www.sheepsqueezers.com

Creating C DLLs for SAS

with a section on compiling for 64-bit Itanium2 (IA-64)

-

This presentation as well as other presentations and documents found on the sheepsqueezers.com website may contain quoted material from outside sources such as books, articles and websites. It is our intention to diligently reference all outside sources. Occasionally, though, a reference may be missed. No copyright infringement whatsoever is intended, and all outside source materials are copyright of their respective author(s).

Table of Contents

Introduction

This document is a brief introduction to creating C DLLs for use with the SAS module functions. These functions allow C DLLs to be called from within a SAS Data Step.

A section at the end of this document outlines how to compile for 64-bit Itanium-2 (IA-64).

Section One – What You Will Need

In order to create C DLLs, you will need to have a compiler for the operating system on which you are running SAS. That is, if you are running SAS on a *Windows* platform, you will need to have either *Microsoft Visual C++* or the *Intel C Compiler* (assuming your CPU is an Intel chip). There are a variety of other compilers, like *Borland C++*, etc., so take your pick. If you are running SAS on a Linux/Unix platform, then you can use `gcc` or `cc`, both of which come with the operating system. Intel also makes a C compiler for Linux. If your chip is not an Intel chip, check out the chip manufacturer's webpage for information on their C compiler.

Section Two – How to Do It

A NOTE ON DECORATED FUNCTION NAMES

Some compilers change the name of compiled functions/subroutines for internal use. These are called "decorated" names and the extent of the decorating depends on the calling convention used.

ACCESSING FUNCTIONS IN A .DLL FROM AN .EXE (C++)

Here is an example of how you would normally create a simple function, called DOIT, in C. This takes one parameter, called anum, which accepts a floating point number, adds one to it ("++anum"), and returns the result to the calling function.

Figure 1: DOIT Function (C vs. VB)

```
C Example:
      double DOIT(double anum)
            {
                  return ++anum;
            }

VB Example:
      Function DOIT(anum As Double) as Double
            DOIT=anum+1
      End Function
```

It is often a preferable programming practice to build a library of useful functions as a DLL (dynamic link library). You can expose these functions to be called from other, standard programs (.exe's), but the code has to contain certain information to do this. The following example shows how the DOIT function would be coded to work from within a DLL:

Figure 2: The DOIT function, within a DLL

```
//The First Program: This will be the DLL.

#include <stdio.h>

extern "C" __declspec( dllexport ) double __cdecl DOIT(double);

extern "C" __declspec( dllexport ) double __cdecl DOIT(double anum)
      {
            return ++anum;
      }
```

The program line…

```
extern "C" __declspec( dllexport ) double __cdecl DOIT(double);
```

…tells the compiler that there is a function called DOIT which takes one parameter (double) and returns a double. Note that this does not actually contain any of the code of the function itself; it's just a heads-up for the compiler.

The line, dissected:

1. `extern "C"`
   Indicates that functions in the DLL will be made externally available to other C++ programs

2. `__declspec( dllexport )`
   Similar to the Public keyword in Visual Basic, you have to let the compiler know which functions are available for use outside the DLL itself. The keywords tell the compiler to export the function.

3. `__cdecl`
   Insures dll compilation will not resulted in "decorated" function names (see below)

In this sample program, the C++ printf function prints output to the screen. The .dll is called that contains the DOIT function, and the parameter '5' is passed to it.

Figure 3: Calling the DOIT function from a C++ EXE

```
//The Second Program: This will be the EXE and is the "main" program.

#include <stdio.h>

extern "C" double __cdecl DOIT(double);

int main(int argc,char* argv)
{
      printf("5+1=%f\n",DOIT(5));
      return 0;
}
```

USING DUMPBIN

As mentioned above, function names tend to be *decorated* by the compiler for their own internal use. Once you compile the first program into a DLL, you can use the Microsoft Visual C++ program `dumpbin.exe` to actually see the function names as they have been decorated by the compiler:

Figure 4: sample output from running Dumpbin

```
>dumpbin.exe /EXPORTS cpgm.dll

Microsoft (R) COFF Binary File Dumper Version 6.00.8447
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

Dump of file cpgm.dll

File Type: DLL

  Section contains the following exports for cpgm2.dll

           0 characteristics
    439626F8 time date stamp Tue Dec 06 19:04:08 2005
        0.00 version
           1 ordinal base
           2 number of functions
           2 number of names

    ordinal hint RVA      name

          1    0 00001000 DOIT
          2    1 0000100C LenOfStr

  Summary

        2000 .data
        1000 .rdata
        1000 .reloc
```

The red text above shows you the names of the functions available (exported) in the DLL.

