

# L'environnement Windows

## 1. Les programmes Win32

La plate-forme Win32 a très vite été supportée par C++ car Microsoft a lancé dès 1992 l'un des premiers environnements intégrés pour ce langage ; il s'agissait de Visual C++, et à cette époque, de nombreux architectes étaient convaincus de l'intérêt d'un langage objet proche du langage C pour créer des applications fonctionnant dans des environnements graphiques.

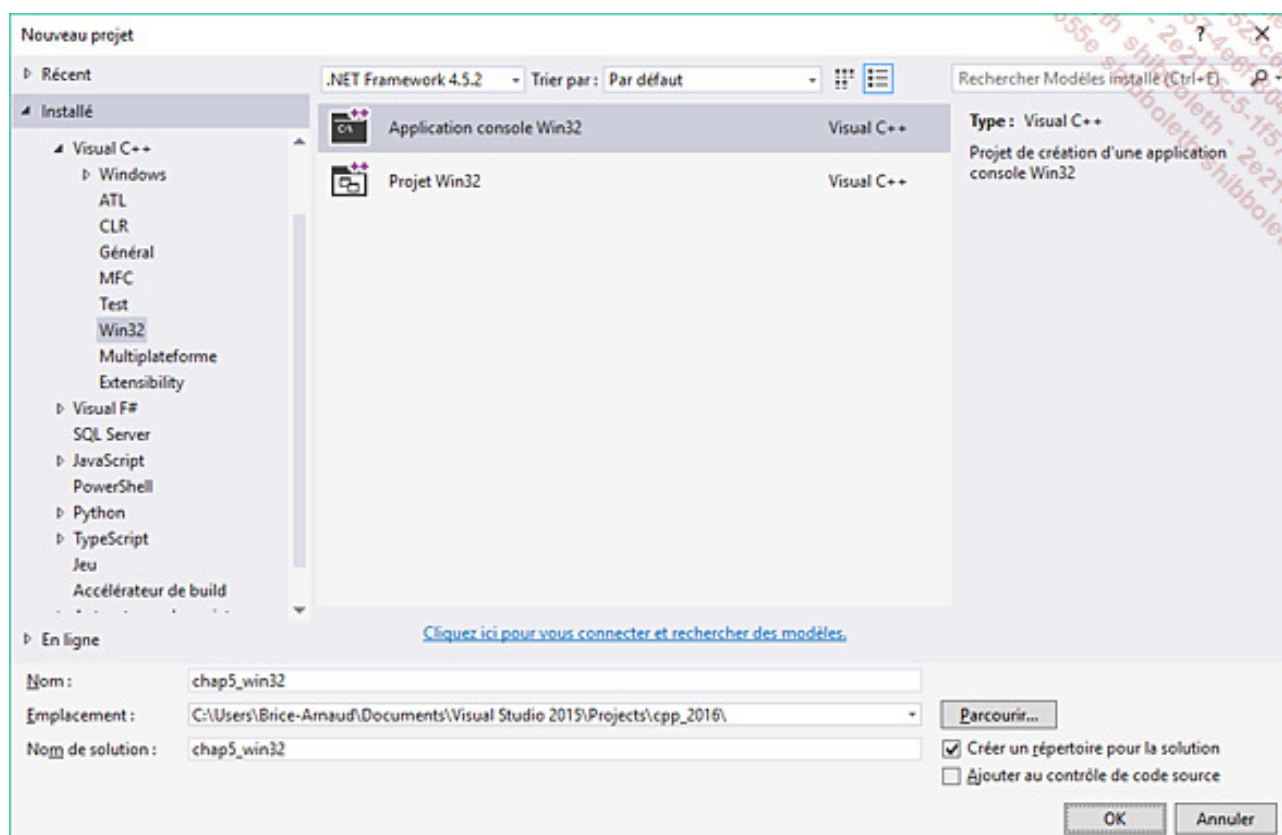
Toutefois, la programmation d'une application fenêtrée, sans framework, n'est pas une tâche aisée. Il n'en demeure pas moins que la plateforme Win32 propose plusieurs formats d'applications : applications "console", services Windows, bibliothèques dynamiques (DLL) et bien entendu les applications graphiques fenêtrées.



