Create DLLs (dynamic loaded libraries) with MingW

Autor: Michel D. Schmid

Datum: April 2, 2009

Contents

1 Introduction

Using open source tools to create dynamic loaded libraries (dll's) for the Windows platform is still no easy job. From my point of view, this depends on the rarely informations you can find on the internet.

1.1 Open source tools

I tested all the steps in this tutorial with following tools:

1. MinGW-5.1.3 (**Min**imal **G**nu for **W**indows), containing:
   (a) GCC 3.4.5, we won't use more for this example!

2 Example 1: Hello World!

2.1 The source code

We will start writing dll's with the world famous example "Hello World". To do this, we are using three files:

1. example_dll.h
2. example_dll.cpp
3. hello.cpp, contains the main function

The content of example_dll.h looks like:

```
#ifdef BUILD_DLL
/* DLL export */
#define EXPORT __declspec(dllexport)
#else
/* EXE import */
#define EXPORT __declspec(dllimport)
#endif

extern "C"
{
    EXPORT void hello(void);
}
```

The file example_dll.cpp contains:

```
#include <stdio.h>
#include "example_dll.h"

EXPORT void hello(void) {
    printf ("Hello\n");
}
```

And last, the containt of hello.cpp:

```
extern "C"
#include <windows.h>
#include <stdio.h>
int main () {
    /*Typedef the hello function*/
    typedef int(*pfunc)(void);
//            typedef int(*PFNAPI2)(void);
// PFNAPI2 hello;

    /*Windows handle*/
    HINSTANCE hdll;

    /*A pointer to a function*/
    pfunc hello;

    /*LoadLibrary*/
    hdll = LoadLibrary("message.dll");

    /*GetProcAddress*/
    hello = (pfunc)GetProcAddress(hdll, "hello");

    /*Call the function*/
    hello();
    return 0;

}
```

2.2 Compiling the code

So let's compile the code. You should have a GNU-C++ compiler and the source code (copied or written by your self) on your local hard disk. If you have both you should be able to compile the example as descriped below.

Firstly, we create the object code of the dll. Enter at the command prompt:

```
g++ -c -DBUILD_DLL example_dll.cpp
```

Secondly, we create the dll. We choose the name *message.dll*.

```
g++ -shared -o message.dll example_dll.o -Wl,–out-implib,message.a
```

After you entered above commands you will see a message at the prompt like:: "Creating library file: message.a"

The **-DBUILD_DLL** compiler option causes the dll's functions to be declared as *dllexport*, this means that these functions will be "exported" from the dll and available to client applications.
The next compiler option **-shared** creates the .dll at the place of a .exe.
Almost at the end of the command prompt line there is one more option. This option is used to control the linker. The option **–out-implib** is used to create an additional library. This one will be *imported* later.

Thirdly, you can read the commands to create the .exe

```
g++ -o hello.exe hello.cpp
```

Now you should have an executable file named *hello.exe*. If you have already studied the code of the *hello.exe*, you will know, that we get in touch with the windows api, at least with a very small part of it.

3 Example 2: more functions

3.1 The source code

For this example, we using the three files we used before again and write some more code to the files:

1. example_dll.h
2. example_dll.cpp
3. hello.cpp, contains the main-Funktion

The file example_dll.h now looks like:

```
#ifdef BUILD_DLL
/* DLL export */
#define EXPORT __declspec(dllexport)
#else
/* EXE import */
#define EXPORT __declspec(dllimport)
#endif

extern "C"
{
    EXPORT void hello(void);
    EXPORT void printString(inti);
    EXPORT int square(inti);
}
```

We export two more functions and of course, we have to implement both of them:

```
#include <iostream>
#include <stdio.h>
#include "example_dll.h"

EXPORT void hello(void) {
    printf ("Hello\n");
}

EXPORT void printString(inti) {
    std::cout « i « "\n";
}

EXPORT int square(inti) {
    return i * i ;
}
```

And now once more the code of hello.cpp:

```
extern "C"
#include <windows.h>
#include <stdio.h>
#include <iostream>
int main () {
    /*Typedef the hello function*/
    typedef void(*pfunc)(void);
    typedef void(*pfunc2)(int);
    typedef int(*pfunc3)(int);

    /*Windows handle*/
    HINSTANCE hdll;

    /*A pointer to a function*/
    pfunc hello;
    pfunc printString;
    pfunc square;

    /*LoadLibrary*/
    hdll = LoadLibrary("message.dll");

    /*GetProcAddress*/
    hello = (pfunc)GetProcAddress(hdll, "hello");
    /*Call the function*/
    hello();

    /*GetProcAddress*/
    pProcAdello = (pfuncg0g0G00000rg000RG0g0G0.320.110.78rg0.320.110.78RG(GetProcAdd)1(ress)T0g



    /*GetProcAddress*/
    square = (pfunc3)GetProcAddress(hdll, "square");
    /*Call the function*/
    intiSquare = square(10);
    std::cout « "iSquare:  " « iSquare « "\n";


    return 0;

}
```

I think I don't have to write a lot about the code above. We did only simple modifications which should be clear.

3.2 Compiling the code

Do it on the same way as in example 1.

4 Example 3: Using classes inside dll's

Will follow shortly (what this ever mean by a family man :-)

5 Bibliography

Following links are used until now:

[1] http://sig9.com/node/35
[2] http://www.users.fh-sbg.ac.at/.../.../20erstellen.pdf