Programmation de librairies dll win32pour Access avec Visual C++ Express

par Thierry GASPERMENT

Date de publication : 24/11/06

Dernière mise à jour : 24/11/06

Programmation de librairies dll win32 pour Access avec Visual C++ Express.

I - Introduction

I-A - Liens

Les instructions d'installation et de paramètrage sont issues de cette page :

Using Visual C++ 2005 Express Edition with the Microsoft Platform SDK

Pour une première prise en main de Visual C++ Express et plus de détails sur l'installation, je vous invite à lire le tuto de arb

Démarrer avec Visual C++ 2005 Express

La lecture du tutoriel suivant s'avère très utile pour résoudre les problèmes de type de données :

Créer des dll en C compatibles avec VB 6

I-B - Pourquoi Visual C++ Express?

Il est d'abord souvent nécessaire de déporter les sous-classements (=Hook) dans une librairie pour assurer la stabilité de l'application.

Ensuite l'environnement de développement Visual Express est gratuit (pour une utilisation personnelle ou professionnelle).

Et enfin C++ parce que ce langage permet de créer des librairies simples (VB ne permet de créer que des librairies ActiveX qui nécéssite un référencement).

I-C - Remerciements

Merci à la rédaction Access de DVP pour le temps passé en relectures et tests.

Merci à tous ceux qui partagent leurs connaissances sur internet et sans qui je n'aurais pas pu réaliser cet article.

II - Installation

Pour programmer une librairie win32 en Visual C++, il faut au préalable installer :

- L'environnement de développement Visual C++ Express

- Le Platform SDK pour disposer des fichiers d'en-tête win32

- Le Framework est installé obligatoirement avec Visual C++, même si nous ne l'utiliserons pas

Il n'est pas nécessaire de graver les images ISO.

Vous pouvez utilisez Daemon tools pour créer un disque virtuel à partir de l'image ISO.

II-A - Installation de Visual C++ Express

Vous pouvez télécharger une image ISO **en anglais** :

Manual Installation Instructions for Express Editions

La **version française** est disponible en installation en ligne ici :

Visual C++ 2005, Express Edition

II-B - Installation du Platform SDK

Pour une installation en ligne :

Windows Server 2003 R2 SDK - March 2006 Edition Web Setup

Pour télécharger une image ISO :

Windows Server 2003 R2 SDK - March 2006 Edition ISO Download

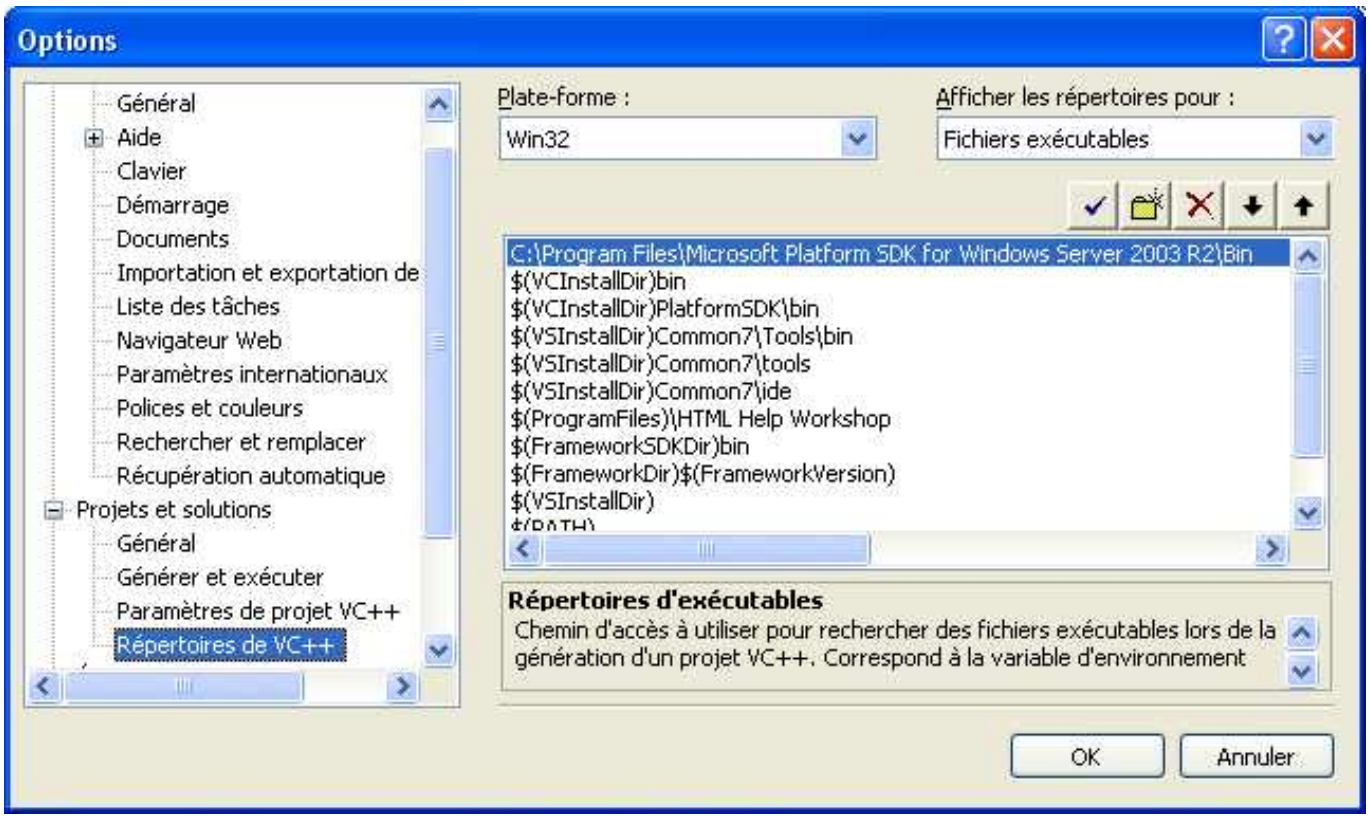
II-C - Paramétrage de l'environnement de développement (IDE)

1 - Ajoutez les répertoires du Platform SDK :

- Ouvrez Visual C++ Express

- Dans le menu : **Outils** --> **Options...**, choisissez **Projets et solutions** et sélectionnez **Répertoires de VC++**

a - Ajoutez : **C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2\Bin** dans **Fichiers exécutables** comme ceci :



b - Puis ajoutez **C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2\Include** dans **Fichiers Include**

c - Et **C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2\lib** dans **Fichiers bibliothèques**

(Ces répertoires sont les répertoires par défaut si vous n'avez rien changé lors de l'installation).

