Paper CC-06

By Your Command: Executing Windows DLLs from SAS® Enterprise Guide®

Darryl Putnam, CACI, Inc. Elkridge, MD

ABSTRACT

Have you ever wanted to use system shell commands (copying and moving files, creating directories, etc) from SAS® Enterprise Guide® 4.1? SAS Enterprise Guide default installation turns off this functionality. You can re-enable the X command using –ALLOWXCMD, but do you really want anyone to be able to issue commands like “format C:\” on your SAS server? Luckily it is possible to run external DLLs from SAS under Windows by using the SASCBTBL Attribute table and MODULE family of call routines and functions.

This paper will demonstrate how to setup the SASCBTBL table to call a select number of Windows DLLs. A brief description of MSDN library and how to turn the Windows function into SAS callable routines will be presented.

INTRODUCTION

SAS Enterprise Guide is fast becoming the preferred method of a SAS installation because of the centralized nature of administering the product. SAS Enterprise Guide is a client server application where the client (SAS Enterprise Guide) can talk to and issue commands to the server where SAS is installed. However, not all of the functionality in a traditional SAS installation works with SAS Enterprise Guide. One of the limitations is that functions used to run operating system commands have been disabled. It is possible to have these capabilities turned back on, but not many system administrators want users to have the ability to run amok on the servers (Hemedinger, 2007).

This paper will show a work-around by demonstrating an alternative method of creating a directory and moving a file from a SAS DATA STEP to the recently created directory by accessing the Windows API from within our SAS session.

THE WINDOWS API

In our quest to run operating system commands, we first need to find which commands are to be run. By searching Microsoft’s MSDN Library (http://msdn.microsoft.com/en-us/library), we can find the functions that we need to move a file and create a directory. The below information was obtained from the MSDN website:

FUNCTIONS TO BE USED

MoveFile Function

```
The syntax for the MoveFile function
BOOL WINAPI MoveFile(
  __in  LPCTSTR lpExistingFileName,
  __in  LPCTSTR lpNewFileName
);
Return Value: If the function succeeds, the return value is nonzero.  If the function
fails, the return value is zero.
Header: WinBase.h
DLL: Kernel32.dll
Unicode and ANSI names: MoveFileW(Unicode) and MoveFileA(ANSI)
Source:  http://msdn.microsoft.com/en-us/library/aa365239(VS.85).aspx
```

CreateDirectory Function

```
The syntax for the CreateDirectory function
BOOL WINAPI CreateDirectory(
  __in      LPCTSTR lpPathName,
  __in_opt  LPSECURITY_ATTRIBUTES lpSecurityAttributes
);
Return Value: If the function succeeds, the return value is nonzero.  If the function
fails, the return value is zero.
Header: WinBase.h
DLL: Kernel32.dll
```

```
Source: http//msdn.microsoft.com/en-us/library/aa363855(VS.85).aspx
```

UNDERSTANDING THESE FUNCTIONS

These Windows DLLs are written in C and understanding how a C function and data structure operate will be helpful in our quest.

Let us deconstruct the syntax for the MoveFile function:

- BOOL: Lets the user know that the function returns a Boolean value. If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.
- WINAPI: Lets the user know that the function is in the Windows API.
- MoveFile: Is the name of the function.
- __in: Lets the user know that the argument is an input parameter.
- LPCTSRT: Lets the user know that the input parameter is expecting a character string. Specifically a null-terminated character string.
- lpExistingFileName: The parameter name for the file to be moved.
- lpNewFileName: The parameter name for new file name.

Let us deconstruct the syntax for the CreateDirectory function:

- BOOL: Lets the user know that the function returns a Boolean value. If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.
- WINAPI: Lets the user know that the function is in the Windows API.
- CreateDirectory: Is the name of the function.
- __in: Lets the user know that the argument is an input parameter
- __in_opt: Lets the user know that the argument is an optional input parameter.
- LPSECURITY_ATTRIBUTES: Lets the user know that the input parameter is expecting a pointer. Fortunately this is of no concern to us and we will just use the Windows default security settings.
- lpPathName: The parameter name for the new directory.
- lpSecurityAttributes: The optional parameter name for security attributes to be assigned to the newly created directory.

