, the macro function %SYSFUNC is a mysterious, misunderstood macro function. There is not much literature on how to use %SYSFUNC. Also, the writings on %SYSFUNC have mainly discussed its use with external files, reading meta-data by opening a dataset, or formatting a macro variable. It is unfortunate that the literature does not proclaim %SYSFUNC as one of the most important tools to the macro language since the %LET statement. Macro variables simply cannot function without it, literally.

**The Problem**
While writing code with macros, how many times have you wanted to perform a Base SAS® function on a macro variable? There were many times that I wondered why SAS stopped creating macro functions like %INDEX or %UPCASE. Would not it be great if one could use %MAX, %EXIST, or %INDEXW?

I’m here to tell all SAS programmers, you can! %SYSFUNC allows one to perform nearly ever SAS function on one’s macro variables. No longer do you have to write witty or complex code to get around the non-macro function issue. Your macro programs can be as free as your open code.

**The Big Mystery**
For many, %SYSFUNC is a mysterious macro function with little documentation or discussion. The name itself, %SYSFUNC, adds to its enigma, blending connotations of “system” and “funk”. It can leave the programmer with thoughts of complex computer systems in a funk. Instead, I hope the reader comes away from this paper remembering that %SYSFUNC implies performing “system functions”.

If one peruses the contents of the past three or four NESUG conferences, one finds nearly every reference to %SYSFUNC to be code similar to below:

```sas
%let dsid=%sysfunc(open(mydata,i));
%let lbl=%sysfunc(attrc(&dsid,'LABEL'));
%let vnum=%sysfunc(varnum(&dsid,a));
%let fmt=%sysfunc(varfmt(&dsid,&vnum));
%let rc=%sysfunc(fetch(&dsid));
%let rc=%sysfunc(close(&dsid));
```

Such program code that is filled with %SYSFUNC references with peculiar function names like “fetch” or “attrc” can be intimidating to those familiar to the smooth linguistic flow of SAS language. Many of these data steps even seem to have little substance beyond the repetitive calling of %SYSFUNC.

And what if a programmer wanted to learn more about %SYSFUNC? Where can one find literature about this macro function? A recent query from the SAS technical web site offers no paper, notes, or advice on how to use this macro function. In fact, nearly every reference to it has code similar to the above. It is good to see that SAS v8.1 “Help” finally contains some information, yet it uses code similar to above in the “Details” section when describing how to use it.

I admit that %SYSFUNC is very useful when one needs access to meta-data or reading external file information. However, this paper attempts to de-mystify the macro function by showing its ease of use and programming power when writing macro code.
Functions on Macro Variables

I use %SYSFUNC in nearly every one of my macro programs. Several colleagues of mine would also profess the same. So why are you not using %SYSFUNC so prevalently?

If you are reading this paper, then you probably already write SAS macro code. If you write macro code, then at some point you will find yourself wanting to perform a function on a macro variable. Maybe you wanted to obtain the maximum value from a list contained in a macro variable or several macro variables? Maybe you wanted to scan a macro variable for a certain word? Or maybe you wanted to replace a character in a macro variable with another character?

SAS functions accomplish these tasks on regular variables in a data step. However, as every programmer soon finds out, SAS functions cannot be performed directly on macro variables. For example, one cannot perform the following:

\%
let a=1;
let b=2;
let c=max(&a, &b);

Nor can one perform the following:

\%
let a= adc;
let b=translate(&a,'b','d');

This can cause great distress as well as very creative programming to get around this problem. What many programmers do not know is that functions can be performed indirectly on macro variables by using %SYSFUNC.

While some may believe that %SYSFUNC is used by an elite few to show off their advanced SAS knowledge, %SYSFUNC should be used by every programmer who writes macro code. Nearly ever SAS function can be utilized via %SYSFUNC. Now that’s a tool.

