SAS1572-2015

When I'm 64-bit: How to Still Use 32-bit DLLs in Microsoft Windows

Rick Langston, SAS Institute Inc., Cary, NC

ABSTRACT

Now that SAS® users are moving to 64-bit Microsoft Windows platforms, some are discovering that vendor-supplied DLLs might still be 32-bit. Since 64-bit applications cannot use 32-bit DLLs, this would present serious technical issues. This paper explains how the MODULE routines in SAS can be used to call into 32-bit DLLs successfully, using new features added in SAS® 9.3.

INTRODUCTION

If you have used external DLLs within SAS on Windows via the MODULE function, and you are transitioning from a 32-bit version of SAS to a 64-bit version, you might encounter problems in trying to use those DLLs.

If those DLLs are the standard Windows DLLs, such as KERNEL32, then you should not encounter problems. However, if those DLLs are supplied by third parties, then you will definitely encounter problems that will need your attention.

An example of such a DLL is APRLIM.DLL, provided by the Agency For Healthcare Research and Quality (AHRQ). It is distributed as a 32-bit DLL, but users of 64-bit SAS would want to use it as they transition to 64-bit SAS on Windows.

WINDOWS SYSTEM LIMITATIONS

The fundamental problem is that a 32-bit DLL cannot be loaded and used by a 64-bit application. So if a third party has supplied you with a DLL to use, and it is a 32-bit DLL, a 64-bit application such as SAS cannot load this DLL and call into it.

Don't confuse this with 32-bit application usage in 64-bit Windows. That is fully acceptable. In fact, the 32-bit version of SAS can run fine on a 64-bit Windows platform. However, a 32-bit DLL will fail to load when requested by a 64-bit application.

Under normal circumstances, you would be forced to stay with a 32-bit version of SAS in order to use the DLL, or you'd have to persuade the third party to provide a 64-bit version of the DLL. Neither of these options may be feasible for you.

THERE IS HOPE!

As the developer of the MODULE functions, I realized that users may be caught in this situation where they needed 32-bit DLLs to work in 64-bit SAS. With this in mind, I created a special feature starting in the second maintenance release of SAS 9.3.

I introduced the DLLTYPE=32 option that indicates the DLL is a 32-bit DLL. This option appears on the ROUTINE statement for the routine and module of interest within the SASCBTBL file. For example:

```
filename sascbtbl temp;
data _null_; file sascbtbl;
     input; put _infile_; cards4;
routine myfunc module=mymod minarg=2 maxarg=2 returns=int dlltype=32;
arg 1 num input format=ib4.;
arg 2 num input format=ib4.;
;;;;
```

When MODULE encounters the DLLTYPE=32 value, it starts a background 32-bit process and requests that the process load the DLL. All data is passed to the process in order to call into the DLL. This is all transparent to the user.

Note that if you are still using a 32-bit version of SAS, the DLLTYPE=32 option can be used, and the DLL will be loaded normally.

SPECIAL CONSIDERATIONS

If you are actually passing the address of a value, and not using the BYADDR attribute, then you will need to change your ARG statement for this argument. This is because you cannot pass a 64-bit pointer value to a 32-bit process, and you will almost certainly cause a failure of the process. For example, in 32-bit SAS you can do the following:

```
filename sascbtbl temp;
data _null_; file sascbtbl;
     input; put _infile_; cards4;
routine myfunc3 module=mymod minarg=2 maxarg=2;
arg 1 char input byvalue format=$char4.;
arg 2 num input byvalue format=pib4.;
;;;;
data temp;
     x='abc'; y=length(x);
     z=modulen(addrlong(x),y);
     put z=;
     run;
```

This will work fine in 32-bit SAS because the pointer will fit in a 4-byte character argument. The myfunc3 routine will receive as its two arguments a pointer to the variable x and the length of the value.

This cannot work with DLLTYPE=32. MODULE will not know that argument 1 is actually a pointer, so it will pass a 4-byte value, which is an 8-byte pointer truncated to 4 bytes! The 32-bit process will get confused with this pointer and unpredictable results will occur.

Instead, you should do the following:

```
filename sascbtbl temp;
data _null_; file sascbtbl;
     input; put _infile_; cards4;
routine myfunc3 module=mymod minarg=2 maxarg=2 dlltype=32;
arg 1 char input byvalue format=$ptr. datalen=8;
arg 2 num input byvalue format=pib4.;
;;;;
data temp;
     length x $8;
     x='abc'; y=length(x);
     z=modulen(addrlong(abc),y);
     put z=;
     run;
```

Inside of "hiding" that argument 1 is a pointer that indicates that it is via the $PTR pseudo-format. The MODULE will now know that a pointer is used, and will instead copy the data down to the process and provide it a proper pointer. Note that the DATALEN= option must also be used. This specifies the length of the data pointed to by the pointer. It should match the length given in the LENGTH statement.

Note that you can use this same approach with 32-bit SAS. With 32-bit SAS, the DATALEN= option is ignored if the DLLTYPE= option is not given.

PERFORMANCE CONSIDERATIONS

As you might expect, there is a performance penalty when a 32-bit DLL is used via the DLLTYPE=32 option within 64-bit SAS. I have found that a 32-bit DLL via DLLTYPE=32 runs at a factor of 17 times slower than a 64-bit DLL. This is because all data must be passed via named pipes to and from the 32-bit process, versus everything running in local memory. However, if you don't have access to a 64-bit DLL, and you're running 64-bit SAS, you may be willing to pay this performance penalty.

Be especially aware if you are working with large character strings, such as in this example:

```
filename sascbtbl temp;
data _null_; file sascbtbl;
     input; put _infile_; cards4;
routine myfunc2 module=mymod minarg=2 maxarg=2 returns=int dlltype=32;
arg 1 char input format=$char500.;
arg 2 char input format=$char500.;
;;;;
data temp; x='abc'; y='def'; z=modulen('myfunc2',x,y);
```

In this example, our variables x and y are only length 3, but because the specifications for the arguments indicate that length 500 arguments are needed, the values of x and y will be padded to 500 characters each, and all 1000 of those characters will be sent to the 32-bit process.

CONTACT INFORMATION

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

Rick Langston
SAS Institute Inc.
Rick.Langston@sas.com