2 - Modifier les dépendances :

Dans le fichier **C:\Program Files\Microsoft Visual Studio 8\VC\VCProjectDefaults\corewin_express.vsprops**, remplacez

```
AdditionalDependencies="kernel32.lib"
```

par

```
AdditionalDependencies="kernel32.lib user32.lib gdi32.lib winspool.lib
                  comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib"
```

Mettez tout le texte sur la même ligne.

(vous pouvez éditer le fichier avec Notepad ou WordPad par exemple)

3 - Activer l'assistant pour créer une application ou une librairie Win32

Dans le fichier **C:\ProgramFiles\Microsoft Visual Studio 8\VC\VCWizards\AppWiz\Generic\Application\html\1036\AppSettings.htm**, remplacez à l'aide d'un éditeur de texte (Notepad ou WordPad par exemple) :

```
WIN_APP.disabled = true;
WIN_APP_LABEL.disabled = true;
DLL_APP.disabled = true;
DLL_APP_LABEL.disabled = true;
```

par:

```
// WIN_APP.disabled = true;
// WIN_APP_LABEL.disabled = true;
// DLL_APP.disabled = true;
// DLL_APP_LABEL.disabled = true;
```

Ceci se trouve en ligne 441 à 444.

Vérifiez que vous modifiez bien au bon endroit, là où se trouvent ces 4 lignes en un seul bloc.

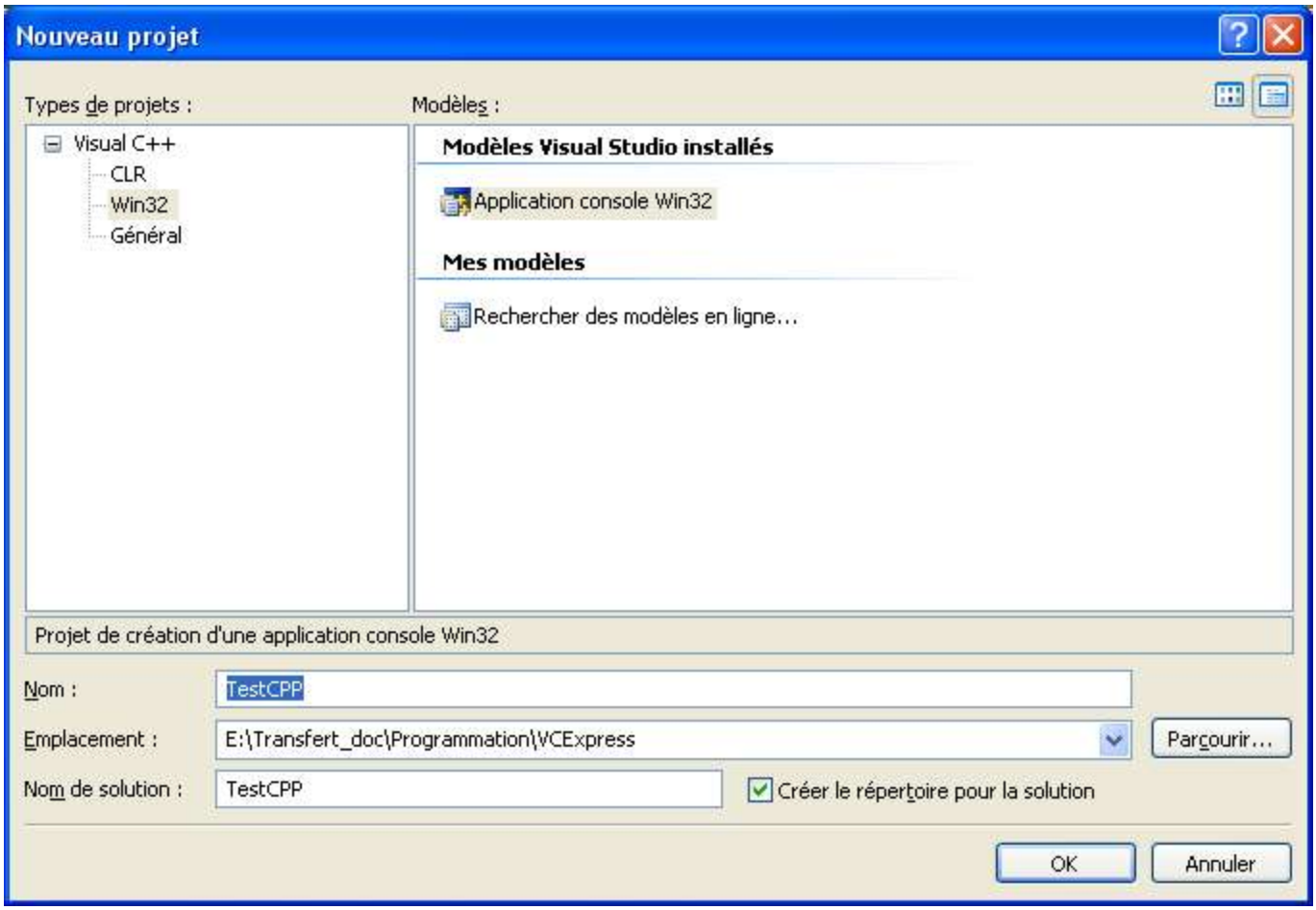
Notez que le **1036** en fin de chemin du fichier **peut différer en fonction de la langue** d'installation.

