Paper 24-2022

Talking to Your Host

Kurt Bremser, formerly of Allianz Technology

ABSTRACT

SAS® provides multiple tools to interact with the underlying operating system. This paper will review those tools, from the built-in functions to simple use of external commands to complex, flexible solutions for problems that cannot be solved with built-in functions.

INTRODUCTION

While doing a stint as a “Code Doctor” at SASGF 2019, I quickly typed in one of my “standard” code sequences to run an external command. Another presenter, who is quite an accomplished SAS coder himself, was astonished that I could do this “from memory”. This reminded me that what may seem to be a simple everyday action can be surprising new stuff for others. So I decided to do a paper to bring coding techniques that are often used by the senior members of the SAS® Communities to the attention of a wider audience.

These techniques can be helpful to the novice and even to the master who has not yet known them.

BUILT-INS

SAS has been active for many years to make interfacing with the base operating system easier by providing functions, statements and other elements. These are designed to make it unnecessary for the user to know details like what command to use or how to use it; they also make porting the code from one system to another easier.

FUNCTIONS FOR FILES

FILENAME(fref,physname<,device><,host-options>)

Assigns a file reference for file physname to the logical name contained in fref; if fref is empty, a name is created and assigned to the variable.
fref can be a character expression, a string or a variable name in a data step; in a macro (used with %sysfunc) it must be a macro variable name without the ampersand. In a data step, define a variable with a length of 8, as that is the maximum length allowed for file references.
Used without a second argument (or an empty second argument), it clears the file reference.
device and host-options are optional parameters, the same as used in a FILENAME statement. If you need to use host-options for a simple file, you need to supply an empty string for device (the order of parameters is important!)

FOPEN(fref)

Tries to open the file pointed to by the file reference in fref; if successful, it returns a non-negative integer (the file identifier), otherwise it returns 0. It will fail if the file reference does not point to an ordinary file or a pipe (e.g. if it points to a directory).

FOPTNUM(fid)

Returns the number of *information items* for the file associated with the identifier fid returned by the FOPEN() function.
The value depends on the operating system and the type of the file.

FOPTNAME(fid,number)

Returns the *name* of a *single information item* for the file. Number must be between 1 and the value returned by FOPTNUM().
The returned name is dependent on the operating system and the locale of the system!

FINFO(fid,inf_item)

Returns the value of a single information item for the file. inf_item has to be the *name* returned by the FOPTNAME() function.

FCLOSE(fid)

Closes the file; this is necessary so that you don't accumulate file handles, especially when using the FOPEN() function in macros.

FCOPY(fref_1,fref_2)

Copies a file from the first reference to the second; how the file is copied (as text or binary) is determined by the options used in the FILENAME() function calls or FILENAME statements which defined the file references.

FDELETE(fref | directory)

Removes a file or directory associated with fref, or a directory named with its physical name; to remove a directory, the directory must be empty.

RENAME(name_1,name_2,"FILE")

Renames an external file; with a different third parameter, RENAME can also be used for datasets, views or catalogs.

There are a lot more functions for reading and writing file content, but for the purposes of this paper only the ones dealing with file metadata are used and presented.

Examples

Retrieving the available information items:

```
data finfo;
rc = filename("fref","$HOME/test.txt");
fid = fopen("fref");
if fid then do;
  do i = 1 to foptnum(fid);
    optionname = foptname(fid,i);
    optionvalue = finfo(fid,optionname);
    output;
  end;
  rc = fclose(fid);
end;
rc = filename("fref");
keep optionname optionvalue;
run;
```

Copying a SAS dataset file:

```
data _null_;
length fref1 fref2 $8;
rc = filename(fref1,"~/mylib/class.sas7bdat","","recfm=n");
put rc=;
rc = filename(fref2,"~/mylib/class2.sas7bdat","","recfm=n");
rc = fcopy(fref1,fref2);
put rc=;
rc = filename(fref1);
rc = filename(fref2);
run;
```

RECFM=N causes a binary copy; the target file is a valid SAS dataset file and can be used like the source.

