Get funky with %SYSFUNC

Elena Muriel, Amadeus Software Limited

ABSTRACT

%SYSFUNC is a powerful macro function with respect to the manipulation of SAS data sets, external files and directories; amongst other uses.

In this paper we will discuss how to use SCL specific functions in a typical Base and Macro environment.

By using simple examples we will illustrate the full capacity of this macro function in obtaining data set attributes, data set variable attributes, checking for existence of external files and how to read the contents of external directories.

Discussion is given to the QSYSFUNC function, along with an explanation and example of when it should be used. Finally, a discussion of the benefits of using SCL functions in the data step will be presented.

This paper will be of interest to SAS programmers familiar with Base SAS and Macro, who wishes to learn the benefits of using SCL functions in the macro and data step.

INTRODUCTION

%SYSFUNC and %QSYSFUNC are two powerful macro functions that make the most of Base SAS software and Screen Control Language (SCL) functions.

Its use increases the number of functions available to a SAS programmer, and in many cases, solutions to common tasks are simpler and more efficient.

Emphasis is placed on available SCL functions, and special interest is taken on those interacting with external files.

From Version 8 onwards, SCL functions are also available to the programmer through the data step environment (in some cases using an easier syntax), but it is the aim of this paper to illustrate all available possibilities using %SYSFUNC.

Note: all examples were run using SAS software version 8.2 on Windows© XP.

SYSFUNC

The %SYSFUNC function, executes SAS functions or user-written functions (created using SAS/TOOLKIT®). Its syntax is very easy as it only needs to reference the name of the desired function and its required arguments:

%SYSFUNC(function-name(function-arguments)[,format])

There are two types of functions that can be referenced: SAS base functions, or SCL functions.

SAS Base functions include all data steps functions except from:

DIF	DIM	HBOUND	IORCMSG
INPUT	LAG	LBOUND	MISSING
PUT	RESOLVE	SYMGET	

All variable information functions (like VNAME or VLABEL)

Instead of INPUT and PUT you can use INPUTC, INPUTN, and PUTC, PUTN for character or numeric formats.

The SCL functions include:

ATTRC	FCLOSE	FPOINT	OPEN
ATTRN	FCOL	FPOS	PATHNAME
CEXIST	FDELETE	FPUT	POINT
CLOSE	FETCH	FREAD	REWIND
CUROBS	FETCHOBS	FREWIND	SPEIDS
DCLOSE	FEXIST	FRLEN	SYSGET
DINFO	FGET	FSEP	SYSMSG
DNUM	FILEEXIST	FWRITE	SYSRC
DOPEN	FILENAME	GETOPTION	VARFMT
DOPTNAME	FILEREF	GETVARC	VARINFMT
DOPTNUM	FINFO	GETVARN	VARLABEL
DREAD	FNOTE	LIBNAME	VARLEN
DROPNOTE	FOPEN	LIBREF	VARNAME
DSNAME	FOPTNAME	MOPEN	VARNUM
EXIST	FOPTNUM	NOTE	VARTYPE
FAPPEND			

This paper will focus on examples using SCL functions.

EXAMPLE 1

This first example relates to a very common task, how to check if a SAS data set exists.

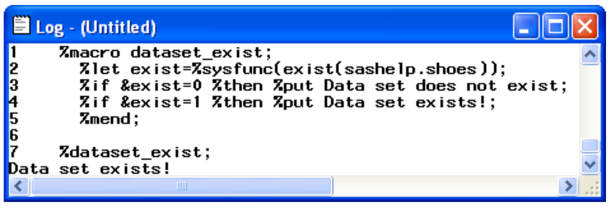
```
%let exist=%sysfunc(exist(sashelp.shoes));
```

We are creating a macro variable named *exist,* that will contain the result of applying the SCL function EXIST on the SASHELP.SHOES data set. If the data set exists, the function will return the value 1, and if the data set doesn't exist, it will return a 0.