III - Créer un nouveau projet

Tous les fichiers sources sont regroupés dans un projet.

Pour créer un nouveau projet, choisissez dans le menu : **Fichier** --> **Nouveau** --> **Projet**

Dans le fenêtre qui apparaît, donnez un nom au projet.

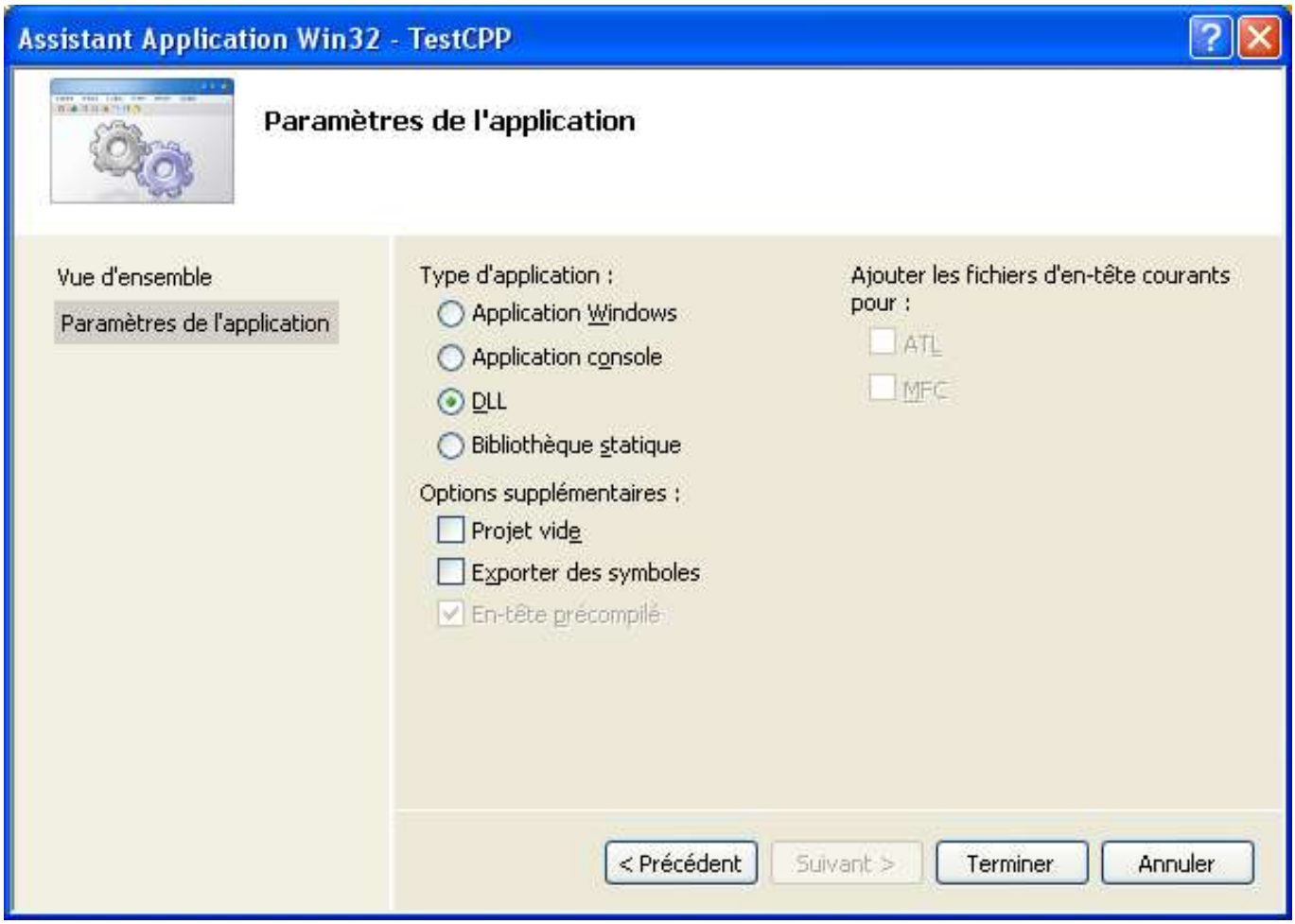


Le nom de la librairie sera celui du projet.

Par exemple j'ai choisi **TestCPP** comme nom de projet, la librairie sera générée sous le nom **TestCPP.dll**.

L'assistant se déclenche, cliquez sur **Suivant**.

Sélectionnez ensuite **DLL** pour générer un projet de librairie.



Puis cliquez sur **Terminer**.

L'assistant crée les fichiers nécessaires à la création d'une librairie :

- **stdafx.h** : fichier d'entête contenant les déclarations.

- **stdafx.cpp** : fichier source qui inclue le précédent fichier d'en-tête.

- **TestCPP.cpp** : c'est le fichier source principal dans lequel on va ajouter notre code.

IV - Configurer le projet

Il y a deux configurations par défaut pour un nouveau projet :

- **Debug**

- **Release**

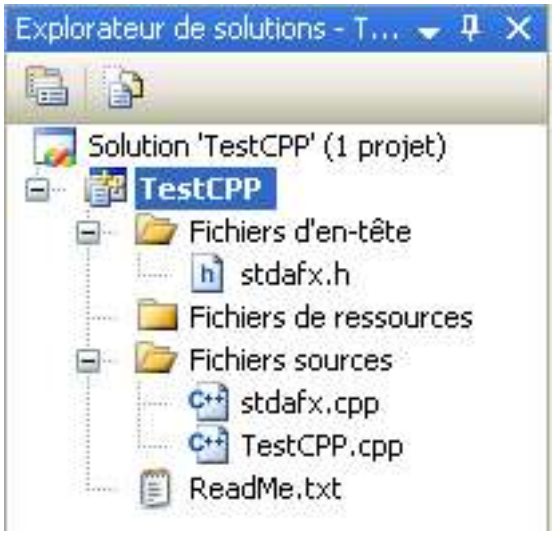
La configuration **Debug** possède par défaut des paramètres utiles au débogage.