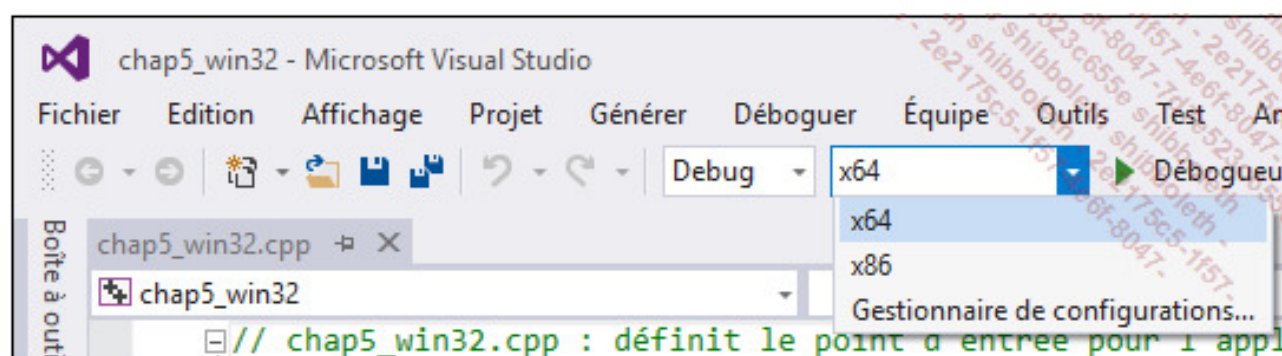
Il n'existe pas de modèle de projet "Win64" car Windows reste un système très ancré dans le mode 32 bits. Toutefois, Visual Studio sait compiler en mode 64 bits depuis un projet Win32.

## 2. Choix du mode de compilation

La création d'un projet s'effectue avec la commande **Fichier - Nouveau Projet** :

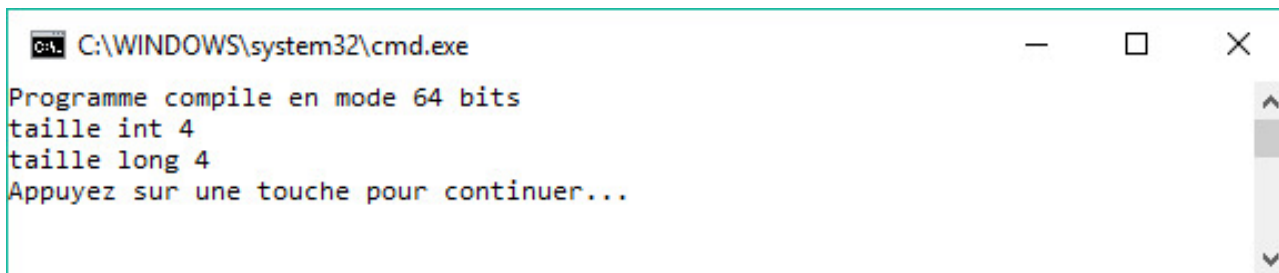


Depuis le **Gestionnaire de configurations**, le mode de compilation 64 bits est activé :



Le programme qui suit est intéressant ; il indique que la taille des types `int` et `long` est identique à un mode 32 bits.

```
printf("Programme compilé en mode 64 bits\n");
printf("taille int %d\n", sizeof(int));
printf("taille long %d\n", sizeof(long));
```



Le mode 64 bits recommande au compilateur de produire des instructions microprocesseur 64 bits, mais cela ne veut pas dire que les types sont ajustés en conséquence. Autrement dit, un calcul sur un type `__int64` pourra prendre plus de temps en 32 bits qu'en 64, mais `__int64` reste toujours 64 bits et `int` reste sur 4 octets quel que soit le mode de compilation. Il s'agit là d'une particularité du compilateur de Microsoft ; d'autres compilateurs peuvent adopter un comportement différent.