Once we have checked if a data set exists, we can perform other operations, and as in this case, write a message to the log window with the result of our operation using macros:

```
%macro dataset_exist;
%let exist=%sysfunc(exist(sashelp.shoes));
%if &exist=0 %then %put Data set does not
      exist;
%if &exist=1 %then %put Data set exists!;
%mend;

%dataset_exist;
```

And in this case, the log window will display



```
   %macro dataset_exist;
     %let exist=%sysfunc(exist(sashelp.shoes));
     %if &exist=0 %then %put Data set does not exist;
     %if &exist=1 %then %put Data set exists!;
     %mend;

   %dataset_exist;
Data set exists!
```

This example checks for the existence of a SAS data set, but in the same way, you can also check for existing Access descriptors, catalogs and views.

EXAMPLE 2

Once we have checked if a data set exists, we may be interested in obtaining certain attributes from it.
A very useful one is to obtain the number of observations and the number of columns in a data set:

```
%let did=%sysfunc(open(sashelp.shoes,i));
%let nobs=%sysfunc(attrn(&did,nobs));
%let nvars=%sysfunc(attrn(&did,nvars));
%let rc=%sysfunc(close(&did));
%put Data set SASUSER.SHOES contains &nobs
  observations and &nvars variables;
```

As in SCL, first we need to open the data set we want to obtain the attributes from, and in this example, we are opening SASHELP.SHOES in SCL in Input mode. When we open a data set in Input mode (I, IN and IS), the data set can be read, but it cannot be modified. All other available modes of opening a data set (N, U, UN, US and V) are not available with %SYSFUNC.

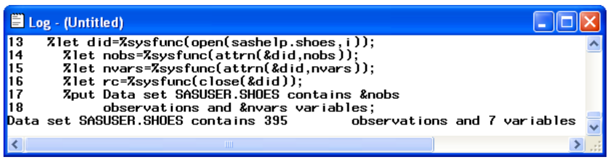
Once the data set is open, we can obtain its attributes using the ATTRN function. In this case the NOBS (number of observations) and NVARS (number of variables) numeric attributes. Character attributes can be obtained using the ATTRC function.

A very useful property for the %SYSFUNC function is that it can be nested. We could in fact, combine the first two lines of code, so we can open and obtain attributes from a data set in just one line:

```
%let nobs=%sysfunc(attrn(
    %sysfunc(open(sashelp.shoes,i)),nobs));
```

After obtaining all attributes, always remember to close the data set.

If we run this code, we will obtain the following message in the log window:



```
  %let did=%sysfunc(open(sashelp.shoes,i));
    %let nobs=%sysfunc(attrn(&did,nobs));
    %let nvars=%sysfunc(attrn(&did,nvars));
    %let rc=%sysfunc(close(&did));
    %put Data set SASUSER.SHOES contains &nobs
         observations and &nvars variables;
Data set SASUSER.SHOES contains 395        observations and 7 variables
```

EXAMPLE 3

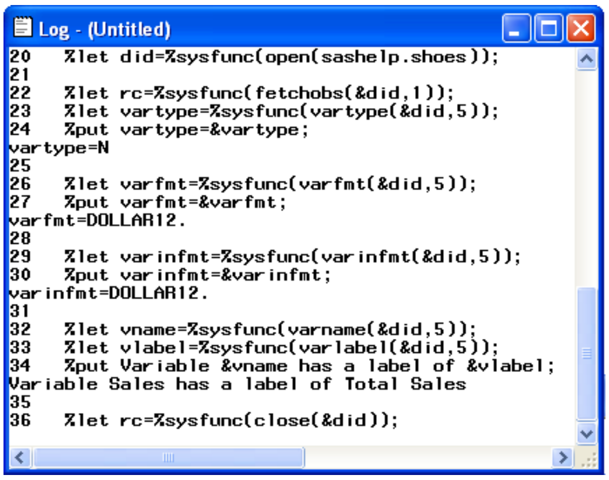
Once we have obtained data set attributes, on this next example, we will obtain attributes regarding the variables present in the data set SASHELP.SHOES.

```
%let did=%sysfunc(open(sashelp.shoes));

%let rc=%sysfunc(fetchobs(&did,1));
%let vartype=%sysfunc(vartype(&did,5));
%put vartype=&vartype;

%let varfmt=%sysfunc(varfmt(&did,5));
%put varfmt=&varfmt;

%let varinfmt=%sysfunc(varinfmt(&did,5));
%put varinfmt=&varinfmt;

%let vname=%sysfunc(varname(&did,5));
%let vlabel=%sysfunc(varlabel(&did,5));
%put Variable &vname has a label of &vlabel;

%let rc=%sysfunc(close(&did));
```

