Paper PA10

The Disk Detective: A Tool Set for Windows SAS© Administrators

Darryl Putnam, Bank of America, Charlotte, NC

ABSTRACT

Ever needed to know detailed information about a file on your disk subsystem, such as: who owns the file, when was it modified, how much space does it take? Ever wonder how much free space is on my server before I run this huge SQL query? Luckily with SAS© we can directly access the functions within the Windows API to accomplish these and many more tasks by using the SASCBTBL attribute table and the MODULE family of call routines and functions.

This paper will demonstrate how to setup the SASCBTBL attribute table to be able to call all the Windows functions needed to get a complete picture of the file structure on a disk drive.

INTRODUCTION

SAS 9.2 has functions that can be used to gather information on your system's files and directories. The FINFO function under SAS will return the file's Name, Create Time, Last Modified Time, Size (bytes), RECFM, and LRECL. These file attributes are import to us but what if we need more file attributes for our SAS applications? In order to get a complete attribute picture of a file, the operating system must be called from within SAS. Utilizing piping we can gather this information by passing operating system commands through the X command. Unfortunately, under SAS Enterprise Guide this functionality is turned off (Hemedinger, 2007).

We need to develop a methodology that can be used across different installations of SAS on the Windows platform whether SAS Enterprise Guide or traditional SAS. The MODULE family of call routines and functions used in the DATA step has been shown to effectively call the Windows API and with this we can get a full picture of the files on our system.

We will focus on understanding the three aspects of successfully executing Windows functions from within your SAS session: 1) the Windows API, 2) SASCBTBL attribute table, and 3) the MODULE family of call routines and functions.

THE WINDOWS API

The Windows API (application programming interface) was designed with interaction between the operating system and an application like SAS in mind. Within the Windows API, the operating system is at our disposal as application programmers, because any Windows application must be able to interface with the operating system in order for the application to even work. Memory management, credentials, file management are all done behind the scenes during our SAS session. Sometimes we need to directly run operating commands from our SAS session, and The Invoking External DLLs section in this paper will show in great detail how to access the Windows API from our SAS session.

The operating system contains a wealth of information for the files residing on the disk system. This is one of primary functions of the operating system. By opening up Windows Explorer we can view files and their attributes via a right mouse click. Windows Explorer in the back ground is executing a series of functions to display that information. A visit to Microsoft's MSDN Library (http://msdn.microsoft.com/en-us/library) points us to the functions that we need to generate a file name list for each directory and the associated attributes of those files.

Fortunately, Microsoft has sample code on MSDN which shows in C++ how to use the Windows functions to get a listing of files in a directory.

http://msdn.microsoft.com/en-us/library/windows/desktop/aa365200(v=vs.85).aspx

By a careful review of this sample, we see that the code calls the **FindFirstFile** function to get the file listing in a handle, then the **FindNextFile** function is used to loop through the handle file by file, and then the

FindClose function is used to end the process. Since we will be calling the Windows API to get our directory listing, we will need to have SAS access these three functions.

The following subsection will explain in detail how the above functions work. Most of the functions in Windows are written in C++, so much of the focus of this paper will be translating these into SAS callable routines. All documentation for the functions is on Microsoft's MSDN Library.

FUNCTION FindFirstFile

The function FindFirstFile searches a directory for a file or subdirectory with a valid path or path with a wild card. This function returns a handle to a collection of pointers for the file attributes of all the files in the directory. With this information we can starting building our picture of the directory searched. The documentation for these functions is extensive and filled with C++ jargon. The below information was obtained from the MSDN website.

The syntax to call the FindFirstFile function prototype is below:

```
HANDLE WINAPI FindFirstFile(
  _In_   LPCTSTR lpFileName,
  _Out_  LPWIN32_FIND_DATA lpFindFileData
);
Return Value:  If the function succeeds, the return value is a search handle
used in a subsequent call to FindNextFile or FindClose.
DLL: Kernel32.dll
Unicode and ANSI names: FindFirstFileW(Unicode) and FindFirstFileA(ANSI)
Source: http://msdn.microsoft.com/en-us/library/windows/desktop/aa364418(v=vs.85).aspx
```

This function only has two parameters the file name or directory and the associated output. Seems simple enough, but the inner workings of operating systems are complex. The output parameter lpFindFileData contains a nested call to the _WIN32_FIND_DATA structure that will be discussed next. When this function is called a virtual listing of all the files in the directory plus their corresponding attributes are held in a handle data structure which is conceptually like a collection of pointers or a stack of data. This function is located in Kernel32.dll and has both an ANSI and Unicode implementation. These will be discussed in the SASCBTBL Attribute Table section later in this paper.

Let us deconstruct the syntax for the FindFirstFile function. The first line of the function describes the function, while the lines between the parentheses describe the arguments to the function.

- HANDLE: Lets the user know that the function returns a handle which is a collection of pointers.
- WINAPI: Lets the user know that the function is in the Windows API.
- FindFirstFile: Is the name of the function.

- After the first line, the parameters used in the function are enclosed with parentheses. The general syntax for the arguments are argument type (input/output), data type, and name.
    - _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.
    - lpFileName: The parameter name for the path or directory to be searched.

    - _Out_: Lets the user know that the argument is an output parameter.
    - LPWIN32_FIND_DATA: Lets the user know that the output parameter is in the _WIN32_FIND_DATA structure. This data structure contains many of the attributes of the file that we need and is deconstructed in the next paragraph.
    - lpFindFileData: The parameter name to the pointer to the WIN32_FIND_DATA structure that received information about the found file or directory.

Syntax of the WIN32_FIND_DATA structure contains our file attributes:

```
typedef struct _WIN32_FIND_DATA {
  DWORD    dwFileAttributes;
  FILETIME ftCreationTime;
  FILETIME ftLastAccessTime;
  FILETIME ftLastWriteTime;
  DWORD    nFileSizeHigh;
  DWORD    nFileSizeLow;
  DWORD    dwReserved0;
  DWORD    dwReserved1;
  TCHAR    cFileName[MAX_PATH];
  TCHAR    cAlternateFileName[14];
} WIN32_FIND_DATA, *PWIN32_FIND_DATA, *LPWIN32_FIND_DATA;
```

Let us deconstruct the syntax for the _WIN32_FIND_DATA structure. The FindFirstFile function outputs all of this information during the search.

