

Always There When I Need You: Macros to Create Directories

Pete Lund
WA State Office of Financial Management
Forecasting Division

Abstract

There are many production processes which output data to directories which may not exist when the job is ready to run. The options are to create all the needed directories before you start the job, or have your SAS code do the work for you.

This paper discusses three macros which create directories "on the fly". The discussion begins with the very simple, a three-line macro which generates a directory and assigns a file reference. We'll conclude with a complicated recursive macro which can generate a directory many levels deep and has extensive error checking. The programs presented are 100% macro code, just to make it fun!

Introduction

All this began with a simple post in response to a question on SAS-L. Someone had asked about a simple way to create directories on the fly in a PC SAS job. I had a little macro I'd used in a number of situations which I passed along. In composing the response I realized that my solution may not meet every need and so a second, more robust, solution was added. The final version grew over the next few days.

The assumption for all of the macros presented here is that the user does not want to just create a directory, but also use it in the SAS program. All three macros have the same parameter list - a fully-qualified path and a file reference (fileref). All of them check for the existence of the specified path and create it if it does not exist. The specified fileref is then assigned to the path.

The Beginning

The first macro is the simplest of all. There is no error checking and the path must exist up to the highest level specified. For example, if the path *c:\mydata\june* is

passed, *c:\mydata* must already exist. The macro consists of only five lines:

```
1. %macro ChkDir(dname,fref);  
2. %if not %sysfunc(fileexist("&dname")) %then  
3. %let rc = %sysfunc(system(md "&dname"));  
4. %let rc = %sysfunc(filename(fref,&dname));  
5. %mend;
```

Let's look at the macro line by line, and assume the following call has been made:

```
%ChkDir(c:\mydata\june,prod)
```

1. As noted, the macro parameters are a fully-qualified path (DNAME) and a fileref (FREF). The above call is equivalent to the following FILENAME statement:

```
filename prod 'c:\mydata\june';
```

2. Check to see if the directory exists. The SCL function FILEEXIST¹ takes a directory name as an argument and returns TRUE (1) if the directory exists and FALSE (0) if not. This condition will be "true" if the directory does not exist - notice the %if **not**.... In that case, the conditional portion of the statement, in line 3, is executed.

3. If the directory does not exist, this part of the statement is executed. The SYSTEM function is used to create the directory, using the DOS MD (make directory) command.

Like the X command, the SYSTEM function spawns a DOS window to execute the specified command. You should set the SAS option NOXWAIT to force the window to close when the command completes.

¹ FILEEXIST and many other SCL functions are available in the dataset and to the macro facility (via the macro function %SYSFUNC). See Polzin and O'Connor (1996) and Lund (1997) for further discussion of %SYSFUNC. Also, see SAS Institute, Inc. (1995) for a complete list of SCL functions and their use.

The SYSTEM function will return 1 if the command was executed successfully, and 0 if not. The value is stored in the macro variable RC, which in this case is never evaluated.

4. Finally, the SCL FILENAME function is used to assign the fileref to the directory. This function is equivalent to the filename statement.

Similar to the SYSTEM function, the FILENAME function returns 1 if the function executes successfully and non-1 if not. This value is stored in the macro variable RC (over-writing the value placed by the SYSTEM function) and is also not evaluated here.

An important note about the argument list to the filename function when it is used in a %SYSFUNC call. The fileref argument must be a macro reference without the ampersand. In this case it is the macro parameter *fref*. The directory argument is a string, without quotes. In this case it is the macro parameter *dname*.

This macro works well in a controlled setting, one in which you always know that the lower level directories exist.

Adding a Little Depth

The next macro goes one step further and will create all levels of a path necessary to create the specified directory. In our earlier example, if *c:\mydata\june* was passed and neither level existed, the macro would create *c:\mydata* followed by *c:\mydata\june*.

Here's the code, again enumerated for explanatory notes:

```
1 %macro ChkDir(dname,fref);
2   %if not %sysfunc(fileexist("&dname")) %then
3     %do;
4       %let revdn = %sysfunc(reverse(&dname));
5       %let newdn = %sysfunc(reverse
6         (%substr(&revdn,%index(&revdn,')+1)));
7       %ChkDir(&newdn,&fref);
8       %let rc = %sysfunc(system(md "&dname"));
9   %end;
10 %let rc = %sysfunc(filename(fref,&dname));
11 %mend;
```

The macro is basically the same as the one above, except that if the specified directory does not exist the highest level subdirectory is stripped off and the macro is called again to check for the existence of that directory. This continues until an existing directory is

found, after which all the previously checked subdirectories are created.

Let's look at where things have changed from above:

4. The easiest way to remove the high-level directory is to look for the last backslash (\) and trim the directory string at that point. However, it's much easier to find the first occurrence of a character than the last. So, the first thing we'll do is reverse the characters in the string and look for the first backslash. In our above example, *c:\mydata\june* becomes *enuj\atadym\c*.

5. The next line has two purposes. The inner %SUBSTR function looks for the first backslash, with %INDEX, and creates a string beginning with the next character. In our example, *atadym\c*. The outer reverse function simply reverses the characters in that string. We are thus left with *c:\mydata*. Just what we wanted.