La librairie étant utilisée à partir de Access, on ne va pas la déboguer.

On choisi donc la configuration Release dans le menu déroulant (en haut dans la barre de menu).

C'est sur cette configuration que l'on va définir les paramètres de notre projet.

Sélectionnez le projet **TestCPP** dans le panneau de gauche :



Choisissez dans le menu : **Projet** --> **Propriétés**

Dans les **Propriétés de configuration** : **Général**, vérifiez les paramètres suivants :

- Type de configuration : Bibilothèque dynamique (dll)

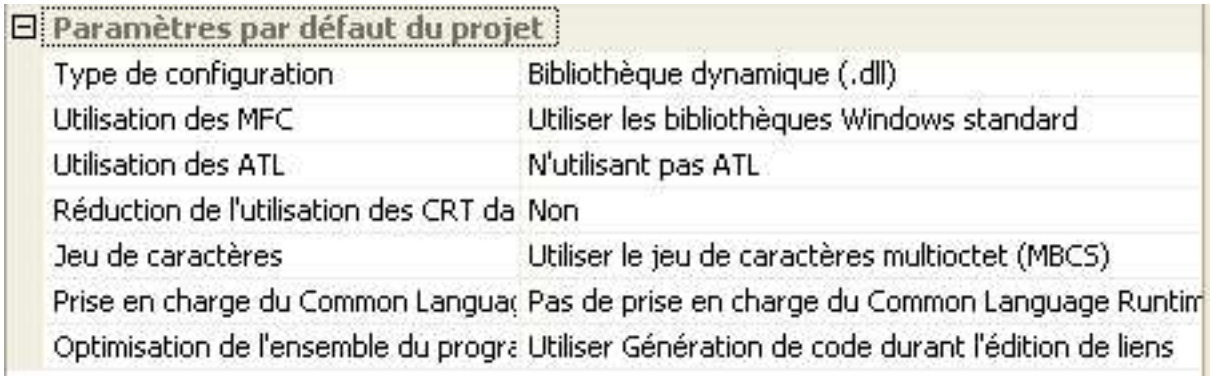
- Utilisation des MFC : Utiliser les bibliothèques Windows standard

- Utilisation des ATL : N'utilisant pas ATL

- Reduction de l'utilisation des CRT dans les ATL : Non

- Jeu de caractères : Utiliser le jeu de caractères multioctet (MBCS)

(ce dernier paramètre simplifie l'utilisation des chaînes de caractères)



Dans les **Propriétés de configuration** : **C/C++** : **Général** : - Détection des problèmes de portabilité 64bits : Non

(Pour une librairie Win32, inutile d'afficher les avertissements liés au 64Bits)

Dans les **Propriétés de configuration** : **C/C++** : **Génération de code** : - Bibliothèque runtime : Multithread (/MT)

(important pour ne pas avoir à installer de runtime sur les PC)

V - Créer une première librairie contenant une procédure simple

On va, pour débuter en douceur, simplement afficher un message à l'aide de l'équivalent de **MsgBox** en C++.

V-A - Fichier d'en-tête Windows

On ajoute tout de suite la déclaration du fichier d'en-tête Windows dans le fichier **stdafx.h**.

(Ajoutez ce code en fin du fichier.)

```
	// Fichiers d'en-tête Windows :
	#include <windows.h>
```

Ce fichier contient les déclarations des fonctions et constantes communément utilisées, notamment pour notre exemple **MessageBox** et **MB_OK**.

V-B - Ajouter une procédure à la librairie

Pour exécuter un traitement sans recevoir de résultat, une procédure est suffisante.

A la fin du fichier **TestCPP.cpp** ajouter la procédure suivante :

```
void __stdcall TestMessage()
{
	MessageBox(NULL,"test message","test titre",MB_OK);
}
```

- **void** signifie que la procédure ne renvoie pas de résultat.

- **_stdcall** est nécessaire pour pouvoir exécuter la fonction depuis **VBA**.

- **TestMessage** est le nom de la procédure.

- **MessageBox** est la fonction C++ qui permet d'afficher un message.

Retrouvez l'aide de la fonction dans la documentation du PlatForm SDK

Pour que la procédure soit exportée, et donc exécutable à partir de VBA, il faut écrire un **fichier de définition de module**.

- Dans le menu : **Fichier** --> **Nouveau** --> **Fichier...**

- Choisissez dans **Général** : **Fichier texte**

- Sauvegardez ce fichier sous le nom **TestCPP.def**.

Ce fichier contient les noms des fonctions à exporter :

```
LIBRARY TestCPP
EXPORTS
TestMessage @1
```

Les fonctions/procédure sont numérotées dans l'ordre d'apparition dans le code.