Note that the names DOIT and LenOfStr aren't decorated, because the "__cdecl" keyword prevented the decoration from happening. Without "__cdecl", the names look something like this:

```
ordinal hint RVA      name

      1    0 00001000 _DOIT@8
      2    1 0000100C _LenOfStr@4
```

The problem with decorated functions for SAS is that the text file containing the definitions of the functions for SAS's own internal use – called sascbtbl.txt and is created by the user (see below) – does not like the at-sign you see above.

COMPILING A DLL

Now, in order to compile the DLL using the Intel C/C++ Compiler, you issue this command at the command line:

```
ICL /I"C:\Program Files\Microsoft Visual Studio\VC98\Include" /O2 /LD /FD
cpgm1.cpp
```

This line, dissected:

1. `/I` "<Path>"
   tells the compiler where to find the headers (like stdio.h, string.h, etc.).

2. `/LD`
   tells the compiler to generate a DLL instead of and EXE file.

3. `/02`
   The recommended optimization level for the compiler

4. /FD
   Generates file dependencies related to the Microsoft C/C++ compiler.

THE OBJECT (.OBJ) FILE

After the compilation of the DLL is completed, the compiler also generates a .OBJ file, or object file. This object file is used in the compilation of the second program (see below).

COMPILING A PROGRAM (.EXE) THAT CALLS A .DLL

To compile a second program in a way that allows that program, you would issue this command at the command line:

```
ICL /I"C:\Program Files\Microsoft Visual Studio\VC98\Include" /O2 /Fe TEST1.exe
cpgm1.obj cpgm2.cpp
```

This line, dissected:

1. /Fe TEST1.exe
   option tells the compiler what to call the resulting executable (.EXE). In this case, it's 'TEST1.exe'

2. cpgm1.obj
   Without this, the compiler would never find the function DOIT.

SAS provides the function 'modulen()', which allows you to call external DLLs (not ActiveX DLLs). In order to do use the modulen() function in SAS, you need to create a text file -- called an Attribute Table -- to let SAS know what to expect. Here is the Attribute Table for the DOIT function. I called this text file sascbtbl.txt.

Figure 5: Attributes Table file for SAS .dll usage

```
ROUTINE DOIT MINARG=1 MAXARG=1 STACKPOP=CALLER MODULE=CPGM1 RETURNS=DOUBLE;

ARG 1 INPUT NUM BYVALUE FORMAT=RB8.;
```

The first line, dissected:

1. ROUTINE DOIT
   Names the embedded function the file is about to describe.

2. MINARG=1 and MAXARG=1
   Specifies the minimum and maximum number of arguments the DOIT function can handle.

3. STACKPOP=CALLER
   This is specified if you used __cdecl in the function name (See above).

4. MODULE=CPGM1.
   Specifies the DLL name (aka, module) name. This module must be in some subdirectory within the PATH environment variable.

5. RETURNS=DOUBLE
   The returning data type of the function, double in this case.

The second line, dissected:

6. ARG 1
   The next line specifies more detailed information about the parameter itself. Since there is just one parameter, we have ARG 1; if we had two parameters we would have one line starting with ARG 1 and the next line starting ARG 2.

7. INPUT

keyword tells SAS that the parameter is an incoming parameter.

8. BYVALUE
   Tells SAS to pass the actual value and not an address of the value (similar to Visual Basic's BYVAL or BYREF).

9. FORMAT=
   Tells SAS how to pass different data types to the function DOIT. Since the parameter of DOIT is a double, SAS's documentation specifies that you use FORMAT=RB8. to pass any values to the function.

ACCESSING FUNCTIONS FROM A .DLL IN SAS

Now that you have an Attribute Table defined, you can call the function DOIT from within SAS. Use the following sample SAS Script to access the .dll created above.