First of all, we need to open the data set in input mode. Note that Input (I) mode is the default mode associated with the OPEN function.

Then read the first observation in the data set using the FETCHOBS function, to obtain the type of variable, format, informat, variable name and label for the fifth variable present in the data set.

The results this program will display in the log window are:



```
  %let did=%sysfunc(open(sashelp.shoes));

  %let rc=%sysfunc(fetchobs(&did,1));
  %let vartype=%sysfunc(vartype(&did,5));
  %put vartype=&vartype;
vartype=N

  %let varfmt=%sysfunc(varfmt(&did,5));
  %put varfmt=&varfmt;
varfmt=DOLLAR12.

  %let varinfmt=%sysfunc(varinfmt(&did,5));
  %put varinfmt=&varinfmt;
varinfmt=DOLLAR12.

  %let vname=%sysfunc(varname(&did,5));
  %let vlabel=%sysfunc(varlabel(&did,5));
  %put Variable &vname has a label of &vlabel;
Variable Sales has a label of Total Sales

  %let rc=%sysfunc(close(&did));
```

In this example we are obtaining information on the fifth variable found in the SHOES data set, but if we want to reference the name of the variable instead of its position in the SAS data set, we need to use the VARNUM function:

```
%let var1=%sysfunc(varnum(&did,sales));
%let rc=%sysfunc(fetchobs(&did,&var1));
```

The VARNUM function gives us the position of the variable in the SAS data set (in the case of SASHELP.SHOES the position of the SALES variable is 5).

And as seen on Example 2, we can nest several functions:

```
%let vartype=%sysfunc(vartype(&did,
     %sysfunc(varnum(&did,sales))));
```

EXAMPLE 4

All previous examples were focused on SAS data set attributes and variable attributes. In the next two examples, we will focus on external files.

One of the first things that a programmer is interested about an external file, is knowing if a file actually exists or not.

On this first example, we are going to check if a CSV file exists:

```
%let rc=%sysfunc(fileexist(
       'c:\elena\data\10Feb2004.csv'));
```

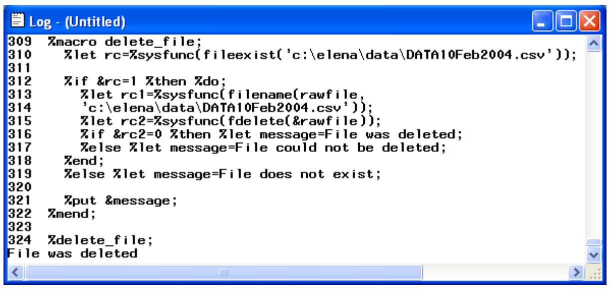
Again if the file exists the function will return a 1, and if it doesn't exist, it will return a 0.

Once we have proved that the file exists, we are going to perform an action with it. In this case we are going to check that a file exists before deleting it:

```
%macro delete_file;
%let rc=%sysfunc(fileexist('c:\elena\data\
     DATA10Feb2004.csv'));

%if &rc=1 %then %do;
  %let rc1=%sysfunc(filename(rawfile,
      'c:\elena\data\DATA10Feb2004.csv'));
  %let rc2=%sysfunc(fdelete(&rawfile));
  %if &rc2=0 %then %let message=File was
     deleted;
  %else %let message=File could not be
     deleted;
%end;
%else %let message=File does not exist;

%put &message;
%mend;

%delete_file;
```

First we check that the CSV file exists and if it does, we are going to associate a filename with it.

