Paper 834 - 2017

I've Got to Hand It to You; Portable Programming Techniques

Arthur L. Carpenter, California Occidental Consultants, Anchorage, Alaska

Mary F. O. Rosenbloom, Alcon, a Novartis Company, Lake Forest, California

ABSTRACT

As technology expands, we have the need to create programs that can be handed off – to clients, to regulatory agencies, to parent companies, or to other projects, and handed off with little or no modification by the recipient. Minimizing modification by the recipient often requires the program itself to self-modify. To some extent the program must be aware of its own operating environment and what it needs to do to adapt to it.

There are a great many tools available to the SAS® programmer, which will allow the program to self-adjust to its own surroundings. These include location-detection routines, batch files based on folder contents, the ability to detect the version and location of SAS, programs that discern and adjust to the current operating system and the corresponding folder structure, the use of automatic and user defined environmental variables, and macro functions that use and modify system information.

Need to create a portable program? We can hand you the tools.

KEYWORDS

Macro, %SYSGET, AUTOEXEC, %SYSFUNC, portable, SAS_EXECFILEPATH, Batch submissions, UNIX, Windows

INTRODUCTION

Programmers rarely have the opportunity to create a program from scratch using a blank slate. Instead, we save and reuse portions of code or entire programs or other utilities such as library macros. As technology grows, systems change, and we may find ourselves migrating our work into a new parent company or into a new operating system. Independent consultants often develop code on their own system, but are then required to deliver a package that the client can run in their computing environment, which may be dramatically different than the platform on which it was developed.

When you employ portable programming techniques, you are practicing defensive programming. When your program is delivered and before it can be put into production, other programmers may need to 'fine tune' it. Robust code requires fewer modifications and therefore fewer introduced problems. But how do you write programs that are portable? How do you construct programs that are portable enough that they can run in a variety of situations with minimal (or better yet without any) programmer intervention?

Portable programs may take advantage of several useful techniques, such as detecting the current file path, detecting the version and location of SAS, and detecting and temporarily modifying system options (Hughes, 2016). A portable program will often take advantage of automatic macro variables and automatic environmental variables to accomplish these tasks. It may use information about the programming platform or version of SAS to conditionally execute some options and functions only where they are available.

The techniques shown in this paper have been applied to a number of operating systems, OS, running various versions of SAS, but they have not been tested on every OS or for all versions of SAS. These macros were not designed to work with a remote submission process, which adds another layer of complexity, albeit a surmountable one. Not all of the macros in this paper have been tested on UNIX. Remember that the UNIX OS is case sensitive and coding everything in lower case is generally a good idea. It is likely that you will need to test and apply modifications to the techniques that are shown here, however they should give you good ideas of how to get started. This paper does not discuss portability issues associated with Enterprise Guide projects, see Billings (2015) for more information on EG portability.

I'VE GOT TO HAND IT TO YOU

To illustrate the use of these techniques, we have created an example scenario. Imagine that you are an independent consultant, and that your client, Acme Shoe Company, wants you to develop a system for them so that they can create weekly cumulative reports on their shoe sales around the world. You will develop your system on the current data provided in SASHELP.SHOES, but the client will update this data weekly and produce a new report. Your programs will be re-run by them in their statistical computing environment. You will develop your suite of programs, and then you've got to hand it over to them.

Unfortunately, you know very little about the statistical computing environment where your program suite will eventually be run. You don't know the location of the files, you don't know the location or version of SAS that will be run, and since the data will be updated, you don't know the number or names of reports that may be produced in the future. You know that the program will be run in Base SAS, but you don't know if it will be run interactively or in batch. You are going to need to take advantage of portable programming techniques!

The Generalization of a REPORT Step

First let's take some baby steps by working with a REPORT step that needs to be generalized. The following report has a number of requirements; an RTF document is created, the current date is in the title, the data location and the generating program are specified in footnotes. Clearly each of these requirements has dependencies that can change with where, when and how the program is executed.

```
ods rtf file="C:\PortableProject\Results\SalesRpt.rtf" ❶
        style=rtf;
title1 "Shoe Sales Report for August 20, 2016"; ❷
footnote1 h=1 j=r "Data Location: C:\PortableProject\Data"; ❸
footnote2 h=1 j=r "Executing Program: C:\PortableProject\SASCode\SalesRpt.sas"; ❹
proc report data=projdata.shoes ; ❺
column  region product sales inventory ;

define  region / group "Region" ;
define  product / group "Product" ;
define  sales / sum format= dollar12. "Total Sales" ;
define  inventory / sum format= dollar12. "Total Inventory" ;
run;
ods rtf close;
```

❶ The location of the report and how it is specified depends on the operating system.

❷ The current date should be automatically shown.

❸ The location of the data should be automatically detected and shown.

❹ The location of the executing program should be automatically detected and shown.

❺ The PROJDATA *libref* is constant, but depending on how the program is used, it will not always point to the same location ❸.

When considering portability issues, common ones include:

- Retrieving and inserting current date and time values
- Determining the operating system (OS)
- Detecting locations of data, programs, and SAS executables
- Adapting to different operating environments including operating oystems and Interactive versus Batch executions
- Programming techniques that increase portability

The portable version of this REPORT step replaces system dependencies and hardcoded information with macro calls that self-adjust according to the operating environment. The first few lines of this step are shown here and explained in the text below. Essentially in terms of your operating environment, as you program you must be asking yourself what parts of my program depend on what I know *now*, as opposed to what I will need to know in the *future*. Then write your program so that it can self-determine what it needs to know when the time comes.

```
ods rtf file="%checkloc(dirloc=%levelup(uppath=%execpath,up=1),dirname=results)%slash%str(SalesRpt.rtf)"
        style=rtf;
title1 "Shoe Sales Report for %currdate";
footnote1 h=1 j=r "Data Location: %qsysfunc(pathname(projdata))";
footnote2 h=1 j=r "Executing Program Location: %execprg";
footnote3 h=1 j=r "Executing Program Path: %execpath";
footnote4 h=1 j=r "Executing Program Name: %execname";
```

CAVEAT: It is anticipated that some modifications will be necessary to these macros if you will be using a non-directory based OS or earlier releases of SAS.

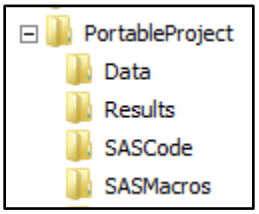
Using Macro Libraries

Generalization requires the use of macros and when making your code portable it is likely that there will be quite a few of them. This means that they need to be available when needed and it will at best be inconvenient to store the %MACRO to %MEND for each macro in the root program. As a general rule the easiest way to use and maintain of a series of macros is through the use of a macro library. There are three basic forms of macro libraries and each has its advantages and disadvantages (Carpenter, 2001). This author finds the autocall library to be the most flexible of the three. The example macros and batch commands used in this paper assume an AUTOEXEC.SAS has been executed. See the section on "Determining a Location", which describes the use of an autoexec file.

In the programs associated with this paper the AUTOEXEC.SAS program is used to establish an autocall library which contains the macro definitions needed in the examples.

Directory Structure

As we strive for portability we should also recognize that some consistency in structure, to the extent possible, will make our programming efforts substantially easier. The base structure used in the examples in this paper assumes a primary directory named "\PortableProject", with subdirectories to hold the data, results, the basic SAS code, and the SAS macros (autocall library). The root portion of this path can be specified in the AUTOEXEC (see below), however in order to establish portability all the programs in this project are completely independent of the directory structure above this level. An easy way to ensure this folder structure would be for us to supply our client with a zip file containing the 'Portable Project' folder and its subfolders, along with a readme.txt file. We will examine this at the end of this paper. There are several ways to create zip files via SAS, which may be helpful for large systems or for repeated tasks (see Hamilton, 2013 for one example). They can also be created manually.