Figure 6: Sample SAS Script that uses an external .DLL

```
/* This is REQUIRED and MUST be called SASCBTBL!! */
filename sascbtbl "C:\TEMP\sascbtbl.txt";
run;

data test1;
      anum=50;
      anewnum=modulen("*iet","CPGM1,DOIT",anum);
run;

proc print data=test1;
run;
```

The first thing is to set up a FILENAME to the sascbtbl.txt Attribute Table text file we created. The FILENAME statement must be called SASCBTBL or everything that follows won't work.

Next, we have a data step which creates the variable anum and sets it to 50. We then create the variable anewnum as a result of calling out C function DOIT. We use the modulen() function.

The modulen line, dissected:

1. "*iet".
   a set of flags useful for debugging, which prints useful information to the SAS Log file.

2. `CPGM1,DOIT"`
   This is a quoted string with the name of the DLL (without the .DLL extension) followed by a comma followed by the name of the actual function DOIT.

3. `anum`
   The next parameter is the actual the parameter to the DOIT function. Here we are passing the value of anum – 50 – to DOIT and we will get back 51 in the variable anewnum.

Section Three – Example C Program, Windows Batch Command File and SASCBTBL

```
//*-------------------------------------------------------------------------*
//* Program:      prjSASDLLTEST.c                                           *
//* Program Type: C                                                         *
//* Authors:                                                                *
//* Date:                                                                   *
//*                                                                         *
//* Application:  Test C Program                                            *
//*                                                                         *
//* Abstract:     Test C Program for SAS.                                   *
//*                                                                         *
//* Invocation:   None.                                                     *
//*                                                                         *
//* Assumptions:  Create a DLL called MYSASFN.DLL.                          *
//*                                                                         *
//* Parameters:   N/A                                                       *
//*                                                                         *
//* Input:        N/A                                                       *
//*                                                                         *
//* Output:       MYSASFN.DLL                                               *
//*                                                                         *
//* Example:      None.                                                     *
//*                                                                         *
//* Notes:        Use this file along with the exports file prjSASDLLTEST.def *
//*               to create MYSASFN.DLL.                                    *
//*                                                                         *
//* Modification History:                                                   *
//* Date          Prog    Mod #   Reason                                    *
//* ---------     ----    -----   ------------------------------------------ *
//*-------------------------------------------------------------------------*


//*-----------------------------------------*
//* Includes                                *
//*-----------------------------------------*
#include "string.h"
#include "wtypes.h"
#include "math.h"
#include "stdio.h"
#include "stdlib.h"
#include "windows.h"
#include "process.h"

//*-----------------------------------------*
//* Function Prototypes                     *
//*-----------------------------------------*
extern "C" __declspec( dllexport ) long __cdecl LenOfStr(char *);

//*-----------------------------------------*
//* FUNCTIONS                               *
//*-----------------------------------------*

//*-----------------------------------------*
//* Name: LenOfStr                          *
//* Purpose: Computes the length of a string.  *
//*-----------------------------------------*
extern "C" __declspec( dllexport ) long __cdecl LenOfStr(char *lpString)
{
 return (long) strlen(lpString);
}
```