Syntax

The syntax for this macro function:

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

For some reason, one tends to see in the literature one of the following syntaxes for %SYSFUNC:

rc = %sysfunc(function(argument(s))
              <, format>);
%let rc = %sysfunc(function(argument(s))
                  <, format>);
dsid = %sysfunc(function(argument(s))
               <, format>);

I have been told the “rc” stands for “return code” since %SYSFUNC always returns a value. The problem for users is a tendency to view “rc” as a special macro variable, thus adding to the mystery of %SYSFUNC. The same is true for “dsid” which simply means “data set ID”.

The truth is that %SYSFUNC is performing a SAS function and will return a value just as if one used a SAS function. One does not see programmers writing:

rc = reverse(var);

So why use it with %SYSFUNC? It is good to see SAS v8.1 “Help” move away from this customary approach and to use more “real life” programming code:

%let string1 =
    %sysfunc(translate(&string1,P, N));

To create the correct syntax, follow these three basic steps:

1. Write the function statement just as one would in a data step, using a macro variable instead of a regular variable.
2. Surround the function statement with %SYSFUNC().
3. Remove any quotation that may be used in the regular function syntax, as macro variables are not character values.

Let us look at the above example from SAS, using the function TRANSFORM. In a data step, one would write the following code:

```sas
string1=translate(string,'P','N');
```

Follow the 3-step process:

1. Write the statement using macro variables instead of regular variables:
   ```sas
   %let string1 = translate(&string,'P','N');
   ```

2. Surround the function statement with %SYSFUNC( ):
   ```sas
   %let string1 = %sysfunc(translate(&string,'P','N'));
   ```

3. Remove any quotes that may exist in the function statement:
   ```sas
   %let string1 = %sysfunc(translate(&string,P,N));
   ```

It is that easy to use %SYSFUNC. If you know how to use SAS functions, then you can start using %SYSFUNC to perform functions on macro variables.

### 1001 Uses for %SYSFUNC

The ability of %SYSFUNC to perform functions on macro variables, as well as to obtain system options and information means that it can be used in many places in a program:

- System settings and options
- Title and footnote statements
- Data step
- Creation and modification of macro variables
- Macro logic

The next portion of the paper will demonstrate each of these uses.

### System Settings and Options

Sometimes a programmer wants to know a specific system setting of an option. For example, if a macro program will set the option COMPRESS=YES, the macro program should return the option to its original setting when it is finished processing. The original information can quickly be obtained and stored in a macro variable through %SYSFUNC:

```sas
%let opt=%sysfunc(getoption(compress));
options compress=yes;
...statements...
options compress=&opt;
```

### Titles and Footnotes

Since %SYSFUNC returns a value, one can use it in a title or footnote to obtain system information or to format a macro variable. For example, one can obtain the current system date or system time. Please note that this is the system date and time, not the SAS session date or time:

**Code**

```sas
%let a = 1234;
title1 "Today's date is %sysfunc(date(), worddate.);"
title2 "The time is %sysfunc(time())"
title3 "Dollar: %sysfunc(putn(&a , dollar8.))"
title4 "Decimal: %sysfunc(putn(&a , 8.2))"
```

**Output for Titles**

<table>
<thead>
<tr>
<th>Today's date is</th>
<th>June 18, 2002</th>
</tr>
</thead>
<tbody>
<tr>
<td>The time</td>
<td>9:53</td>
</tr>
<tr>
<td>Dollar</td>
<td>$1,234</td>
</tr>
<tr>
<td>Decimal</td>
<td>1234.00</td>
</tr>
</tbody>
</table>
Data Step
In a data step, one can use %SYSFUNC in creating variables by performing functions on macro variables. The below example creates a new variables depending on the maximum and minimum values of the variable VALUE and the macro variables &VAL1 and &VAL2 that were assigned before the data step:

