CREACION Y USO DE DLLs UTILIZANDO DEV-C++

CAPITULO I: *Creación de la DLL y de la aplicación de test*

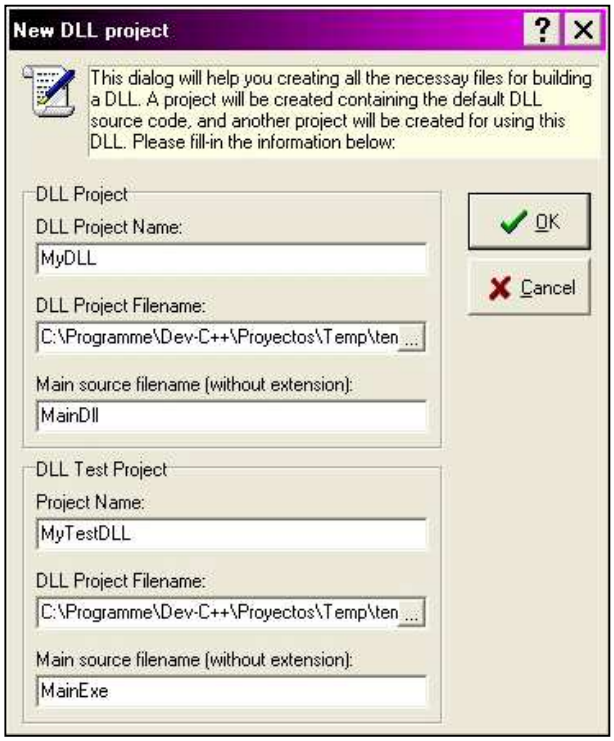
El objetivo es crear una DLL utilizando Dev-C++. Para ello necesitamos tener instalado en nuestro ordenador el programa Dev-C++, programa que te puedes bajar libremente de:
http://www.bloodshed.net/devcpp.html

Una vez que lo tenemos instalado, vamos a crear la DLL. Lo que vamos a conseguir es una DLL programada en C++, más una aplicación de „test“ que utiliza las funciones de la DLL. Para ello, vamos a realizar los siguientes pasos. Hay que tener en cuenta que puede que las ventanas tengan otro aspecto, dependiendo de la versión de Dev-C++ utilizada.

1. Opción del menú "File -> New Project..."

2. Tipo "DLL" y lenguaje "C++". Pulsar OK

3. Cuadro de diálogo "New DLL project"



3.1. Opciones DLL Project: este proyecto va a crear la DLL en sí
- DLL Project name: MyDLL

- DLL Project filename: Pulsar el boton con los 3 ptos y seleccionar un nombre (se puede definir un nombre nuevo)
- Main source filename: Dejar el que sale por defecto (MainDll)
3.2. Opciones DLL Test Project: este proyecto va a crear la aplicacion que usa la DLL
- DLL Project name: MyTestDLL
- DLL Project filename: Pulsar el boton con los 3 ptos y seleccionar un nombre (se puede definir un nombre nuevo)
- Main source filename: Dejar el que sale por defecto (MainExe)
3.3 Pulsar OK. Nos sale un mensaje que dice que va a abrir el projecto DLL y que luego debemos usar la DLL con el programa de test. Le damos a OK

4. Ya tenemos la DLL. Le damos a compilar y nos genera la DLL

5. Ahora, para ver si funciona correctamente, cerramos el proyecto de la DLL y abrimos el proyecto que la usa, el cual hemos creado en el punto 3.2.

6. Compilamos el proyecto abierto en el punto 5. y lo ejecutamos y "tachachán!" llama a las funciones de la DLL

CAPITULO II: *Pero, Qué es una DLL? Qué es lo que genero?*

Vamos a mirar "dentro" de la DLL, para ver qué es lo que tenemos y comprender cómo funciona, pero primero, una explicación previa: existen dos formas de integrar una DLL en nuestro programa:

- A través de la información que la propia DLL contiene, sin ningún archivo externo y basándonos solo en la documentación de la DLL. Este método NO es el que usaremos (de momento)
- O a través del archivo de cabecera (*.h) y la librería estática (*.lib) que la DLL genera al compilarse. Este método es el que usaremos.

