Paper 835 - 2017

Building Intelligent Macros: Using Metadata Functions with the SAS® Macro Language

Arthur L. Carpenter
California Occidental Consultants, Anchorage, AK

ABSTRACT

The SAS macro language gives us the power to create tools that to a large extent can think for themselves. How often have you used a macro that required your input and you thought to yourself "Why do I need to provide this information when SAS already knows it?" SAS may well already know what you are being asked to provide, but how do we direct our macro programs to self-discern the information that they need? Fortunately there are a number of functions and other tools within SAS that can intelligently provide our programs with the ability to find and utilize the information that they require.

If you provide a variable name, SAS should know its type and length; given a data set name, the list of variables should be known; given a library or *libref*, the full list of data sets that it contains should be known. In each of these situations there are functions that can be utilized by the macro language to determine and return these types of information. Given a *libref* these functions can determine the library's physical location and the list of all the data sets it contains. Given a data set they can return the names and attributes of any of the variables that it contains. These functions can read and write data, create directories, build lists of files in a folder, and even build lists of folders.

Maximize your macro's intelligence; learn and use these functions.

KEYWORDS

Metadata, macro language, DATA step functions, %SYSFUNC, OPEN, ATTRN, ATTRC, FETCH, CLOSE

INTRODUCTION

There are a number of ways for a macro to determine information about the operating system, the SAS environment, libraries and the files that they contain, data sets and their variables, and variable attributes and the values that they contain (Carpenter, 2016). Among others these include the use of the X statement (and related statements and functions), DICTIONARY tables, SASHELP views, data sets created by PROC CONTENTS, system options, and automatic macro variables (Carpenter and Rosenbloom, 2016). Of these various approaches DATA step functions tend to be fastest and as a bonus they support the creation of user defined macro functions.

There are also a number of DATA step functions that can be used to obtain information about the SAS environment. Although used less often in the DATA step itself, these functions are invaluable to the macro programmer. Collectively they are often referred to as metadata functions, because they tend to return information about data sets and locations. In fact they actually have a much broader list of capabilities that extends beyond the metadata. These functions can also read and write data; create, count, and eliminate both directories and files; return path and location information; and much more. Of these functions, some of the more commonly used are highlighted here.

Within the macro language, these DATA step functions are typically accessed using the %SYSFUNC or %QSYSFUNC macro functions. For each of these functions, the first argument is the DATA step function (along with its own arguments) that is to be executed, and an optional format in the second argument to control the appearance of the value returned by the function in the first argument. In this example of a TITLE statement, the DATE function is called. It returns the current date as a SAS date, and this value is then formatted using the WORDDATE18. format. The generated TITLE statement will then contain the formatted date.

```
title4 "%qtrim(%qleft(%qsysfunc(date(),worddate18.)))";
```

```
title4 "October 22, 2016";
```

Most of the macros in this paper which demonstrate these DATA step functions are themselves macro functions. As macro functions the macro itself returns a value. If you are unfamiliar with the coding techniques used to create user written macro functions, you may want to review Carpenter, 2002 for more detail than will be described here.

VIEWING DATA SET METADATA

Metadata is data about the data set. Each SAS data set automatically stores, as a part of the data set, information about the data set itself, hence metadata. This is the information that you see when you execute a PROC CONTENTS. Although of interest to the DATA step programmer, access to the metadata can provide huge benefits to the macro programmer.

Typically the CONTENTS procedure is used just to display the metadata, however it can also be used to store the metadata in a data set as well. This data set has one row per variable and it contains information on the individual variables as well as data set specific information such as the number of observations in the data set. A portion of the metadata data set generated by CONTENTS for the SASHELP.CLASS data set is shown below.

```
proc contents data=sashelp.class
              out=classvar
              noprint;
   run;
```

Metadata Surfaced by CONTENTS

Obs	LIBNAME	MEMNAME	MEMLABEL	TYPEMEM	NAME	TYPE	LENGTH	VARNUM	LABEL	FORMAT	FORMATL	FORMATD	INFORMAT	INFORML	INFORMD	JUST	NPOS	NOBS	EI
1	SASHELP	CLASS	Student Data		Age	1	8	3			0	0		0	0	1	0	19	V9
2	SASHELP	CLASS	Student Data		Height	1	8	4			0	0		0	0	1	8	19	V9
3	SASHELP	CLASS	Student Data		Name	2	8	1			0	0		0	0	0	24	19	V9
4	SASHELP	CLASS	Student Data		Sex	2	1	2			0	0		0	0	0	32	19	V9
5	SASHELP	CLASS	Student Data		Weight	1	8	5			0	0		0	0	1	16	19	V9

One row per variable per data set can be obtained by using `data=sashelp._all_`. This is similar to the information contained in the view SASHELP.VCOLUMNS which is shown in the next example.

Without executing a PROC CONTENTS, similar information can be found in the SASHELP view VCOLUMNS, however this view has one row per variable per data set per library known to SAS.

