The Dynamic Duo - SAS® Software and Dynamic Link Libraries (DLLs) - Capable Of Anything!

Holly Scholz, Innovative Alternatives, Inc., Schaumburg, IL

ABSTRACT

Have you ever tried to create a directory from within a SAS/AF® software application? How about deleting a text file from a DATA step? You could use the X command or the Screen Control Language SYSTEM function to shell out to DOS, then use the appropriate DOS command. Effective, but not very elegant.

The Dynamic Duo comes to the rescue! The SAS System® for Microsoft Windows provides some powerful features that allow you to call Dynamic Link Libraries (DLLs) to perform just about any task that can be accomplished in a Windows environment.

This paper will give you a good working knowledge of how to use the new MODULE functions and CALL routine to call DLL routines from your SAS/AF software application or DATA step programs. It will also provide you with several real life methods that will help get you up and running with DLLs in no time!

INTRODUCTION

Developing applications in SAS/AF software often requires performing tasks at the operating system level. These can run the gamut from copying files or removing directories to editing the system registry to creating icons and graphical images. The benefits of using DLLs versus the X command or SYSTEM function are:

- Smooth integration with the Windows environment - there is no flashing DOS window.
- Consistent look to your applications - using standard Windows features helps give your application a professional appearance.
- Speed - DLL routines are usually faster than the equivalent DOS command.
- More options - there are things you can do with DLLs that you cannot do with a DOS command like displaying a printer setup window.

Where Do I Get A DLL?

If you have Microsoft Windows 3.x or higher on your PC, you already have the power of DLLs at your fingertips. The DLLs that come with Windows 95 and Windows NT are part of the Win32 API (Application Programming Interface). There is also a Win32s API that is a subset of Win32 designed to work with Windows 3.x platforms.

The Win32 API allows systems developers to call commonly used routines from their applications, even though the DLL may be written in another languages, such as C or Visual Basic.

You can even write your own DLLs using many different languages - C is the most popular one - if you cannot find what you need in the Win32 API. You will need a very good understanding of Windows programming to do this however!

SAS Software And DLLs - Together At Last

The SAS System® provides its version of calling DLLs through the use of a set of MODULE functions and CALL routines. These can be used from the DATA step, Screen Control Language (SCL) or from the IML procedure. Their use is documented in the book, *Microsoft Windows Environment: Changes and Enhancements to the SAS System, Release 6.11*. This excellent reference is a good place to start to learn the syntax of these functions. You will find below a brief explanation of this reference material along with some comments about what I found to work the best.

The basic steps for using DLL routines with SAS software are:

1. Create the SASCBTBL attribute table file. This file describes the individual DLL routines you will be using.

2. Assign a FILEREF to the attribute table file.

3. Use the CALL routine or function (MODULE, MODULEN, MODULEC) to call the DLL routine.

CREATING THE SASCBTBL ATTRIBUTE TABLE

The attribute table is the most important part in the process. If you do not get it right - I mean exactly right - it will not work! It can be quite tricky to convert the data types used in a DLL to SAS software data types. There is not always an equivalent.

A reference book that describes the DLL routines and what their parameters are can be very useful. The one I use is *Windows 95 Win 32 Programming API Bible* by Richard Simon. It also helps to have an understanding of C code to interpret the data types.

The ROUTINE Statement (Its anything but routine!)

The ROUTINE statement describes the DLL routine you want to call. It also contains ARG statements that describe each parameter to pass to the DLL routine.

The ROUTINE statement options that I used most are shown in Table 1. Note: If you use the default value for a statement option, you do not need to specify it. The syntax for the ARG statement is displayed in Table 2.

