EXOOD4 Tutorials
Video games development

Création et utilisation d'une DLL

PROGRAMMATION WINDOWS

Table des matières

1 Introduction

La programmation Windows est essentiellement basée sur l'utilisation des DLLs (Dynamic Link Library). Nous avons tous, à un moment donné, été confrontés à certaines d'entre elles, par exemple KERNEL32.dll qui est le cœur même de Windows, sûrement l'une des plus utilisées et souvent à l'origine de plantage système ...

Une DLL est une sorte de programme indépendant de toute application, non exécutable, mais contenant de nombreuses fonctions accessibles depuis un programme exécutable (ou une autre DLL). On les associe généralement à des librairies de fonctions, un peu comme les librairies statiques (LIB), à la seule différence que celles-ci n'ont pas besoin d'être liées lors de la compilation. La DLL est installée une fois pour toutes en mémoire, lors de son premier chargement par une application.

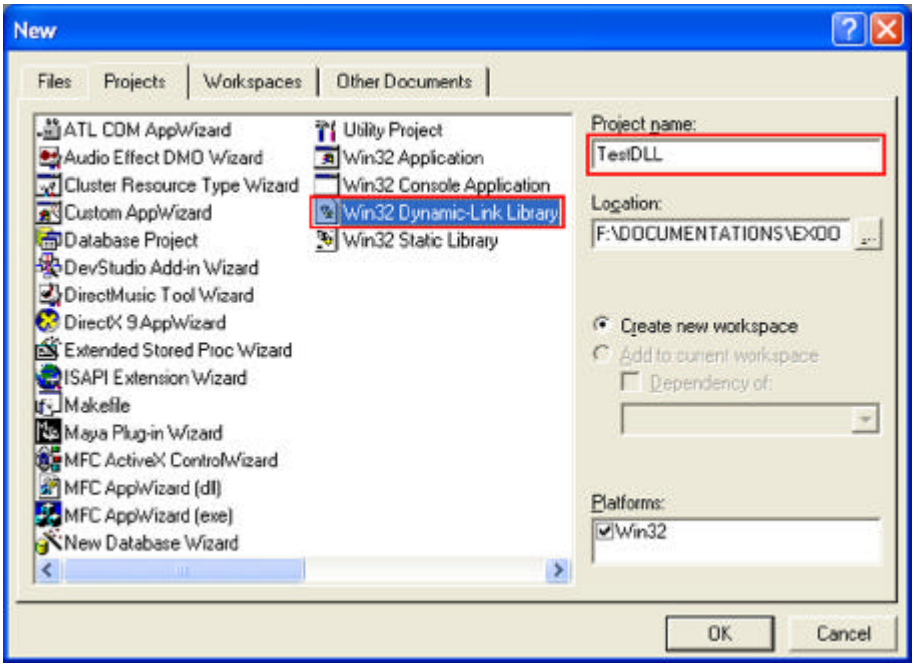
Pour une application donnée, chaque instance de la DLL partage le même espace mémoire, ce qui signifie que toute variable globale d'une DLL peut être modifiée à partir de n'importe quelle de ses instances. Cela peut engendrer de nombreux problèmes si le programmeur n'y fait pas attention, c'est pourquoi on déconseille fortement d'utiliser de telles variables (globales). Ces problèmes surviennent en général dans une application multi-thread.