SASHELP.VCOLUMN

Obs	libname	memname	memtype	name	type	length	npos	varnum	label	format	informat	idxusage	sortedby	xtype	notnull	precision	scale	transcode
68	SASHELP	CLASS	DATA	Name	char	8	24	1					0	char	no	.	.	yes
69	SASHELP	CLASS	DATA	Sex	char	1	32	2					0	char	no	.	.	yes
70	SASHELP	CLASS	DATA	Age	num	8	0	3					0	num	no	.	.	yes
71	SASHELP	CLASS	DATA	Height	num	8	8	4					0	num	no	.	.	yes
72	SASHELP	CLASS	DATA	Weight	num	8	16	5					0	num	no	.	.	yes
73	SASHELP	CLASSFIT	DATA	Name	char	8	64	1					0	char	no			yes

Almost the same information can also be surfaced through the use of the SQL DICTIONARY table COLUMNS. The DESCRIBE statement can be used to list the names of the columns, while the SELECT statement can write the values held in the table.

```
options nolabel;
title1 "DICTIONARY.COLUMNS";
proc sql;
   describe table dictionary.columns;
   select *
      from dictionary.columns
         where memname='CLASS';
   quit;
```

Both SASHELP.VCOLUMN and DICTIONARY.COLUMNS are created when they are requested. This means that they will always contain current information. This also means that there can be a significant wait while these tables are created. The metadata functions described in this paper do not have this limitation.

DICTIONARY.COLUMNS

libname	memname	memtype	name	type	length	npos	varnum	label	format	informat	idxusage	sortedby	xtype	notnull	precision	scale	transcode
SASHELP	CLASS	DATA	Name	char	8	24	1					0	char	no	.	.	yes
SASHELP	CLASS	DATA	Sex	char	1	32	2					0	char	no	.	.	yes
SASHELP	CLASS	DATA	Age	num	8	0	3					0	num	no	.	.	yes
SASHELP	CLASS	DATA	Height	num	8	8	4					0	num	no	.	.	yes
SASHELP	CLASS	DATA	Weight	num	8	16	5					0	num	no	.	.	yes

Viewing the metadata is oftentimes sufficient, but what if during the execution of a program we need to gather some metadata item and use it in the program? How can we have our program do this dynamically? Yes we can! Enter the DATA step's metadata functions.

DATA SET METADATA FUNCTIONS

The DATA step metadata functions operate directly against the metadata of a SAS data set. This means that we can avoid the expenditure of resources that are needed to create the SASHELP.VCOLUMN view or the DICTIONARY.COLUMNS table. It also means that through the use of these functions we can dynamically gather and use metadata in our programs directly.

Since these are DATA step functions, when they are to be used with the macro language you will need to invoke them through the use of the %SYSFUNC or %QSYSFUNC macro functions. This allows the DATA step function to execute during macro execution and to return the value to the macro facility.

Opening the Metadata – making it available for our use

The data set's metadata is not automatically available for our use. Before we can access it we need to get "permission" to look at the data set's descriptor record which contains the metadata. We gain this access through the use of the OPEN function. This function checks to see if the data set is currently available for our use. If the data set is available, the OPEN function returns a non-zero data set identifier. We never really care what the identifier is, as long as it is not zero (which would indicate that we have been denied access to the data set). After we have utilized the information in the metadata we need to release the data set so that other programs or programmers can use it. We do this by closing our access through the use of the CLOSE function.

The macro %META shown here is just a shell that highlights the use of the OPEN and CLOSE functions. These functions will almost always be present when working with a data set's metadata.

```
%macro meta(dsn=class);
   %local dsid;
   %let dsid = %sysfunc(open(&dsn)); ❶
   %if &dsid %then . . . .

      <<<<. . . .macro statements. . . .>>>>

   %let dsid = %sysfunc(close(&dsid)); ❷
%mend meta;
```

❶ The OPEN function is used to gain access to the data set's metadata. When the data set is successfully opened for use, it returns a data set identifier, which, in this macro, is stored in the local macro variable &DSID. When the data set is not opened successfully the value returned is a 0. This allows us to test whether or not the metadata is available.

❷ After we have finished using the data set, it is closed with the CLOSE function. This function returns a 0 for success. In this example macro the identifier is also cleared by replacing it with the value returned by the CLOSE function.

The data set identifier that is returned by the OPEN function is used by many of the other metadata functions. Having a unique identifier associated with a given data set is necessary as you may wish to open the metadata of more than one data set at a time. Usually this identifier will be stored in a local macro variable. Remember although we need to store the value of the identifier, the value itself is rarely of specific interest.

Variable Information Functions

As the name implies, variable information functions return information about the variables in the data set. These functions are commonly used when you need to ask questions such as; "Is the ABC variable on this data set?", "Does the ABC variable have a format?", and "What is the type of the ABC variable?". The functions of this type all start with the letters 'VAR', however not all functions that start with 'VAR' fit the category.