Table 1

`ROUTINE CopyFileA`	The name of the DLL routine you are calling.
`MINARG=3`	Specifies the minimum number of arguments the DLL expects.
`MAXARG=3`	Specifies the maximum number of arguments (some may be optional).
`CALLSEQ=BYADDR`	I use BYADDR calling sequence. You can specify BYVALUE for ARGs as needed.
`STACKPOP=CALLED`	The 32 bit API requires the called routine to pop the stack.
`MODULE=kernel32;`	The name of the DLL (most routines you will use are in the kernel32 DLL).
`ARG 1 INPUT format=$cstr200.;`	You will need an ARG statement for each parameter to the DLL (including the optional ones).
`ARG 2 INPUT format=$cstr200.;`	
`ARG 3 NUM INPUT BYVALUE FORMAT=PIB4.;`	

Table 2

`ARG argument number`	
`NUM or CHAR`	Numeric or character.
`INPUT or OUTPUT or UPDATE`	INPUT is for passing values to the DLL routine; use UPDATE to pass values back from the DLL. I do not recommend using OUTPUT because the data types are not converted.
`NOTREQD or REQUIRED`	I do not recommend using NOTREQD unless you are positive that the DLL will not try to access this argument. Use a comma to hold the omitted argument's place in the call if you use NOTREQD.
`BYADDR or BYVALUE`	If you used BYADDR as the CALLSEQ in the ROUTINE statement, you only need to specify BYVALUE to pass an actual value. BYADDR passes an address to a value (a pointer in C terms).
`FDSTART`	This signifies the beginning of a block of arguments that define a structure. Many DLLs require complex data structures as one or more of their arguments. You will have to look up the definition of the structure as well as the DLL routine, but do not be too intimidated - you just need to break it down into smaller pieces.
`FORMAT`	It is very important to determine the correct format for SAS software to convert the argument to or from. There is a table in the Release *6.11 Changes and Enhancements* document that helps. For the Win32 API, use the C Language Formats.

THE MODULE FUNCTION

The MODULE function or CALL routine is how SAS software communicates with the DLL routine. It contains the name of the DLL routine and passes all of the necessary parameters. It also may receive a return value from the DLL routine.

The syntax for the MODULE function is the same in both SCL and DATA step applications (for IML calls, just add and 'I' after the 'E' in module, i.e.: CALL MODULEI()).

Use the CALL routine when you do not have a return value from the DLL routine.

Otherwise, use the MODULEN function for numeric return values or the MODULEC function for character return values.

The syntax for the MODULE function and CALL routine is displayed in Table 3.

Table 3

```
CALL MODULE(<cntl>, module, arg1, arg2, ..., arg-n);
num=MODULEN(<cntl>, module, arg1, arg2, ..., arg-n);
char=MODULEC(<cntl>, module, arg1, arg2, ..., arg-n);
```

The CNTL value

The CNTL value is an optional parameter that is used mostly for debugging. To use CNTL, precede the value with an '*'. *Hint*: The value '*I' is most useful for debugging (E is implied when you use I). Remove the CNTL value when moving programs to production to reduce LOG output.

The possible values for CNTL are shown in Table 4.

Table 4

I	Prints a hexadecimal representation of the arguments before and after calling the DLL routine.
E	Prints detailed error messages.
S*x*	Specifies that *x* will be used as a separator in the argument list. This is only used when there is no table entry for the routine (not a recommended route!).
H	Provides help on syntax.

The MODULE name and parameters

The module name needs to be specified next. This can be the module name and routine name (i.e.: Kernel32,DeleteFileA) or just the routine name if you included the MODULE option in the ROUTINE statement.

After the module name, all of the non-optional arguments to be passed to the DLL routine are listed. These can be passed as variables or literals, depending on how they are defined.

DLLS IN ACTION: A REAL LIFE EXAMPLE

I have learned quite a bit through trial and error (a lot of error!) in trying to implement these functions in an application I wrote for a client. This application manages source code for the programmers at my client's site. It does a lot of file manipulation and thus needs to interact with the operating system (Windows NT) quite a bit. It provides features to copy and delete files, create and remove directories, etc.

I needed a way to implement these features seamlessly within the application, so DLLs were the answer. The following examples show some of the ways I used DLLs to provide these functions in my application.

The SASCBTBL Attribute Table File

Here is the SASCBTBL attribute table file I created for use with my application (Figure 1). It contains function descriptions for a copy file function, a delete file function, a create directory function and a remove directory function.