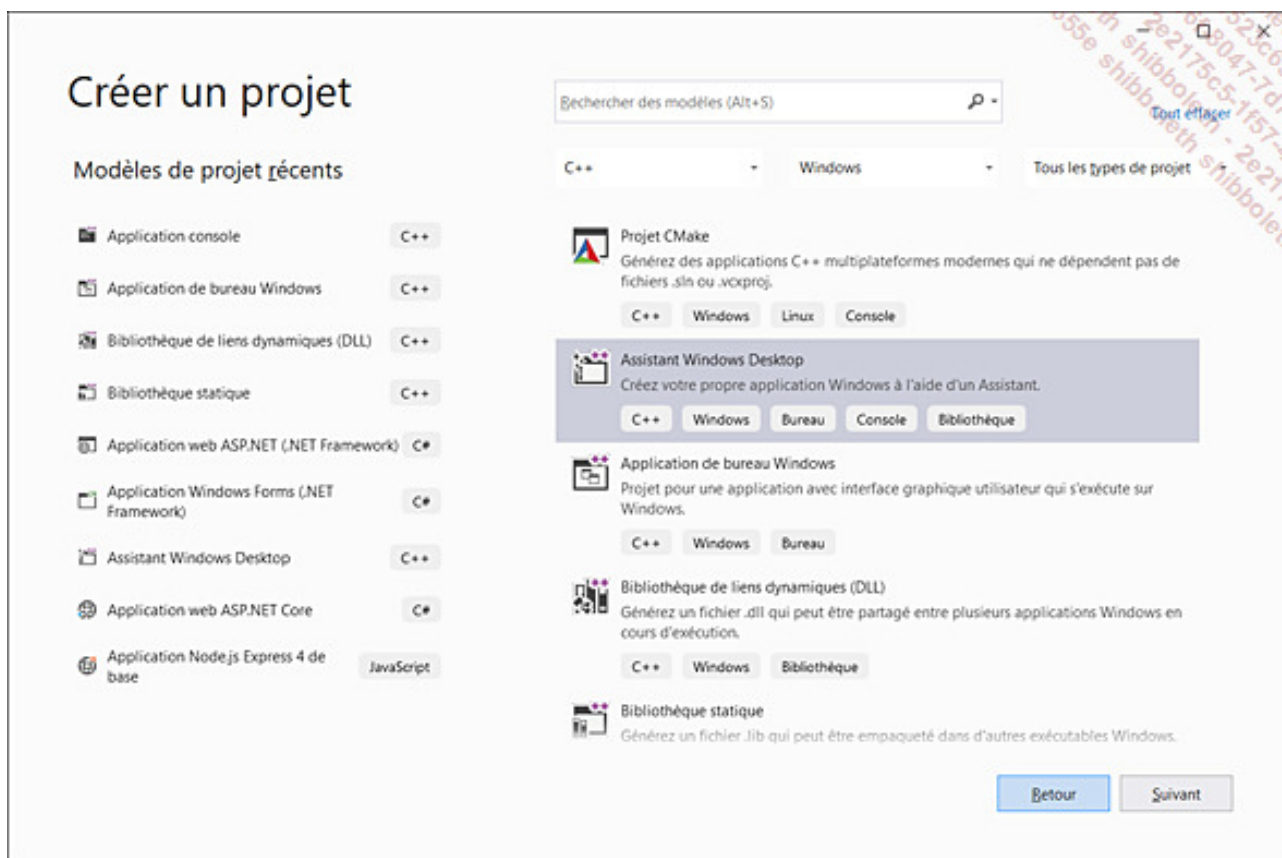
### 3. Fonctionnement des applications fenêtrées Win32

Il n'y a pas de différence structurelle entre l'exécutable console et le desktop. Ce sont des appels système qui déclenchent la bascule vers le mode fenêtré.

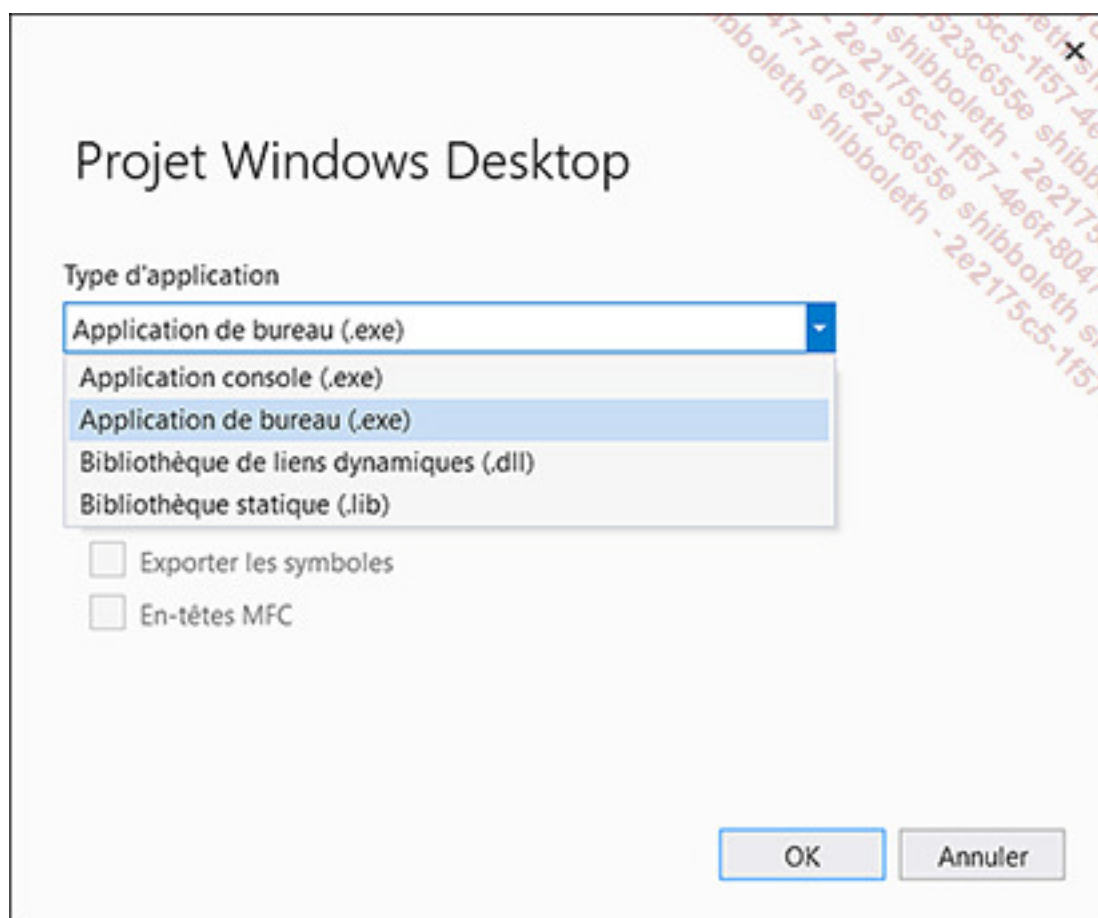
Cependant, le fonctionnement d'une application fenêtrée est assez particulier, car le programme doit réagir aux sollicitations de l'utilisateur - clic de souris, raccourci clavier, superposition de fenêtres - ainsi qu'aux événements externes comme la lecture de programmes multimédias ou encore des dialogues sur le réseau.