INVOKING EXTERNAL DLLS

Dynamic Linked Libraries (DLLs) are executable files that can contain many routines. SAS provides routines and functions that let you invoke these external routines from within your SAS session. You can access the DLL routines from the DATA STEP by using the MODULE family of functions and call routines. Three general steps are needed to invoke an external DLL (SAS OnlineDoc).

1. Create a text file that describes the function to be called. This information is stored in a specific format in the SASCBTBL attribute table.
2. Use the FILENAME statement with the CATALOG access method to assign the SASCBTBL fileref to the attribute file you created.
3. Use the MODULE family of functions and call routines to invoke the DLL routine.

THE SASCBTBL ATTRIBUTE TABLE

In order to invoke an external function from within SAS, SAS needs to know information about the function to be called. This information is stored in the SASCBTBL attribute table. Starting in version 6, the SASCBTBL attribute table was originally designed to execute COBOL routines from within SAS. Now, we can use the SASCBTBL attribute table to store information from any external DLL, in this case the Windows API.

The SASCBTBL attribute table is a text file that consists of 2 statements ROUTINE and ARG. The ROUTINE statement consists of the function's "metadata". Let us look at the syntax:

From the SAS Online Documentation http://support.sas.com/onlinedoc/913/docMainpage.jsp

```
ROUTINE name MINARG=minarg MAXARG=maxarg
       <CALLSEQ=BYVALUE|BYADDR>
       <STACKORDER=R2L|L2R>
       <STACKPOP=CALLER|CALLED>
       <TRANSPOSE=YES|NO> <MODULE=DLL-name>
       <RETURNS=SHORT|USHORT|LONG|ULONG |DOUBLE|DBLPTR|CHAR<n>>
       <RETURNREGS=DXAX>;

ARG argnum NUM|CHAR <INPUT|OUTPUT|UPDATE> <NOTREQD|REQUIRED> <BYADDR|BYVALUE>
<FDSTART> <FORMAT=format>;
```

MOVE FILE FUNCTION SYNTAX

Below is the syntax for setting up the SASCBTL attribute table for the MoveFile function.

```
ROUTINE MoveFileA
       MINARG =2
       MAXARG =2
       CALLSEQ = BYADDR
       STACKORDER =R2L
       STACKPOP = CALLED
       TRANSPOSE = YES
       MODULE =kernel32
       RETURNS = LONG;
ARG 1 CHAR INPUT FORMAT =$CSTR260.;
ARG 2 CHAR INPUT FORMAT =$CSTR260.;
```

This daunting syntax deconstructed:

- ROUTINE is the statement name.
  - The Windows function name is MoveFile. This must be the exact name as the external function to be called. Most Windows functions have a Unicode name and an ANSI name. This is case sensitive. The main difference is the MAX_PATH. In the ANSI version the MAX_PATH is 260 and the Unicode version extends the MAX_PATH to 32,757. We will use the ANSI name for this example.
  - MINARG=2. For this function to work we need a “from file” and a “to file” so 2 arguments are needed. In most cases this value will be the same as MAXARG.
  - MAXARG=2. For this function to work we need a “from file” and a “to file” so 2 arguments are needed.
  - CALLSEQ=BYADDR. This indicates the calling sequence method used by the DLL. The default value is BYADDR. Use BYVALUE only if the function passes data from or to another function.
  - STACKORDER=R2L. Almost all Windows functions are R2L (Johnson, 2005).
  - STACKPOP=called. Almost all Windows functions are CALLED (Johnson, 2005).
  - Transpose=no. Transpose=yes only if using PROC IML.
  - MODULE=kernel32. In the Windows documentation the MoveFile function is located in the Kernell32.dll
  - RETURNS=LONG. Notice the BOOL keyword in front of the Window MoveFile function definition. This means that the function returns a Boolean value 0 for success and not 0 for failure. A long is a signed integer data type that is used to return to Boolean values.

- ARG 1 CHAR INPUT FORMAT=$CSTR260.
  - Corresponds to the information in this line __in LPCTSTR lpExistingFileName.
  - 1 is the first argument which corresponds to lpExistingFileName in MoveFile function.
  - CHAR is the SAS datatype to be passed to the function. LPCTSTR is Windows “speak” for char.
  - INPUT means that the argument is an input argument. __in tells us that it is an input parameter.
  - FORMAT=$CSTR260. In almost all cases of passing character strings to Windows you will need the $CSTR format which is a null-terminated format. 260 characters is the MAX_PATH limit in Windows.