Function Name	Returns the:
VARFMT	Variable's assigned format.
VARINFMT	Variable's informat.
VARLABEL	Label of a variable.
VARLEN	Length of a variable.
VARNAME	Name of a SAS data set variable.
VARNUM	Number of a variable's position.
VARTYPE	Data type of a SAS data set variable (C or N)

Table 1: Variable Information Functions

One of the most commonly accessed metadata attributes is information about variable names and the existence of variables on a data set. The two functions that apply specifically to the variable name is VARNAME and VARNUM. Given a variable number (position on the PDV), the VARNAME function returns the name of the variable. The VARNUM function is the opposite as it provides the position number given the variable name.

The macro %VAREXIST checks to see if a specific variable exists on a specified data set. The name of the variable of interest is passed to the VARNUM function, which in turn returns the variable's position. If the variable does not exist on the data set the VARNUM function returns a 0.

```
%macro varexist(dsn=class,varname=age);
%local dsid vnum;
%let vnum=0;
%let dsid = %sysfunc(open(&dsn));
%if &dsid %then %do;
   %let vnum = %sysfunc(varnum(&dsid,&varname)); ❶
   %let dsid = %sysfunc(close(&dsid));
%end;
&vnum ❷
%mend varexist;
```

❶ The first argument of the VARNUM function is the data set identifier (obtained from the OPEN function), and the second is the name of the variable of interest. In this example the value returned by the VARNUM function is stored in the local macro variable &VNUM.

❷ The macro %VAREXIST is written as a macro function. The value returned by VARNUM is passed out of the macro, and the macro is said to resolve to the returned value. This means that the macro call can be used in other statements, such as an %IF, to make decisions about further processing.

```
%if %varexist(dsn=&dset,varname=&var) %then %do;
```

Data Set Attribute Functions

When a data set is created or modified a number of attributes of the data set are stored in the metadata. These attributes include things like:

- when the data set was created
- how many variables it contains
- how many observations it contains (there can be more than one answer)
- data set size
- status of indexes, WHERE clauses, and passwords.

The two functions that return this type of attribute information are the ATTRC (returns character information) and ATTRN (returns numeric information) functions. Each of these functions can return a number of different attributes that can be selected by the user. These requests are made by specifying an attribute as the second argument to the function. Some of the available attribute request options are shown in Table 2.

Function Name	Attribute Request	Returns the:
ATTRC	COMPRESS	Compression status
	ENGINE	Name of the engine used to create the data set
	LABEL	Data set's label
	LIB	Location of the data set, the Library (*libref*) name
	MEM, DSNAME	Name of the data set
	SORTEDBY	List of BY variables used to sort the data
	TYPE	Data set type
ATTRN	CRDTE, MODTE	Datetime the data set was created or last modified
	ISINDEX, INDEX	Status on indexes for this data set
	NDEL, NLOBS, NLOBSF, NOBS	Number of observations in the data set based on various conditions (primarily whether or not to count those marked for deletion)
	NVAR	Number of variables in the data set
	various	Read, write, and alter password status

Table 2: Data Set Attribute Functions

The use of the ATTRN function is demonstrated in the macro %VARLIST which creates a list of variable names of a specified type (numeric or character) for the data set of interest. The user passes in the data set name and whether or not to select a specific type of variable. The macro then returns the list of variables (or a blank if no variables meet the criteria.

```
%macro varlist(dsn=, vtype=);
   %local varlist dsid i;
   %let vtype = %upcase(&vtype);
   %let dsid = %sysfunc(open(&dsn));

   %if &dsid %then %do;
      %do i=1 %to %sysfunc(attrn(&dsid,nvars)); ❶
        %if %sysfunc(vartype(&dsid,&i))❷=&vtype or &vtype= %then
             %let varlist=&varlist %sysfunc(varname(&dsid,&i)); ❸
      %end;

      %let dsid = %sysfunc(close(&dsid));
   %end;
   &varlist ❹
%mend varlist;
```

❶ The ATTRN function is used with the NVARS attribute to request the number of variables in the data set. The index (&I) for the %DO loop will then cycle from 1 to the number of variables. The loop index (&I) is used to identify a specific variable in both the VARTYPE function ❷ and the VARNAME function ❸.

❷ The VARTYPE function will return the type of the &Ith variable. The returned value is either a C or N and this value is compared to the requested type stored in &VTYPE.

❸ The VARNAME function returns the name of the &ith variable. The name of each variable that meets the criteria is added to the growing list stored in &VARLIST.

❹ Once the list of variables has been created it is passed out of the macro.