- Typedef struct _WIN32_FIND_DATA: Lets the user know that a type of data structure named _WIN32-FIND_DATA is being defined.
- Data types used:
  - DWORD: A 32-bit unsigned integer data type. The range is 0 through 4294967295 decimal.
  - FILETIME: A FILETIME structure.
  - TCHAR: Character data type.
- dwFileAttributes: The file attributes of a file. The file attributes are bit wise indicators and are displayed as decimal or hexadecimal values. Below are the attributes that can be obtained.

Attribute Name	Value Dec/(Hex)
FILE_ATTRIBUTE_ARCHIVE	32 (0x20)
FILE_ATTRIBUTE_COMPRESSED	2048 (0x800)
FILE_ATTRIBUTE_DEVICE	64 (0x40)
FILE_ATTRIBUTE_DIRECTORY	16 (0x10)
FILE_ATTRIBUTE_ENCRYPTED	16384 (0x4000)
FILE_ATTRIBUTE_HIDDEN	2 (0x2)
FILE_ATTRIBUTE_INTEGRITY_STREAM	32768 (0x8000)
FILE_ATTRIBUTE_NO_SCRUB_DATA	131072 (0x20000)
FILE_ATTRIBUTE_NORMAL	128 (0x80)
FILE_ATTRIBUTE_NOT_CONTENT_INDEXED	8192 (0x2000)
FILE_ATTRIBUTE_OFFLINE	4096 (0x1000)
FILE_ATTRIBUTE_READONLY	1 (0x1)
FILE_ATTRIBUTE_REPARSE_POINT	1024 (0x400)
FILE_ATTRIBUTE_SPARSE_FILE	512 (0x200)
FILE_ATTRIBUTE_SYSTEM	4 (0x4)
FILE_ATTRIBUTE_TEMPORARY	256 (0x100)
FILE_ATTRIBUTE_VIRTUAL	65536 (0x10000)

- ftCreationTime: A FILETIME structure when the file or directory was created.
- ftLastAccessTime: A FILETIME structure when the file was last read from, written to, or for executable files, run.
- ftLastWriteTime: A FILETIME structure when the file was last written to, truncated, or overwritten.

FUNCTION FindNextFile

The function FindNextFile continues a file search on the handle from a previous call to the FindFirstFile function. This function when executed properly will iterate through the directory listing file by file until the end. The below information was obtained from the MSDN website.

The syntax to call the FindFirstFile function prototype is below:

```
BOOL WINAPI FindNextFile(
  _In_   HANDLE hFindFile,
  _Out_  LPWIN32_FIND_DATA lpFindFileData
);
Return Value:  If the function succeeds, the return value is nonzero and the
lpFindFileData parameter contains information about the next file or directory
found.
DLL: Kernel32.dll
Unicode and ANSI names: FindNextFileW(Unicode) and FindNextFileA(ANSI)
Source: http://msdn.microsoft.com/en-us/library/windows/desktop/aa364428(v=vs.85).aspx
```

This function is similar to the FindFirstFile function with two notable exceptions: the return value is Boolean and the input parameter is the handle returned from the FirstFileFunction. Recall that a handle is a collection of pointers and this function is used to go through that collection file by file.

Let us deconstruct the syntax for the FindFirstFile 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.
- _in_ HANDLE hFindFile: Is the input parameter of type handle and the handle is from the return value of the FindFirstFile function.
- _Out_ LPWIN32_FIND_DATA lpFindFileData: Is the output parameter containing all of the file attributes and is structured exactly like the output from the FindFirstFile function.

FUNCTION FindClose

The function FindClose, closes a file search handle opened by the FindFirstFile function. If we do not close the search then the memory allocation is not released which can cause a memory leak on our SAS box that will eventually crash the computer. (Johnson, 2005).

The below information was obtained from the MSDN website.

```
BOOL WINAPI FindClose(
  _Inout_  HANDLE hFindFile
);
Return Value:  If the function succeeds, the return value is nonzero.  If the
function fails, then the return value is zero.
DLL: Kernel32.dll
Source: http://msdn.microsoft.com/en-us/library/windows/desktop/aa364413(v=vs.85).aspx
```

Let us deconstruct the syntax for the FindClose 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.
- _inout_ HANDLE hFindFile: The file search handle from the output of the FindFirstFile function.

INVOKING EXTERNAL DLLS

Dynamic Linked Libraries (DLLs) are executable files that can contain many routines and functions. The three Windows functions discussed above are all located in Kernel32.dll. 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 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 which is simply a text file containing SAS statements. 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 DLLs in 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:

```
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>;
```

Next we will build the SASCBTBL attribute table for the three functions: 1) FindFirstFile, 2) FindNextFile, and 3) FindClose.

FindFirstFile FUNCTION AS A SAS ROUTINE

```
ROUTINE FindFirstFileA
        minarg=11
        maxarg=11
        callseq=byaddr stackorder=R2L stackpop=called transpose=no
        module=Kernel32
        returns=long;

ARG 1 char input format=$cstr260.;      /* lpFileName */
ARG 2 num update fdstart format=pib4.;  /* lpFindFileData, see _WIN32_FIND_DATA */
                                        /*  - dwFileAttributes                  */
ARG 3 num update format=pib8.;          /*  - FILETIME ftCreationTime          */
ARG 4 num update format=pib8.;          /*  - FILETIME ftLastAccessTime        */
ARG 5 num update format=pib8.;          /*  - FILETIME ftLastWriteTime         */
ARG 6 num update format=pib4.;          /*  - DWORD    nFileSizeHigh           */
ARG 7 num update format=pib4.;          /*  - DWORD    nFileSizeLow            */
ARG 8 num output format=pib4.;          /*  - DWORD    dwReserved0             */
ARG 9 num output format=pib4.;          /*  - DWORD    dwReserved1             */
ARG 10 char update format=$CSTR260.;    /*  - TCHAR    cFileName[MAX_PATH]     */
ARG 11 char update format=$CSTR14.;     /*  - TCHAR    cAlternateFileName[14]  */
```

This daunting syntax deconstructed:

- ROUTINE is the statement name.
  - The Windows function name is FindFirstFile. 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 paper.
  - MINARG=11. For this function to work we need a file name and all the file attributes so 11 arguments are needed. In most cases this value will be the same as MAXARG. This argument list maps to the input parameters and all the output parameters for the FindFirstFile function.
  - MAXARG=11. For this function to work we need a file name and all the file attributes so 11 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 FindFirstFile function is located in Kernell32.dll
  - RETURNS=LONG. Recall the HANDLE keyword in front of the Window FindFirstFile function definition. Handle data structures typically resolve to long data types.