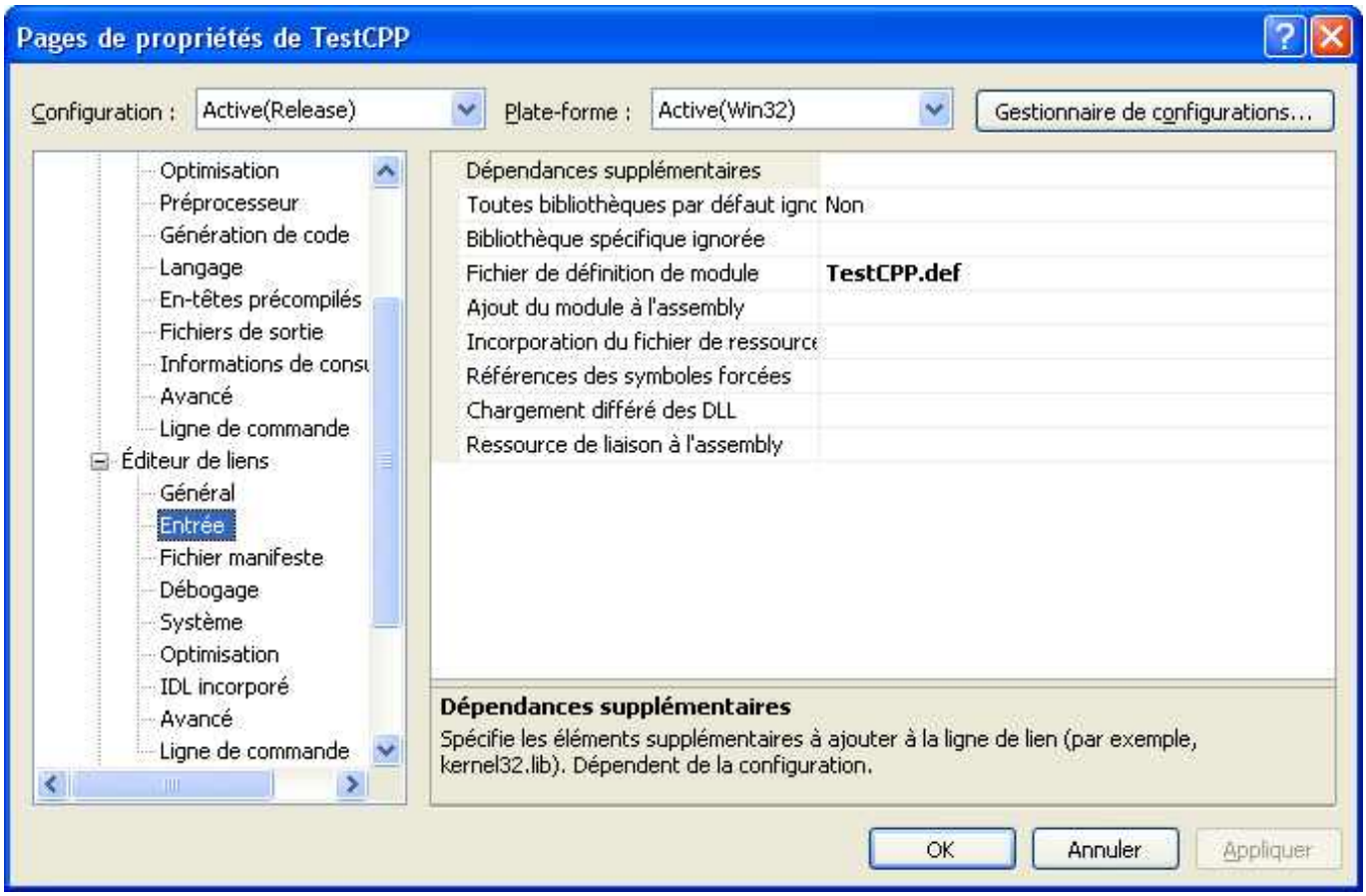
Cela signifie qu'on exporte la première procédure (**@1**) sous le nom **TestMessage**.

Il faut ensuite demander au compilateur d'utiliser ce fichier de définition.

Dans les propriétés du projet :

Propriétés de configuration : **Editeur de liens** : **Entrée**.

Fichier de définition de module : TestCPP.def



V-C - Générer la librairie

Dans le menu, choisissez : **Générer** --> **Générer TestCPP**.

Voici le protocole de génération :

```
------ Début de la génération : Projet : TestCPP, Configuration : Release Win32 ------
Compilation en cours...
stdafx.cpp
Compilation en cours...
TestCPP.cpp
Édition des liens en cours...
   Création de la bibliothèque E:\Transfert_doc\Programmation\VCExpress\TestCPP\Release\TestCPP.lib
             et de l'objet E:\Transfert_doc\Programmation\VCExpress\TestCPP\Release\TestCPP.exp
Génération de code en cours
Fin de la génération du code
Incorporation du manifeste en cours...
Le journal de génération a été enregistré à l'emplacement
            "file://e:\Transfert_doc\Programmation\VCExpress\TestCPP\TestCPP\Release\BuildLog.htm"
TestCPP - 0 erreur(s), 0 avertissement(s)
========== Génération : 1 a réussi, 0 a échoué, 0 mis à jour, 0 a été ignoré ==========
```

La librairie a été générée dans le répertoire :

E:\Transfert_doc\Programmation\VCExpress\TestCPP\Release

Le nom du fichier est :

TestCPP.dll

On peut maintenant utiliser cette librairie à partir d'**Access**.

V-D - Exécuter la procédure à partir d'Access

La procédure **TestMessage** étant dans un fichier séparé, il faut tout d'abord la déclarer.

Dans un module VBA, ajouter la déclaration suivante :

```
Public Declare Sub TestMessage Lib
"E:\Transfert_doc\Programmation\VCExpress\TestCPP\release\TestCPP.dll" ()
```

N'oubliez pas de modifier le répertoire pour indiquez celui où se trouve votre librairie.

Ensuite vous pouvez exécuter la procédure, par exemple sur click sur un bouton de formulaire :

```
Private Sub CmbTestDll_Click()
TestMessage
End Sub
```

Il est heureusement possible de **définir dynamiquement le chemin** de la librairie.

Pour cela, **ne précisez pas le chemin dans la déclaration de la procédure**.

La librairie sera chargée grâce à la fonction **LoadLibrary** (c'est une API à déclarer).

Nouvelles déclarations

```
Public Declare Sub TestMessage Lib "TestCPP.dll" ()
Public Declare Function LoadLibrary Lib "Kernel32" Alias "LoadLibraryA" (ByVal lpLibFileName As
String) As Long
```

Appel de la fonction avec chargement de la dll dans le répertoire de l'application Access

```
Private Sub CmbTestDll_Click()
LoadLibrary Left(CurrentDb.Name, Len(CurrentDb.Name) - Len(Dir(CurrentDb.Name))) & "TestCPP.dll"
TestMessage
End Sub
```

V-E - Passer des paramètres à la procédure

V-D-1 - Variables

Pour personnaliser la boîte de message on va passer en paramètre :

- le texte du message;

- le titre de la boîte de dialogue.