The SAS Log to the right shows how the %VARLIST macro could be used to create a list of variable names of varying types.

```
59   %put %varlist(dsn=sashelp.class,vtype=n);
Age Height Weight
60   %put %varlist(dsn=sashelp.class,vtype=c);
Name Sex
61   %put %varlist(dsn=sashelp.class);
Name Sex Age Height Weight
```

Because the number of observations in a data set is stored in the data set's metadata, the fastest way to determine the number of observations is to access the metadata directly using the macro language. The data set is opened and the ATTRN function is used with the NLOBS (number of non-deleted observations) to return the number of observations contained in the data set.

```
%macro obscnt(dsn);
%local nobs dsnid rc;
%let nobs=.;

%* Open the data set of interest;
%let dsnid = %sysfunc(open(&dsn)); ❺

%* If the OPEN was successful get the;
%* number of observations and CLOSE &dsn;
%if &dsnid %then %do;
     %let nobs=%sysfunc(attrn(&dsnid,nlobs)); ❻
     %let rc  =%sysfunc(close(&dsnid)); ❼
%end;
%else %do;
     %put WARNING: Unable to open &dsn;
     %put %sysfunc(sysmsg()); ❽
%end;

%* Return the number of observations;
&nobs ❾
%mend obscnt;
```

❺ The data set is opened. If the data set cannot be opened, this macro returns a dot (.) for the number of observations.

❻ The NLOBS attribute is used with the ATTRN function to return the number of non-deleted observations. This takes into account any observations that may have been marked for deletion during an interactive session using PROC FSEDIT or a similar tool.

❼ The data set is closed after retrieving the value of interest.

❽ If the data set is not available, a warning along with the reason (using the SYSMSG function) is written to the SAS Log.

❾ The number of observations is returned by the %OBSCNT macro.

Notice that all the macro variables created by the %OBSCNT macro are forced onto the local symbol table.

READING DATA USING FUNCTIONS

Although not a common requirement, it is possible to both read and write data held in a SAS data set using the macro language. Once a data set has been opened, you can read observations both sequentially and using random access techniques.

Function Name	Action:
CUROBS	Current observation number
FETCH	Reads the next non-deleted observation from a SAS data set into the Data Set Data Vector (DDV)
FETCHOBS	Reads a specified observation from a SAS data set into the Data Set Data Vector (DDV)
GETVARN	Returns the value of a numeric variable
GETVARC	Returns the value of a character variable
NOTE/DROPNOTE	NOTE stores a unique observation ID number
POINT	Locates the observation identified by NOTE
REWIND	Returns the observation pointer to the beginning of the data set
CALL SET	Links data set variables to macro variables of the same name

Table 3: Data Access Functions

The macro %SYMCHECK (Carpenter, 2016, Exp. 9.2.1b) can be used to determine if a given macro variable is currently defined on a specific symbol table. The view SASHELP.VMACRO is used as input for the macro. As it is opened, a WHERE clause is used to limit the read to the specific row of interest.

```
%macro symcheck(mscope,mvname);
   %* Determine if a specific macro variable
   %* has been defined in a specific scope.;
   %local fetchrc dsnid rc;
   %let rc  = 0;
   %let dsnid  = %sysfunc(open(sashelp.vmacro  ❶
                  (where=(scope=%upcase("&mscope") and
                          name=%upcase("&mvname"))),i));
   %let fetchrc = %sysfunc(fetch(&dsnid,noset));  ❷
   %if &fetchrc eq 0  %then %let rc=1; ❸
   %let dsnid = %sysfunc(close(&dsnid));
   &rc ❹
%mend symcheck;
```

❶ The view is opened using a WHERE= data set option to limit the number of rows available – there should be exactly one row if the macro variable of interest exists within the specified scope.
❷ The FETCH function is used to determine if a row meets the criteria in the WHERE clause. The NOSET attribute specifies that the values stored in the view are not to be transferred to macro variables.
❸ A successful read indicates that the row exists, and the FETCH function returns a 0 for success.
❹ &RC will contain a 0 if the specified macro variable does not exist, and a 1 if it does. Regardless this value is returned by the macro.

The %M_ALL_DATA macro shown next, can be used to mimic the functionality of the DATA step's CALL EXECUTE routine. In this macro both the FETCHOBS function and the SET routine are used (Carpenter, 2016, Exp. 9.2.2d) to build the macro variables. In fact it creates a macro variable for each variable in the named data set, and these macro variables are then populated using the data in the data set.

```
%macro M_all_data(dsn=);
%local dsid i nobs;
%let dsid = %sysfunc(open(&dsn));

%let nobs= %sysfunc(attrn(&dsid,nlobs));
%syscall set(dsid); ❺
%do i = 1 %to &nobs; ❻
   %let rc=%sysfunc(fetchobs(&dsid,&i)); ❼
   %* Local process goes here; ❽
   %put ****** Observation &i *********;
   %put _local_;
%end;
%let dsid = %sysfunc(close(&dsid));
%mend m_all_data;
```

❺ The SET routine is used to match the names of the variables in the data set with the names of the macro variables to be created.
❻ The %DO loop is used to step through all of the observations in the incoming data set (there will be NLOBS of them).
❼ A FETCHOBS function is used to read the next observation from the data set. Because of the use of the SET routine ❺, the values of the variables read from the data set are automatically written to corresponding macro variables of the same name.
❽ A typically usage of this macro would be to call another macro at this point that would utilize the newly created observation specific macro variables.