RETRIEVING DATE VALUES

The title in the sales report should reflect the current execution date, however it is currently hardcoded and will require a change each time the report is executed. We need the ability to determine and insert today's date automatically.

```
title1 "Shoe Sales Report for August 20, 2016"; ❷
```

The user defined %CURRDATE macro returns today's date in WORDDATE form. The SAS date is retrieved using the DATE function, which is then formatted, trimmed, and left justified. This macro could easily be modified to use other date formats or to retrieve time or datetime values as well.

```
%macro currdate;
%qtrim(%qleft(%qsysfunc(date(),worddate18.)))
%mend currdate;
```

When used in the TITLE statement, this macro function assures that the date value displayed by the title will always be current, and it does it without user intervention.

```
title1 "Shoe Sales Report for %currdate";
```

DETERMINING THE OPERATING SYSTEM

When programs are moved from one platform to another they need to be adapted to the new operating environment. The first step in this process is determining the current operating system. The easiest way to do this is to take advantage of the automatic macro variables &SYSSCP and &SYSSCPL. The %PUT statement shown in this SAS Log was executed under the Windows (7 Pro) operating system for SAS 9.4. In the following LIBNAME statement &SYSSCP is used to determine the correct path. When the first argument is true (Windows OS), the second argument is returned, otherwise the non-windows path in the third argument is returned. In a 2016 SAS Dummy blog Chris Hemedinger also uses &SYSSCP in a similar usage.

```
45   %put &=sysscp &=sysscpl;
SYSSCP=WIN SYSSCPL=X64_7PRO
```

```
libname mydata "%sysfunc(ifc(&SYSSCP = WIN,
                            c:\portableproject\data,
                            /myloc/portproj/data))";
```

The macro %SLASH returns either a forward or backward slash depending on the OS. For a Windows application paths are designated using back slashes. The macro %SLASH has been written to mimic the properties of a macro function, and the macro call resolves to the value that is to be passed back. You can read more about writing macro functions in Carpenter (2002). A %PUT executed using windows shows that the %SLASH resolves to a back slash in the SAS Log.

```
%macro slash;
%qsysfunc(ifc(&sysscp = WIN,\,/))
%mend slash;
```

```
53   %put &=sysscp |%slash|;
SYSSCP=WIN |\|
```

Notice that this macro does not create any macro variables. A macro programming best practice is to define macro variables in the most local library where they can be used. In this case, the macro program %SLASH will be used in filename expressions (ODS), *libref* declarations, and more. The macro will resolve to the environmentally appropriate slash, and following the resolution of the macro, the slash will not be stored.

DETERMINING LOCATION

One of the hardest and perhaps one of the more pervasive issues when dealing with these types of problems is location detection. This problem is aggravated with changes to operating environments. We need to be able to determine things like the location of the data, where to write the results, the location of the SAS executable files, and where the currently executing program resides.

Hard Coding a Path in the AUTOEXEC.SAS Program

One of the easiest ways to control or specify path information, when it is known, is through the use of an AUTOEXEC program. The AUTOEXEC.SAS is a SAS program that is automatically executed at the initiation of the SAS session or batch job. In this program it is straightforward to specify global macro variables, system options, and autocall macro libraries.

```
%global path;
%let path=C:\PortableProject; ❻

libname projdata "&path\data"; ❼
filename sasmac  "&path\sasmacros"; ❼
options nodate nonumber; ❽
options mautosource sasautos=(sasmac, sasautos); ❾
```

❻ The macro variable (&PATH) containing the upper portion of the path is specified. Although it is safest to make this macro variable read only, we will avoid the use of %GLOBAL/READONLY here as this syntax is not available in versions of SAS before SAS 9.4.

❼ &PATH is used to specify *every* definition of every *libref* and *fileref* in the application or suite of programs.

❽ System options are specified.

❾ The autocall macro library is established.

The first few lines of the sales report program shown above can now be simplified by replacing the root portion of the hardcoded path information with the macro variable &PATH. When this program is transferred to another location only the path in the AUTOEXEC.SAS needs to be changed.

```
ods rtf file="&path\Results\SalesRpt.rtf" ❶
        style=rtf;
title1 "Shoe Sales Report for August 20, 2016"; ❷
footnote1 h=1 j=r "Data Location: &path\Data"; ❸
footnote2 h=1 j=r "Executing Program: &path\SASCode\SalesRpt.sas"; ❹
```

This may be all that you need to do to make your programs more portable, but it still requires user intervention (to change the path in the AUTOEXEC.SAS program), and it does not adjust to changes in OS.

The AUTOEXEC.SAS program shown here could also be executed through a %INCLUDE statement instead of as an autoexec. While Art prefers the use of the autoexec initialization option, this is not always a possibility or even feasible in some operating environments. Fortunately the same result would be accomplished if the AUTOEXEC.SAS shown above were executed through the use of a %INCLUDE statement at the start of your program.

An alternative to the user specification of &PATH is to add a location-detection macro to the AUTOEXEC.SAS program. We will discuss this later.

Using the PATHNAME Function to Return a Physical Location

When you need to determine a physical location and already have a known *libref* or *fileref*, you can use the PATHNAME function to retrieve the physical location. In this example we want to convert the current data set into an Excel table through the use of a DATA step. To do this we need to establish a new libref using an EXCEL engine and in this case we want it to point to the same physical location as the existing data.

The PATHNAME function returns the current location specified in PROJDATA l*ibref.* The value or path returned by the PATHNAME function will be appropriate for the OS on which the function is used. This means that when you know a *libref* or a *fileref* and need the location information this function will return the appropriate value. In the sales report shown above, one of the footnotes is to display the location of the data. The hardcoded location can be replaced with a call to the PATHNAME function. The footnote will no longer need to be changed if the location of the PROJDATA *libref* (established at ❼ above) is changed.

```
footnote1 h=1 j=r "Data Location: C:\PortableProject\Data"; ❸
```

```
footnote1 h=1 j=r "Data Location: %qsysfunc(pathname(projdata))";
```

When you need to build a longer path based on existing path information you will probably need to append information onto the path returned by PATHNAME. Since under Windows the path elements are separated with back slashes and front slashes in UNIX, you will also need to know which to use. While the value returned by PATHNAME will be appropriate for the operating system executing the step, the programmer will not necessarily know where the program will be executed. In this example the %SLASH macro described above is used to provide the OS dependent forward or backward slash. The %STR quoting function allows us to append the file name onto the resolved slash. For Windows the resultant path might be specified as: `C:\PortableProject\data\shoes.xls`.

```
libname toxls pcfiles
        path="%qsysfunc(pathname(projdata))%slash%str(shoes.xls)";
data toxls.shoes;
   set projdata.shoes;
   run;
libname toxls clear;
```

Returning the Name and Location of the Executing Program

At the operating system level, environmental variables are used in a similar manner to macro variables within SAS. We can access these OS level environmental variables through the use of the %SYSGET function. When executing a SAS program interactively under Windows, the environmental variables SAS_EXECFILENAME and SAS_EXECFILEPATH take on the value of the currently executing SAS filename and currently executing SAS file path (including filename). We can access these values from within interactive SAS. When a program is run in batch mode, or on a UNIX platform, these

environmental variables will not be available, but for batch execution the path and name of the currently executing file can be obtained from the SYSIN system option. The GETOPTION function allows us to obtain the value of SYSIN.