FUNCTIONS FOR DIRECTORIES

As for files, you need to first define a file reference with the FILENAME() function (or the FILENAME statement).

DOPEN(fref)

Tries to open the directory pointed to by the file reference in fref; if successful, it returns a non-negative integer. It will fail for anything that is not a directory.

DOPTNUM(did), DOPTNAME(did,number), DINFO(did,inf_item), DCLOSE(did)

These work similar to the respective functions for files.

DNUM(did)

Returns the number of member entries in the directory.

DREAD(did,n)

Retrieves the name of the nth member in the directory.

Examples

Reading a directory into a SAS dataset:

```
data members;
length dref $8 name $200;
rc = filename(dref,'$HOME');
did = dopen(dref);
if did then do;
  do i = 1 to dnum(did);
    name = dread(did,i);
    output;
  end;
  rc = dclose(did);
end;
rc = filename(dref);
keep name;
run;
```

And a more complicated example showing how to retrieve the names of all files in a directory tree:

```
%macro find(directory);
%local did i name subdir fref fref2;
%let did=%sysfunc(filename(fref,&directory));
%let did=%sysfunc(dopen(&fref));
%if &did ne 0
%then %do;
  %do i = 1 %to %sysfunc(dnum(&did));
    %let name=&directory/%sysfunc(dread(&did,&i));
    %let subdir=%sysfunc(filename(fref2,&name));
    %let subdir=%sysfunc(dopen(&fref2));
    %if &subdir ne 0
    %then %do;
      %let subdir=%sysfunc(dclose(&subdir));
```

```
      %find(&name)
    %end;
    %else %put &name;
    %let subdir=%sysfunc(filename(fref2));
  %end;
  %let did=%sysfunc(dclose(&did));
%end;
%let did=%sysfunc(filename(fref));
%mend;
%find($HOME)
```

A macro is used because the macro language allows recursion; doing recursion in a data step with custom functions is quite non-trivial, but one of my fellow super users on the SAS Communities has developed a clever way to do pseudo-recursion via a modified dataset:

```
data filelist;
  length dname filename $256 dir level 8 lastmod size 8;
  format lastmod datetime20.;
  input dname;
  retain filename ' ' level 0 dir 1;
cards4;
$HOME
;;;;

data filelist;
  modify filelist;
  rc1=filename('tmp',catx('/',dname,filename));
  rc2=dopen('tmp');
  dir = not not rc2;
  if not dir then do;
    fid=fopen('tmp','i',0,'b');
    lastmod=input(finfo(fid,foptname(fid,5)),NLDATM100.);
    size=input(finfo(fid,foptname(fid,6)),32.);
    fid=fclose(fid);
  end;
  else do;
    dname=catx('/',dname,filename);
    filename=' ';
    lastmod=input(dinfo(rc2,doptname(rc2,5)),NLDATM100.);
  end;
  replace;
  if dir;
  level=level+1;
  do i=1 to dnum(rc2);
    filename=dread(rc2,i);
    output;
  end;
  rc3=dclose(rc2);
run;
```

FUNCTIONS TO WORK WITH ENVIRONMENT VARIABLES

SYSEXIST(name)

Checks if an environment variable exists.

SYSGET(Name), %SYSGET(Name)

Retrieves the value of an environment variable.

Automatic Libraries and File References

If an environment variable was present at SAS start that meets the requirements (valid SAS name of a maximum length of 8) and contains either a path to a directory or a filename, then this will automatically be usable as a library or file reference in SAS. It will not be visible in the Explorer before you actually use it (SAS does a behind-the scenes LIBNAME or FILENAME).

MACRO VARIABLES

SAS provides several automatic macro variables that let you retrieve information about your process and the system (excerpt):

SYSSCP, SYSSCPL

Contain the name of the operating system; useful to select operating-system specific syntax elements (e.g. forward slash vs. backslash in path names) or command names/syntax when using external commands.

SYSUSERID

Contains the name of the user running the SAS executable.

SYSJOBID

Contains the process number of the current process (UNIX); is also coded (in hex notation) into the name of the WORK directory.