- The ARG statement defines all the arguments to the function and is listed in ascending order, starting with the first function argument. As with SAS data types, the argument can be either character or numeric and need to be specified as either “char” or “num”.
- Since the arguments are being passed from SAS to Windows and from Windows to SAS, the data needs to be structured in a way for the operating system and SAS to both understand. Most numeric arguments need to be either formatted as a long (PIB8.) or a short (PIB4.) and all character arguments need to have the format $CSTR.
- An INPUT argument is fairly straight forward. They are passed to the function and not changed. If an argument is sent from the function then we specify OUTPUT. If we expect the argument we send to be changed then we specify UPDATED. We can gather the type of argument from the documentation of the function.

- ARG 1 CHAR INPUT FORMAT=$CSTR260.
  - Corresponds to the information in this line _In_ LPCTSTR lpFileName.
  - 1 is the first argument which corresponds to lpFileName in FindFirstFile function.
  - CHAR is the SAS datatype to be passed to the function. LPCTSTR is Windows “speak” for char.
  - INPUT means that the argument is an input argument. _In_ tells us that it is an input parameter.
  - FORMAT=$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 for the ANSI version.
- ARG 2 NUM UPDATE FDSTART FORMAT=PIB4.
  - Corresponds to the information in the _Out_ LPWIN32_FIND_DATA lpFindFileData in the FindFirstFile function. Recall that the LPWIN32_FIND_DATA structure resolves to a number of file attributes. The FDSTART definition tells us that this argument and the subsequent arguments will be concatenated into a single variable. Windows is expecting two parameters, the file name and the structure of the output. Our argument specification is for the benefit of SAS (Johnson, 2005).
- ARG 3-11 show the corresponding parts of the WIN32_FIND_DATA structure.

FindNextFile FUNCTION AS A SAS ROUTINE

```
ROUTINE FindNextFileA
        minarg=11 maxarg=11
        callseq=byaddr stackorder=R2L stackpop=called transpose=no
        module=Kernel32
```

```
        returns=long;

ARG 1 num input  byvalue format=pib4. ;  /* HANDLE hFindFile */
ARG 2 num update fdstart format=pib4.;  /* lpFindFileData, see _WIN32_FIND_DATA */
                                        /*  - dwFileAttributes                  */
ARG 3 num update format=pib8.;          /*  - FILETIME ftCreationTime          */
ARG 4 num update format=pib8.;          /*  - FILETIME ftLastAccessTime        */
ARG 5 num update format=pib8.;          /*  - FILETIME ftLastWriteTime         */
ARG 6 num update format=pib4.;          /*  - DWORD    nFileSizeHigh           */
ARG 7 num update format=pib4.;          /*  - DWORD    nFileSizeLow            */
ARG 8 num output format=pib4.;          /*  - DWORD    dwReserved0             */
ARG 9 num output format=pib4.;          /*  - DWORD    dwReserved1             */
ARG 10 char update format=$CSTR260.;    /*  - TCHAR    cFileName[MAX_PATH]     */
ARG 11 char update format=$CSTR14.;     /*  - TCHAR    cAlternateFileName[14]  */
```

The syntax for the FindNextFile is virtually the same as FindFirstFile except for ARG 1 which is an input with the additional **byvalue** definition.

- ROUTINE is the statement name.
  - The Windows function name is FindNextFile. This must be the exact name as the external function to be called. The ANSI name is used throughout this paper.
  - MODULE=kernel32. In the Windows documentation the FindNextFile function is located in Kernell32.dll
  - RETURNS=LONG. Notice the BOOL keyword in front of the Window FindNextFile function definition. Boolean values are typical long data type. In general, if the user is unsure try the long return data type for all non-character returns.

- ARG 1 NUM INPUT BYVALUE FORMAT=PIB4.
  - Corresponds to the information in this line _In_ HANDLE hFindFile.
  - The additional BYVALUE parameter indicates that the value itself is the key to the function (Johnson, 2005). We will be passing the value of the handle returned from the FindFirstFile function into this argument. Since the handle is a memory address, we can pass only the value of handle.
- ARG 2 NUM UPDATE FDSTART FORMAT=PIB4.
  - Corresponds to the information in the _Out_ LPWIN32_FIND_DATA lpFindFileData in the FindNextFile function. Recall that the LPWIN32_FIND_DATA structure resolves to a number of file attributes. The FDSTART definition tells us that this argument and the subsequent arguments will be concatenated into a single variable. Windows is expecting two parameters, the file name and the structure of the output.
- ARG 3-11 show the corresponding parts of the WIN32_FIND_DATA structure.

FindClose FUNCTION AS A SAS ROUTINE

```
ROUTINE FindClose
        minarg=1 maxarg=1
        callseq=byaddr stackorder=R2L stackpop=called
        module=Kernel32
        returns=long;

ARG 1 num input byvalue format=pib4. ;  /* HANDLE hFindFile */
```

The syntax for the FindClose file is much simpler than FindFirstFile and FindNextFile. Only one parameter is passed, which is the returned handle from FindFirstFile. Recall that the handle returned from FindFirstFile contains the memory addresses of the directory listing and closing that memory allocation is good programming practice.

- ROUTINE is the statement name.

- The Windows function name is FindClose. This must be the exact name as the external function to be called. This function does not have an ANSI nor Unicode version because only a handle not a string is passed.
- MINARG=1 MAXARG=1. We have only one mandatory argument.
- MODULE=kernel32. In the Windows documentation the FindClose function is located in Kernell32.dll
- RETURNS=LONG. Notice the BOOL keyword in front of the Window FindClose function definition.