In the sales report's second footnote, we would like to list the name and location of the program that is generating the report. Since your program will likely have been moved into another computing environment, the program itself will need to be able to detect its own location. How the program determines its own name and location depends on whether the program is being executed interactively or through a batch process.

```
footnote2 h=1 j=r "Executing Program: C:\PortableProject\SASCode\SalesRpt.sas"; ❹
```

The %EXECPRG macro detects whether the program is being executed with a batch process or as a part of an interactive session and then uses this information to retrieve the path and name of the executing program. Notice that this macro has no input parameters. Because all of the inputs are either environmental variables or system options, we can simply place a call to %EXECPRG anywhere that we want to reference the path of the currently executing program.

```
%macro ExecPrg;
%if %sysfunc(getoption(sysin)) ne %str() %then %do;
      /* Batch Execution */
   %sysfunc(getoption(sysin))
%end;
   %else %do;
      /* Interactive Execution */
      %sysget(SAS_EXECFILEPATH)
   %end;
%mend execprg;
```

For batch processes this information is stored in the SYSIN system option which can be retrieved through the use of the GETOPTION function. Otherwise the %SYSGET macro function is used to retrieve the value from the environmental variable SAS_EXECFILEPATH. Either way this macro returns the full path and name of the executing program.

This macro allows us to replace the hardcoded file path in ❹, with a call to %EXECPRG. The new footnote statement is shown to the right:

```
footnote2 h=1 j=r "Executing Program: %execprg";
```

Under windows the returned path may include a mapped drive. When you need to determine the physical drive location (UNC path) of a mapped drive it is possible to do so, and the process for determining the UNC path is described in Carpenter (2008).

Returning Just the Name or Location of the Executing Program

The %EXECPRG macro, which was derived from the %GRABPATH macro (Carpenter, 2008) returns the whole path of the executing program. You will sometimes, however, want to retrieve or use the name and path as two distinct pieces of information. The macros %EXECNAME and %EXECPATH can be used to separately determine the name and path of the executing program. These two macros, shown below, are simple extensions of the %EXECPRG macro shown above.

The %EXECNAME macro extracts the full path using the same techniques as the %EXECPRG macro and stores it in a local macro variable (&FULLNAME). The %QSCAN function is then used to extract the last two words from the path. This macro will require slight modification if your OS uses single word file names, or if the path separators are other than slashes and dots.

```
%macro ExecName;
   %local fullname;
   %if %sysfunc(getoption(sysin)) ne %str() %then %do;
      /* Batch Execution */
      %let fullname=%sysfunc(getoption(sysin));
   %end;
   %else %do;
      /* Interactive Execution */
      %let fullname = %sysget(SAS_EXECFILEPATH);
   %end;
   /* Return the program name without the path*/
   %qscan(&fullname,-2,%str(\/.)).%qscan(&fullname,-1,%str(\/.))
%mend execname;
```

The %QSCAN function is used to retrieve the program name from the full path. The negative word number causes %QSCAN to read from the right and the third argument contains the three potential word delimiters of interest.

Similar to %EXECNAME, the %EXECPATH macro returns only the path (without the program name). The %QSUBSTR macro function is used to separate the program name from the path.

```
%macro ExecPath;
   %local fullname name_len full_len;
   %if %sysfunc(getoption(sysin)) ne %str() %then %do;
      /* Batch Execution */
      %let fullname=%sysfunc(getoption(sysin));
   %end;
   %else %do;
      /* Interactive Execution */
      %let fullname = %sysget(SAS_EXECFILEPATH);
   %end;
   /* Length of the name only */
   %let nam_len=%length(%qscan(&fullname,-2,%str(\/.)).%qscan(&fullname,-1,%str(\/.)));
   /* Length of the whole path - including the name */
   %let full_len=%length(&fullname);
   /* Return the path without the program name */
   %qsubstr(&fullname,1,&full_len-&nam_len-1)
%mend execpath;
```

The footnotes in the reporting program could now reflect the use of these two path and location macros.

```
footnote1 h=1 j=r "Data Location: %qsysfunc(pathname(projdata))";
footnote2 h=1 j=r "Executing Program Location: %execprg";
footnote3 h=1 j=r "Executing Program Path: %execpath";
footnote4 h=1 j=r "Executing Program Name: %execname";
```

Data Location: C:\PortableProject\data
Executing Program Location: C:\PortableProject\SASCode\SalesRpt_Auto.sas
Executing Program Path: C:\PortableProject\SASCode
Executing Program Name: SalesRpt_Auto.sas

ASIDE: Some macro programmers have the habit of ending every macro call with a semicolon. Since the macro facility ultimately does text substitution, it is not necessary to end each call with a semicolon. In fact in these examples, you can see that if any of these macro calls were followed by a semicolon, the FOOTNOTE statement would fail to compile, let alone execute, and would result in an ERROR message in the SAS Log.

Adding a Location-Detection Macro to the AUTOEXEC.SAS Program

Previously we saw that one way to manage an AUTOEXEC.SAS program is to hard code the file location. Another option is to place a macro definition into the AUTOEXEC.SAS file, and then use that macro to determine the location automatically and then assign the value of &PATH for us. The macro %AUTOEXECPATH is used simply to determine the location of the AUTOEXEC.SAS file. It takes advantage of the view SASHELP.VEXTFL which notes the known external files, including the autoexec.

The macro %AUTOEXECPATH functions similarly to %EXECPATH (shown previously), although it creates &PATH inside of the macro. The biggest difference is the code used for the interactive execution. Although SAS_EXECFILEPATH and SAS_EXECFILENAME are indispensable to a portable programmer for interactive SAS, they are not available for use with the AUTOEXEC.SAS file (see Usage Note 36613). However, through the use of the SASHELP.VEXTFL view and a DATA step (see Usage Note 24301), we can obtain the path information that we need.

```
%macro AutoExecPath;
   %local fullname name_len full_len;

   /* Batch Execution */
   %if %sysfunc(getoption(sysin)) ne %str() %then %do;
      %let fullname=%sysfunc(getoption(sysin));
   %end;
      /* Interactive Execution */
      %else %do;
         /* The VEXTFL view contains a list of known external files. */
         /* This list includes the autoexec that is currently executing. */
         data _null_; ❶
            set sashelp.vextfl;
            if (substr(fileref,1,3)='_LN' or substr
               (fileref,1,3)='#LN' or substr(fileref,1,3)='SYS') and
               index(upcase(xpath),'AUTOEXEC.SAS')>0 then do; ❷
                  call symputx("fullname",xpath); ❸
                  stop;
            end;
         run;
      %end;

   /* Length of the name + next higher level */
   %let nam_len=%length(%qscan(&fullname,-3,%str(\/.)).%qscan(&fullname,-
        2,%str(\/.)).%qscan(&fullname,-1,%str(\/.))); ❹
   /* Length of the whole path - including the name */
   %let full_len=%length(&fullname);
   /* Return the path without the program name */
   %let path=%qsubstr(&fullname,1,&full_len-&nam_len-1); ❺
%mend AutoExecPath;
```

❶ A DATA step is used to read the information about the external files.

❷ We are interested in the row that contains the path for the AUTOEXEC.SAS file.

❸ The full name and path is written to the macro variable &FULLNAME. Once we detect this file we are finished with the DATA step, and the STOP is executed.

❹ The path that is stored in &PATH is assumed to be one level higher than the location of the autoexec file. The length of the name and the next higher level is determined.

❺ The value of &PATH is the upper portion of the path less the name and the next higher level.