LIMITS OF THE BUILT-IN TOOLS

While all these statements and functions make it quite easy to write code that can be ported across platforms, as no system-specific code is necessary, they do have their limitations.

The names of information items for files and directories are specific to operating system and locale (a major shortcoming in the author's opinion), and lots of interesting data cannot be retrieved (like the timestamp of the last read access to a file or directory).

Whenever the built-in tools run out of options, it is good when the coder can make use of tools external to SAS.

RUNNING EXTERNAL COMMANDS

Use of the following statements, functions and methods requires that the system option XCMD is set; out of the box, SAS BI Server installations have this disabled for workspace servers, but it only requires a quick setting in Management Console to enable it.

It is the express opinion of the author that (in most cases) security cannot be an issue here; in a properly administrated environment, no end user can harm anything that does not explicitly belong to him/her, and even in the case of a catastrophic mistake by the user, nothing more than a day's work can be lost (as everything else of importance is retrievable from the backup that ran incrementally during the night). Security and privacy can be an issue in installations that work for multiple organizations, where (e.g.) information about other users would easily be accessible through the OS commands, while it can be hidden from other user groups in SAS metadata.

X AND %SYSEXEC STATEMENTS

```
X 'command';
%sysexec command;
```

These statements execute the command and set the automatic macro variable SYSRC to the return code of the external command; does not provide any information in the log about results other than SYSRC. X is a global statement, immediately executed when encountered in the code, and can therefore not be executed conditionally. But it can be "part" of a data step in terms of time consumed, see this log:

```
27         %put %sysfunc(time(),time8.);
15:37:49
28         data _null_;
29         y = time();
30         put y time8.;
31         do x = 1 to 10;
32         x 'sleep 5'
32       !           ;
33         end;
34         y = time();
35         put y time8.;
36         run;

15:37:54
15:37:54
NOTE: DATA statement used (Total process time):
      real time           5.02 seconds
      cpu time            0.01 seconds
```

The X statement causes the DATA step to "wait" for 5 seconds, but only once, because it is executed while the step is compiled, not while it runs.

Output from the external command is routed back to the process that started SAS; this can be the scheduler in case of batch jobs, but with today's interactive clients this means that the output usually goes nowhere.

SYSTEM() FUNCTION AND CALL SYSTEM() ROUTINE

```
rc = system('command');
call system('command');
```

In contrast to the X statement, these are part of DATA step code and can be executed repeatedly and conditionally. Both work very similar, the function returns the exit code from the system. Both set the macro variable SYSRC, just like the X or %SYSEXEC statement, but only the result of the *last* call in a DATA step will be recorded. Output from the external commands does not go back to SAS in any way.

THE FILENAME PIPE METHOD

With the PIPE file reference, SAS provides a way to either feed data to an external process or read data from it.

So we can retrieve a directory listing with this:

```
filename dirlist pipe "ls -l";

data dirlist;
infile dirlist truncover;
input perms :$10. links owner :$8. group :$8. size mth :$3. day _time :$5.
name :$100.;
format mod_time e8601dt19.;
if perms ne "total";
if length(_time) = 5
then do;
  mod_time = dhms(
    input(put(day,z2.)!!mth!!put(year(today()),z4.),date9.),
```

```
      0,
      0,
      input(_time,time5.)
    );
    if mod_time > datetime() then mod_time = intnx('dtyear',mod_time,-1,'s');
  end;
  else mod_time = dhms(input(put(day,z2.)!!mth!!strip(_time),date9.),0,0,0);
  drop day mth _time;
  run;
```

But that is not all. It's not intuitive at first, but we can use the same method with an external command that does not produce output. *Normally* would not provide output:

```
filename copylist pipe "cp filea fileb 2>&1";

data _null_;
infile copylist;
input;
put _infile_;
run;
```

Normally, cp would not create any output, but if it fails, it will put a message to the stderr output stream; in the above example, the 2>&1 construct reroutes this to stdout, and now SAS can read it. If the command fails for any reason, we can see the error message in the SAS log. So now our previous "black box" turns into a very talkative entity if it has something to complain about.