For the first observation in the SASHELP.CLASS data set, the SAS Log shows that a local macro variable has been created for each variable in the incoming data set. The values of those macro variables correspond to the values of the data set variables.

```
20   %m_all_data(dsn=sashelp.class)
****** Observation 1 *********
M_ALL_DATA AGE 14
M_ALL_DATA DSID 1
M_ALL_DATA DSN sashelp.class
M_ALL_DATA HEIGHT 69
M_ALL_DATA I 1
M_ALL_DATA NAME Alfred
M_ALL_DATA NOBS 19
M_ALL_DATA RC 0
M_ALL_DATA SEX M
M_ALL_DATA WEIGHT 112.5
```

The important thing to note in this example is that for each variable in the data set, a macro variable has been created with the same name and value as the variable. This can be very advantageous when there are a large number of macro variables that need to be created, and even more especially so if we do not necessarily know the names of the macro variables. This is demonstrated in the following example which calls the macro %SHOE_RPT once for each observation in a control file, and demonstrates the utility of this approach.

A control file is constructed with the desired attributes. In this case the two variables in the WORK.SHOE_RPT data set will become the parameters that will be used with a reporting macro named %SHOE_RPT. This macro depends on two macro variables (&REGION and &PRODUCT).

VIEWTABLE: Work.Shoe_rpt

	region	product
1	Africa	Boots
2	Asia	Slipper

```
%macro shoe_rpt;
title Sales Report for &product in &region;
proc report
data=sashelp.shoes(where=(region="&region" and
                          product="&product"));
   column subsidiary sales;
   rbreak after / summarize;
   run;
%mend shoe_rpt;
```

To take advantage of the control data set (WORK.SHOE_RPT) the macro %M_ALL_DATA has been slightly modified at ❾. It will now call a macro with the same name as the name of the control data set (SHOE_RPT), and since it is called inside the %DO loop, it will be called once for each observation. For slippers sold in Asia the report to the left is generated.

Sales Report for Slipper in Asia

Subsidiary	Total Sales
Bangkok	$3,019
Seoul	$149,013
	$152,032

```
%macro M_all_data(dsn=);
%local dsid i nobs;
%let dsid = %sysfunc(open(&dsn));

%let nobs= %sysfunc(attrn(&dsid,nlobs));
%syscall set(dsid);
%do i = 1 %to &nobs;
   %let rc=%sysfunc(fetchobs(&dsid,&i));
   %&dsn ❾
%end;
%let dsid = %sysfunc(close(&dsid));
%mend m_all_data;
%m_all_data(dsn=shoe_rpt)
```

❾ The macro variable &DSN will resolve before the SAS attempts to call the macro. Because &DSN will resolve to SHOE_RPT %&DSN will resolve to %SHOE_RPT, which of course is interpreted as a macro call. The macro %SHOE_RPT will execute using the current values of &REGION and &PRODUCT, which are both updated for each iteration of the %DO loop and therefore for each observation in the control file.

In a blog Leonid Batkhan (Batkhan, 2016) uses the SET routine along with the FETCH function to read variable attributes.

WORKING WITH SAS FILES (LIBRARIES, DATA SETS, AND CATALOGS)

Creating and maintaining libraries of SAS files, like data sets and catalogs, can also be managed using functions. While these functions do not use the metadata of SAS data sets, they return information about the libraries – effectively metadata about the directory. These functions are often used in conjunction with those already described.

Function Name	Action:
CEXIST, EXIST	Whether or not the entity exists
LIBNAME	Establish a *libref*
LIBREF	Check for *libref* existence
PATHNAME	Return the physical path for a *libref* or *fileref*
RENAME	Rename a data set or catalog

Table 4: SAS File Functions

When you need to work with libraries of SAS files (data sets or catalogs) or with SAS files as entities, the functions in Table 4 can be of assistance. The LIBNAME and LIBREF functions can be used to establish or check the existence of a library. While the CEXIST and EXIST functions are used to determine whether or not a catalog or data set exists.

The PATHNAME function returns a physical path given a *libref* or a *fileref*. In this example a series of data sets are to be copied from a SAS data directory with a *libref* of PROJMETA into an Excel workbook using the PCFILES engine. Because we want to create the workbook in the same directory as the original data, the PATHNAME function is used to return the current path to the *libref* PROJMETA.

```
libname toxls pcfiles ❶
        path="%qsysfunc(pathname(projmeta))❷\MyExcelData.xls";

proc datasets nolist;
   copy inlib=sashelp outlib=toxls; ❸
      select class heart prdsale; ❹
   quit;

libname toxls clear; ❺
```

❶ The PCFILES interpretation engine is selected for the new *libref*.
❷ The PATHNAME function is used to return the physical path used by the PROJMETA *libref*.
❸ The PROC DATASETS COPY statement is used to point to the incoming and outgoing *libref*s.

❹ The SELECT statement specifies which data sets are to be copied.
❺ The *libref* must be cleared before the new workbook can be used.