We can place this macro definition at the top of our AUTOEXEC.SAS program, and then we simply need to call it. ❻ Once established, we can then use &PATH as we did in the previous AUTOEXEC.SAS example.

```
%global path;

%AutoExecPath ❻

libname projdata "&path\data";
filename sasmac  "&path\sasmacros";
options nodate nonumber;
options mautosource sasautos=(sasmac, sasautos);
```

MANAGING LOCATIONS AND FOLDERS

Clearly, before our programs can write to a location, that location must exist. Perhaps your report will be written to the \RESULTS folder. Your program will need to determine if the location exists, and if not, it must be created. This process, of course, has to be done automatically. In the sales report program the path for the RTF output was specified directly. This statement depends on the availability of the \RESULTS folder and will fail if it does not already exist.

```
ods rtf file="C:\PortableProject\Results\SalesRpt.rtf"
        style=rtf;
```

Does the Location Exist?

One of the easiest ways to detect whether or not a location exists is through the use of the FILEEXIST function. This DATA step function returns a 1 when the location specified in its argument exists. A simple usage in a %PUT is shown to the left.

```
%put %sysfunc(fileexist(&fname));
```

This function can be used to test for the existence of a specific file or for a location. In the %CHECKLOC macro below it is used to determine whether or not a folder exists.

Creating a Location

When you need to create a location that does not already exist, you can use the OS-independent DCREATE function. The macro %CHECKLOC uses the FILEEXIST function to determine if the desired location already exists, and if the location does not exist it is created. This is one of several of ways to create a folder with SAS. Oftentimes, if the location already exists, it will not be re-created and any contents will be preserved, but we have chosen to build a location check into this macro.

The DREATE function takes two arguments. The first argument is the name of the root directory, stored in a variable, expression, or string, and the second is the name of the new subfolder(s), also stored in a variable, expression or string.

```
%macro CheckLoc(DirLoc=, DirName=);
   %local slash;
   %let slash=%slash;
   %* if the directory does not exist, make it;
   %if %sysfunc(fileexist("&dirloc&slash&dirname"))=0 %then %do;
      %put Create the directory: "&dirloc&slash&dirname";
      %* Make the directory and return the path;
      %sysfunc(dcreate(&dirname,&dirloc))
   %end;
   %else %do;
      %put The directory "&dirloc&slash&dirname" already exists;
      &dirloc&slash&dirname ❶
   %end;
%mend checkloc;
```

Our portable program code will take advantage of %EXECPATH (defined above) to dynamically pass in the parameter value of the root directory based on where the currently executing program resides. So, when our client moves this program suite to their statistical computing environment, the subfolder will be created in that location. The code from this example will work in both interactive SAS and in batch mode.

The %CHECKLOC macro function has the same two arguments as the DCREATE function: the base or root portion of the path (&DIRLOC) and the portion of the folder structure that is to be added (&DIRNAME). The macro returns the name of the completed path, which could then be used in a FILENAME or LIBNAME statement. ❶

In the example that we are considering in this paper, we would like to place the results of the sales report in a RESULTS folder under the same path as the program that is being executed to generate the reports. We know that the %EXECPATH macro returns the path for the location that contains the SAS code. For these examples the SAS programs reside in the folder `c:\portableproject\sascode`. If we want the results to be under this folder we could create this location using: `%CHECKLOC(dirloc= c:\portableproject\sascode, dirname=results)`, which would give us the path `c:\portableproject\sascode\results`. It is more likely that we will instead want to create the \RESULTS location at the same level as \SASCODE (up one level) rather than under \SASCODE. To do this we need to be able to move up the path and select a higher root portion. If we move up one level the \RESULTS can be at the same level as \SASCODE, and the path could now be specified as: `c:\portableproject\results`. We can move up the levels of the path through the use of the %LEVELUP macro.

The macro %LEVELUP accepts a path and a displacement and returns a path with the lower levels removed. Since we already have the ability to return the location of the executing program using the %EXECPATH macro, we can use %LEVELUP to determine the next higher level.

```
%macro levelUp(uppath=,up=1);
   %local up_len full_len lev;
   %if &up gt 0 %then %do;
      %let up_len=0;
      %do lev= 1 %to &up;
         /* Add the length of this level to total length to be eliminated */
         %let up_len=%eval(&up_len + %length(%qscan(&uppath,-&lev,%str(\/.))));
      %end;
      /* Return the path without the lower &up levels */
      /* Each eliminated level has a delimiter to remove as well */
      %qsubstr(&uppath,1,%length(&uppath)-&up_len-&up)
   %end;
   %else %do;
      /* No change requested return the incoming path */
      &uppath
   %end;
%mend levelup;
```

In this macro the %QSCAN and %LENGTH functions are used to determine the length of the text of the levels to be removed. This length along with the overall length is then used with the %QSUBSTR function to obtain the desired portion of the path.

The number of levels to eliminate is stored in the parameter &UP. When this number is zero the current path, with no levels removed, is returned. Although this macro does not check if the number of levels to be removed exceeds the overall number of levels, that check could easily be added.

Using the %LEVELUP macro, the new root portion of the path could now be written as:

`%levelup(uppath= c:\portableproject\sascode, up=1)`, which results in the path `c:\portableproject`. Of course this will be more practical when the path is not hardcoded. The explicitly specified path can be replaced with a call to %EXECPATH, and the call to %LEVELUP becomes: `%levelup(uppath= %execpath, up=1)`. Since the call to %LEVELUP returns the upper portion of the path that we want to use for the results, the call to %LEVELUP can now be used with the %CHECKLOC macro to ensure that the desired location exists. Using %LEVELUP with %CHECKLOC will be coded as: `%checkloc(dirloc=%levelup(uppath=%execpath,up=1),dirname=results)`. This call to %CHECKLOC will return a path that we know exists and is based on the location of the executing program. The ODS statement can now be modified to take advantage of this known location.

```
ods rtf file="%checkloc(dirloc=%levelup(uppath=%execpath,up=1),dirname=results)%slash%str(SalesRpt.rtf)"
        style=rtf;
```

DETECTING THE LOCATION AND VERSION OF SAS

There are several reasons why we sometimes want to detect the version and location of the SAS executable file. Your programs may have version specific code and if the client or server is running an older version of SAS, certain procedures and options will need to be avoided or replaced. You may also need to create programs that are to be executed in a batch process that requires the location of the SAS executable file.

Determining the SAS Version

If you need to know the currently executing version of SAS, there are three automatic macro variables (&SYSVER, &SYSVLONG, and &SYSVLONG4) that can be used to display the version in varying levels of detail. These are displayed in a SAS Log to the left. Of the three &SYSVER is the most basic and simply shows the version number. The other two show not only the version number, but the maintenance release and the corresponding release date (&SYSVLONG4 shows a four digit year for the release

```
386  %put &=SYSVER;
SYSVER=9.4
387  %put &=SYSVLONG;
SYSVLONG=9.04.01M0P061913
388  %put &=SYSVLONG4;
SYSVLONG4=9.04.01M0P06192013
```

date) as well. Because these are automatic macro variables, they will always be available and will always reflect the status of the currently executing version of SAS.

Retrieving the Location of the SAS Executable File

Batch execution of SAS requires access to the SAS executable file location. Because batch execution is common on UNIX, local scripts have generally been written to make this easier, however under windows, batch processing is less common. In either case, you will usually need to know, directly or indirectly, the location of the SAS executable file. This is the SAS.EXE file. The executable file is stored in the SAS root directory in the environmental variable SASROOT, and this location is both OS and SAS Version-dependent.

```
1    %put |%sysget(sasroot)|;
|C:\Program Files\SASHome2\SASFoundation\9.4|
2    %let sasloc=%sysget(sasroot)\sas.exe;
3    %put location of executable file is: &sasloc;
location of executable file is: C:\Program Files\SASHome2\SASFoundation\9.4\sas.exe
```

In the SAS Log shown here, the %SYSGET function is used to retrieve the folder location under a Windows system.