2 Programmation d'une DLL

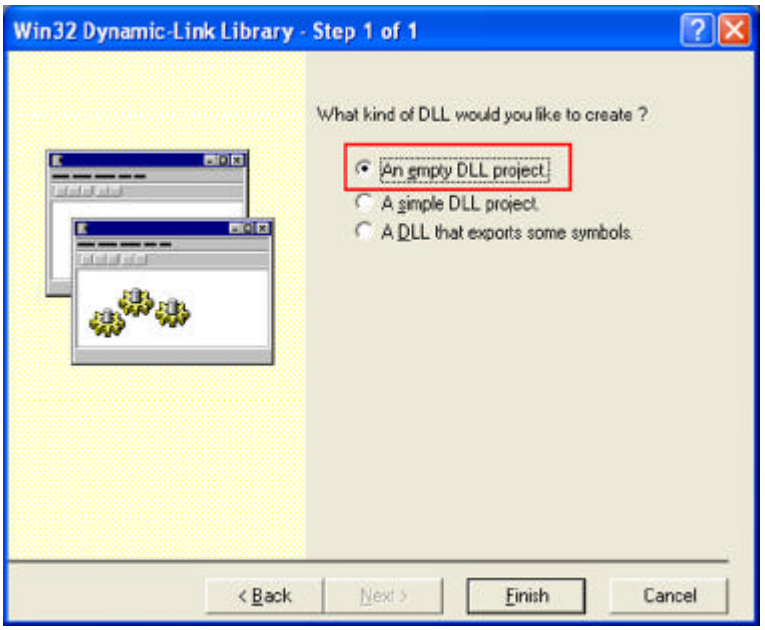
2.1 Première méthode

La première façon d'écrire une DLL consiste à passer par un fichier de définition des fonctions de la DLL. On utilise pour cela un fichier portant l'extension .DEF. Pour commencer nous allons créer une DLL très simple contenant 3 fonctions.

- Créez un nouveau projet "Win32 Dynamic-Link Library", par exemple : TestDLL



- Choisissez l'option "An Empty DLL project"



- Créez maintenant un fichier texte : TestDLL.def et écrivez les lignes suivantes :

```
; TestDLL.def : Declares the module parameters for the application

LIBRARY        TestDLL
DESCRIPTION    'Example of DLL'

EXPORTS
        ; Explicitly exported initialization routine
        Initialize          @1
        Release             @2
        TestFunction        @3
```

Nous allons ainsi déclarer nos 3 fonctions : Initialize, Release et TestFunction

Remarque : un fichier DEF peut contenir d'autres informations, vous pouvez vous référer à la documentation MSDN pour obtenir une description plus complète des autres paramètres.

- Ensuite créez le fichier TestDLL.cpp dans lequel vous écrirez les fonctions de la DLL :

```
#include <windows.h>

BOOL APIENTRY DllMain(HANDLE hModule, DWORD ul_reason_for_call,
                      LPVOID lpReserved)
{
        switch (ul_reason_for_call)
        {
                case DLL_PROCESS_ATTACH:
                case DLL_THREAD_ATTACH:
                case DLL_THREAD_DETACH:
                case DLL_PROCESS_DETACH:
                        break;
        }
        return true;
}

extern "C" void WINAPI Initialize()
{
        // Write here all the code you need to initialize the DLL
}

extern "C" void WINAPI Release()
{
        // Write here all the code you need to free everything ...
}

extern "C" int WINAPI TestFunction(int _Value)
{
        return (_Value * _Value);
}
```

- Pour finir nous allons créer un header contenant la description de notre API (plutôt succincte …) qui sera utilisée par tous les programmes clients :

```
#ifndef __API_TESTDLL_H
#define __API_TESTDLL_H

#if defined (__cplusplus)
extern "C" {
#endif

// Main API functions declaration
typedef void  (WINAPI *DLL_Function_Initialize)        ();
typedef void  (WINAPI *DLL_Function_Release)           ();
typedef int   (WINAPI *DLL_Function_TestFunction)      (int _Value);

#if defined (__cplusplus)
}
#endif
#endif __API_TESTDLL_H
```

N'oubliez pas d'inclure le fichier de définition .DEF à votre projet Visual C++ pour que les noms de fonctions soient exportés lors de la compilation.

Si tout s'est bien passé jusqu'à maintenant vous devriez obtenir après la compilation, dans le répertoire (par défaut) Debug, la DLL TestDLL.dll.

Remarque :
Si vous tentez d'obtenir les propriétés du fichier TestDLL.dll (avec l'explorateur Windows) vous remarquerez qu'il n'y a aucune information associée à la DLL. Pour personnaliser le détail de cette DLL il nous faut y ajouter une ressource particulière.