On s'aperçoit également que la boîte de dialogue ne reste pas en avant plan:

il faut utiliser le premier paramètre de la fonction **MessageBox** pour préciser l'identifiant de la fenêtre parent

On ajoute donc trois paramètres :

- pHwnd : un entier long qui contiendra l'identifiant de l'application Access ou du formulaire;

- pTexte : une chaîne de caractères qui contient le texte du message;

- pTitre : une chaîne de caractères qui contient le titre de la boîte de dialogue.

Pour un **entier long**, on utilise le type C++ **DWORD**.

Pour une **chaîne de caractères**, on utilise le type C++ **BSTR**.

*Le type **BSTR** est défini dans le fichier d'entête **"olectl.h"**.*

On ajoute d'abord ce fichier d'entête dans le fichier **stdafx.h**, à la fin du fichier dans la partie **"// Fichiers d'en-tête Windows :"**:

```
// Fichiers d'en-tête Windows :
#include <windows.h>
#include "olectl.h"
```

Ensuite on modifie la déclaration de la procédure :

```
void __stdcall TestMessage(DWORD pHwnd,BSTR pTexte,BSTR pTitre)
```

Puis on utilise ces paramètres dans la fonction **MessageBox**:

```
void __stdcall TestMessage(DWORD pHwnd,BSTR pTexte,BSTR pTitre)
{
	MessageBox(HWND(pHwnd),LPCSTR(pTexte),LPCSTR(pTitre),MB_OK);
}
```

Le premier paramètre de la fonction **MessageBox** est de type **HWND**.

On converti donc notre paramètre de type **DWORD** en ajoutant **HWND** lors de son utilisation.

Les paramètres suivant sont de type **LPCSTR**, on les converti de la même manière

N'oubliez pas de régénérer la librairie après ces modifications.

Il peut être nécessaire de fermer l'application Access pour pouvoir générer la librairie.

Les paramètres ajoutés doivent être préciser dans la déclaration VBA de la fonction dans Access.

Notez l'importance du mot-clé **ByVal** qui précise que les paramètres sont passés par valeur.

```
    Public Declare Sub TestMessage Lib
"E:\Transfert_doc\Programmation\VCExpress\TestCPP\release\TestCPP.dll" _
            (ByVal pHwnd As Long, ByVal pTexte As String, ByVal pTitre As String)
```

L'appel de la fonction dans le formulaire devient :

```
    Private Sub CmbTestDll_Click()
    TestMessage Me.Hwnd, "Mon message", "Mon Titre"
    End Sub
```

Me.Hwnd est l'identifiant du formulaire. Cela permet de lier la boîte de message au formulaire.

Si la fonction est utilisée dans un module standard, on peut utiliser **Application.hWndAccessApp** pour lier la boîte de dialogue à l'application Access.

On obtient une boîte de dialogue toute simple:

On a tout de même réussi à passer des paramètres à notre librairie!

V-D-2 - Tableaux

Le sujet est largement traité dans ce tutoriel : Créer des dll en C compatibles avec VB 6

Nous n'allons pas le developper ici.

V-F - Transformer la procédure en fonction

La procédure ne renvoie pas de résultat.

On ne connaît donc pas le choix de l'utilisateur.

La prochaine étape consiste à transformer la procédure en fonction pour appeler une boîte de message de type OUI/NON et récupérer dans Access la choix de l'utilisateur.

Pour transformer la procédure en fonction, il suffit de remplacer **Void** par le type de donnée à retourner.

D'après la documentation, la fonction **MessageBox** retourne un résultat de type entier **int**.

```
int __stdcall TestMessage(DWORD pHwnd,BSTR pTexte,BSTR pTitre)
{
    int lResultat;
        lResultat = MessageBox(HWND(pHwnd),LPCSTR(pTexte),LPCSTR(pTitre),MB_YESNO);
        Return lResultat;
}
```

On utilise une variable locale **lResultat** de type **int** pour récupérer la résultat de la boîte de dialogue.

L'instruction **Return** permet de quitter la fonction en renvoyant le contenu de la variable spécifiée.

Dans Access, la définition de la procédure doit être modifiée :

- ce n'est plus une procédure donc on remplace **Sub** par **Function**;

- et on ajoute le type de donnée à retourner par la fonction : **int** en C++ donc **long** en VBA.

```
Public Declare Function TestMessage Lib
"E:\Transfert_doc\Programmation\VCExpress\TestCPP\release\TestCPP.dll" _
         (ByVal pHwnd As Long, ByVal pTexte As String, ByVal pTitre As String) As Long
```

Le résultat de **MessageBox** dans C++ est un entier dont la valeur peut-être **IDYES** ou **IDNO**.

Ces constantes ne sont pas définies dans VBA, il faut donc les déclarer.

Pour trouver les valeurs correspondantes à ces constantes C++, tapez **IDYES** dans l'éditeur C++ puis dans le menu contextuel (click sur bouton droit sur le mot IDYES) sélectionnez **Atteindre la déclaration**.

On trouve les valeurs des constantes :

```
#define IDYES                6
#define IDNO                 7
```

Il suffit alors de déclarer l'équivalent dans VBA :

```
Public Const IDYES = 6
Public Const IDNO = 7
```

On peut ensuite comparer le résultat de la fonction **TestMessage** avec ces constantes.

```
Private Sub CmbTestDll_Click()
Dim lResultat As Long
lResultat = TestMessage(Me.Hwnd, "Mon message", "Mon Titre")
MsgBox "Vous avez cliquez sur " & IIf(lResultat = IDYES, "Oui", "Non")
End Sub
```

Remarquez qu'on peut également directement utiliser les constantes VBA **vbYes** et **vbNo**.