EXECUTING A BATCH FILE

Many programmers prefer to use some kind of batch file to execute a suite of programs, often in a specific order. To create a batch file for the Windows OS we create a text file, but with a file extension of .BAT rather than .TXT. To run the batch file, one would simply double click on it. To view the file contents, one would right click and 'Edit'.

The Batch Command Line

Within the batch file, we need to have one line of text for every SAS file that we want to execute. Each individual line must include the program to execute (SAS), the name of the SAS code to execute (your SAS program), and supplemental information like where to write the SAS Log or what autoexec program to use. Because batch command line is too long to fit on a single line in this paper, the batch command line has been broken up in the code to the right.

```
Start/w "Sales_Rpt" ❶
"C:\Program Files\SASHome2\SASFoundation\9.4\sas.exe" ❷
-sysin "C:\PortableProject\SASCode\SalesRpt_Auto.sas" ❸
-nosplash ❹
-log "c:\portableproject\sascode\SalesRpt_Auto.log" ❺
-autoexec "c:\portableproject\sascode\autoexec.sas" ❻
```

The batch command line can also contain initialization options. These are designated with a dash and a few of these are included in this example batch command.

❶ Start/w: a windows execution command submits the command line for execution. This command is not needed if the line is submitted from the RUN command line. The /W or /WAIT switch forces sequential execution when there are multiple batch executions requested. Each 'Job' should be named (here it is 'Sales Rpt'). Without the job name an interactive session is initiated when using Windows.

❷ the location of the SAS executable file is specified

❸ -SYSIN: Names the file containing the SAS program to be executed

❹ -NOSPLASH: Do not show the splash screen when SAS initiates

❺ -LOG: Redirect the SAS Log to this location. The -PRINT option (not used here) could be used to redirect the listing.

❻ -AUTOEXEC: Use this autoexec file as SAS initialization

Remember although this text box shows these options on separate lines they should be on a single line in the .BAT file.

Building the .BAT File

The single batch command shown in the previous text box will execute a single SAS program. What if you want to execute a series of SAS programs? Either you will need to create the .BAT manually (boring), or you will need to have the ability to have SAS build it for you. The following DATA step creates a .BAT file with one entry for each SAS program in the stated directory that matches a specified criterion.

```
data _null_;
   attrib Stmnt length = $32000 ❶
          name  length = $40;

   * Specify the fileref location;
   rc = filename('codedir', "%execpath"); ❷

   file "%execpath%slash%str(runall.bat)"; ❸

   * Open the directory of interest;
   dirid = dopen('codedir'); ❹

   * Loop through the list of items in the directory of interest;
   do i = 1 to dnum(dirid); ❺
      * Read the list of programs that match the criteria;
      name = dread(dirid,i); ❻
      if index(upcase(name),'_AUTO') and index(upcase(name),'SAS') then do; ❼
         * Create the batch command;
         stmnt=catt('Start/w "', scan(name,1,'.'),'"'); ❽
         stmnt=catt(stmnt,' "',%tslit(%sysget(sasroot)),%tslit(%slash),'sas.exe"'); ❾
         stmnt=catt(stmnt,' -sysin "',%tslit(%execpath),%tslit(%slash),name,'"');
         stmnt=catt(stmnt,' -nosplash -log');
         stmnt=catt(stmnt,' "',%tslit(%execpath),%tslit(%slash),scan(name,1,'.'),'.log"');
         stmnt=catt(stmnt,' -autoexec "',%tslit(%execpath),%tslit(%slash),'autoexec.sas"');
         put stmnt;
      end;
   end;
   * Close the directory;
   dirid = dclose(dirid); ❿

   * Remove the fileref;
   rc = filename('codedir',, 'clear'); ⓫
   run;
```

❶ The variable STMNT will temporarily hold the batch command. It can be no longer than 32k characters.

❷ A fileref is established to point to the directory that contains the programs to be executed. This DATA step function allows us to create the reference and then use it right away. The variable RC will take on the value of zero if this assignment is successful, but we won't actually use RC in this process.

❸ Establish the name and location of the batch file. In this case the file will be named RUNALL.BAT and will be written to the same location as the executing DATA step.

❹ The directory containing the programs of interest is opened for use by the DATA step using the DOPEN function. This function returns an identification number (DIRID) that is used by several other functions that access the information in this directory.

❺ The DNUM function returns the number of items in the directory identified by the DIRID variable.

❻ The DREAD function reads the name of the Ith item in the directory.

❼ For this particular example we are only interested in SAS programs that contain the characters '_AUTO'. You will want to consider some thoughtful nomenclature or other strategy to ensure that only the desired programs are incorporated into your .BAT file.

❽ The STMNT variable holds the batch command call for this SAS program. This first assignment statement adds the START/W command and the job name which in this case is the name of the executing program, surrounded with double quotes. Successive calls to the CATT function iteratively build the command one element at a time.

❾ The SAS executable file location and name is added to the batch command. Additional elements are added in the remaining STMNT assignment statements. %TSLIT is a SAS supplied autocall macro that surrounds text with single quotes.

❿ The open directory is closed.

⓫ The *fileref* pointing to the directory containing the SAS programs is cleared.

There will now be a file named RUNALL.BAT that can be double clicked to execute all the appropriate SAS programs in the directory. These same files could be executed through the Windows batch scheduler by removing the START/W and the job name. Each line in RUNALL.BAT will initiate a separate SAS session. This means that they are independent with separate SAS Logs. If one fails the others will still execute, and as long as the /Wait (or /W) switch is used with START each of these jobs will be executed in succession (one at a time). To examine this process further, at least two options are available. One can right click on the .BAT file and "edit" to see the commands that were created. Alternatively, you may specify a different output dataset name in place of '_null_' in the DATA statement, and then examine the contents of that dataset. Using PUT statements within a data step is another way to understand this process. It may be helpful to add them after each definition of STMT.

Sometimes you will want to have a batch job that executes a single SAS program that in turn executes a secondary series of SAS programs. Done this way, the successive jobs do not have to be independent. There can be a single SAS Log and an error in one of the programs can be used to determine if the successive programs should be executed (not shown in this paper). In this example, our resulting batch file will look like this (remember this is really one long line that has been wrapped for inclusion in this paper):

```
Start/w "Run Include List" "C:\Program Files\SASHome2\SASFoundation\9.4\sas.exe" -sysin
"C:\PortableProject\SASCode\INCDriver.sas" -nosplash -log
"C:\PortableProject\SASCode\INCDriver.log" -autoexec "C:\PortableProject\SASCode\autoexec.sas"
```

This batch command executes a SAS program (INCDRIVER.SAS) that in turn includes other programs for execution. It is of course the execution of these included programs that is most interest. Because only one SAS program is to be initiated by the batch process, the file containing this batch command (RUNINC.BAT in this example) would have only a single command. This will cause the execution of our driver file, a SAS program. Both the batch command (shown above) and the SAS driver program are created in the DATA step (below). For this example the driver program will contain two %INCLUDE statements, which are shown to the right.

```
%include "C:\PortableProject\SASCode\SalesRpt_Auto.sas";
%include "C:\PortableProject\SASCode\SubsidiaryRpt_Auto.sas";
```