Example SAS Test Program

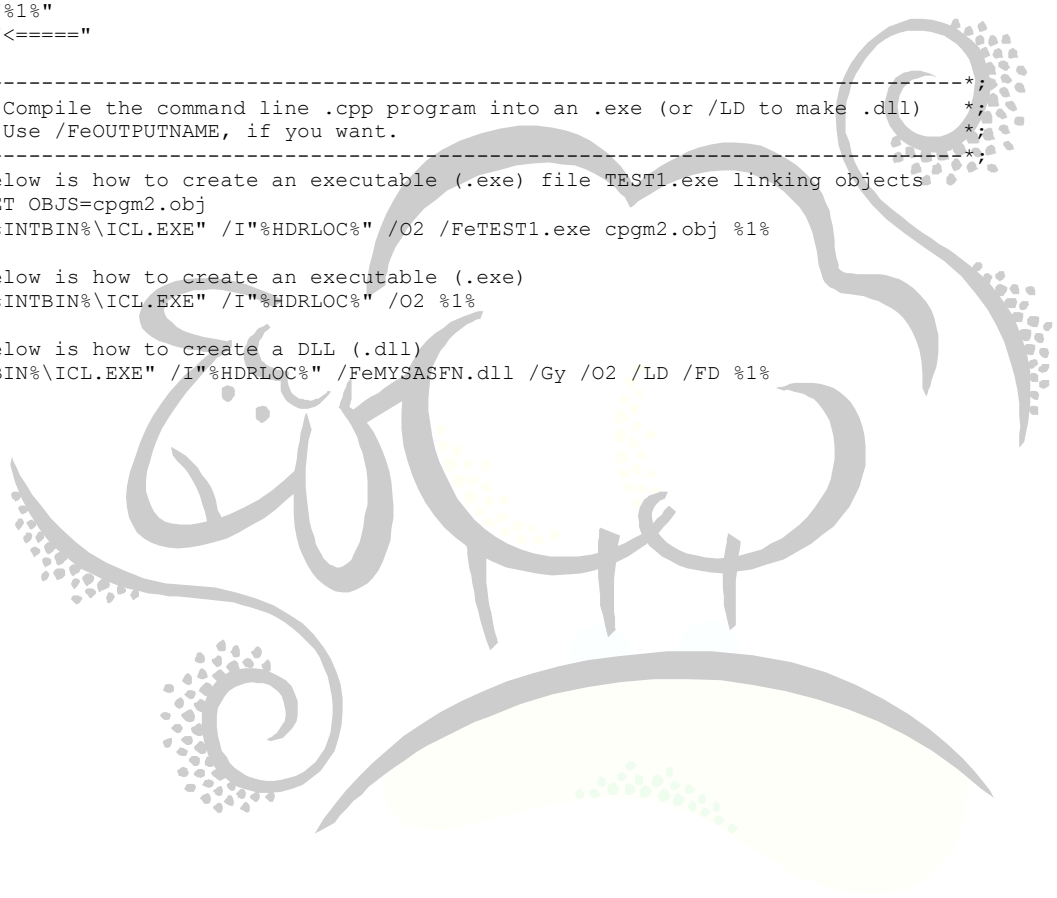
```
filename sascbtbl "C:\TEMP\sascbtbl.txt";
run;

data test1;
 A_String="This is a string!";
 Len_of_A_String=modulen("*","MYSASFN,LenOfStr",A_String);
 output;
run;

proc print data=test1;
run;

filename sascbtbl clear;
run;
```

Example SASCBTBL.TXT File

```
*------------------------------------------------------------------------------*;
* Program:      sascbtbl.txt                                                   *;
* SAS Version:                                                                 *;
* Authors:                                                                     *;
* Date:                                                                        *;
*                                                                              *;
* Application:  Example SASCBTBL File                                          *;
*                                                                              *;
* Abstract:     SASCBTBL file for the SAS call module MYSASFN.dll.             *;
*                                                                              *;
* Invocation:   None.                                                          *;
*                                                                              *;
* Assumptions:  None.                                                          *;
*                                                                              *;
* Parameters:   None.                                                          *;
*                                                                              *;
* Input:        None.                                                          *;
*                                                                              *;
* Output:       None.                                                          *;
*                                                                              *;
* Example:      None.                                                          *;
*                                                                              *;
* Notes:        1) The DLL MYSASFN.DLL contains three functions as outlined    *;
*                  below.                                                      *;
*                  a) The function LenOfStr is used for test and is not        *;
*                     necessarily a replacement for SAS`s LEN() function.      *;
*      Ex: intLenOfStr=modulen("*","MYSASFN,LenOfStr",strname)                 *;
*                                                                              *;
*               2) Don`t forget to copy the DLL call module MYSASFN.dll to     *;
*                  C:\WINNT\SYSTEM32 or the calls to this module will fail.    *;
*                  (or to a directory defined within the path).                *;
*                                                                              *;
* Modification History:                                                        *;
* Date          Prog    Mod #   Reason                                         *;
* ---------     ----    -----   ---------------------------------------------- *;
*------------------------------------------------------------------------------*;

*------------------------------------------------------------------------------*;
* LenOfStr Function                                                            *;
*------------------------------------------------------------------------------*;
ROUTINE LenOfStr MINARG=1 MAXARG=1 STACKPOP=CALLER MODULE=MYSASFN RETURNS=LONG;
 ARG 1 UPDATE FORMAT=$CSTR200.;
```