- ARG 2 CHAR INPUT FORMAT=$CSTR260.
  - 2 is the second argument which corresponds to lpNewFileName in MoveFile function.
  - CHAR is the SAS datatype to be passed to the function.
  - INPUT means that the argument is an input argument.
  - FORMAT=$CSTR260. In almost all cases passing character strings to Windows you will need the $CSTR format which is a null-terminated format. 260 characters is the MAX_PATH limit in Windows.

CREATE DIRECTORY FUNCTION SYNTAX

```
ROUTINE CreateDirectoryA
       MINARG=2
       MAXARG=2
       CALLSEQ=BYADDR
       STACKORDER=R2L
       STACKPOP=CALLED
       RETURNS=LONG;

ARG 1 CHAR INPUT FORMAT=$CSTR260.;
ARG 2 NUM INPUT BYVALUE  FORMAT= ib4.;
```

Most of the parameters and function descriptions are the same as the MoveFile function, except for the second argument.

- In the ARG 2 line BYVALUE is used because this references a pointer which references a handle. The format `ib4.` is a 4 byte signed integer and pointers can use this data type (Johnson, 2005).

CREATING THE SASCBTBL ATRIBUTE TABLE

Now that we have the appropriate syntax for SAS to know about our two functions, how do we get the ROUTINE and ARG statements into a text file and have SAS know what to do with it? We can load the description of the DLLs into the reserved fileref SASCBTBL as a source entry in a SAS catalog. The FILENAME statement with the CATALOG access method is a good way to get this done. The below SAS code creates the source entry by using the PUT statement and creates the SAS CATALOG in WORK.

```
filename SASCBTBL catalog "WORK.WINDOWS_ROUTINES.ATTRIBUTE_TABLE.SOURCE";
* SAS Catalogs have 4 part names: libref.catalog.entry-name.entry-type *;

data _null_;
  file SASCBTBL;
  put "**** SASCBTBL FILE ****;";
  put;
  put "* NOTES:;";
  put "* C Language Formats;";
  put "*    C Type            SAS Format/Informat ;";
  put "*    ---------------   ------------------- ;";
  put "*    double            RB8.                ;";
  put "*    float             FLOAT4.             ;";
  put "*    signed int        IB4.                ;";
  put "*    signed short      IB2.                ;";
  put "*    signed long       IB4.                ;";
  put "*    char*             IB4.                ;";
  put "*    unsigned int      PIB4.               ;";
  put "*    unsigned short    PIB2.               ;";
  put "*    unsigned long     PIB4.               ;";
  put "*    char[w]           $CHARw. or $CSTRw.  ;";
  put;
  put "* NOTES: $CSTRw. Format character argument as a null-terminated string;";
  put "* NOTES: All Definitions from http://msdn2.microsoft.com/en-
us/library/default.aspx ;";
  put ;

  put "** Move a File **;";
  /* http://msdn2.microsoft.com/en-us/library/aa365239(VS.85).aspx
     BOOL WINAPI MoveFile(
       __in  LPCTSTR lpExistingFileName,;";
       __in  LPCTSTR lpNewFileName
     );
  */
  put "routine MoveFileA minarg=2 maxarg=2 callseq=byaddr stackorder=R2L
stackpop=called transpose=no module=kernel32 returns=long;";
```

```
  put " arg 1 char input format=$CSTR260.;";
  put " arg 2 char input format=$CSTR260.;";



  put "routine CreateDirectoryA minarg=2 maxarg=2 callseq=byaddr stackorder=R2L
stackpop=called returns=long module=kernel32;";
  put " arg 1 char input format=$CSTR260.;";
  put " arg 2 num input format=ib4. byvalue;";
  put;
run;
```

EXECUTING THE DLLS FROM THE DATA STEP.
The MODULE family of call functions and routines look into the SASCBTBL fileref for the attribute information needed to run the DLL. The purpose of the SASCBTBL attribute table is to define how the MODULE function should interpret its supplied arguments when building a parameter list to pass to the called DLL routine. The MODULE function builds a parameter list by using the information from the ROUTINE statement and ARG statement that is defined in our SASCBTL fileref. We have three different MODULE functions to choose in order to run our external DLL:

1. CALL MODULE is used when no return value is expected.
2. MODULEN is used when a numeric return value is expected. In our case, since both the MoveFile and CreateDirectory function return Boolean values, the MODULEN function will be used.
3. MODULEC is used when a character return value is expected.