The process for building this type of batch file is similar to what was done before in the previous DATA step, however instead of writing a single batch file with a series of execution commands, we will write a SAS program to drive the process and it will contain a series of %INCLUDE statements. The batch file has one execution command to execute the SAS program that includes the programs of interest. These two programs (the driver SAS program that has the %INCLUDE statements and the batch file that executes the driver) are created by the DATA step that follows.

```
data _null_;
   attrib Stmnt length = $32000
          name  length = $40;

   * Prep to write to the Batch file;
   file "%execpath%slash%str(runINC.bat)"; ❶
   *put 'file ' "%execpath%slash%str(runINC.bat)"; ❷

   stmnt=catt('Start/w "Run Include List"'); ❸
   stmnt=catt(stmnt,' "',%tslit(%sysget(sasroot)),%tslit(%slash),'sas.exe"');
   stmnt=catt(stmnt,' -sysin "',%tslit(%execpath),%tslit(%slash),'INCDriver.sas"');
   stmnt=catt(stmnt,' -nosplash -log');
   stmnt=catt(stmnt,' "',%tslit(%execpath),%tslit(%slash),'INCDriver.log"');
   stmnt=catt(stmnt,' -autoexec "',%tslit(%execpath),%tslit(%slash),'autoexec.sas"');
   put stmnt;


   * Prep to write to the driver program;
   file "%execpath%slash%str(INCDriver.sas)"; ❹
   *put 'file ' "%execpath%slash%str(INCDriver.sas)"; ❷

   * Specify the fileref location;
   rc = filename('codedir', "%execpath");
   * Open the directory of interest;
   dirid = dopen('codedir');

   * Loop through the list of items in the directory of interest;
   do i = 1 to dnum(dirid);
      * Read the list of programs that match the criteria;
      name = dread(dirid,i);
      if index(upcase(name),'_AUTO') and index(upcase(name),'SAS') then do;
         * Create the INCLUDE statement;
         stmnt=catt('%include', ' "', %tslit(%execpath),%tslit(%slash),name,'";');  ❺
         put stmnt;
      end;
   end;
   * Close the directory;
   dirid = dclose(dirid);

   * Remove the fileref;
   rc = filename('codedir',, 'clear');
   run;
```

❶ The batch file is named and the location for the batch file is specified to be at the same location as the program that holds this DATA step.

❷ During testing the FILE statement can be commented and this and all other PUT statements will write to the SAS Log.

❸ The START/W is a windows command. This DATA step could be made more portable at this point by detecting the OS and inserting the correct command for the OS. A series of assignment statements are used to create the batch file (which in this case contains only one command – to execute the driver file). This DATA step resides in the directory `C:\portableproject\sascode` as will the resulting batch file, which contains the single command line shown above.

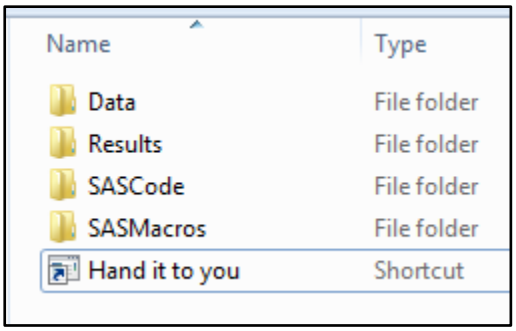
❹ The SAS program that will contain the %INCLUDE statements is named.

❺ The individual %INCLUDE statements are written to the driving SAS program, which is shown here.

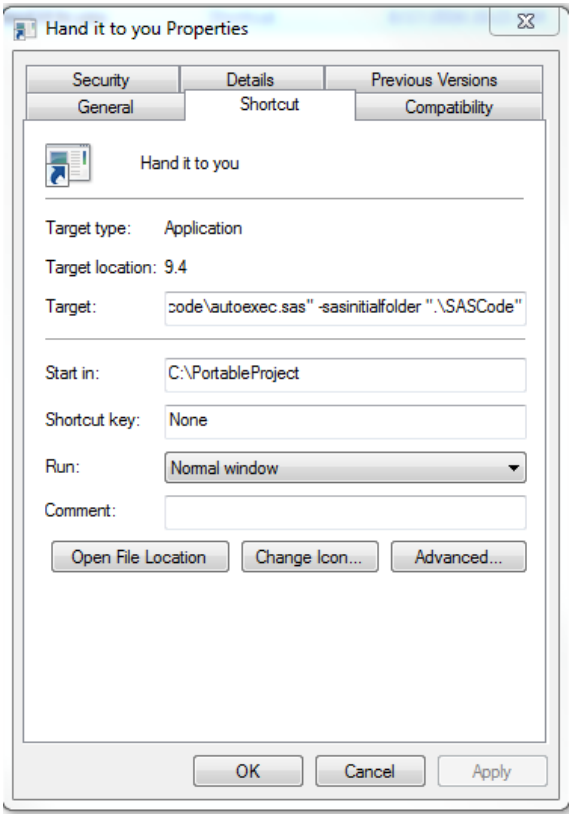
In this example we were able to build a driver file and a batch file, incorporating the current file locations into the commands. To create a driver file for a larger suite of programs that need to be run in a specific order, a control file may be a useful tool. For more information on this, see Rosenbloom and Carpenter, 2015.

THE HANDOFF

Let's return to our example consulting project with Acme Shoe Company. We have created the main reporting program for them, as requested, and with it, we will send the macro suite that we have described. We have packaged this all into a zip file (including folders and subfolders), which they will extract to a location within their statistical computing environment. Once extracted, they will see the contents pictured here. In the SASCode folder, we have supplied an AUTOEXEC.SAS file. Depending on what we know about the end-user(s) this file may contain a location-detection macro, as we have described earlier, or it may be set up for hard coding of the location.



The user has several options for working with this system. For a Windows transfer we have created a shortcut file, called "Hand it to you", located in the outermost folder. This can be used to boot up an interactive instance of SAS, calling the AUTOEXEC.SAS file, and also specifying a "home" folder which opens first. If you do not already have a SAS shortcut on your desktop, you can create or copy one, and then modify its properties. To do this, we right click on the shortcut file, and select 'Properties'. The 'Start in:' dialogue box is used to specify the root or home location for the project. In the properties window to the right this location is `c:\PortableProject`. Next we will modify the 'Target' field. This is used when a SAS session is initiated.



The value for the target line specifies the location and version of the SAS application on the system where the code is to be executed, and usually a configuration file (CONFIG) as well. The target line may also contain initialization options, such as were used in the batch program described earlier. In this example these include the -AUTOEXEC and –

```
"C:\Program Files\SASHome2\SASFoundation\9.4\sas.exe" -CONFIG "C:\Program Files\SASHome2\SASFoundation\9.4\nls\en\sasv9.cfg" -autoexec ".\sascode\autoexec.sas" -sasinitialfolder ".\SASCode"
```

SASINITIALFOLDER options. The location uses relative notation to indicate that the file, 'AUTOEXEC.SAS' will be located in the '\SASCODE' folder.

If our set of portable programs were to be executed on SAS 9.2 we would need to modify the locations of the SAS executable and the CONFIG file (if used). To do this we will need to modify the path to the 'SAS.EXE' file, and remove the CONFIG statement if we are not using a CONFIG file. So, our new value for the Target line might contain something like this:

```
"C:\Program Files\SAS\SASFoundation\9.2\sas.exe" -autoexec ".\sascode\autoexec.sas" -sasinitialfolder ".\SASCode"
```

In this example, the current user is working with SAS 9.2, and not using a CONFIG file. When the code (and shortcut file) were developed, the file location was 'C:\PortableProject'. In the new statistical computing environment, we may change this to 'C:\Desktop\Ive Got To Hand It To You\PortableProject'. Once this information is updated, the icon for the shortcut changes to a more familiar SAS icon, which tells us that we can double click the shortcut to boot up SAS and to run the AUTOEXEC.SAS file that we have sent. This has the additional benefit of landing us in the "Start in:" folder when we open files in SAS, making things even more convenient for the new user. Our colleagues at ACME Shoes can easily open up the reporting program for shoe sales and re-run it in interactive SAS with no further modifications.