In the previous example, because an existing *libref* is used, we know that its associated folder also exists. Often we will be given a folder or path to use and we will need to either verify that it is valid or to create if it does not already exist. Given a path/location, the %CHECKLIB macro uses the LIBNAME and LIBREF functions to establish or clear a *libref*.

```
%macro checklib(libref=,libpath=);
   %local rc;
   %if &libpath= %then %do;
      %* Clear this libref;
      %let rc=%sysfunc(libname(&libref)); ❻
   %end;
   %else %if %sysfunc(libref(&libref)) ❼ ne 0 %then %do;
      %* Establish this libref;
      %let rc=%sysfunc(libname(&libref,&libpath)); ❽
      %put %sysfunc(sysmsg());
   %end;
   %else %do;
      %sysfunc(sysmsg()); ❾
      %put WARNING: LIBREF not reassigned;
   %end;
%mend checklib;
```

❻ When only a *libref* is passed to the LIBNAME function, the *libref* is cleared.

❼ The LIBREF function determines whether or not the *libref* is defined. The function returns a non-zero value if the *libref* does not exist.

❽ When a second argument is present and the *libref* does not already exist , the LIBNAME function is used to establish the l*ibref*.

❾ The SYSMSG function is used to return error messages when the macro is unable to establish the *libref*.

WORKING WITH DIRECTORIES

There are a number of functions available that have been designed to work with directories or folders in much the same way that the functions discussed earlier work with data sets. These functions can be used to create new folders as well as to read the names of the files within a folder.

Common approaches to working with folders include the use of the X statement or one of its equivalents (%SYSTASK, CALL SYSTEM, etc.). Because these techniques tend to be OS dependent, they require the programmer to understand the language/commands of the OS sub session. For the Windows OS the sub session, these are DOS commands. You can learn more about DOS commands by using the help command or by pairing help with the name of a command to get command specific help.

```
x help;
x help md;
```

Here the X statement is used to create a Windows directory (using the MD command), and to write a list of SAS programs and data sets to a text file (using the DIR command).

```
x md "c:\temp\test"; ❶
x dir "c:\temp\*.sas" /o:n /b > "c:\temp\test\pgmlist.txt"; ❷
```

❶ The MD command is used to create a directory.

❷ The DIR command is used to write the names of the files that contain SAS in the extension to the text file PGMLIST.TXT.

The DATA step's file and directory functions allow us to accomplish the same basic types of tasks as the X statement without resorting to OS specific syntax and without stepping out of the macro language. Some of the more common functions of this type are shown in Table 5.

Function Name	Action:
DOPEN and DCLOSE	Open and close a directory
DCREATE	Create a directory or sub folder
DREAD	Read the names of items within a directory
DNUM	Returns the number of items in a directory
FILEEXIST	Checks existence of a file or directory
MOPEN	Open a file within a directory

Table 5: Directory and File Functions

The %CHECKLOC macro utilizes the FILEEXIST function to determine if a directory exists, and if it does not exist, the DCREATE function is used to create the directory. This macro is written as a macro function that returns the full path of the directory (whether it already exists or if it is created).

```
%macro CheckLoc(DirLoc=, DirName=); ❸
   %* if the directory does not exist, make it;
   %if %sysfunc(fileexist("&dirloc\&dirname"))=0 %then %do;❹
      %put Create the directory: "&dirloc\&dirname";
      %* Create the directory;
      %sysfunc(dcreate(&dirname,&dirloc)) ❺
   %end;
   %else %do;
      %put The directory "&dirloc\&dirname" already exists;
      &dirloc\&dirname ❻
   %end;
%mend checkloc;
```

❸ The upper portion of the path along with the folder name is passed into the macro.
❹ The FILEEXIST function is used to determine whether or not the specified folder already exists. This function returns a 0 when the folder does not already exist.
❺ The DCREATE function can be used to make a directory. The first argument is the directory name and the second is the upper portion of the path. Notice that the call to %SYSFUNC stands alone and is not a part of a complete statement. Because the DCREATE function returns the full path, this line will resolve to the full path, and it is this value that is passed out of the macro.
❻ When the directory already exists, the full path is returned at this point in the macro.

❼ Because %CHECKLOC is itself written as a function that returns an existing path, it can be used in a LIBNAME statement. Here %CHECKLOC establishes the path `c:\temp\test` (if it does not already exist), and it creates the *libref* TEMTEST.

```
libname temtest "%checkloc(dirloc=c:\temp, dirname=test)"; ❼
```

When you want to access the names of the files in a directory you will need to open and close the directory using the DOPEN and DCLOSE functions. Once the directory is opened, you can use the DREAD and DNUM functions to step through the files in the directory. The %FILLIST macro writes the names of all the SAS programs in a directory to the SAS Log.

```
%macro fillList(filerf=);
%local rc fid i fname;
%let fid = %sysfunc(dopen(&filerf)); ❽
%if &fid %then %do i = 1 %to %sysfunc(dnum(&fid)); ❾
   %let fname= %sysfunc(dread(&fid,&i)); ❿
   %if %upcase(%qscan(&fname,-1,.))=SAS %then %put &fname;
%end;
%let fid = %sysfunc(dclose(&fid)); ⓫
%mend fillist;

filename saspgms 'c:\temp';
%fillist(filerf=saspgms)
filename saspgms clear;
```

❽ The directory specified in the *fileref* passed into the macro is opened for use. The directory identification number is saved in the macro variable &FID.
❾ The DNUM function returns the number of files in the directory.
❿ The name of the &Ith file is returned by the DREAD function, and the names are written to the SAS Log.
⓫ After using the directory, it is closed.