Figure 1

```
* FILE FUNCTIONS IN DLL kernel32;
* ------------------------------;
* COPY FILE FUNCTION;
ROUTINE CopyFileA
MINARG=3
MAXARG=3
CALLSEQ=BYADDR
STACKPOP=CALLED
MODULE=kernel32;
ARG 1 INPUT format=$cstr200.;
ARG 2 INPUT format=$cstr200.;
ARG 3 NUM INPUT BYVALUE FORMAT=PIB4.;
*;

* CREATE DIRECTORY FUNCTION;
ROUTINE CreateDirectoryA
MINARG=2
MAXARG=2
CALLSEQ=BYADDR
STACKPOP=CALLED
MODULE=kernel32;
ARG 1 INPUT FORMAT=$CSTR200.;
ARG 2 NUM INPUT BYVALUE FORMAT=IB4.;
*;

* DELETE FILE FUNCTION;
ROUTINE DeleteFileA
MINARG=1
MAXARG=1
CALLSEQ=BYADDR
STACKPOP=CALLED
MODULE=kernel32;
ARG 1 input FORMAT=$Cstr200.;
*;

* REMOVE DIRECTORY FUNCTION;
ROUTINE RemoveDirectoryA
MINARG=1
MAXARG=1
CALLSEQ=BYADDR
STACKPOP=CALLED
MODULE=kernel32;
ARG 1 INPUT FORMAT=$CSTR200.;
*;
```

SCL Methods To Implement The DLL Calls

The following SCL catalog entry (Figure 2) contains methods to call the DLLs. Most of these examples are written in SCL.

However, the delete directory method is shown as a DATA step to illustrate just how alike the two are.