Vamos a abrir el proyecto de la DLL y a hacer un par de cambios en él, para aclarar su funcionamiento. Abrimos, pues, el proyecto MyDLL y vemos que tenemos 2 archivos, `MainDLL.cpp` y `MainDLL.h`. Nos fijamos primero en el archivo de cabecera.

Abrimos el archivo `MainDll.h` y al principio, vemos lo siguiente:

```
#if BUILDING_DLL
# define DLLIMPORT __declspec (dllexport)
#else /* Not BUILDING_DLL */
# define DLLIMPORT __declspec (dllimport)
#endif /* Not BUILDING_DLL */
```

Ufff! Parece complicado, pero no hay que desanimarse, sólo son macros!
Vemos que se distingue enrtre estar **creando** la DLL (`BUILDING_DLL`) o estar **utilizando** la DLL. Hay que darse cuenta que:

- al **crear** la DLL se define la macro `DLLIMPORT` como **dllexport**
- mientras que al **utilizar** la DLL, se define `DLLIMPORT` como **dllimport**

Es decir, al crear la DLL, vamos a exportar el código/programa que hagamos a otros programas, mientras que al utilizar la DLL, lo que hacemos es importar ese programa anteriormente creado.

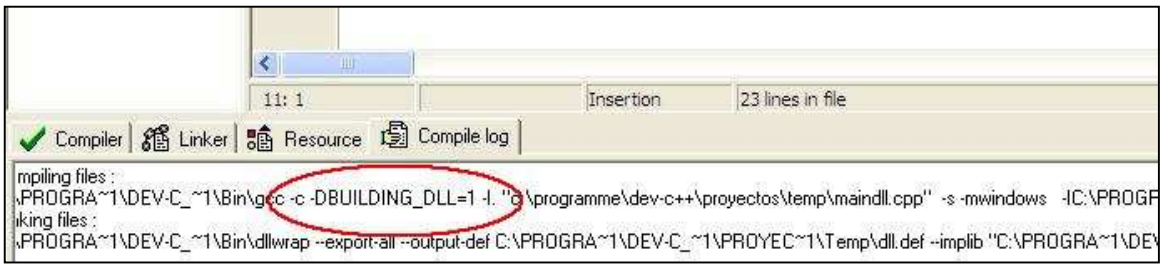
El resto del archivo de cabecera es la implementación de una clase que realiza „algo"; en este ejemplo, la DLL solo muestra un mensaje. Un punto importante es que en las declaraciones tenemos lo siguiente:

```
struct DLLIMPORT
DllClassBase {
  virtual int virtual_method () const;
};
```

Como se ve, ahí se utiliza la macro `DLLIMPORT`, de tal manera que cuando utilicemos la DLL, estaremos importando el código que se haya creado en la DLL a nuestro programa, mientras que cuando estemos programando la DLL en sí, estaremos exportando el código. Así, por medio de una simple macro, el mismo archivo de cabecera sirve para crear Y para utilizar la DLL.

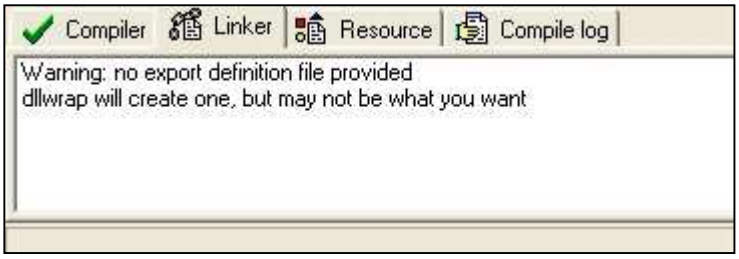
Ahora, vamos a detenernos en el archivo `MainDll.cpp`. Lo abrimos y lo primero que vemos es que en este archivo está incluído `MainDLL.h`. Esto es obvio, pues necesitamos para compilar la definición de las funciones y clases que vamos a necesitar. Ahora,

compilamos el proyecto para empezar a ver cosas. Todo tiene que ir con normalidad, sin errores, pero aún así pulsamos en la pestaña de „Compile Log“, vemos lo siguiente:



