Paper 281-26

Accessing and Utilizing the Win32 API From SAS

Christopher A. Roper, National Committee for Quality Assurance, Washington, D.C.

Introduction

The Win32 API is actually a collection of dynamic link libraries (DLLs), consisting three primary DLLs, and several additional DLLs added to extend the functionality of the various Windows® 32 bit operating systems(i.e. Windows 95, 98, 2000, NT, etc. but not WIN3.1X). Some of these additional DLLs were added by Microsoft, and some are added by third party vendors when their software is installed. The three primary DLLs contain the lion's share of the functionality that makes up the core of the Windows operating system. These DLLs are named KERNEL.DLL, USER32.DLL, and GDI32.DLL. When Microsoft adds new functionality to their Windows operating systems, they do so by adding new DLLs, not by adding to existing DLLs. This 'daisy-chain' approach has several advantages. First, it actively promotes backward compatibility since future releases of the operating system will contain exactly the same DLLs as the earlier releases (plus a few new ones of course!). Second, it allows third party vendors to also extend the functionality of their own products in the same manner. Most of us have seen references to one or more of the primary DLLs (usually when something really bad has happened!), and have then realized that these DLLs are doing something behind the scenes. But what are they exactly? A DLL is nothing more than a collection of functions that the Windows operating system can interpret and therefore it is these functions that actually control the behavior of the Windows operating system. Having stated this, it does not mean that the Win32 API is simple, or easy to comprehend. But it can be utilized to take advantage of the functions available within those DLLs. Having the ability to access these functions allows a SAS programmer to seamlessly integrate Windows functionality into SAS-based applications. This enables the automation of many tasks, such as email, basic file management, and others. The Win32 API encompasses thousands of functions, most of which are beyond the scope of this paper (not to mention the author!). A very good and thorough explanation can be found in Dan Appleman's "Visual Basic® Programmer's Guide To The Win32 API". The purpose of this paper is to introduce the SAS programmer to the Win32 API and illustrate how to take advantage of some of the more useful Win32 functions.

Terminology

There are some keywords associated with using SAS to access the Win32 API. Understanding these keywords, and more importantly, understanding their purpose is critical to successfully utilizing the Win32 API. They are: Attribute Table, SASCBTBL, and the MODULEN and MODULEC functions.

Attribute Table:

The attribute table is a separate file that describes the Win32 API function to SAS. This includes the name of the DLL, the function name within the DLL, some parameters and their required values for this function, and a declaration of the arguments used by the function. A typical attribute table might be 6-12 lines long. For simplicity and to keep maintenance simple, multiple attribute tables can exist in one physical file. Later in this paper this will be demonstrated. The name of the file containing the attribute table(s) has no restrictions (other than those imposed by the Window operating system); you may name it whatever you wish. A common convention among SAS programers is WIN32API.TXT, but again, this is up to you. An attribute table consists of several parameters and associated values appropriate for the particular Win32 function. The following is an attribute table shell:

```
ROUTINE  win32functionalias
  MODULE =win32dll
  MINARG=n
  MAXARG=n
  STACKPOP=CALLED/CALLER
  RETURNS=type;
ARG n action type addressing          format;
```

An attribute table can contain more parameters than this shell, but for most uses with SAS, these seven parameters are sufficient. In an attribute table, the ROUTINE parameter must be first, and the ARG parameter(s) must be last, but the remaining parameters have no positional authority, so they can be shuffled within the ROUTINE and ARG parameters as desired.

The ROUTINE parameter names the Win32 function that will be addressed. The value of the ROUTINE parameter is the alias for the Win32 function that was defined when the function was created.

The MODULE parameter identifies the DLL that contains the function referenced in the ROUTINE statement. Typical values for this parameter are KERNEL32, USER32, etc.

The MINARG and MAXARG parameters define the minimum and maximum, respectively, number of arguments that will be passed to the function when it is invoked. There must be at least as many ARG statements as the MINARG value and no more than the MAXARG value, (not exactly rocket science here!).

The STACKPOP argument refers to internal memory management. Every time a function is executed, Windows must reserve a block of memory for that function to execute; this block of memory is called a stack. When SAS calls a Win32 function, Windows puts the function parameters and return address into this stack. The return address enables Windows to return to SAS after the function executes. STACKPOP may take one of two possible values, CALLED or CALLER. For use with SAS, this should usually be CALLED.

The RETURNS parameter describes the value of the return code from the function. These values are C/C++ data types, such as BOOL, BYTE, INT, SHORT, LONG, etc. The correct value to use for a particular Win32 function should be found in the documentation for that function. Again, Dan Appleman's 'The Visual Basic Programmer's Guide to the WIN32 API' is an excellent source for this information.

The ARG statement describes the argument list passed to/from the Win32 function. This statement consists of at least five components; n, Action, Type, Addressing, and Format.