Voici la procédure à suivre pour ajouter les informations de version :
- Dans le menu **Insert** de Visual C++, cliquez sur **Resource**
- Choisissez ensuite le type **Version** et cliquez sur **New**
- Remplissez les champs que vous désirez modifier
- Sauvegardez (CTRL+S) la nouvelle ressource (par exemple sous le nom TestDLL.rc)
- Ajoutez ce fichier au projet de la DLL, et recompilez

2.2 Deuxième méthode

Il existe une autre façon d'écrire une DLL. Elle consiste à utiliser l'option *"A DLL that exports some symbols"* lors de la création d'un nouveau projet Win32 Dynamic-Link Library. Visual C++ va alors générer les fichiers sources contenant des indications sur la procédure à suivre pour créer une telle DLL. Vous aurez ainsi la possibilité d'exporter des fonctions, des classes, des variables (globales) …

L'un des points importants à noter dans cette méthode concerne l'utilisation des mots-clés __declspec(dllexport) et __declspec(dllimport). Visual C++ génère automatiquement les defines correspondants, l'un étant utilisé pour la DLL elle-même (dllexport) et l'autre pour les programmes clients (dllimport).

Pour pouvoir exporter des variables ou des classes il faut linker le programme client avec le fichier LIB généré lors de la compilation de la DLL. Ce qui n'est pas forcément très pratique et revient en quelque sorte à utiliser une librairie statique. Nous verrons dans un autre article comment contourner ces problèmes.

3 Utilisation d'une DLL

Maintenant que nous avons une DLL prête à l'emploi, nous devons créer le programme qui va l'utiliser. Pour cela nous allons voir les différentes étapes pour charger en mémoire une DLL, faire appels à ses fonctions et, pour finir, la décharger.

3.1 Chargement d'une DLL

Le chargement d'une DLL s'effectue très simplement en utilisant la fonction **LoadLibrary**.

```
HMODULE hDLL = LoadLibrary("TestDLL.dll");
```

Le seul paramètre à fournir est le chemin de la DLL, la fonction retourne un handle (unique) permettant de faire appel plus tard aux autres fonctions de gestion des DLL. En cas d'échec, le handle retourné sera nul.

Lorsque le programme client charge la DLL en mémoire, la fonction **DllMain** est appelée et le paramètre **ul_reason_for_call** a pour valeur **DLL_PROCESS_ATTACH**.

*Remarque : dans le cas d'une application multi-threads, chaque thread exécuté provoquera un appel à la fonction **DllMain** avec la valeur **DLL_THREAD_ATTACH**. Cette fonction peut dans certains cas être utilisée pour initialiser ou libérer des informations liées au process (ou thread) appelant. Mais en général cette fonction est inutilisée.*

3.2 Déclaration des fonctions

Une fois la DLL en mémoire, il nous faut accéder à ses fonctions. Nous utiliserons des pointeurs de fonction qui nous faciliteront les appels :

```
typedef int (WINAPI *DLL_Function_TestFunction) (int _Value);

DLL_Function_TestFunction pfn_TestFunction;
```

Vient ensuite la déclaration des fonctions à partir de celles encapsulées dans la DLL. Pour cela il nous faut utiliser leur nom exact donné à la création (et dans le fichier de définitions .DEF), on utilise alors la fonction **GetProcAddress** ayant pour paramètres le handle obtenu avec **LoadLibrary** et le nom exact de la fonction à référencer :

```
pfn_TestFunction = (DLL_Function_TestFunction)GetProcAddress(hDLL,"TestFunction");
```

Attention :
- "TestFunction" doit être le véritable nom de la fonction
- Ne pas oublier de "caster" le résultat de l'appel à **GetProcAddress**

On utilise ensuite la fonction normalement, comme dans tout programme C/C++ :

```
int Result = pfn_TestFunction(12345);
```