Then we will use the FDELETE function to delete the file, and then write a message to the log window with the output of our operation.



```
309  %macro delete_file;
310    %let rc=%sysfunc(fileexist('c:\elena\data\DATA10Feb2004.csv'));
311
312    %if &rc=1 %then %do;
313      %let rc1=%sysfunc(filename(rawfile,
314      'c:\elena\data\DATA10Feb2004.csv'));
315      %let rc2=%sysfunc(fdelete(&rawfile));
316      %if &rc2=0 %then %let message=File was deleted;
317      %else %let message=File could not be deleted;
318    %end;
319    %else %let message=File does not exist;
320
321    %put &message;
322  %mend;
323
324  %delete_file;
File was deleted
```

Note: Operating system security will have to be considered when dealing with external files.

EXAMPLE 5

This example will be focused on how to read the contents of an external directory, and how to automatically import all the CSV files found into SAS data sets.

```
%macro read_files;
%let rc=
  %sysfunc(filename(rawdata,'c:\elena\data'));
%let did=%sysfunc(dopen(&rawdata));
%let dnum=%sysfunc(dnum(&did));

%do i=1 %to &dnum;
  %let name_file=%sysfunc(dread(&did,&i));
  %put file=&name_file;
  %let csv=%sysfunc(scan(&name_file,2,'.'));
  %if &csv=csv %then %do;
    %let name_file_short=
           %sysfunc(scan(&name_file,1,'.'));
       %put file=&name_file_short;

       PROC IMPORT OUT= WORK.&name_file_short
         DATAFILE= "C:\Elena\data\&name_file"
         DBMS=CSV REPLACE;
       RUN;
  %end;
%end;
%mend;

%read_files;
```

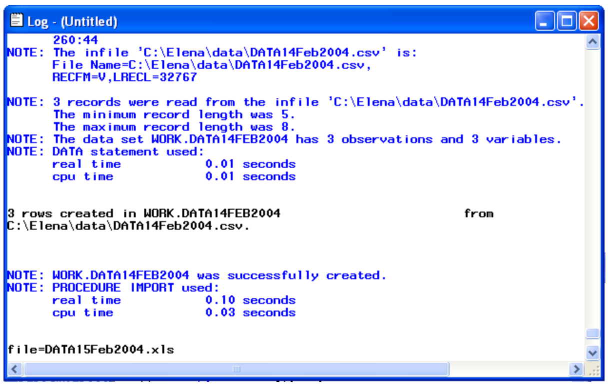
First we associate a filename with the directory containing all our files. Then we open it in order to obtain the number of members it contains (this is achieved with the DNUM function).

We start reading every file located under the directory using a do loop and the DREAD function.

Once we obtain the name of the object found, we need to make sure it is a CSV file before importing it. By using the SCAN function, we will be able to discount any other non-CSV files found or other subdirectories.

Finally, we use a proc import to generate a SAS data set with the same name as the CSV file.

Part of the log window will appear as follows:



```
      260:44
NOTE: The infile 'C:\Elena\data\DATA14Feb2004.csv' is:
      File Name=C:\Elena\data\DATA14Feb2004.csv,
      RECFM=V,LRECL=32767

NOTE: 3 records were read from the infile 'C:\Elena\data\DATA14Feb2004.csv'.
      The minimum record length was 5.
      The maximum record length was 8.
NOTE: The data set WORK.DATA14FEB2004 has 3 observations and 3 variables.
NOTE: DATA statement used:
      real time           0.01 seconds
      cpu time            0.01 seconds


3 rows created in WORK.DATA14FEB2004                       from
C:\Elena\data\DATA14Feb2004.csv.



NOTE: WORK.DATA14FEB2004 was successfully created.
NOTE: PROCEDURE IMPORT used:
      real time           0.10 seconds
      cpu time            0.03 seconds


file=DATA15Feb2004.xls
```

EXAMPLE 6

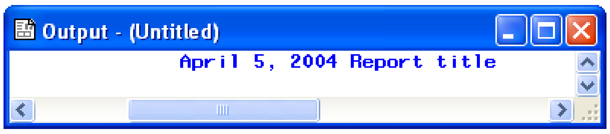
Another of the major uses of the %SYSFUNC function is to format macro variables.