Example Batch Command File for Windows

```
REM *---------------------------------------------------------------------------*;
REM * Location of the Intel compiler \bin folder.                               *;
REM *---------------------------------------------------------------------------*;
SET INTBIN=C:\Program Files\Intel\Compiler\C++\9.0\IA32\Bin

REM *---------------------------------------------------------------------------*;
REM * Location of C++ headers (*.h) Files.                                      *;
REM *---------------------------------------------------------------------------*;
SET HDRLOC=C:\Program Files\Microsoft Visual Studio\VC98\Include

REM *---------------------------------------------------------------------------*;
REM * Location of needed library (*.lib) Files.                                 *;
REM *---------------------------------------------------------------------------*;
SET LIB=C:\Program Files\Intel\Compiler\C++\9.0\IA32\Lib\;C:\Program Files\Microsoft Visual
Studio\VC98\Lib

ECHO "COMPILING THE FOLLOWING PROGRAM====>"
ECHO "%1%"
ECHO "<====="

REM *---------------------------------------------------------------------------*;
REM * Compile the command line .cpp program into an .exe (or /LD to make .dll)  *;
REM * Use /FeOUTPUTNAME, if you want.                                           *;
REM *---------------------------------------------------------------------------*;
REM Below is how to create an executable (.exe) file TEST1.exe linking objects
REM SET OBJS=cpgm2.obj
REM "%INTBIN%\ICL.EXE" /I"%HDRLOC%" /O2 /FeTEST1.exe cpgm2.obj %1%

REM Below is how to create an executable (.exe)
REM "%INTBIN%\ICL.EXE" /I"%HDRLOC%" /O2 %1%

REM Below is how to create a DLL (.dll)
"%INTBIN%\ICL.EXE" /I"%HDRLOC%" /FeMYSASFN.dll /Gy /O2 /LD /FD %1%
```



Section Four – Creating a 64-Bit DLL for Itanium2 (IA-64) for Use with SAS

1. In order to compile your SAS DLL for Itanium-2, you will need to download the Windows 2003 R2 Platform Software Development Kit (PSDK) from this location:

http://www.microsoft.com/downloads/details.aspx?FamilyId=E15438AC-60BE-41BD-AA14-7F1E0F19CA0D&displaylang=en

Download the ISO Image and burn it on a CD-ROM using your CD-ROM burning software and using the **Burn ISO Image** option (or similarly named feature).

2. Install the software. Insert the CD-ROM in your CD player and it should start automatically. Make sure to choose to install all of the components.
3. Copy your 32-bit version of your SAS code over to another subdirectory, say, SASIA64.
4. Copy the following into a text document:

```
cl /G2 /Wp64 /FeMYSASFNIA64.dll /LD /O2 /GS /MD prjSASDLLTEST.cpp
```

Save this as CompileSAS64.cmd. Note that you will have to change the text after the /Fe option to reflect your DLL name, and change the text prjSASDLLTEST.cpp to reflect your C++ program name.

5. Next, click on Start…Program Files…Microsoft Platform SDK for Windows 2003 Server R2…Open Build Environment Window…Windows Server 2003 64-Bit Build Environment…Set Win Svr 2003 IA64 Build Environment (Debug). This will open up a command window, set the appropriate environment variables and allow you to compile and link your programs from the command line. The other options are Retail (when you are done debugging) or for x64 (AMD, etc.)
6. Begin to compile and link your SAS DLL. Test it as described in the previous sections above.

Support sheepsqueezers.com

If you found this information helpful, please consider supporting sheepsqueezers.com. There are several ways to support our site:

- ☐ Buy me a cup of coffee by clicking on the following link and donate to my PayPal account: Buy Me A Cup Of Coffee?.
- ☐ Visit my Amazon.com Wish list at the following link and purchase an item: http://amzn.com/w/3OBK1K4EIWIR6

Please let me know if this document was useful by e-mailing me at comments@sheepsqueezers.com.