- ARG 1 NUM INPUT BYVALUE FORMAT=PIB4.
  - Corresponds to the information in this line _In_ HANDLE hFindFile.
  - We will be passing the value of handle returned from the FindFirstFile function into this argument. Since the handle is a memory address, we will pass the value of handle.

EXECUTING THE DLLS FROM THE DATA STEP

The MODULE family of call routines and functions 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 the FindFirstFile function returns a handle and the FindNextFile and FindClose functions 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

To use the Windows functions that we defined in the SASCBTBL attribute file containing the ROUTINE and ARG statements, we can use the DATA step. To load the Window API definitions into the SASCBTBL attribute table, simply save the SAS code containing the ROUTINE and ARG statements in a text file and make sure to use SASCBTBL as the fileref in your SAS program. The author saves his SASCBTBL attribute file with a .sas extension, so it is obvious that the file is a program. The below code generates a list of the files plus their attributes by calling the Windows API discussed above. Please note that a wild card "*" must be used in our search.

```
%let search_path=C:\*;
filename SASCBTBL 'C:\WINAPI_SASCBTBL_ATTRIBUTE_TABLE.sas';

DATA FILE_LIST;
attrib /* Windows DLL Parameters */
```

```
   hFindFile            length=8     /* Search Handle Return Value from Windows
*/
   path                 length=$260  /* Directory to Search
*/
   /*  lpFindFile Data Structure
*/
   dwFileAttributes     length=8     /* dwFileAttributes
*/
   ftCreationTime       length=8     /* FILETIME ftCreationTime
*/
   ftLastAccessTime     length=8     /* FILETIME ftLastAccessTime
*/
   ftLastWriteTime      length=8     /* FILETIME ftLastWriteTime
*/
   nFileSizeHigh        length=8     /* DWORD    nFileSizeHigh
*/
   nFileSizeLow         length=8     /* DWORD    nFileSizeLow
*/
   dwReserved0          length=8     /* DWORD    dwReserved0
*/
   dwReserved1          length=8     /* DWORD    dwReserved
*/
   cFileName            length=$260  /* TCHAR    cFileName[MAX_PATH]
*/
   cAlternateFileName   length=$14   /* TCHAR    cAlternateFileName[14]
*/
   ;

   ** initialize variables;
   ** parameters with an UPDATE type need to be set as NULL;
   array num_null (*) hFindFile dwFileAttributes ftCreationTime ftLastAccessTime
                      ftLastWriteTime
                      nFileSizeHigh nFileSizeLow;

   array char_null (*) $ cFileName cAlternateFileName;

   ** numeric parameters with an OUTPUT type need to be set as ZERO;
   array num_zero (*) dwReserved0 dwReserved1;

   do i=1 to dim(num_null);
     call missing(num_null[i]);
   end;
   do i=1 to dim(num_zero);
     num_zero[i]=0;
   end;
   do i=1 to dim(char_null);
     call missing(char_null[i]);
   end;

   path="&search_path";
   put path=;

   ** Get the handle with the results of our search;
   ** Non-zero positive integer values indicate a successful search;
   hFindFile=modulen('FindFirstFileA',path, dwFileAttributes
                    ,ftCreationTime, ftLastAccessTime, ftLastWriteTime
                    ,nFileSizeHigh, nFileSizeLow
                    ,dwReserved0, dwReserved1
                    ,cFileName, cAlternateFileName);


   if hFindFile ge 1 then do;
     ** All attribute conversions to be done in the linked statement;
```

```
      ** GetFileAttributes;
      link GetFileAttributes;
      output;
      found=1;
      do while(found);
        ** Non-zero positive integer values indicate a successful search;
        found = Modulen('FindNextFileA', hFindFile, dwFileAttributes
                        ,ftCreationTime, ftLastAccessTime, ftLastWriteTime
                        ,nFileSizeHigh, nFileSizeLow
                        ,dwReserved0, dwReserved1
                        ,cFileName, cAlternateFileName);
        if found then do;
          link GetFileAttributes;
          output;
        end;
      end;
     ** Close out the search;
     rc=modulen('*e','FindClose',hFindFile);
    end;

    GetFileAttributes:
    ** Convert Filesize from bits to bytes;
    FileSize = nFileSizeLow + nFileSizeHigh * 2**32;
    return;

run;
```

REVIEWING THE RESULTS

Our program ran successfully and we have a data set that contains all the files in the C:\ directory. A few of the columns have odd results. First, the dwFileAttributes column contains integers. This is because Windows stores the file attributes such as: Directory, Read Only, Compressed, Hidden, etc. as bit wise hexadecimal values. The below code is used to convert the dwFileAttribute values into easy to understand yes/no flags. This code should be placed in the GetFileAttributes: branch of the DATA step.

```
   FL_READONLY  =substr('NY',(band(dwFileAttributes,00000001x)>0)+1,1);
   FL_HIDDEN    =substr('NY',(band(dwFileAttributes,00000002x)>0)+1,1);
   FL_SYSTEM    =substr('NY',(band(dwFileAttributes,00000004x)>0)+1,1);
   FL_DIRECTORY =substr('NY',(band(dwFileAttributes,00000010x)>0)+1,1);
   FL_COMPRESSED=substr('NY',(band(dwFileAttributes,00000800x)>0)+1,1);
   FL_ENCRYPTED =substr('NY',(band(dwFileAttributes,00004000x)>0)+1,1);
```

Secondly, the three file time variables are unreadable. Like SAS, Microsoft has its own way of displaying dates and fortunately for us the Windows function FileTimeToSystemTime converts the data into year, month, day of week, day, hour, minute, second, and millisecond. One caveat to just using this function is that Windows stores its dates as Greenwich Mean Time (GMT). Once again Windows has a function, FileTimeToLocalTime, that will adjust the file time to the local time. Then we can convert the local file time into a human readable format with the FileTimeToSystemTime function.