Alternatively, we may also include a program called 'MakeBatch.sas' in the SASCode folder, which could be used to create a batch file for our system, as we have previously described. Since the AUTOEXEC.SAS program is sitting in the same directory as 'MakeBatch.sas', we can batch submit 'MakeBatch.sas' and it will use the AUTOEXEC.SAS file to detect the program location.

SYSTEM OPTIONS AND THEIR TEMPORARY MODIFICATION

Often times your macros will need to modify or change the settings of system options during execution. Since we anticipate that our programs will be passed around, we can't be sure of the system options settings that they will encounter. After execution it is always wise to reset these options to the setting that they enjoyed prior to the execution of your macro. This is easily accomplished by grabbing and saving the initial setting, and then restoring it afterwards. The easiest way to do this is through the use of the GETOPTION function. In the macro %WHATEVER we want to specify values for the DATE and LINESIZE system options.

```
%macro Whatever;
   %* prep to change DATE and LS system options;
   %let old_date = %qsysfunc(getoption(date)); ❶
   %let old_ls   = %qsysfunc(getoption(ls ,keyword)); ❷
   %put &=old_date;
   %put &=old_ls;

   %* your macro code here;
   options nodate ls=85; ❸
   %* your macro code here;

   %* reset options to original values;
   options &old_date &old_ls; ❹
%mend whatever;
```

❶ The GETOPTION function is used to retrieve the value of the DATE option. Depending on the current setting of this option, the macro variable &OLD_DATE will contain either DATE or NODATE.

❷ When KEYWORD is used with GETOPTION the value saved in &OLD_LS includes the name of the option along with its value. The %PUT statements are included here only to show the values stored in these two macro variables. The SAS Log shows that either option can be re-established simply by using the macro variable in an OPTIONS statement as is done at ❹. KEYWORD is not used with DATE as it is not specified using an equal sign (it is either on or off, but is never assigned a value). Using KEYWORD with an option such as DATE would not cause an error, however a NOTE is written to the SAS Log.

```
1243  %whatever
OLD_DATE=NODATE
OLD_LS=LS=75
```

❸ The values of the options needed by your macro are specified here. These may or may not change the settings of these options, either way the original values have been saved.

❹ When you are ready to restore the options to their original level, all you need to do is include the macro variables that hold the original values in an OPTIONS statement.

For additional information on the capture and re-establishment of system options see Lund (2003).

OTHER MACRO LANGUAGE ELEMENTS USED WITH PORTABILITY

In addition to the ones mentioned previously in this paper, there are a number of other macro language elements that can be used to increase the portability of your applications and programs. Some of these have fairly specific uses, however it is incumbent on the user to have a familiarity with them so that they can be incorporated into your programs when they are needed.

Automatic Macro Variables

&_METAFOLDER

If you are using SAS/IntrNet® and executing stored processes you can use &_METAFOLDER, which holds the metadata path for the currently executing stored process. See Henderson (2009) for more details including other related stored process macro variables.

&SYSPARM

The automatic macro variable &SYSPARM shares its value with the SYSPARM system option. Loading or specifying one provides a value to the other. Through the –SYSPARM initialization option, it can be loaded during the SAS initialization phase. This allows the value to be supplied before SAS is executed, and gives us the ability to pass values into SAS from an outside process or program. The value stored in this system option can be retrieved in a number of ways including the SYSPARM( ) DATA step function and the automatic macro variable &SYSPARM.

Because this option is most useful when its value is loaded when SAS is initially executed, it is most commonly used when SAS is executed in a batch execution environment. The SYSPARM initialization option specifies a character string that can be passed into SAS programs. The maximum length of this macro variable is 32K characters.

In the following example, you would like your programs to automatically direct your data to either a test or production environment. To make this switch, assign &SYSPARM the value TST or PROD when you start the SAS session. In the batch command shown here the –SYSPARM initialization option is given the value 'TST', and this value is automatically loaded into the macro variable &SYSPARM.

```
Start/w c:\. . .\sas.exe . . . .  -sysparm "TST"
```

This macro variable is now available for use, and a typical LIBNAME statement which uses this value would specify &SYSPARM wherever the resolved value is desired. The resolved LIBNAME statement substitutes the resolved value.

```
libname projdat "c:\portableproject\&sysparm\data";
```

```
libname projdat "c:\portableproject\TST\data";
```

The syntax that you use to load a value into &SYSPARM depends on the operating environment that you are using.

&SYSSCP and &SYSSCPL

The automatic macro variables &SYSSCP and &SYSSCPL were briefly described earlier in this paper. They hold OS specific information and can be used to detect operating system name and type.

&SYSVER, &SYSVLONG, and &SYSVLONG4

The three automatic macro variables (&SYSVER, &SYSVLONG, and &SYSVLONG4) can be used to display the SAS version in varying levels of detail, and were also described earlier in this paper.

&SYSMACRONAME

The automatic macro variable &SYSMACRONAME contains the name of the most local macro program that is currently executing. If you only need to know the name of the innermost currently executing macro, then &SYSMACRONAME is available for your use.

&SYSPROCESSMODE, &SYSPROCESSNAME, and &_CLIENTAPP

These macro variables return information on the current SAS process. This portion of a SAS Log shows the values of these macro variables when running an interactive SAS session using the Display

```
6    %put &=sysprocessmode &=sysprocessname;
SYSPROCESSMODE=SAS DMS Session SYSPROCESSNAME=DMS Process
```

Manager. If SAS is being accessed by a client, such as SAS Studio or Enterprise Guide, &_CLIENTAPP will exist and will hold the name of the client.

The macro %CHECK shown here was presented in a blog written by Russ Tyndall. It uses these automatic macro variables to determine the SAS process.

```
%macro check;

  %if %symexist(_clientapp) %then %do;
   %if &_clientapp = SAS Studio %then %do;
    %put Running SAS Studio;
   %end;
   %else %if &_clientapp= 'SAS Enterprise Guide' %then %do;
    %put Running SAS Enterprise Guide;
   %end;
  %end;

  %else %if %index(&sysprocessname,DMS) %then %do;
    %put Running in Display Manager;
  %end;
  %else %if %index(&sysprocessname,Program) %then %do;
     %let prog=%qscan(%superq(sysprocessname),2,%str( ));
     %put Running in batch and the program running is &prog;
  %end;

%mend check;
%check
```

&SYSENV

Indicates foreground or background execution.

Macro Functions

Macro functions are available that will help with navigation. This can be especially important when your application is executing in an unknown environment

%SYSMEXECDEPTH and %SYSMEXECNAME

When you have developed a series of nested macros (macros that call other macros), it can sometimes become important to be able to determine which macros are being called and in which order. The nesting depth and the name of the executing macro at each depth can be surfaced using the %SYSMEXECDEPTH and %SYSMEXECNAME functions. These two functions are usually used together, however it is not necessary to do so. The %SYSMEXECDEPTH function returns the total number of nesting levels, and the %SYSMEXECNAME function returns the macro name for the specified nesting level.

```
%macro ShowMacNest;
   %local i;
   %do i = 1 %to %sysmexecdepth;
      %put Level &i, Macro name is: %sysmexecname(&i);
   %end;
%mend showmacnest;
```

Execution of the %SHOWMACNEST macro will surface the nesting structure of a series of macros that call one another.