Establishing a process for each file in a directory would be a fairly straightforward expansion of the %FILLIST macro. Since the %DO loop will execute once for each file in the directory, any process that depends on the name of the file will have sufficient information.

In the next macro %FILLIST is expanded in a couple of important ways. The critical elements of %FILLIST remain, however in this macro, instead of just a %PUT, we include a process (to convert all CSV files in a directory to SAS data sets using PROC IMPORT). Not only do we need to convert the CSV files in the current directory, but in all subdirectories as well. This can be accomplished using a technique known as recursion.

A recursive macro is a macro that calls itself. The macro %FILLIST can be expanded to search for all files of a given type across a directory, including sub-directories, by making it recursive. The macro %RECURSIVEIMPORTDATA shown here is a simplified version of a macro written by Phuong Dinh of Cornerstone Research Inc. A similar, albeit even more simplistic, version of this macro appears in the SAS 9.4 Macro Language reference manual. Ostensibly this macro converts all CSV files in a folder (and subfolders) to SAS data sets and appends them into a single data set, however the important take away is that it uses recursion and a series of directory and file functions to search the sub-directories as well.

In this macro each file in a directory is examined by passing the macro the path of the directory of interest. If the file is a CSV file it is converted to a SAS data set, however if the file is a subdirectory, the macro is called again, this time with the path to the subdirectory. Because it is recursively called, this macro will crawl through an entire directory including all levels of subdirectories.

```
%macro RecursiveImportData(folder=,levelcounter=0);
   %* written by Phuong Dinh of Cornerstone Research Inc.;
   %local _rc _dsid _i _s_ext _filename _ref;
   %let _ref=filrf&levelcounter; ❶
   %if %sysfunc(fileexist(&folder)) = 0 %then %do; ❷
      %put ERROR: Folder does not exist - &folder;
      %return;
   %end;
   %let _rc = %qsysfunc(filename(_ref, &folder)); ❸

   %if &_rc=0 %then %do;
      %let _dsid = %qsysfunc(dopen(&_ref)); ❹
      %if &_dsid ne 0 %then %do;
         %do _i=1 %to %qsysfunc(dnum(&_dsid)); ❺
            %let _filename = %qsysfunc(dread(&_dsid, &_i)); ❻
            %let _s_ext = %scan(&_filename, -1, .); ❼
            %if %upcase(&_s_ext)=CSV %then %do; ❽
               %put NOTE: Importing &folder\&_filename;
               proc import file="&folder\&_filename"
                           out=temp dbms=csv replace;
                  getnames=yes;
                  run;
               proc append base=bigdata
                           data=temp force;
                  run;
            %end;
            %else %if &_s_ext= &_filename %then %do; ❾
               %put This is a folder: &folder\&_filename;
               %recursiveImportData(folder=&folder\&_filename,
                                    levelcounter=%eval(&levelcounter+1))
            %end;
         %end;
      %end;
      %let _dsid=%qsysfunc(dclose(&_dsid)); ❿
   %end;
   %let _rc = %qsysfunc(filename(_ref)); ⓫
%mend recursiveImportData;
```

❶ A unique *fileref* name is created. A level counter is used so that when the macro is called recursively the *fileref* created by the inner macro will not replace a *fileref* that already exists. Because a fileref is restricted to 8 characters, this macro cannot accommodate more than 1,000 levels (the highest level defaults to &LEVELCOUNTER=0).
❷ The FILEEXIST function is used to check to make sure that the requested folder exists. If it does not exist a custom error message is written to the SAS Log and the macro terminates execution.
❸ The FILENAME function is used to establish the *fileref* for this folder. Notice that although the name of the *fileref* is stored in &_REF, the ampersand is not used with the FILENAME function.
❹ Assuming that the *fileref* (&_REF) is successfully established (&_RC=0), the folder designated by the *fileref* &_REF is opened for processing. The folder's identification number is saved in &_DSID.

❺ When the directory is successfully opened (%_DSID ne 0), a %DO loop is used to step through all of the files in the current folder. The number of files to be processed is returned by the DNUM function. The index of the %DO loop (&_I) will be used by the DREAD function.
❻ The DREAD function is used to read the name of the &_Ith file. The file's name is stored in the macro variable &_FILENAME.
❼ The extension of the current file name is extracted using the %SCAN function. If there is no extension the name of the file will be returned. This macro assumes that all files except folders have extensions.
❽ The file name has an extension of CSV. Import the CSV file, create a data set and append it to the growing data table.
❾ This file does not have an extension and is therefore assumed to be a sub-directory. The macro %RECURSIVEIMPORTDATA is called with the name of the subfolder and with a level indicator increased by 1.
❿ The directory is closed using the DCLOSE function.
⓫ The *fileref* for this folder is cleared. Remember that the macro variable &_REF is specified without the ampersand in the FILENAME function.