THE FILETIME FUNCTIONS

The function FileTimeToSystemTime, converts a file time to a local file time. The below information was obtained from the MSDN website.

```
BOOL WINAPI FileTimeToLocalFileTime(
  _In_   const FILETIME *lpFileTime,
  _Out_  LPFILETIME lpLocalFileTime
);
```

```
Return Value:  If the function succeeds, the return value is nonzero.  If the
function fails, then the return value is zero.
DLL: Kernel32.dll
Source: http://msdn.microsoft.com/en-us/library/windows/desktop/ms724277(v=vs.85).aspx
```

The syntax for the FileTimeToLocalFileTime function is simple. We input the file time and the local file time is outputted.
Let us deconstruct the syntax for the FileTimeToLocalFileTime 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.
- FileTimeToLocalFileTime: Is the name of the function.
- _In_ const FILETIME *lpFileTime: Is the input parameter for the file time to convert.
- _Out_ LPFILETIME lpLocalFileTime: Is the output parameter of the converted time.

Our SAS definition in the SASCBTL attribute table for this function is as follows:

```
ROUTINE FileTimeToLocalFileTime
        minarg=2 maxarg=2 callseq=byaddr stackorder=R2L stackpop=called
        returns=long module=Kernel32;
ARG 1 num  input        format=pib8.;
ARG 2 num  output       format=pib8.;
```

Just like the same named Windows function, our SAS definition has only two parameters of the same data type.

- ROUTINE is the statement name.
  - The Windows function name is FileTimeToLocalFileTime. This must be the exact name as the external function to be called. This function does not have an ANSI nor Unicode version because strings such as the file name are not passed.
  - MINARG=2 MAXARG=2. We have two mandatory arguments. The input file time and the converted output file time.
  - MODULE=kernel32. In the Windows documentation the FileTimeToLocalFileTime function is located in Kernell32.dll
  - RETURNS=LONG. Notice the BOOL keyword in front of the Window FileTimeToLocalFileTime function definition.

- ARG 1 NUM INPUT FORMAT=PIB8.
  - This is the definition of the file time to be converted.
  - Since file time is a long we use the PIB8. format.
- ARG 2 NUM OUTPUT FORMAT=PIB8.
  - This is the definition of the converted file time.
  - Since file time is a long we use the PIB8. format.

The Windows function FileTimeToSystemTime will convert the Windows file time into a human readable format. Unlike the previous function, the FileTimeToSystemTime function results in the deconstructed time as year, month, day of week, day, hour, minute, second, millisecond. Many of the Windows functions output a data structure that is nested, so the application developer needs to keep a keen eye on the Windows documentation. The below information was obtained from the MSDN website.

```
BOOL WINAPI FileTimeToSystemTime(
  _In_   const FILETIME *lpFileTime,
  _Out_  LPSYSTEMTIME lpSystemTime
```

```
);

Return Value:  If the function succeeds, the return value is nonzero.  If the
function fails, then the return value is zero.
DLL: Kernel32.dll
Source: http://msdn.microsoft.com/en-us/library/windows/desktop/ms724280(v=vs.85).aspx
```

Let us deconstruct the syntax for the FileTimeToSystemTime function prototype.

- 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.
- FileTimeToSystemTime: Is the name of the function.
- _In_ const FILETIME *lpFileTime: Is the input parameter for the file time to convert.
- _Out_ LPSYTEMFILETIME lpSystemTime: Is the output parameter which is a pointer the LPSYSTEMTIME structure. The entire contents of the structure is outputted and in our SASCBTBL attribute table we will need to account for this. The components of the LPSYSTEMTIME structure is listed below.

  LPSYSTEMTIME Structure

```
typedef struct _SYSTEMTIME {
  WORD wYear;
  WORD wMonth;
  WORD wDayOfWeek;
  WORD wDay;
  WORD wHour;
  WORD wMinute;
  WORD wSecond;
  WORD wMilliseconds;
} SYSTEMTIME, *PSYSTEMTIME;
```

Our SAS definition in the SASCBTL attribute table for this function is as follows:

```
ROUTINE FileTimeToSystemTime
        minarg=9 maxarg=9 callseq=byaddr stackorder=R2L stackpop=called
        returns=long  module=kernel32;

ARG 1 num  input          format=pib8.;
ARG 2 num  output fdstart format=pib2.;    /*   WORD wYear         *
ARG 3 num  output         format=pib2.;    /*   WORD wMonth        */
ARG 4 num  output         format=pib2.;    /*   WORD wDayOfWeek    */
ARG 5 num  output         format=pib2.;    /*   WORD wDay          */
ARG 6 num  output         format=pib2.;    /*   WORD wHour         */
ARG 7 num  output         format=pib2.;    /*   WORD wMinute       */
ARG 8 num  output         format=pib2.;    /*   WORD wSecond       */
ARG 9 num  output         format=pib2.;    /*   WORD wMilliseconds */
```

- ROUTINE is the statement name.
  - The Windows function name is FileTimeToSystemTime
  - MINARG=9 MAXARG=9. The input file time and the 8 deconstructed time components.
  - MODULE=kernel32. In the Windows documentation the FileTimeToSystemTime function is located in Kernell32.dll
  - RETURNS=LONG. Notice the BOOL keyword in front of the Window FileTimeToSystemTime function definition. Boolean values are typical long data type. In general, if the user is unsure try the long return data type for all non-character returns.

- ARG 1 NUM INPUT FORMAT=PIB8.

  - Corresponds to the file time to be converted
- ARG 2 NUM OUTPUT FDSTART FORMAT=PIB8.
  - Corresponds to the LPSYTEMFILETIME data structure, specifically the year. The optional FDSTART definition was used since the output data structure has multiple parts.
- ARG 3-9 NUM OUTPUT FORMAT=PIB8.
  - Deconstructed parts of the file time.