From the SAS Online Documentation:

```
CALL MODULE(<cntl>,module,arg-1,arg-2. . . ,arg-n);
num=MODULEN(<cntl>,module,arg-1,arg-2...,arg-n);
char=MODULEC(<cntl>,module,arg-1...,arg-2,arg-n);
```

The parameters are described below:

- cntl is an option control string.
- module is the name of the module wrapped in quotes and is case sensitive
- arg-1,..arg-n are the values and/or variables passed to and from the routine as defined in the SASCBTBL attribute table.

EXAMPLE

The example below shows a simple yet effective piece of code that will create a new directory and move a file to that newly created directory. A more robust solution would include error checking items, such as:

- Handling a request to move a non-existent file.
- What to do if the file being moved already exists.
- Creating a directory when it already exists.

Windows functions such as GetLastError and SAS I/O functions to add error checking can be utilized, however, this is beyond the scope of this paper.

```
*** sample code to create a directory and move a file to that directory ***;

%let from_file=%str(c:\test.txt);
%let to_file=%str(c:\test_folder\test.txt);;
%let new_directory=%str(c:\test_folder\);

Filename SASCBTBL catalog “WORK.WINDOWS_ROUTINES.ATTRIBUTE_TABLE.SOURCE”;

DATA _NULL_;
  From_file=”&from_file”;
  To_file=”&to_file”;
  New_directory=”&new_directory”;

  Rc1=modulen(‘CreateDirectoryA’,new_directory,0);
      * Since the CreateDirectory function returns a Boolean value,;
      * the MODULEN function is used.;
      * 0 makes sure the security settings are the default settings in windows,;
      * change at your own risk;

  Rc2=modulen(‘MoveFileA’,from_file,to_file);
      * Since the MoveFile function returns a Boolean value,;
      * the MODULEN function is used.;

  Put rc1= rc2=;

Run;
```

Notice that the value of rc1 and rc2 should be 1, which indicates a successful invocation of the CreateDirectory and MoveFile function. Recall that Windows returns a 0 for a failure.

CONCLUSION

As usual with SAS, there is more than one way to solve a problem. Even though in SAS Enterprise Guide the X command and SYSTASK function are disabled, we can still run operating system commands by an alternative method. This paper demonstrated using the SASCBTBL attribute table to describe the Windows functions and use the MODULE family of call routines and functions within the DATA STEP to invoke the MoveFile and CreateDirectory function in the kernel32.dll of the Windows API. With a basic understanding of the Windows API, a SAS programmer is able to invoke any function available in the Windows API.

The author has a library of Windows functions in his SASCBTBL attribute table and hopes the reader will add this tool in the programming arsenal.

REFERENCES

Hemedinger, Chris, “Find Out What You're Missing: Programming with SAS® Enterprise Guide”, Paper # 329-2008 SGF 2008.

Hemedinger, Chris, “The SAS Dummy Blog, Thursday, November 19. 2009, Using the X and SYSTASK commands from SAS Enterprise Guide”.

Hemedinger, Chris, “The SAS Dummy Blog, October 5, 2007, You are under my command (prompt)”

SAS OnlineDoc® 9.1.3, Operating Environment Specific Information, SAS Companion for Microsoft Windows. Available at http://support.sas.com/onlinedoc/913/docMainpage.jsp.

SAS Documentation: (TS-575) Preliminary documentation for CALL MODULE

Johnson, David H., “SAS® with the Windows API”, Paper 248-30, SUGI 30, 2005”

Matthews, Michael, “SAS, the System and Sudoku, ‘Using SAS Software to access the Operating System, and solve other problems … like Sudoku!” , SNUG (SAS NSW Users Group) Q1 2006

Carpenter, Arthur L., “The Path, The Whole Path, And Nothing But the Path, So Help Me Windows” , Paper 023-2008”, SAS Global Forum 2008.

MSDN Library: available at http://msdn.microsoft.com/en-us/library

ACKNOWLEDGMENTS

Richard A. DeVenezia’s web site http://www.devenezia.com/downloads/sas/sascbtbl was an invaluable source of information for this paper.

CONTACT INFORMATION

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

Darryl Putnam
CACI Inc.
6835 Dearpath Road
Elkridge, MD 21075
Work Phone: 410-762-6535
E-mail: dputnam@caci.com