The first component of the ARG statement (n) is a positional indicator; i.e. ARG 1 would describe the first argument, ARG 2 the second, etc.

The second component (Action) describes how the function will use the argument. Typically, this either INPUT or UPDATE. INPUT indicates that the argument will be passed to the Win32 function. UPDATE indicates that the Win32 function will be passing a value back to SAS.

The third component (Type) is the variable type, just as in the RETURNS statement. This can be any valid C/C++ data type, and the correct value would depend on the Win32 function. Refer to the documentation for the Win32 function for this value.

The fourth component (Addressing) refers to how the argument is actually passed to the Win32 function. There are two customary ways an argument can be made available to the Win32 function, a BYVALUE, which is a call by value or BYADDR, which is a call by address (i.e. memory address). For character arguments, this component can be omitted, as BYVALUE is understood. However, for numeric arguments this compoent should be specified. As with other components of the ARG statement, consult the documentation for the particular Win32 function to be sure.
The fifth component (Format) describes to SAS how the argument is stored. The valid values for this are any of the SAS formats and again will be described in the documentation for the Win32 function. However, since most (if not all!) authors of Win32 API documentation know nothing of SAS formats, you will need to translate their description of this component into the appropriate SAS format.

Finally, the order of the components (after the ARG n) is unimportant, just getting them correct is difficult enough!

SASCBTBL:

This is a special fileref that you must assign to the file that contains the attribute table(s) mentioned above. If you don't use SASCBTBL as the fileref, the DLL call will fail. Fortunately, there are no other requirements for this filename statement; in all other respects it is a simple, typical filename statement. An example of this filename statement could be:

```
Filename sascbtbl ‘c:\temp\win32api.txt’;
```

Modulen/Modulec

These are SAS functions that access the Win32 API DLL functions. They act as a gateway to communicate with the Win32 functions by utilizing the attribute table referenced by the SASCBTBL fileref. These two functions perform the same service, they differ in what they expect to be returned from the Win32 function. Modulen expects a numeric value returned from the Win32 function, and Modulec expects a character value to be returned. More often then not, a numeric value is returned by the Win32.

Advantages of Using Win32 Functions

Often, you can use DOS commands to perform some of the functions that you could use the Win32 API to accomplish. Executing a DOS command from SAS is relatively simple. For example, a code segment to copy a file (i.e. c:\test.txt to d:\test.txt) might look like:

```
Data _null_;
  rc = system(‘copy c:\test.txt d:\test.txt’);
Run;
```

Very simple, no attribute tables to deal with, no weird functions, and no special filename statements. Performing the same task using the Win32 API is a good bit more involved.

First, the attribute table:

C:\my_api\attr_tbls.txt

```
Routine CopyFileA
  module=KERNEL32
  minarg=3   maxarg=3  stackpop=called
  returns=ushort;

arg 1 input char format=$cstr200.; *  FROM ;
arg 2 input char format=$cstr200.;  *  TO     ;
arg 3 input num format=pib4. byvalue;
                 *  1=Do Not Overwrite Existing
*;
                 *  0=Overwrite Existing File
*;
```

Next, the SAS program:

```
filename sascbtbl
c:\my_api\attr_tbls.txt';

data _null_;
 rc = modulen('*e','CopyFileA',
             'c:\temp\orig.txt',
        'c:\temp\api\orig1.txt',
              0);
run;

filename sascbtbl clear;
```

Notice, we never actually reference the SASCBTBL fileref in the SAS code. It is required by the MODULEN function and is used internally by the SAS supervisor. So why might it be advantageous to use the more complicated Win32 API approach over the simpler DOS command? First, the Win32 API function operates much faster than the DOS command. Also, using the DOS command forces SAS to pop-up the DOS window during the execution of the command. This may seem trivial if you are only executing this command infrequently, but when part of a loop that is executed repeatedly, this can be a real issue. Also, if you have users that interact with a system that "keeps popping up those ugly black DOS boxes", you (as the developer) will hear about it! Furthermore, you only need to create the attribute table once, it can be used any number of times in any number of programs as long as the SASCBTBL fileref points to it.

Conclusion

The Win32 API is the heart of the various 32 bit Windows operating systems. Understanding how to access and utilize the functions that make up the DLLs that comprise the Win32 API greatly enhances the ability of the SAS developer to integrate SAS programs and applications with the native windows operating system. However, care must be taken when using the Win32 functions. In most cases, incorrectly accessing Win32 functions will just simply not do anything, but sometimes bad things can happen. Since these functions deal directly with the operating system, memory management, and basic, low-level commands, it is possible to cause program halts and even crash the computer by incorrectly using these functions. But like most very useful tools, just because something bad can happen by using incorrectly doesn't justify not using it at all. The benefits can be truly significant.