UPDATING OUR CODE WITH CALLS TO THE FILETIME FUNCTIONS

The below code is added to the GetFileAttributes: branch in the DATA step.

```
** Convert File Time from GMT to Local Time;
rc = modulen ("FileTimeToLocalFileTime", ftCreationTime, LocalCreationTime);
rc = modulen ("FileTimeToLocalFileTime", ftLastAccessTime, LocalLastAccessTime);
rc = modulen ("FileTimeToLocalFileTime", ftLastWriteTime, LocalLastWriteTime);

** Convert File Time to readable format;
rc = modulen ("FileTimeToSystemTime", LocalCreationTime,
              SystemCreationTime_Year,
              SystemCreationTime_Month,
              SystemCreationTime_DayOfWeek,
              SystemCreationTime_Day,
              SystemCreationTime_Hour,
              SystemCreationTime_Minute,
              SystemCreationTime_Second,
              SystemCreationTime_Millisecond);

rc = modulen ("FileTimeToSystemTime", LocalLastAccessTime,
              SystemLastAccessTime_Year,
              SystemLastAccessTime_Month,
              SystemLastAccessTime_DayOfWeek,
              SystemLastAccessTime_Day,
              SystemLastAccessTime_Hour,
              SystemLastAccessTime_Minute,
              SystemLastAccessTime_Second,
              SystemLastAccessTime_Millisecond);

rc = modulen ("FileTimeToSystemTime", LocalLastWriteTime,
              SystemLastWriteTime_Year,
              SystemLastWriteTime_Month,
              SystemLastWriteTime_DayOfWeek,
              SystemLastWriteTime_Day,
              SystemLastWriteTime_Hour,
              SystemLastWriteTime_Minute,
              SystemLastWriteTime_Second,
              SystemLastWriteTime_Millisecond);

** Convert dates to SAS dates;
created_date  = dhms (mdy(SystemCreationTime_Month,
                          SystemCreationTime_Day,
                          SystemCreationTime_Year)
                     ,SystemCreationTime_Hour,
                     ,SystemCreationTime_Minute
                     ,SystemCreationTime_Second);

accessed_date = dhms (mdy(SystemLastAccessTime_Month,
                          SystemLastAccessTime_Day,
                          SystemLastAccessTime_Year)
                     ,SystemLastAccessTime_Hour
                     ,SystemLastAccessTime_Minute
                     ,SystemLastAccessTime_Second );
```

```
   modified_date = dhms (mdy(SystemLastWriteTime_Month,
                             SystemLastWriteTime_Day,
                             SystemLastWriteTime_Year)
                            ,SystemLastWriteTime_Hour
                            ,SystemLastWriteTime_Minute
                            ,SystemLastWriteTime_Second);
```

FILE OWNER

It has been a long journey exploring how SAS calls the Windows API. With our revised code we have a listing of all the files in a directory, the attributes of the files, readable file times, and the file size. But we are missing the file owner attribute. The process of getting the file owner is complex but we will use our newly gained knowledge of accessing the Windows API to tackle this complexity.

The process to get the file owner name has five steps and calls three Windows functions:

1. Pass the file name to the **GetFileSecurity** function to get the buffer size needed to hold the file security information.
2. Call **GetFileSecurity** again with file name and the newly obtained buffer size outputted from step 1 to return the file security information.
3. Pass the file security information to the **GetSecurityDescriptorOwner** function, to get the SID (security identifier) of the owner.
4. Pass the SID to the **LookupAccountSid** function to get the buffer sizes needed to hold the account and domain name.
5. Call the **LookupAccountSid** function again with the SID and the newly obtained buffer size for the names. The result is the account and domain name of the file owner.

Let us examine these functions and the SASCBTBL attribute table definition associated with these functions. All of the functions used to capture the owner of the file are in Advapi.dll as opposed to Kernel32.dll previously mentioned in this paper. We will put the Advapi.dll in the MODULE part of the ROUTINE statement in the SASCBTBL attribute table.

FUNCTION GetFileSecurity

The function GetFileSecurity obtains specified information about the security of a file or a directory. One caveat on the available information available is the caller's access rights and privileges. This function needs to be called twice. Once to obtain the number of bytes needed to store the complete security description. Secondly, pass the preceding bytes size value in order to retrieve the security information. The below information was obtained from the MSDN website.

```
BOOL WINAPI GetFileSecurity(
  _In_       LPCTSTR lpFileName,
  _In_       SECURITY_INFORMATION RequestedInformation,
  _Out_opt_  PSECURITY_DESCRIPTOR pSecurityDescriptor,
  _In_       DWORD nLength,
  _Out_      LPDWORD lpnLengthNeeded
);

Return Value:  If the function succeeds, the return value is nonzero.  If the
function fails, then the return value is zero.
DLL: Advapi32.dll
Unicode and ANSI names: GetFileSecurityW (Unicode) and GetFileSecurityA (ANSI)
Source: http://msdn.microsoft.com/en-us/library/windows/desktop/aa446639(v=vs.85).aspx
```

Let us deconstruct the syntax for the GetFileSecurity function:

- Note the DLL: is Advapi32.dll not Kernel32.dll like the previous examples.

- 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.
- GetFileSecurity: The name of the function.
- _In_ LPCTSTR lpFileName: The input parameter containing the file name.
- _In_ SECURITY_INFORMATION RequestedInformation: Since we are only interested in the file owner's name we will only pass that request. Hexadecimal (1) points to bit SE_OWNER_DEFAULTED which indicates the SID of the owner.
- _Out_opt_ PSECURITY_DESCRIPTOR pSecurityDescriptor: A pointer to a buffer that receives a copy of the security descriptor: This is our desired output.
- _In_ DWORD nLength: This is the length of the buffer used to store the security descriptor. This value is unknown, so we need to obtain it.
- _Out_ LPDWORD lpnLengthNeeded: This will return the value needed to populate the above nLength parameter.
- Note: This function needs to be called twice, because the input value nLength is not known. The first call to the function returns the value of nLength in the parameter lpnLenghthNeeded. Once the value of the buffer is known, then the SID can be returned.

Our SAS definition in the SASCBTBL attribute table for this function is as follows:

```
   ROUTINE GetFileSecurityA
           minarg=5 maxarg=5
           callseq=byaddr stackorder=R2L stackpop=called
           module=Advapi32
           returns=long;

   ARG 1 char input          format=$CSTR260.; /* -LPCTSTR lpFileName */
   ARG 2 num  input  byvalue format=pib4.;   /* -SECURITY_INFORMATION
RequestedInformation */
   ARG 3 num  update byvalue format=pib4.;   /* -PSECURITY_DESCRIPTOR pSecurityDescriptor
*/
   ARG 4 num  input  byvalue format=pib4.;   /* -DWORD   nLength */
   ARG 5 num  output         format=pib4.;   /* -LPWWORD lpnLengthNeeded */
```

- ROUTINE is the statement name.
  - The Windows function name is GetFileSecurityA. Recall that we are using the ANSI version of the functions where available throughout this paper.
  - MINARG=5 MAXARG=5. The total number of arguments.
  - MODULE= Advapi32.dll In the Windows documentation the GetFileSecurity function is located in the Advapi.dll
  - RETURNS=LONG. Notice the BOOL keyword in front of the Window GetFileSecurity function definition. Boolean values are typical long data type.