%SYSMACEXEC and %SYSMACEXIST

When you are executing an application that has a series of macros that call other macros, it is not always easy to determine which macro is currently executing or sometimes even if a macro definition currently exists. Fortunately, we are not without tools to help us.

The %SYSMACEXEC and %SYSMACEXIST functions can be used to determine if a macro is currently executing or if it has been compiled. Each of these functions return a true / false. %SYSMACEXEC determines if the macro named in the argument is currently executing and %SYSMACEXIST determines if the named macro has been compiled in the WORK.SASMACR catalog.

System Options

When you have a series of autocall libraries it is sometimes difficult to determine from which location a given macro was drawn. This can be especially true if different versions of a macro reside in different libraries. There are several system options that can be used with autocall libraries that can assist you with determining where a given macro definition resides.

MAUTOCOMPLOC

This option displays in the SAS Log the macro location at the time of macro compilation.

MAUTOLOCDISPLAY

The MAUTOLOCDISPLAY option displays the macro library location in the SAS Log when a macro is called.

MAUTOLOCINDES

When this option is turned on the full location of the macro is added to the SASMACR catalog at macro compilation.

MPRINTNEST and MLOGICNEST

When you are using nested macros (macros that call macros), MPRINT and MLOGIC only show the innermost level, and it is difficult to discern the hierarchy of calling macros. MPRINTNEST and MLOGICNEST overcome this limitation by listing each of the calling macros in order. MPRINTNEST and MLOGICNEST, respectively, require the use of the MPRINT and MLOGIC system options, which must also be turned on.

CONCLUSION

We all know how important it is to write reusable code. Writing portable code takes reusable code to the next level. In this paper you have seen how building in location-detecting syntax, application-detecting syntax, and data-driven code can create a system that is functional, even when the file structure, programming environment, execution method, SAS location and version and even the number of files changes. By employing these techniques, you can write programs that may be passed around your company or by your client, long after you are gone.

They say: “you can’t take it with you”, but with portable programming techniques, we beg to differ!

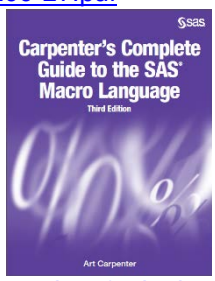
REFERENCES

Billings, Thomas E., 2015, “Enhancing the Portability and Reusability of SAS® Enterprise Guide® Projects”, presented at the 2015 Western Users of SAS Software Conference. http://wuss.org/Proceedings15/11_Final_Paper_PDF.pdf

Carpenter, Arthur L., 2001, “Building and Using Macro Libraries”, presented at the 9th Western Users of SAS Software Conference (September, 2001). http://sascommunity.org/wiki/Building_and_Using_Macro_Libraries

Carpenter, Arthur L., 2002, “Macro Functions: How to Make them – How to Use Them”, presented at the 27th annual SAS User Group International Conference, SUGI, Paper 100-27. http://www2.sas.com/proceedings/sugi27/p100-27.pdf

Carpenter, Arthur L., 2008, “The Path, The Whole Path, Nothing But the Path, So Help Me Windows”, presented at the San Diego SAS Users Group 11/2007 and at the SAS Global Forum 2008 Conference in San Antonio, Texas (paper 023-2008). http://www2.sas.com/proceedings/forum2008/023-2008.pdf



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

Documentation for batch commands under Windows can be found at: http://support.sas.com/documentation/cdl/en/hostwin/67962/HTML/default/viewer.htm#p16esisc4nrd5sn1ps5l6u8f79k6.htm

Furdal, Stanislaw, 2008, “Quick Windows Batches to Control SAS® Programs Running Under Windows and UNIX”, SAS Global Forum 2008 Conference in San Antonio, Texas (paper 017-2008). http://www2.sas.com/proceedings/forum2008/017-2008.pdf

Hamilton, Jack, 2013, “Creating ZIP Files with ODS”, presented at the SAS Global Forum 2013 Conference in San Francisco, California (paper 131-2013). http://support.sas.com/resources/papers/proceedings13/131-2013.pdf

Hemedinger, Chris, 2016,”Assign A SAS Library to A Different Path Depending On your OS”, The SAS Dummy; A Blog Post for the rest of us, June 24, 2016. http://blogs.sas.com/content/sasdummy/2016/06/24/assign-libname-based-on-os/

Henderson, Don, 2009, “Get the Metadata Path for the Currently Running Stored Process”. http://sascommunity.org/wiki/Get_the_Metadata_Path_for_the_Currently_Running_Stored_Process

Hughes, Troy Martin, 2016, “SAS® Spontaneous Combustion: Securing Software Portability through Self – Extracting Code”, presented at the SAS Global Forum 2016 Conference in Las Vegas, NV (Paper 11768-2016). http://support.sas.com/resources/papers/proceedings16/11768-2016.pdf

Jackson, Clarence Wm., 2007, ‘’Using SYSPARM and Other Automatic SAS Variables”, http://sascommunity.org/wiki/Using_SYSPARM_and_Other_Automatic_SAS_Variables

Lund, Pete, 2003, “Make Your Life a Little Easier: A Collection of SAS® Macro Utilities”, presented at the SUGI 28 in Seattle, WA (paper 85-28). http://www2.sas.com/proceedings/sugi28/085-28.pdf

Rosenbloom, Mary F. O. and Carpenter, Arthur L., 2015, “Are You a Control Freak? Control Your Programs – Don’t Let Them Control You!”, presented at the SAS Global Forum 2015 Conference in Dallas, Texas (paper 2220-2015). http://support.sas.com/resources/papers/proceedings15/2220-2015.pdf

Tyndall, Russ, 2016, “Macro Variables That Provide Information About Your SAS® Environment”, http://blogs.sas.com/content/sgf/2016/10/21/macro-variables-that-provide-information-about-your-sas-environment/

Usage Note 36613: The availability of the SAS_EXECFILEPATH and SAS_EXECFILENAME environment variables. http://support.sas.com/kb/36/613.html

Usage Note 24301: How to retrieve the program name that is currently running in batch mode or interactively. http://support.sas.com/kb/24/301.html

A SAS forum discussion that focuses on the use of SAS_EXECFILEPATH and SAS_EXECFILENAME in UNIX and remote environments can be found at: https://communities.sas.com/t5/General-SAS-Programming/SAS-EXECFILENAME-and-SAS-EXECPATHNAME-into-UNIX-environment/td-p/112056

A sasCommunity.org article titled “SAS Filesystem Toolbox”, which discusses a number of file discovery and manipulation options can be found here. http://sascommunity.org/wiki/SAS_Filesystem_Toolbox

ACKNOWLEDGMENTS

Mary would like to thank Art for being a great mentor and friend, and for another fun collaboration!

ABOUT THE AUTHORS

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.

Mary Rosenbloom is a statistical programmer, who has been using SAS for over 15 years, and is especially interested in using macros to generate data-driven code, DDE, and program validation methods.

CONTACT INFORMATION

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

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

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

View Art Carpenter's paper presentations page at:
http://www.sascommunity.org/wiki/Presentations:ArtCarpenter_Papers_and_Presentations

Mary F. O. Rosenbloom
Alcon, a Novartis Company

mary.rosenbloom.sas@gmail.com

View Mary Rosenbloom's paper presentations page at:
http://www.sascommunity.org/wiki/Presentations:Otterm1_Papers_and_Presentations

TRADEMARK REFRENCES