Code
%let val1=17, 24, 35, 76;
%let val2=87, 32, 45, 6;
%let x=abc;

data temp;
   max=%sysfunc(max(&val1, &val2));
   min=%sysfunc(min(&val1, &val2));
   n=%sysfunc(n(&val1, &val2));
   rev=%sysfunc(reverse(&x));
run;

proc print;run;

Output
Obs   max   min   n   rev
1    87     6     8    cba

Creating or Modifying Macro Variables
An important ability of %SYSFUNC is to create or modify macro variables. The above example of a data step could be easily modified to create macro variables:

Code
%let val1=17, 24, 35, 76;
%let val2=87, 32, 45, 6;
%let x=abc;

%let max=%sysfunc(max(&val1, &val2));
%let min=%sysfunc(min(&val1, &val2));
%let n=%sysfunc(n(&val1, &val2));
%let rev=%sysfunc(reverse(&x));

%put min=&min;
%put n=&n;
%put rev=&rev;

Log
max=87
min=6
n=8
rev=cba

Two functions that I use quite often in modifying macro variables are the TRANSLATE and TRANSWRD functions. Here is an example of using both:

Code
%let x=abc~def;

%let new1=%sysfunc(translate(&x,$,~));
%let new2=%sysfunc(tranwrd(&x,def,zzz));

%put new1=&new1;
%put new2=&new2;

Log
new1=abc$def
new2=abc~zzz

Macro Logic
Using %SYSFUNC in macro logic statements is probably the most common use of %SYSFUNC outside of using it to retrieve meta-data. One can conditionally check or verify the existence of data sets, librefs, and filerefs, before conducting further processing. Each of these can quickly be done by using %SYSFUNC. Caution, be sure to know the return value for the true statement.

Checking the existence of a data set (1=true):
%if %sysfunc(exist(data-set)) eq 1

Checking the existence of a libref (0=true):
%if %sysfunc(libref(libref)) eq 0
Checking the existence of a fileref (0=true):
%if %sysfunc(fileref(fileref)) eq 0

Almost any SAS function could be used in macro logic:

%if %sysfunc(max(&value)) ge 5 %then
  %do;
  …statements…;
  %end;

%if %sysfunc(indexw(&var,TREAT)) > 0
%then
  %do;
  …statements…;
  %end;

Conclusion
As I said, the uses of %SYSFUNC to perform functions on macro variables are limitless. One just needs to review the SAS Language book to obtain the available functions. In the appendix, I have compiled a brief list of functions that I use most often. So, if one writes macro code, then be sure to add %SYSFUNC to one’s repertoire. It should be no mystery that this instrument will do wonders for one’s program.

Acknowledgments
The author would like to thank Jeremy Gratt and John Adams in our mutual discovery and expanded use of this most appreciated macro language tool.

For an excellent quick-reference of available SAS functions, please see Rick Aster’s book, *Professional SAS® Programmer’s Pocket Reference*, noted in the reference portion of this paper.

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. Other brand and product names are registered trademarks or trademarks of their respective companies.

References

Contact Information
The author welcomes comments, suggestions, or questions:

Chris Moriak
Quantitative Decision Sciences
AstraZeneca
1800 Concord Pike
PO Box 15437
Wilmington, DE 19850

Tel: 302-885-9977
E-mail: christopher.moriak@astrazeneca.com
## Appendix A: Various Uses of %SYSFUNC