- ARG 1 CHAR INPUT FORMAT=$CSTR260.
  - The input parameter for the file name.
- ARG 2 NUM INPUT BYVALUE FORMAT=PIB4.
  - The input parameter for the security information requested. In this case this will be the SID.
- ARG 3 NUM UPDATE BYVALUE FORMAT=PIB4.
  - The output parameter for the optional security descriptor.
  - Since this is classified as an optional output argument, the update definition was used.
- ARG 4 NUM INPUT BYVALUE FORMAT=PIB4.
  - The input parameter that references the buffer size needed to hold the security information.
- ARG 5 NUM OUTPUT FORMAT=PIB4.
  - The output parameter containing the buffer size needed to hold the security information.

FUNCTION GetSecurityDescriptorOwner

The function GetSecurityDesciptorOwner retrieves the owner information from the security descriptor obtained from GetFileSecurity. Microsoft does not make it easy for us. Instead of returning the SID from the GetFileSecurity function, the GetFileSecurity function returns the security descriptor structure. This function gets the SID of the file which can then be passed to the LookUpAccountSid function to get the file owner's name. The below information was obtained from the MSDN website.

```
BOOL WINAPI GetSecurityDescriptorOwner(
  _In_   PSECURITY_DESCRIPTOR pSecurityDescriptor,
  _Out_  PSID *pOwner,
  _Out_  LPBOOL lpbOwnerDefaulted
);
Return Value:  If the function succeeds, the return value is nonzero.  If the
function fails, then the return value is zero.
DLL: Advapi32.dll
Source: http://msdn.microsoft.com/en-us/library/windows/desktop/aa446651(v=vs.85).aspx
```

Let us deconstruct the syntax for the GetSecurityDescriptorOwner function:

- Note the DLL is Advapi32.dll not Kernel32.dll.
- 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.
- GetSecurityDescriptor: The name of the function.
- _In_ PSECURITY_DESCRIPTOR pSecurityDescriptor: The input parameter containing the security descriptor returned from the GetFileSecurity function.
- _Out_ PSID *pOwner: The output parameter containing the SID of the file.
- _Out_ LPBOOL lpbOwnerDefaulted: A pointer to a flag that is set to the value of the SE_OWNER_DEFAULTED flag. This is unneeded for our use.

Our SAS definition in the SASCBTBL attribute table for this function is as follows:

```
ROUTINE GetSecurityDescriptorOwner
        minarg=3 maxarg=3
        callseq=byaddr stackorder=R2L stackpop=called
        module=Advapi32
        returns=long;

ARG 1 num input  byvalue format=pib4.;      /* -pSecurityDescriptor */
ARG 2 num output         format=pib4.;      /* -pOwner        */
ARG 3 num input          format=pib4.;      /* -lpbOwnerDefaulted */
```

- ROUTINE is the statement name.
  - The Windows function name is GetSecurityDescriptorOwner.
  - MINARG=3 MAXARG=3.
  - MODULE= Advapi32.dll In the Windows documentation the GetSecurityDescriptorOwner function is located in the Advapi.dll.
  - RETURNS=LONG. Notice the BOOL keyword in front of the Window GetSecurityDescriptorOwner function definition. Boolean values are typical long data type.

- ARG 1 NUM INPUT BYVALUE FORMAT=PIB4.
  - The input parameter for the security descriptor.
  - The structure of SECURITY_DESCRIPTOR is a variant of a pointer, so its value is passed byvalue.
- ARG 2 NUM OUTPUT FORMAT=PIB4.
  - The output parameter that contains the SID.

- The SID is an integer that represents the owner.
- ARG 3 NUM INPUT FORMAT=PIB4.
  - Corresponds to the lpbOwnerDefaulted flag.

FUNCTION LookUpAccountSid

Our journey to find the file owner is nearly complete. The function LookUpAccountSID returns the name of the account for the SID and the name of the first domain for the SID. This function needs to be called twice, first to obtain the buffer sizes for the owner and domain names. Secondly, to return the account name and domain name associated with the account. The below information was obtained from the MSDN website.

```
BOOL WINAPI LookupAccountSid(
  _In_opt_   LPCTSTR lpSystemName,
  _In_       PSID lpSid,
  _Out_opt_  LPTSTR lpName,
  _Inout_    LPDWORD cchName,
  _Out_opt_  LPTSTR lpReferencedDomainName,
  _Inout_    LPDWORD cchReferencedDomainName,
  _Out_      PSID_NAME_USE peUse
);

Return Value:  If the function succeeds, the return value is nonzero.  If the
function fails, then the return value is zero.
DLL: Advapi32.dll
Unicode and ANSI names: LookupAccountSidW (Unicode) and LookupAccountSidA (ANSI)
Source: http://msdn.microsoft.com/en-us/library/windows/desktop/aa379166(v=vs.85).aspx
```

Let us deconstruct the syntax for the LookUpAccountSid 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.
- LookupAccountSid: The name of the function.
- _In_opt LPCTSTR lpSystemName: Points to the name of the target computer. If this parameter is NULL, the account name translation begins on the local system. When calling this function, we will pass a null value.
- _In_ PSID lpSid: A pointer to look up the SID.
- _Out_opt_ LPTSTR lpName: A pointer to a buffer that corresponds the account name of the SID.
- _Inout_ LPDWORD cchName: On input, specifies the size of the lpName buffer. On output, if the value is zero then the required buffer size is returned.
- _Out_opt_ LPTSTR lpReferencedDomainName: A pointer to a buffer that corresponds to the domain name of the account name.
- _Inout_ LPDWORD cchReferencedDomainName: On input, specifies the size of the lpReferencedDomainName buffer. On output, if the value is zero then the required buffer size is returned.
- _Out_ PSID_NAME_USE peUse: A pointer to a variable that indicates the type of the account.