3.3 Libération d'une DLL

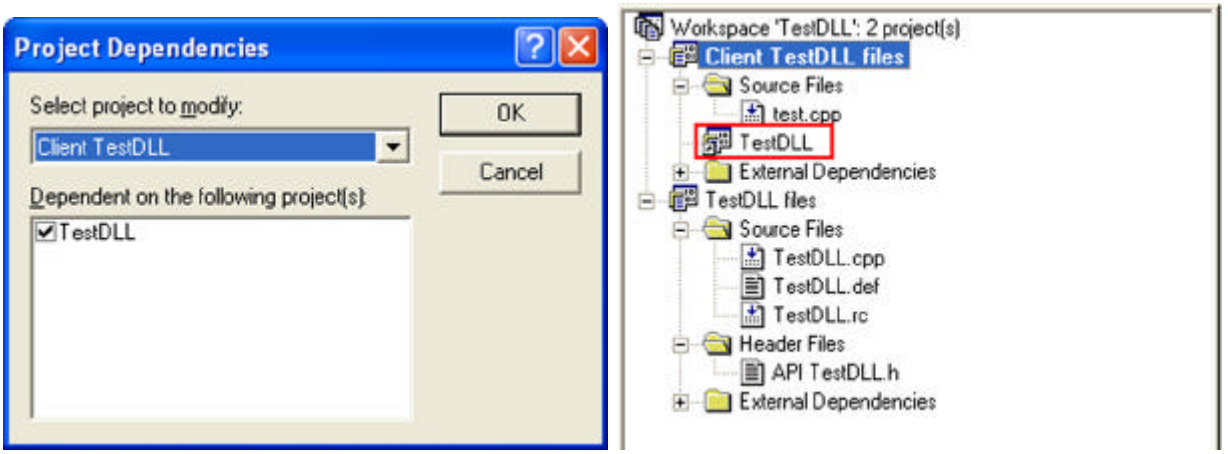
Une fois le programme terminé, il faut décharger la DLL en utilisant la fonction **FreeLibrary** avec, en paramètre, le handle obtenu avec **LoadLibrary** :

```
FreeLibrary(hDLL);
```

La fonction **DllMain** est alors appelée avec la valeur **DLL_PROCESS_DETACH**.

*Remarque : si l'application exécute des threads, chaque terminaison de thread provoquera un appel de la fonction **DllMain** avec la valeur **DLL_THREAD_DETACH**.*

Pour finir, nous allons créer un lien de dépendance entre le projet du programme client et celui de la DLL. Ce qui aura pour effet de recompiler le programme client automatiquement si la DLL a été modifiée. Il suffit de cliquer sur l'option **Dependencies** du menu **Project** :



Une nouvelle icône apparaît dans le projet Client TestDLL

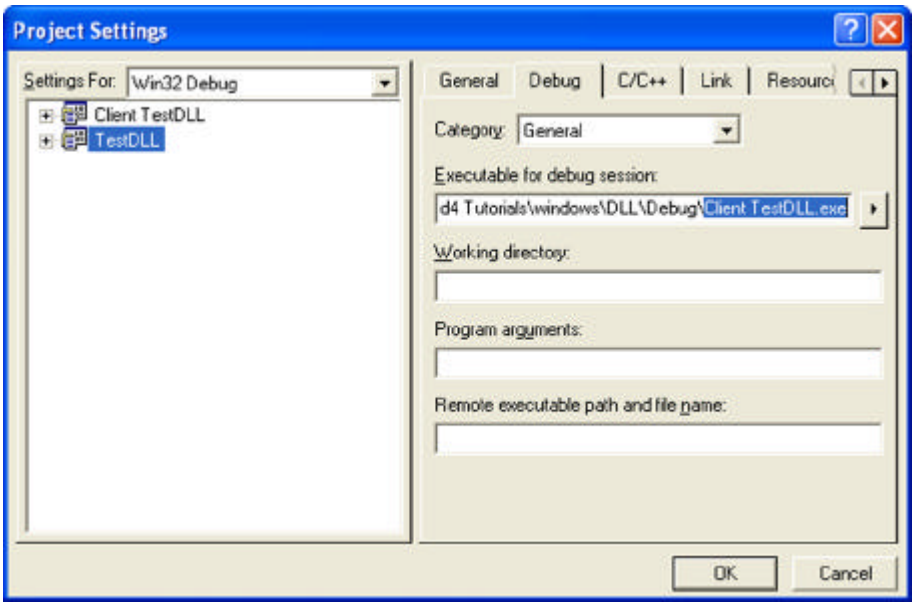
Nous venons d'avoir un premier aperçu sur la création d'une DLL et son utilisation dans un programme client. Nous verrons bientôt plus en détail l'écriture de DLLs, notamment pour développer un moteur de jeu.

4 Débuguer une DLL

Comme nous avons pu le voir précédemment, une DLL ne peut être exécutée mais seulement chargée par un programme client ou éventuellement à l'intérieur même d'une autre DLL. Il serait donc difficile d'utiliser le debugger de Visual C++ puisqu'il fonctionne, en principe, que pour des programmes exécutables. Heureusement il existe plusieurs façons de suivre pas à pas l'exécution d'une DLL.