6. We now want to try and create that directory. So, we pass it to macro and try again. Keep in mind that the initial call to the macro hasn't finished, it's just suspended at this point and will continue to the next line when this second call is finished. Each call to the macro has its own memory space the values of the macro variables are local to the current call.

7. Remember, control of one invocation of the macro returns to this point after completion of a subsequent call. The attempt to create this directory the first time through the macro (in line 3) failed, so we need to call the SYSTEM function again after all lower levels of the path are created.

Appendix A tracks the progress of the macro through three recursions.

Checking Before Progressing

The final version of the macro expands on the recursive version by adding some error handling and reporting. An enumerated copy of the macro code appears in Appendix B. The error handling sections is meant to demonstrate a type of situation which may cause some problems and each could be expanded if necessary. Let's examine each section briefly:

1. Check to make sure that the directory passed does not contain any restricted characters.
2. Check to make sure that the fileref meets the conditions for a valid file reference. In this case, we've checked that the length is not greater than 8 characters.

We could also check that the string began with a letter and didn't contain any restricted characters.

3. Check that the drive exists.

4. This is basically the same code as above. There is one addition to the call to the macro. Notice that there is an additional parameter, recurs. This parameter is a named parameter that is not passed in the original user call and defaults to 0. All recursive calls set the value of recurs to 1. This value is used to limit assignment of the file reference to the originally passed path.

5. Check to see that the file reference was successful. If not, send a message to the log that the fileref was not assigned.

6. If the file reference was successful, write a message to the log displaying the fileref and path. To display the path information the directory is "opened" with the DOPEN function and the path is obtained with the DINFO function.

This may seem like a round-about way to get the information we already passed to the macro in the &dnname parameter. This is true if a fully qualified path was passed. However, if a relative path was passed it will be created under the current working directory. This method of obtaining the path information will display the complete path to the log.

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.

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

Appendix A - Tracking the Recursive Macro

Assume we call the simple recursive macro with the following and that c:\a already exists:

```
%ChkDir(c:\a\b\c, test)
```

Execution would proceed as follows:

Call	Line	Action
1	2	Check for c:\a\b\c - doesn't exist
	3	Flip to c\b\c
	4	Trim and flip to c:\a\b
2	5	Call macro with c:\a\b
	2	Check for c:\a\b - doesn't exist
	3	Flip to b\c
3	4	Trim and flip to c:\a
	5	Call macro with c:\a
	2	Check for c:\a - exists
2	8	Set fileref TEST to c:\a
	6	Try to create c:\a\b - OK
	8	Set fileref TEST to c:\a\b
1	6	Try to create c:\a\b\c - OK
	8	Set fileref TEST to c:\a\b\c

Appendix B - Error Handling Recursive Macro

```
%macro ChkDir(dname,fref,recurs=0);

%global ChkDirRC;
%local DrivePos Drive rc RevDN NewDN did FOption;

%if %sysfunc(indexc(&dname,/?*<>|%str(%))) gt 0 %then
%do;
%put ERROR: File name cannot contain /?*<>|%str(%);
%goto quit;
%end;

%if %length(&fref) gt 8 %then
%do;
%put ERROR: Fileref &fref is longer than 8 characters;
%goto Quit;
%end;

%let drivepos = %index(&dname,:);
%if &DrivePos gt 0 %then
%do;
%let drive = %substr(&dname,1,&drivepos);
%if %sysfunc(fileexist(&drive)) eq 0 %then
%do;
%put ERROR: Drive &Drive does not exist;
%goto quit;
%end;
%end;

%if not %sysfunc(fileexist("&dname")) %then
%do;

%let revdn = %sysfunc(reverse(&dname));
%let newdn = %sysfunc(reverse(%substr(&revdn,%index(&revdn,\)+1)));

%ChkDir(&newdn,&fref,recurs=1);

%put CREATING DIRECTORY: &dname;
%let rc = %sysfunc(system(md "&dname"));
%end;

%if recurs eq 0 %then
%do;
%let rc = %sysfunc(filename(fref,&dname));
%if %sysfunc(fileref(&fref)) ne 0 %then
%do;
%put ERROR: Fileref not assigned;
%goto quit;
%end;
%else
%do;
%let did=%sysfunc(dopen(&fref));
%let foption=%sysfunc(doption(&did,1));
%put FILEREF (%upcase(&fref)) ASSIGNED: %sysfunc(dinfo(&did,&foption));
%let ChkDirRC=%sysfunc(dclose(&did));
%end;
%end;

%quit:;
%mend;
```

1 - check to make sure that the directory passed does not contain any restricted characters.

2 - Check to make sure that the file reference is valid. This section could be expanded.

3 - if a drive is passed, check to make sure it exists.

4 - trim off the highest directory level and call the macro again. Set the recurs flag to 1 (see *RECURS Note* below).

5 -check to see that the file reference was successfully assigned.

6 - display the file reference and path to the log.

RECURS Note - the macro parameter is not assigned by the user. It is set automatically to 0 on the initial call and set to 1 on all recursive calls. This allows for the assigning of the file reference to only the initially specified directory.