Our SAS definition in the SASCBTBL attribute table for this function is as follows:

```
ROUTINE LookupAccountSidA
        minarg=7 maxarg=7
        callseq=byaddr stackorder=R2L stackpop=called
        module=Advapi32
        returns=long;

ARG 1 num  input  byvalue format=pib4.;    /* lpSystemName */
```

```
ARG 2 num  input  byvalue format=pib4.;    /* lpSid        */
ARG 3 num  update byvalue format=pib4.;    /* lpName       */
ARG 4 num  update         format=pib4.;    /* cchName      */
ARG 5 num  update byvalue format=pib4.;    /* lpReferencedDomainName */
ARG 6 num  update         format=pib4.;    /* cchReferencedDomainName */
ARG 7 num  output         format=pib4.;    /* peUse */
```

- ROUTINE is the statement name.
  - The Windows function name is LookupAccountSidA. The ANSI version of this function is being used.
  - MODULE= Advapi32.dll In the Windows documentation the LookupAccountSid function is located in Advapi.dll
  - RETURNS=LONG. Boolean values are typical long data type.

- ARG 1 NUM INPUT BYVALUE FORMAT=PIB4.
  - Corresponds to lpSystemName and we will pass a null value.
- ARG 2 NUM INPUT BYVALUE FORMAT=PIB4.
  - Corresponds to SID.
- ARG 3 NUM UPDATE BYVALUE FORMAT=PIB4.
  - Corresponds to the account name.
- ARG 4 NUM UPDATE FORMAT=PIB4.
  - Corresponds to the buffer size of the account name.
  - Notice that the UPDATE definition is used since the argument will output the buffer size needed and input the obtained buffer size.
- ARG 5 NUM UPDATE BYVALUE FORMAT=PIB4.
  - Corresponds to the domain name.
- ARG 6 NUM UPDATE FORMAT=PIB4.
  - Corresponds to the buffer size of the domain name.
  - Notice that the UPDATE definition is used since the argument will output the buffer size needed and input the obtained buffer size.
- ARG 7 NUM OUTPUT FORMAT=PIB4.
  - Corresponds to SID name use parameter.

EXECUTING THE FUNCTIONS TO GET THE FILE OWNER

By placing our SAS definitions of the Windows functions needed to get the file owner into our SASCBTBL attribute table, we can then access those functions in the DATA step with MODULEN. The addition of the below code into the GetFileAttributes: branch in the DATA step will complete our long journey of getting a complete picture of a directory. You will notice that some of the functions called with MODULEN, has the parameter in the SAS function ADDR. The ADDR returns the memory address of the variable being passed. Since the Windows function are passing pointers back and forth and pointers are numbers, the ADDR function in SAS is used sometimes to help us convert the pointers into text value.

```
** The section below gets the owner name;
* Initialize some required variables.;
OWNER_SECURITY_INFORMATION = 1x;
/* Private Const OWNER_SECURITY_INFORMATION = &H1, SAS 'x' is hex notation */
ERROR_INSUFFICIENT_BUFFER = 122;
/* Private Const ERROR_INSUFFICIENT_BUFFER = 122&, BV '&' is a long */

name = "00"x;
domain_name = "00"x;
name_len = 0;
domain_len = 0;
pOwner = 0;
```

```
szfilename=substr(path,1,length(path)-1)||strip(cfilename);

sizeSD=0;
deUse=0;

* Call GetFileSecurity the first time to obtain the size of the;
* buffer required for the Security Descriptor.;
rc=modulen('GetFileSecurityA'
           ,szfilename
           ,OWNER_SECURITY_INFORMATION
           ,0
           ,0
           ,sizeSD);

sdBuf=repeat('00'x,sizeSD-1);
array AsdBuf(256) $1 _temporary_; ** array works better than single string;
*Fill the buffer with the security descriptor of the object specified;
*by the szfilename parameter. The calling process must have the right;
*to view the specified aspects of the objects security status.;
rc=modulen('GetFileSecurityA'
          ,szfilename
          ,OWNER_SECURITY_INFORMATION
          ,addr(AsdBuf[1])
          ,sizeSD
          ,addr(sizeSD));

lpbOwnerDefaulted_flag=0;
rc = modulen('GetSecurityDescriptorOwner'
             ,addr(AsdBuf[1])
             ,pOwner
             ,lpbOwnerDefaulted_flag);

*Retrieve the name of the account and the name of the first;
*domain on which this SID is found.  Passes in the Owners SID;
*obtained previously.  Call LookupAccountSid twice, the first time;
*to obtain the required size of the owner and domain names.;

system='00'x;
rc = modulen('LookupAccountSidA',
                         addr(system),
                         pOwner,
                         addr(name),
                         name_len,
                         addr(Domain_name),
                         Domain_len,
                         deUse);

rc = modulen('LookupAccountSidA',
             addr(system),
             pOwner,
             addr(name),
             name_len,
             addr(Domain_name),
             Domain_len,
             deUse);
```

CONCLUSION

Our final program was able to list select file attributes and the owner of a file for a specific directory. Since whether the file is a directory or not is returned from the Windows functions, a program could be written to loop through the entire disk drive to the get the attributes of every directory, sub-directory, and file. Image the usages for a data set containing all that information.

With SAS we are able to run operating system commands from our SAS session by calling the operating system directly in order to get a file listing and select file attributes. In order to do that, we need to understand how the Windows functions work and mimic the process in SAS. This paper demonstrated using the SASCBTBL attribute table to describe the Windows functions and use the MODULEN function within the DATA step to invoke DLLs 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.

REFERENCES:

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

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

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

Putnam, Darryl, "By Your Command: Executing Windows DLLs from SAS® Enterprise Guide®" SESUG 2010, NESUG 2010.

SAS OnlineDoc® 9.2

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

SAS support site "Sample 25074: Listing all files that are located in a specific directory" http://support.sas.com/kb/25/074.html

ACKNOWLEDGMENTS

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

All errors are solely the work of the author.

CONTACT INFORMATION

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

Darryl Putnam
Phone: 301-788-3347
Email: darryl.putnam@gmail.com