V-G - Conclusion

Cette première procédure nous a permis de voir les bases de la programmation de librairie pour Access en C++.

Elle n'a pas d'autre intérêt, la fonction **MsgBox** de VBA suffirait à faire la même chose.

VI - Sous-classement / Hook

Nous allons dans les exemples qui suivent utiliser des sous-classements, également appelés Hook en anglais.

Le principe du sous-classement est simple : détourner les messages vers une fonction spécifique.

On peut ensuite :

- modifier le message avant de l'envoyer au gestionnaire de messsages standard;

- effectuer des actions supplémentaires avant ou après le traitement du message;

- ne pas envoyer le message vers le gestionnaire de messages standard, pour annuler un événement.

> *L'article Contrôle du clavier et de la souris sous Windows de gRRosminet est une bonne introduction au sous-classement.*

Notez dans cet article le chapitre Les différents types de hooks

VII - Exemples

VI-A - Bloquer la roulette de la souris dans un formulaire

VI-A-1 - Introduction

Vous trouverez la librairie compilée et le tutoriel d'utilisation ici :

Gestion de la roulette de la souris dans les formulaires (dll sans référencement)

Dans ce chapitre nous allons programmez cette librairie étape par étape :

1 - Sous-classement d'un formulaire;

2 - Annuler le message envoyé par la roulette de la souris;

3 - Utilisation de la librairie pour plusieurs formulaires simultanément;

3 - Aller plus loin en utilisant la roulette pour faire défiler le formulaire de haut en bas.

VI-A-2 - Création du projet

Créez un nouveau projet en suivant les sections Créer un nouveau projet et Configurer le projet.

Pour la suite, cette librairie est nommée **TutoCPPRoulette**.

On ajoute tout de suite la déclaration du fichier d'en-tête Windows dans le fichier **stdafx.h**.

```
// Fichiers d'en-tête Windows :
#include <windows.h>
```

Ce fichier contient les déclarations des APIs et constantes communément utilisées.

VI-A-3 - Sous-classement d'un formulaire

Le sous-classement d'un formulaire (donc d'un fenêtre Windows) s'effectue à l'aide de la fonction **SetWindowLong**

L'identifiant **GWL_WNDPROC** demande le changement de procédure de traitement des messages pour la fenêtre concernée.

La fonction **SetWindowLong** nous renvoit l'identifiant de la procédure standard de traitement des messages.

Il est indispensable de conserver cet identifiant : en effet tous les messages ne seront pas gérés par notre procédure.

Il faut donc rediriger les messages vers la procédure standard pour permettre aux messages d'être traités normalement.

Code du fichier TutoCPPRoulette.cpp pour sous-classement d'un formulaire

```
// TutoCPPRoulette.cpp : définit le point d'entrée pour l'application DLL.
//

#include "stdafx.h"

WNDPROC gWndProc;

#ifdef _MANAGED
#pragma managed(push, off)
#endif

BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                                          )
{
    return TRUE;
}

#ifdef _MANAGED
#pragma managed(pop)
#endif

// Procédure de gestion des messages
LRESULT CALLBACK  FormSubclassProc(
    HWND hwnd,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam)
{
        // Appel de la procédure standard de gestion des messages
        return CallWindowProc(gWndProc, hwnd, uMsg, wParam, lParam);
}

void __stdcall MouseWheelHook(DWORD pHwnd)
{
// Déréférence le formulaire et conserve l'identifiant de la précédédante procédure de gestion des
messages
        gWndProc = (WNDPROC)SetWindowLong((HWND)pHwnd, GWL_WNDPROC,  (LONG)FormSubclassProc);
}

void __stdcall MouseWheelUnHook(DWORD pHwnd)
{
// // Réactive la procédure standard de gestion des messages
        SetWindowLong((HWND) pHwnd, GWL_WNDPROC,(LONG)gWndProc);
}
```

La fonction **MouseWheelHook** pourra être appelée depuis Access VBA pour mettre en place le sous-classement.

La fonction **MouseWheelUnHook** annule le sous-classement.

Suivre la section Ajouter une procédure à la librairie pour créer le fichier de définition.

Fichier TutoCPPRoulette.def

```
LIBRARY TutoCPPRoulette
EXPORTS
MouseWheelHook @1
MouseWheelUnHook @2
```

Il faut ensuite demander au compilateur d'utiliser ce fichier de définition.

Dans les propriétés du projet :

Propriétés de configuration : **Editeur de liens** : **Entrée**.

Fichier de définition de module : TestCPP.def

L'identifiant de la procédure standard de gestion des messages est stocké dans la variable **gWndProc**. Il est ensuite utilisé en paramètre de la fonction **CallWindowProc** pour transmettre les messages à traiter.

Où en est-on?

Après appel de la fonction **MouseWheelHook** depuis Access VBA, lorsque le formulaire reçoit un message, celui-ci est envoyé à notre procédure **FormSubclassProc**.

Dans cette procédure la fonction **CallWindowProc** renvoit le message vers la procédure standard qui traite alors le message.

Comment s'assurer alors que le sous-classement fonctionne?

Ajoutons une boîte de message lors de l'utilisation de la roulette.

La procédure de gestion des messages devient :

```
// Procédure de gestion des messages
LRESULT CALLBACK  FormSubclassProc(
    HWND hwnd,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam)
{
        if (uMsg == WM_MOUSEWHEEL)
        { MessageBox(hwnd,"test roulette","",MB_OK); }
        // Appel de la procédure standard de gestion des messages
        return CallWindowProc(gWndProc, hwnd, uMsg, wParam, lParam);
}
```

WM_MOUSEWHEEL est l'identifiant du message envoyé lors de l'action sur la roulette de la souris.

Si le message reçu correspond à un roulement de la souris, on affiche une boîte de message "Test roulette".