4.1 Débuguer depuis la DLL

Cela est possible en modifiant les paramètres du projet (menu **Project>Settings**) :



Le champ "**Executable for debug session**" permet de déclarer l'application qui sera exécutée par le debugger et susceptible de charger la DLL que l'on souhaite tracer. Bien entendu si l'exécutable n'est pas compilé en version Debug et que les sources ne sont pas disponibles vous n'aurez pas la possibilité de suivre pas à pas l'exécution du programme. Mais lorsque la DLL sera appelée vous prendrez la main dans son code source (à condition bien sûr d'avoir placé des breakpoints).

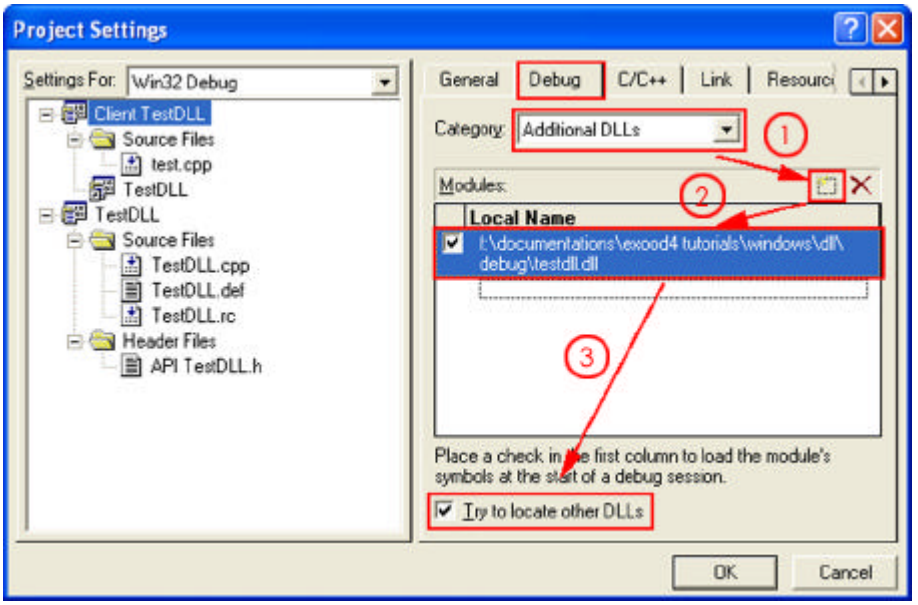
Vous pouvez également renseigner les autres champs de la fenêtre (Working directory, Program arguments etc...) si l'application en a besoin.

Ensuite, il suffit de placer des breakpoints dans le code de la DLL et lancer l'exécution en mode pas à pas (F5). Normalement, dans notre exemple vous devriez prendre la main très rapidement.

Une variante, consiste à se servir du menu **Build>Start Debug>Attach to Process**. Cette technique est souvent utilisée pour développer des DLLs appelées par des programmes résidents tels que les services Windows ou tout autre application ne pouvant être arrêtée ou lancée simplement (DLLs liées à la programmation système : programmation Shell pour enrichir les fonctionnalités de l'explorateur Windows, développement IIS etc...).

4.2 Débuguer depuis le programme client

Une autre méthode consiste à débuguer depuis le programme client. Seulement, les informations de debug de la DLL ne sont pas toujours accessibles au démarrage du programme. Ce qui vous empêche de placer vos breakpoints dans le code source de la DLL tant que celle-ci n'a pas été chargée en mémoire (après l'appel à **LoadLibrary**). Heureusement Visual C++ nous permet de contourner ce problème.
Toujours dans le menu **Project>Settings** vous avez la possibilité de localiser les DLLs pour lesquelles vous souhaitez obtenir les informations de Debug au démarrage du mode pas à pas de l'application.



Vous pouvez alors placer vos breakpoints à tout moment et prendre la main dans la fonction **DllMain** par exemple, ce qui serait impossible autrement puisque cette fonction est appelée la première fois avec **LoadLibrary**.

Retrouvez les sources de cet article à l'adresse suivante :

http://www.exood4.com/tutorials/articles/windows/TestDLL.zip