There is one disadvantage to the FILENAME PIPE method: it does not set the automatic macro variable SYSRC like the X or %SYSEXEC statements do, nor does it return the exit code of the external command like the SYSTEM() function does. To retrieve the exit code, the external command must be expanded:

```
filename copycmd pipe "cp filea fileb 2>&1;echo $?";

data _null_;
infile copycmd end=done;
input;
put _infile_;
if done
then do;
  rc = input(_infile_,best.);
  put rc=;
end;
run;
```

The command line now has two separate parts, where the second captures the return code of the first and sends it to stdout, where we read it as the last row of the infile.

Let us apply the method to the problem covered in the first section, and use the operating system utility `find` to get our list of files:

```
filename filelist pipe "find $HOME/dollar/sas/ -type f 2>&1";

data filenames;
length fname $200;
infile filelist;
input fname;
run;
```

You can see how much simpler this code looks than the original macro, as it harnesses the power of a tool that was specifically designed to tackle a certain kind of task in an easy, quick way.

But what if we have a list of things to do, stored in a dataset (we always store our lists in datasets, don't we?), and want to run that in one quick step, getting all the information, and have everything coming from it once again in a dataset?

Then we need the

DYNAMIC PIPE

Suppose we want to apply the above find command to a series of locations; we have the locations in a dataset, and want to run the command for each location.

We will make use of several features of the INFILE statement that allows us to create a dynamic call of the external command, and let the DATA step do all the looping for us.

First, create a dataset of locations:

```
data locations;
input location $80.;
datalines;
/location1
/location2
/location3
;
```

Then, use a DATA step to create the system calls, catch the output, and allow us to check if something unusual happened:

```
data
  filenames (keep=location fname)
  results (keep=location rc)
;
length cmd fname $200;
set locations;
cmd = catx(' ',"find",location,'-type f','2>&1',';echo $?');
infile filelist pipe filevar=cmd end=end_of_pipe eof=done truncover;
start:
  input fname $200.;
  if not end_of_pipe
  then output filenames;
  else do;
    rc = input(fname,best.);
    output results;
  end;
  go to start;
done:
run;
```

The DATA step reads the table of locations, builds the command, and uses the FILEVAR= option to hand it to the INFILE PIPE. The EOF= option is used with a GO TO statement to create a loop for reading the output of the external command; this is necessary because, without the EOF= option, an input that reaches the end of the current pipe would automatically terminate the DATA step even if there were still observations in the input dataset. Once again, there is a second external command that displays the return code; when the end of the pipe is reached (we use the variable of the END= option to determine this), we record this return code in a secondary table (RESULTS).

We can now use any non-zero return code in the secondary table to extract the error message(s) from the table FILENAMES with a simple join.

A REAL-WORLD EXAMPLE

A poster on the SAS Communities needed to check for the existence of files on remote servers. This is the method I suggested, and it was accepted as the solution:

```
data datasets;
input servername :$100 dsname :$200;
datalines;
server_a /path/test1.sas7bdat
server_b /path/test1.sas7bdat
;

data results;
set datasets;
length command $200;
command = "ssh " !! strip(servername !! " test -f '" !! strip(dsname) !!
"'; echo $?";
infile dummy pipe filevar=command truncover eof=done end=last;
start:
  input line $100.;
  if last then result = input(line,best.);
  output;
  go to start;
done:
run;
```

CONCLUSION

Although SAS provides multiple ways of interacting with the underlying operating system without having to resort to external commands, there will be cases where the tools provide by SAS are not sufficient, or where using dedicated system commands will lead to a more efficient solution. The INFILE with the PIPE engine provides an interface to make the most of the responses of external commands.

The author therefore recommends to have XCMD enabled; security or privacy concerns should not play a role on properly set up and administrated servers, except in special cases

One thing that would be nice to have: if SAS set the automatic macro variable &SYSRC after using a PIPE.

REFERENCES

SAS Communities. “Set &SYSRC from FILENAME PIPE.” Accessed February 27, 2020. https://communities.sas.com/t5/SAS-Programming/Unix-Files-capture-through-SAS-dataset/m-p/624081/.