Pour tester la librairie, créez un formulaire puis placez ce code dans son module de code :

Code à écrire dans le module du formulaire

```
Option Compare Database
Option Explicit

Private Declare Sub MouseWheelHook _
    Lib
"E:\Transfert_doc\Programmation\VCExpress\Tutoriels\TutoCPPRoulette\release\TutoCPPRoulette.dll" _
    (ByVal pHwnd As Long)
Private Declare Sub MouseWheelUnHook _
    Lib
"E:\Transfert_doc\Programmation\VCExpress\Tutoriels\TutoCPPRoulette\release\TutoCPPRoulette.dll" _
```

Code à écrire dans le module du formulaire

```
      (ByVal pHwnd As Long)

Private Sub Form_Load()
MouseWheelHook Me.Hwnd
End Sub

Private Sub Form_Unload(Cancel As Integer)
MouseWheelUnHook Me.Hwnd
End Sub
```

Utilisez le chemin où est stocké votre librairie ou utilisez **LoadLibrary**.

Voir Exécuter la procédure à partir d'Access pour plus d'infos.

Si tout fonctionne correctement, vous devez voir apparaître une boîte de message lors de l'utilisation de la roulette de la souris.

VI-A-4 - Annuler le message envoyé par la roulette

Pour annuler le message envoyée par la roulette de la souris, il suffit de ne pas envoyer ce message à la procédure standard de gestino des messages.

La procédure de gestion des messages devient :

```
// Procédure de gestion des messages
LRESULT CALLBACK  FormSubclassProc(
    HWND hwnd,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam)
{
        if (uMsg != WM_MOUSEWHEEL)
        {
                // Appel de la procédure standard de gestion des messages
                return CallWindowProc(gWndProc, hwnd, uMsg, wParam, lParam);
        }
}
```

Tester le nouveau code dans Access en créant un formulaire basé sur une tabe.

Par défaut, la roulette fait défiler les enregistrements.

Si on ajoute le précédent code VBA au formulaire, le défilement des enregistrements est annulé.

VI-A-5 - Utilisation de la librairie pour plusieurs formulaires simultanément

Le code de la librairie n'est pas prévu pour gérer le blocage de la roulette sur plusieurs formulaires.

Pour pouvoir gérer plusieurs formulaires, il faut conserver les identifiants des procédures standards de gestion des messages de chacun des fomulaires.

Pour stocker ces identifiants et pouvoir facilement les relire, on utilise une **map**.

Cette **map** est plus ou moins équivalente à une **collection** en VBA.

Pour utiliser les maps, on doit ajouter les déclarations nécessaires dans le fichier **stdafx.h**.

Ajouter les déclarations pour les maps à la suite de la déclaration du fichier d'en-tête Windows

```
// Fichiers d'en-tête Windows :
#include <windows.h>
// Fichiers d'en-tête Windows :
#include <windows.h>
// déclarations pour les maps
#include <map>
using namespace std;
```

La procédure de gestion des messages était stockée dans la variable **gWndProc**.

On remplace la déclaration de cette variable par la déclaration d'une map, que l'on peut assimiler à un tableau d'identifiants.

```
// Identifiants des procédures de gestion des messages
map<HWND, WNDPROC> MapWndProc;
```

Le premier paramètre de la map est de type **HWND**, c'est l'identifiant du formulaire.

Ce premier paramètre est la clé de la map qui sert à distinguer chaque entrée du tableau.

Le deuxième paramètre de la map est de type **WNDPROC**, c'est l'identifiant de la procédure standard de gestion des messages du fomulaire.

Ce deuxième paramètre est la valeur de la map.

Pour obtenir la valeur d'une entrée en fonction de sa clé on utilise la syntaxe :

```
Valeur = MapWndProc[Clé]
```

Dans la procédure **MouseWheelHook**, au lieu de stocker le retour de l'API **SetWindowLong** dans la variable **gWndProc**, on va le stocker dans la map **MapWndProc**.

Au préalable on utilise la méthode **find** de la **map** pour rechercher si l'identifiant du formulaire est déjà dans la map.

```
void __stdcall MouseWheelHook(DWORD pHwnd)
{
// Test si le sous-classement de ce formulaire est déjà activé
        if (MapWndProc.find((HWND)pHwnd) == MapWndProc.end())
        {
        // Déréférence le formulaire et conserve l'identifiant de la précédédante procédure de
gestion des messages
             MapWndProc[(HWND)pHwnd] = (WNDPROC)SetWindowLong((HWND)pHwnd, GWL_WNDPROC,
(LONG)FormSubclassProc);
        }
}
```

(HWND)pHwnd est l'identifiant du formulaire sous-classé.

Dans la procédure **MouseWheelUnHook**, on doit rechercher l'identifiant de procédure correspondant à l'identifiant de formulaire passé en paramètre.

Une fois le sous-classement annulé, on peut supprimer l'entrée correspondante dans la map.

```
void __stdcall MouseWheelUnHook(DWORD pHwnd)
{
// Test si le sous-classement de ce formulaire est activé
        if (MapWndProc.find((HWND)pHwnd) != MapWndProc.end())
        {
                // Réactive la procédure standard de gestion des messages
                        SetWindowLong((HWND) pHwnd, GWL_WNDPROC,(LONG)MapWndProc[(HWND)pHwnd]);
                // Supprime les entrées dans les maps
                        MapWndProc.erase((HWND)pHwnd);
        }
}
```

Idem dans la procédure de gestion des messages **FormSubclassProc**.

```
// Procédure de gestion des messages
LRESULT CALLBACK  FormSubclassProc(
    HWND hwnd,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam)
{
        if (uMsg != WM_MOUSEWHEEL)
        {
                // Appel de la procédure standard de gestion des messages
                return CallWindowProc(MapWndProc[(HWND)hwnd], hwnd, uMsg, wParam, lParam);
        }
}
```

On peut maintenant utiliser la même librairie pour plusieurs formulaires.

VI-A-6 - Utiliser la roulette pour faire défiler le formulaire de haut en bas.