Normally, if we want to format the value of a macro variable, we will need to do it in a data null step as follows:

```
data _null_;
  formatdate=put(date(),worddate.);
  call symput('wdate',formatdate);
run;
```

and then maybe use it as part of a report title:

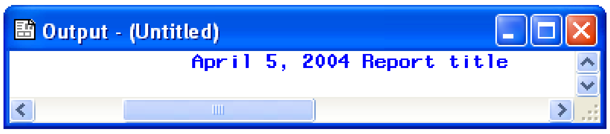
```
title "&wdate Report title";
```



```
            April 5, 2004 Report title
```

With the %SYSFUNC function, we don't need to have the _NULL_ data step, as we can format macro variables directly within the title line:

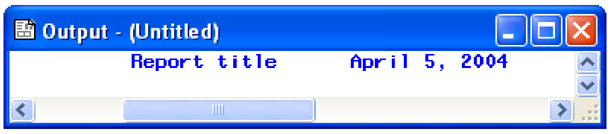
```
title "%sysfunc(date(),worddate.) Report
     title";
```



```
             April 5, 2004 Report title
```

Now if we want to include this date at the end of our title:

```
title "Report title %sysfunc(date(),
     worddate.)";
```



```
Report title        April 5, 2004
```

We find that the title contains several trailing blank spaces.
As we saw on previous examples, %SYSFUNC can be nested, so the obvious step is to apply the LEFT function over the date, to try to remove the trailing blanks:

```
  title "Report title %sysfunc(left
        (%sysfunc(date(),worddate.)))";
```

But when we try to apply this, we obtain an error message in the log window:

```
ERROR: The function LEFT referenced by the
%SYSFUNC or %QSYSFUNC macro function has too
many arguments.
```

When the first %SYSFUNC is resolved, we obtain:

```
  title "Report title
        %sysfunc(left(April 5, 2004))";
```

so the LEFT function finds a comma and considers the year as a second argument of the function, and therefore generates the error message.
To solve this problem we will need to use the %QSYSFUNC function.

QSYSFUNC

The %QSYSFUNC function is used to mask special characters, including commas, &, %, and mnemonic operators, in a resolved value at macro execution.

Its syntax is the same as the one for the %SYSFUNC:

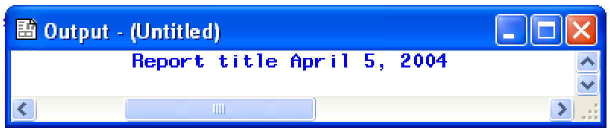
%QSYSFUNC(function-name(function-arguments)[,format])

EXAMPLE 7

Following on from Example 6, we can now resolve the problem in the number of arguments found by the LEFT function, by using the %QSYSFUNC function:

```
  title "Report title %sysfunc(
        left(%qsysfunc(date(),worddate.)))";
```

to obtain:



```
Report title April 5, 2004
```

CONCLUSION

Although utilization of SCL functions in a data step can be more efficient in terms of processing time, SYSFUNC should be considered as an alternative in which the programmer won't find those limitations and restrictions typical of the data step environment.

SYSFUNC has also proven to be a great tool when formatting macro variables, reducing the number of steps required. When used along with QSYSFUNC, it proves to be a powerful formatting tool within the macro language.

REFERENCES

V8 OnLine help documentation
SAS Advanced Tutorials. %SYSFUNC – The Brave New Macro World. Chris Yindra.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author Name	Elena Muriel
Company:	Amadeus Software Limited
Address:	The Old School Hall 11 Wesley Walk Witney, Oxon OX28 6ZJ
Work Phone:	+44 (0) 1993 848010
Email:	Elena.Muriel@amadeus.co.uk
Web:	www.amadeus.co.uk