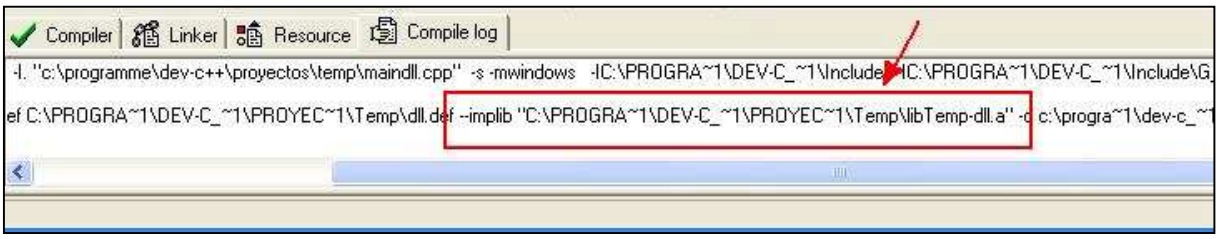
Osea, el archivo de cabecera se usará para exportar la DLL (vamos, para crearla).

Otro punto importante que vemos es que el linker nos muestra un „warning“:



Qué significa esto?
Eso significa que la DLL define nombres para las funciones de forma interna, y que esos nombres son convertidos en un formato „legible“ de forma automática; para ver lo que ha hecho, podemos abrir el archivo `dll.def` que se nos ha generado. De momento, esto no nos interesa.

Lo que sí nos interesa es el archivo LIB que antes he nombrado de pasada. Dicho archivo contiene la implementación en sí de las funciones, y lo necesitaremos en nuestro programa de test después. Si abrimos de nuevo la ventana del „Compile log“ y desplazamos un poco la barra a la derecha vemos lo siguiente:



Eso quiere decir que hemos generado un archivo LIB con el nombre `libTemp-dll.a`

Ahora abrimos el archivo `MainDll.cpp`, que contiene el resto del código de la DLL. En el archivo está incluído `MainDll.h` (ya lo esperábamos) y contiene la implementación de las clases DllClassBase y DllClass. Pero antes de ese código, tenemos lo siguiente:

```
/* Externs */
DllClass DLLIMPORT global_dllclass1 (1);
DllClass DLLIMPORT global_dllclass2 (2);
int DLLIMPORT global_int_variable = 5;
```

Estas son variables globales que la DLL crea y exporta para que otras aplicaciones las puedan usar. Recordar que `DLLIMPORT`, cuando estamos generando la DLL, está definida como **export**. Que no os confundan los nombres![1]

Posteriormente, en el código, tenemos la implementación de los datos y funciones de las clases, y al final, se llega a la parte interesante: la función `DllMain`. Tenemos la definición de la función y su implementación. Esta función es para la DLL lo que la función `main` para los programas, es decir, su punto de entrada. Cuando una aplicación carga la DLL, lo primero que hace Windows es llamar a esta función. Como veis, en realidad sólo se trata de un switch en el cual se definen ciertas funcionalidades, dependiendo del modo en que se inicialice/finalice la DLL. En la práctica, nada de esto se utiliza muy a menudo, sino otros métodos que iremos viendo.

Como resumen, en realidad al generar la DLL se utilizan otros 2 archivos que son los necesarios para compilar aplicaciones que usen la DLL:
- Al compilar, necesitamos el archivo de cabecera y el LIB
- Al ejecutar, necesitamos la DLL en sí

[1] Es igual que el nombre „mano“. A la pregunta, Cuántas manos tienes? La respuesta intuitiva es 2, pero desde el punto de vista de software, eso es problemático: tenemos una mano derecha y una mano izquierda, que casualmente se llaman igual (mano) pero que no son lo mismo

CAPITULO III: *Cómo usan otros programas la DLL*

Para este Capítulo, necesitamos el proyecto MyTestDLL, descrito en los puntos 5 y 6 del Capítulo I. Una vez abierto, vemos que dicho proyecto consta de un único archivo: `MainExe.cpp`, pero dentro de el archivo, tenemos la siguiente línea de código (el contenido exacto puede variar):