Figure 2

```
/*******************************************
  DLLMETHS.SCL
   Methods to perform file functions using DLLs.
*******************************************/
LENGTH dirtomk remdir $150;

COPYFILE:
  /* Method to copy a file to a designated
directory */
 method location $150 filename $50 copydir $150
   optional=copyname $50;

   /* Only continue if file to copy exists */
   if fileexist(trim(location)||'\'||filename)
     then do;

     /* Get the fileref for the DLL file */
     call method('filemeth.scl','getDLL',rcDLL);
     if rcDLL ne 0 then do;
       call display('message.frame','E','Cannot'
         ||' access DLL file!');
       return;
     end;

     /* If copying to the same filename but in
        a different directory */
     if copyname='' then copyname=filename;

     /* Check if copy to directory exists */
     rcfn=filename('copydir',copydir);

     /* Create the copy-to directory if it
        doesn't exist */
     if not fexist('copydir') then do;
       dirtomk=scan(copydir,1,'\');
       i=2;

       /* Create subdirectories at each level
          of the directory path */
       do until(scan(copydir,i,'\')='');
         remdir=scan(copydir,i,'\');
         dirtomk=dirtomk||'\'||remdir;
         if remdir ne '' then do;
           rcfn=filename('copydir',dirtomk);

           /* Create the subdirectory if it
              doesn't already exist */
           if not fexist('copydir') then do;
             rcmd=modulen('CreateDirectoryA',
               dirtomk,0);
           end;
         end;
         i+1;
       end;
     end;

     /* Perform the actual file copy */
     rcc=modulen('CopyFileA',trim(location)||'\'
       ||trim(filename),trim(copydir)||'\'
       ||trim(copyname),0);

     /* Issue error message if file not
        copied */
     if not rcc then
       call display('message.frame','E',
         'Could not copy file '||
         trim(filename)||'!');

     /* Release the DLL fileref */
     call method('filemeth.scl','relsDLL');

   end;
 endmethod;
RETURN;

CRTDIR:
    /* Method to create a directory */
  method dirtocrt $200;

    /* Get the fileref for the DLL file */
    call method('filemeth.scl','getDLL',rcDLL);
    if rcDLL ne 0 then do;
      call display('message.frame','E',
        'Cannot access DLL file!');
      return;
    end;

    /* Only continue if directory to create
       doesn't exist */
    rcfn=filename('crtdir',dirtocrt);
    if not fexist('crtdir') then do;

      /* Create the directory */
      rccd=modulen('CreateDirectoryA',
        dirtocrt,0);

      /* Issue an error message if unable to
         create */
      if not rccd then
        call display('message.frame','E',
          'Unable to create directory ',
          trim(dirtocrt)||'!');
    end;

    /* Release the DLL fileref */
    call method('filemeth.scl','relsDLL');
  endmethod;
RETURN;

DELDIR:
    /* Method to delete a directory */
  method dirtodel $200;

    /* Get the fileref for the DLL file */
    call method('filemeth.scl','getDLL',rcDLL);
    if rcDLL ne 0 then do;
      call display('message.frame','E',
        'Cannot access DLL file!');
      return;
    end;

    /* Only continue if directory to delete
       exists */
    rcfn=filename('deldir',dirtodel);
    if fexist('deldir') then do;

      /* This is an example of how to use the
         modulen function in a datastep.  As
         you can see, it is quite similar. */
      submit continue;
        data _null_;
          /* Delete the directory */
          rcrd=modulen('RemoveDirectoryA',
            trim("&dirtodel"));
          call symput('rcrd',rcrd);
        run;
      endsubmit;

      /* Gets the return code from the delete
         DLL */
      rcrd=symgetn('rcrd');

      /* Issue an error message if unable to
         delete */
      if not rcrd then
        call display('message.frame','E',
          'Unable to delete directory ',
          trim(dirtodel)||'!');
    end;

    /* Release the DLL fileref */
    call method('filemeth.scl','relsDLL');
  endmethod;
RETURN;

DELFILE:
 /* Method to delete a file */
 method location $150 filename $50;

   /* Only continue if the file exists */
   if fileexist(trim(location)||'\'
     ||filename) then do;
```

```
      /* Get the fileref for the DLL file */
      call method('filemeth.scl','getDLL',
        rcDLL);
      if rcDLL ne 0 then do;
        call display('message.frame','E',
          'Cannot access DLL file!');
        return;
      end;

      /* Delete the file */
      rcd=modulen('DeleteFileA',trim(location)
        ||'\'||trim(filename));

      /* Issue an error message if unable to
         delete */
      if not rcd then
        call display('message.frame','E',
          'Unable to delete file ',
          trim(filename)||'!');

      /* Release the DLL fileref */
      call method('filemeth.scl','relsDLL');
   end;

 endmethod;
RETURN;

GETDLL:
  /* Method to get a fileref to the DLL file */
  method rcDLL 8;
    /* The root path is stored in a macro
       variable */
    dirpath=symget('dirpath');

    /* Determine what operating system running
       on */
    sysscpl=upcase("&sysscpl");

    /* If 32 bit o.s., use 32 bit DLL file,
       otherwise use 16 bit DLL file */
    if sysscpl in ('WIN_95','WIN_NT','WIN_NTSV')
      then
      rcDLL=filename('SASCBTBL',trim(dirpath)||
        '\common\DLL32\attrfile\filefunc.txt');
    else
      rcDLL=filename('SASCBTBL',trim(dirpath)||
        '\common\DLL16\attrfile\filefunc.txt');
  endmethod;
RETURN;

RELSDLL:
  /* Method to release the DLL fileref */
  method;
    rc=filename('SASCBTBL','');
  endmethod;
RETURN;
```

REFERENCES

Richard Simon, Michael Gouker and Brian Barnes (1996), *Windows 95 WIN 32 Programming API Bible*, Corte Madera, CA: Waite Group Press

SAS Institute Inc. (1995), *Microsoft Windows Environment: Changes and Enhancements to the SAS System*, Release 6.11, Cary, NC: SAS Institute Inc.

CONTACTING THE AUTHOR

Holly Scholz
Email: holly.scholtz@tappharma.com
Phone: (847) 267-5888.
Innovative Alternatives, Inc.
999 Plaza Drive, Suite 670
Schaumburg, IL 60173