SAS Communities. “Unix Files capture through SAS dataset.” Accessed February 27, 2020. https://communities.sas.com/t5/SASware-Ballot-Ideas/Set-amp-SYSRC-from-FILENAME-PIPE/idi-p/579324.

ACKNOWLEDGMENTS

Most of the coding techniques presented here were brought to the author’s attention by other members of the SAS Communities. A big **Thank You** goes to all of them.

RECOMMENDED READING

- *The SAS® Communities (communities.sas.com)*
- *The SAS® Documentation (documentation.sas.com)*

CONTACT INFORMATION

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

Kurt Bremser
kurt57b@gmail.com
https://communities.sas.com/t5/user/viewprofilepage/user-id/11562

BASIC INSTRUCTIONS

WRITING GUIDELINES

Trademarks and product names

To find correct SAS product names (including use of trademark symbols), if you are a SAS employee, see the Master Name List. Otherwise, see SAS Trademarks.

- Use superscripted trademark symbols in the first use in title, first use in abstract, and in graphics, charts, figures, and slides.
- Do not abbreviate product names. For example, you cannot use "EM" for SAS® Enterprise Miner™. After having introduced a SAS product name, you can occasionally omit "SAS" for certain products, provided that your editor agrees. For example, after you have introduced SAS® Simulation Studio, you can occasionally use "Simulation Studio."

Writing style

- Use active voice. (Use passive voice only if the recipient of the action needs to be emphasized.) For example:

  The product creates reports. (active)
  Reports are created by the product. (passive)

- Use second person and present tense as much as possible. For example:

  You get accurate results from this product. (second person, present tense)
  The user will get accurate results from this product. (future tense)

- Run spellcheck, and fix errors in grammar and punctuation.

Citing references

All published work that is cited in your paper must be listed in the REFERENCES section.

If you include text or visuals that were written or developed by someone other than yourself, you must use the following guidelines to cite the sources:

- If you use material that is copyrighted, you must mention that you have permission from the copyright holder or the publisher, who might also require you to include a copyright notice. For example: "Reprinted with permission of SAS Institute Inc. from *SAS® Risk Dimensions®: Examples and Exercises*. Copyright 2004. SAS Institute Inc."
- If you use information from a previously printed source from which you haven't requested copyright permission, you must cite the source in parentheses after the paraphrased text. For example: "The minimum variance defines the distance between cluster (Ward 1984, p. 23)

TIPS FOR USING WORD

These instructions are written for MS Word 2007 and MS Word 2010. The steps are similar for MS Word 2003.

To select a paragraph style

1. Click the HOME tab. The most common styles in your document are displayed in the top right area of the Microsoft ribbon. If you don't see a style that you want, click the slanted down arrow at the bottom right corner of the Styles area, and scroll through the list. The main styles for this template are headings 1 through 4, PaperBody, and Caption. Avoid using other styles.
2. To change a paragraph style, click the paragraph to which you want to apply a style, and then click the style that you want in the ribbon.
3. PaperBody (used for most text) is automatically applied when you press Enter at the end of any heading style or the Caption style.

To insert a caption

1. Click **REFERENCES** on the main Word menu.
2. Click **Insert Caption**.
3. Select the **Label** type that you want.
4. Click **OK**.

To insert a cross-reference

1. Click **REFERENCES** on the main Word menu.
2. Click **Cross-reference**.
3. In the **Reference type** list box, select Heading, Figure, Table, Display, or Output.
4. For a heading:
    a. In the **For which heading** list, select the heading that you want.
    b. From the **Insert reference to** list, select **Heading text.**
5. For a figure, table, display, or output:
    a. In the **For which caption** list, select the caption that you want.
    b. From the **Insert reference to** list, select **Only label and number**.

To insert a graphic from a file

1. Click **INSERT** on the main Word menu.
2. Click **Picture**.
3. In the Insert Picture dialog box, navigate to the file that you want to insert.
4. When the name of the file that you want to insert is displayed in the **File name** box, click **Insert**.