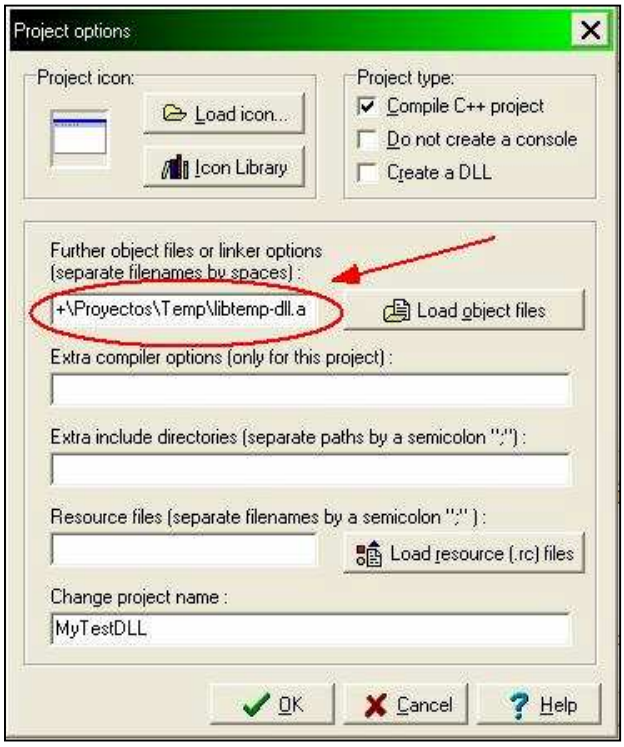
```
#include "C:\Programme\Dev-C++\Proyectos\Temp\MainDll.h"
```

Es el archivo de cabecera del proyecto MyDLL, es decir de la DLL en sí. Para verlo con más detalle, incluimos este archivo en nuestro proyecto: Elegimos la opción de menú `Project - Add to project...` y seleccionamos el archivo que ponga en el include. Abrimos dicho archivo `MainDll.h` y tenemos las conocidas líneas:

```
#if BUILDING_DLL
# define DLLIMPORT __declspec (dllexport)
#else /* Not BUILDING_DLL */
# define DLLIMPORT __declspec (dllimport)
#endif /* Not BUILDING_DLL */
```

Ahora, estamos **utilizando** la DLL, no creando, con lo que la macro `DLLIMPORT` está definida como **dllimport**, es decir, vamos a es importar este programa la DLL. Pero antes, al definir en el Capítulo II la DLL, hemos visto que además, existe un archivo LIB. Pero, dónde está?

Para ver dónde está este archivo, tenemos que abrir la opción de menú Project – Project options..., en el cual vemos:



Y se ve que automáticamente, Dev-C++ ha introducido el archivo LIB `libtemp-dll.a` de nuestra DLL en este proyecto, bajo la opción „Further object files“

Pero volvamos sobre nuestro programa MyTestDLL. Después del archivo de cabecera, tenemos lo siguiente:

```
DLLIMPORT DllClass global_dllclass1;
DLLIMPORT DllClass global_dllclass2;
DLLIMPORT int global_int_variable;
```

No he leído eso antes? Sí! Estas son las variables que se definen en MyDLL! Pero, por qué están aquí también?
Parece que las cosas vuelven a complicarse, y en realidad, es así. Ha pasado lo siguiente: en la DLL que hemos creado, hemos definido también variables globales; dichas variables son las que usaremos en nuestro programa. Y de nuevo vemos la macro `DLLIMPORT`, lo cual quiere decir que en la DLL, los objetos se exportan y en cambio, aquí, se importan.

El compilador necesita saber qué tipo de variables y funciones se utilizan, y esa información está incluída en el archivo de cabecera. Posteriormente, el linkador necesita tener la información de cómo y qué está realmente implementado; esta información se encuentra en el archivo LIB generado por la DLL.

El resto del código que se encuentra en `MainExe.cpp` son llamadas a funciones de visualización para mostrar los objetos creados en la DLL.