References:

"Visual Basic® Programmer's Guide to the Win32 API, The Authoritative Solution" by Dan Appleman
Copyright © 1999 by Macmillan Computer Publishing, USA. All rights reserved.
ISBN 0-672-31590-4

PW Consulting
http://www.pwcons.com/Tips/af/modulen.html

Author Contact

Christopher A. Roper
National Committee for Quality Assurance
2000 L St. NW Suite 500
Washington, D.C.
(202) 955-3562
croper@ncqa.org

Acknowledgments

SAS is a trademark and registered trademark of SAS Institute in the USA and other countries.

MS Excel, MS Word, and MS Access are trademarks and registered trademarks of the Microsoft Corporation in the USA and other countries.

Examples

A flat file (‘c:\temp\win32api.txt’) containing attribute tables for three Win32 functions; GetUserNameA, CopyFileA, and MoveFileA.

C:\temp\win32api.txt

```
Routine GetUserNameA
  minarg=2
  maxarg=2
  stackpop=called
  returns=long;
arg 1 update        format=$cstr200.;
arg 2 input byaddr format=pib4.;

Routine CopyFileA
   module=KERNEL32
   minarg=3
   maxarg=3
   stackpop=called
   returns=ushort;

arg 1 input char format=$cstr200.;  *  FROM filename
*;
arg 2 input char format=$cstr200.;  *  TO   filename  *;
arg 3 input num  format=pib4. byvalue;  *  1=Do Not
Overwrite Existing  *;
                                        *  0=Overwrite Existing File
*;

Routine MoveFileA
   module=KERNEL32
   minarg=2
   maxarg=2
   stackpop=called
   returns=ushort;

arg 1 input char format=$cstr200.;  *  FROM filename
*;
arg 2 input char format=$cstr200.;  *  TO   filename  *;

Routine MessageBoxA
   module=USER32
   minarg=4
   maxarg=4
   stackpop=called
   returns=short;

arg 1 input num format=pub4.  byvalue.;
arg 2 input char format=$cstr200.;  *  FROM filename
*;
arg 3 input char format=$cstr200.;  *  TO   filename  *;
arg 4 input num format=pub4.  byvalue.;  *Push Buttons
;
```

Now, some code that would execute each of these Win32 functions:

```
/**********************************/
/*  Get the network login name              */
/**********************************/
filename SASCBTBL ‘c:\temp\win32api.txt’;

data _null_;
  length usrname $200;
  buffer = 199;
  rc=
 modulen(‘*e’,’Advapi32,GetUserNameA,
  usrname,buffer);
  put USRNAME=;
run;

filename SASCBTBL clear;

/**********************************/
/*  Copy C:\TEMP\ORIG.TXT to            */
/*  D:\TEMP\COPY.TXT                         */
/**********************************/
filename SASCBTBL ‘c:\temp\win32api.txt’;

data _null_;
  rc=modulen('*e','CopyFileA',
     'c:\temp\orig.txt',
     'd:\temp\copy.txt',
      0); /* Replace if exists  */
/*    1);    Fail    if exists  */

run;

filename SASCBTBL clear;

/**********************************/
/*  Move C:\TEMP\ORIG.TXT to            */
/*  D:\TEMP\COPY.TXT                         */
/**********************************/
filename SASCBTBL ‘c:\temp\win32api.txt’;


data _null_;
  rc=modulen('*e','MoveFileA',
     'c:\temp\orig.txt',
     'd:\temp\copy.txt',
      0);
run;

filename SASCBTBL clear;
```

```
/*************************************
/
/*  Display a message box with three buttons */
/*  Yes, No, Cancel.
*/
/*************************************/
filename SASCBTBL ‘c:\temp\win32api.txt’;

data _null_;
  title = 'SUGI Message Box';
  msg = 'Are We Having Fun?';
  msg_box = 3;
  rc=modulen('*ie','MessageBoxA',
                   0,
                   msg,
                   title,
                   msg_box);
run;

filename SASCBTBL clear;
```

Notes about the MessageBoxA API call:

Return codes:
The return codes from the MessageBoxA API call are explaned in the following table.

Return Code	Description
0	Out of memory
1	OK pressed
2	Cancel pressed
3	Abort pressed
4	Retry pressed
5	Ignore pressed
6	Yes pressed

There are 14 types of Message Boxes that can be called; these are determined by the value for the fourth argument to the MessageBoxA function. In the above code, the SAS data step variable MSG_BOX contains this value. The following table lists the valid values for MSG_BOX and their descriptions.

Message Box	Description of Displayed Buttons/Icons
0	OK Button Only
1	OK and Cancel Buttons
2	Abort, Retry, Ignore Buttons
3	Yes, No, Cancel Buttons
4	Yes, No Buttons
5	Retry, Cancel Buttons
16	Icon=Stop Sign
32	Icon=Question Mark
48	Icon=Exclamation Mark
64	Icon=I (info) symbol
0	Button 1 is default
256	Button 2 is default
512	Button 3 is default
4096	All Windows apps are inaccesible until user responds.