<table>
<thead>
<tr>
<th>%SYSFUNC Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System Settings and Options</strong></td>
<td></td>
</tr>
<tr>
<td>%sysfunc(exist(data_set));</td>
<td>- Check if data_set exists (1=True)</td>
</tr>
<tr>
<td>%sysfunc(cexist(catalog));</td>
<td>- Check if catalog exists (1=True)</td>
</tr>
<tr>
<td>%sysfunc(libref(libref));</td>
<td>- Check if libref exists (0=True)</td>
</tr>
<tr>
<td>%sysfunc(fileref(fileref));</td>
<td>- Check if fileref exists (0=True)</td>
</tr>
<tr>
<td>%sysfunc(fileexist(file.sas));</td>
<td>- Check if file.sas exists (1=True)</td>
</tr>
<tr>
<td>%sysfunc(pathname(libref));</td>
<td>- Get file path (directory) for libref</td>
</tr>
<tr>
<td>%sysfunc(getoption(option));</td>
<td>- Get option setting</td>
</tr>
<tr>
<td>%sysfunc(time(),time5.);</td>
<td>- Get system time with format of time5.</td>
</tr>
<tr>
<td>%sysfunc(date(),worddate.);</td>
<td>- Get system date with format of worddate.</td>
</tr>
<tr>
<td><strong>Statistics and Operations</strong></td>
<td></td>
</tr>
<tr>
<td>%sysfunc(min(&amp;val1,&amp;var2));</td>
<td>- Calculates the minimum of the values</td>
</tr>
<tr>
<td>%sysfunc(max(&amp;val1,&amp;var2));</td>
<td>- Calculates the maximum of the values</td>
</tr>
<tr>
<td>%sysfunc(n(&amp;val1,&amp;var2));</td>
<td>- Calculates the number of non-missing values</td>
</tr>
<tr>
<td>%sysfunc(range(&amp;val1,&amp;var2));</td>
<td>- Calculates the range of the values</td>
</tr>
<tr>
<td>%sysfunc(sum(&amp;val1,&amp;var2));</td>
<td>- Calculates the sum of the values</td>
</tr>
<tr>
<td>%sysfunc(std(&amp;val1,&amp;var2));</td>
<td>- Calculates the standard deviation of the values</td>
</tr>
<tr>
<td>%sysfunc(mean(&amp;val1,&amp;var2));</td>
<td>- Calculates the mean of the values</td>
</tr>
<tr>
<td>%sysfunc(floor(&amp;val1));</td>
<td>- Calculates the smallest integer value ( \leq ) the value</td>
</tr>
<tr>
<td>%sysfunc(ceil(&amp;val1));</td>
<td>- Calculates the smallest integer value ( \geq ) the value</td>
</tr>
<tr>
<td>%sysfunc(int(&amp;val1));</td>
<td>- Calculates the integer value of the value</td>
</tr>
<tr>
<td>%sysfunc(round(&amp;val1,value));</td>
<td>- Rounds the value to the specified precision</td>
</tr>
<tr>
<td><strong>Value Functions</strong></td>
<td></td>
</tr>
<tr>
<td>%sysfunc(putn(&amp;var1,format));</td>
<td>- Returns numeric value with specified format applied</td>
</tr>
<tr>
<td>%sysfunc(putc(&amp;var1,format));</td>
<td>- Returns character value with specified format applied</td>
</tr>
<tr>
<td>%sysfunc(indexc(&amp;var1,string));</td>
<td>- Returns the first location of the string</td>
</tr>
<tr>
<td>%sysfunc(indexw(&amp;var1,string));</td>
<td>- Returns the first location of the string as a word</td>
</tr>
<tr>
<td>%sysfunc(translate(&amp;var1, replacement_char, char));</td>
<td>- Replaces the character with the specified character</td>
</tr>
<tr>
<td>%sysfunc(tranwrd(&amp;var1,string, replacement_string));</td>
<td>- Replaces the string with the specified string</td>
</tr>
<tr>
<td>%sysfunc(compress(&amp;var1,char));</td>
<td>- Removes blanks or specified character</td>
</tr>
<tr>
<td>%sysfunc(compbl(&amp;var1));</td>
<td>- Replaces consecutive blanks with a single blank</td>
</tr>
<tr>
<td>%sysfunc(quote(&amp;var1));</td>
<td>- Quotes the value</td>
</tr>
<tr>
<td>%sysfunc(dequote(&amp;var1));</td>
<td>- Unquotes the value</td>
</tr>
</tbody>
</table>