RELATED FUNCTIONS AND OTHER TOOLS

There are a number of less commonly used directory functions. Remember less commonly used does not necessarily mean less useful. These are most useful when you want to handle the directory as an entity.

Function Name	Action:
DINFO	Returns information about a directory
DOPTNAME	Returns directory attribute information
DOPTNUM	Returns the number of attribute items

Table 6: Other Directory Functions

The macro %DIRINFO can be used to show the directory information returned by the functions in Table 6. It has been my experience that, under Windows at least, only a limited amount of information is returned. It is possible that other operating systems and directory structures will return more useful information. Experiment - execute %DIRINFO on your OS.

```
%macro DirInfo(fileref=);
%local dirid i rc;
%let dirid = %sysfunc(dopen(&fileref));
%if &dirid %then %do i = 1 %to %sysfunc(doptnum(&dirid));
   %let dname= %sysfunc(doptname(&dirid,&i));
   %let dinfo= %sysfunc(dinfo(&dirid,&dname));
   %put &=i &=dname &=dinfo;
%end;
%mend dirinfo;
filename my2temp 'c:\temp';
%dirinfo(fileref=mytemp)
filename my2temp;
```

Much like the functions that can be used to read and write data in a SAS data set, there are also a number of similar functions that can be used to read and write information to and from non-SAS controlled files. Some of those functions are shown in Table 7.

Function Name	Action:
FOPEN and FCLOSE	Opens and closes a specific file
FREAD	Reads a row into the File Data Buffer (FDB)
FAPPEND	Appends the current record to an existing file
FCOL	Current position on the FDB
FGET	Retrieves item from the FDB
FWRITE	Writes a record to an external file
FDELETE	Deletes an external file
FPOINT	Contains number of the next row to read

Table 7: File I/O Functions

SUMMARY

There are a number of SAS DATA step functions that you will likely never use in the DATA step. However when paired with the macro language through the use of the %SYSFUNC macro function, these DATA step functions have extensive utility. Functions that are designed to work with the metadata of SAS data sets and OS folders are especially valuable. These functions allow us to retrieve, manipulate, and use information that in many cases would be otherwise unavailable to us; *and* we do not need to leave the macro language in order to take advantage of them!

ABOUT THE AUTHOR

Art Carpenter is a SAS Certified Advanced Professional Programmer and his publications list includes; five books and numerous papers and posters presented at SAS Global Forum, SUGI, PharmaSUG, WUSS, and other regional conferences. Art has been using SAS since 1977 and has served in various leadership positions in local, regional, and international user groups.

Recent publications are listed on my sasCommunity.org Presentation Index page.
http://sascommunity.org/wiki/Presentations:ArtCarpenter_Papers_and_Presentations

AUTHOR CONTACT

Arthur L. Carpenter
California Occidental Consultants
10606 Ketch Circle
Anchorage, AK 99515

(907) 865-9167
art@caloxy.com
www.caloxy.com

REFERENCES

Adams, John H., 2003, "The power of recursive SAS® macros - How can a simple macro do so much?", presented at the 2003 SAS User Group International Conference, SUGI28. http://www2.sas.com/proceedings/sugi28/087-28.pdf

Batkhan, Leonid, 2016, "Modifying variable attributes in all datasets of a SAS library", a SAS Blog Post, http://blogs.sas.com/content/sgf/2016/11/25/modifying-variable-attributes-in-all-datasets-of-a-sas-library/

Carpenter, Arthur L., 2002, "Macro Functions: How to Make Them - How to Use Them", Presented at the SAS User Group International Conference, SUGI27. http://www2.sas.com/proceedings/sugi27/p100-27.pdf.

Carpenter, Art, 2016, *Carpenter's Complete Guide to the SAS® Macro Language, Third Edition*; SAS Institute, Cary NC. http://support.sas.com/publishing/authors/carpenter.html

Carpenter, Arthur L. and Mary F. O. Rosenbloom, 2016, "I've Got to Hand It to You; Portable Programming Techniques", presented at the Midwest SAS Users Group Conference, MWSUG, Cincinnati, OH. http://sascommunity.org/wiki/I%E2%80%99ve_Got_to_Hand_It_to_You;_Portable_Programming_Techniques

There are a number of nice examples in the SAS 9.4 Macro Language Reference manual that have to do with reading files within directories, start here: http://support.sas.com/documentation/cdl/en/mcrolref/69726/HTML/default/viewer.htm#n02xowj8yuqfo4n0zzi98shu8qup.htm

TRADEMARK INFORMATION