#### a. Création d'un projet d'application fenêtrée

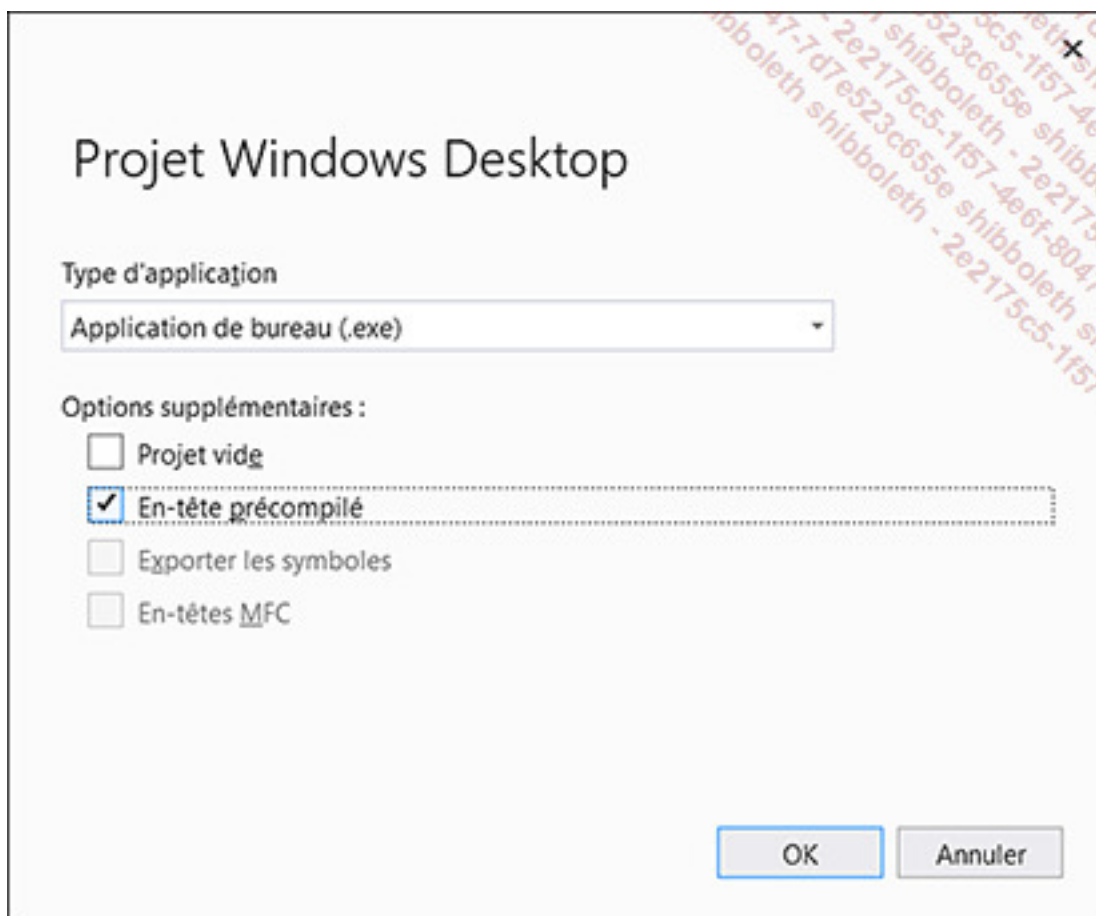
Visual Studio propose un modèle de projet d'application fenêtrée appelé **Application Windows Desktop** :



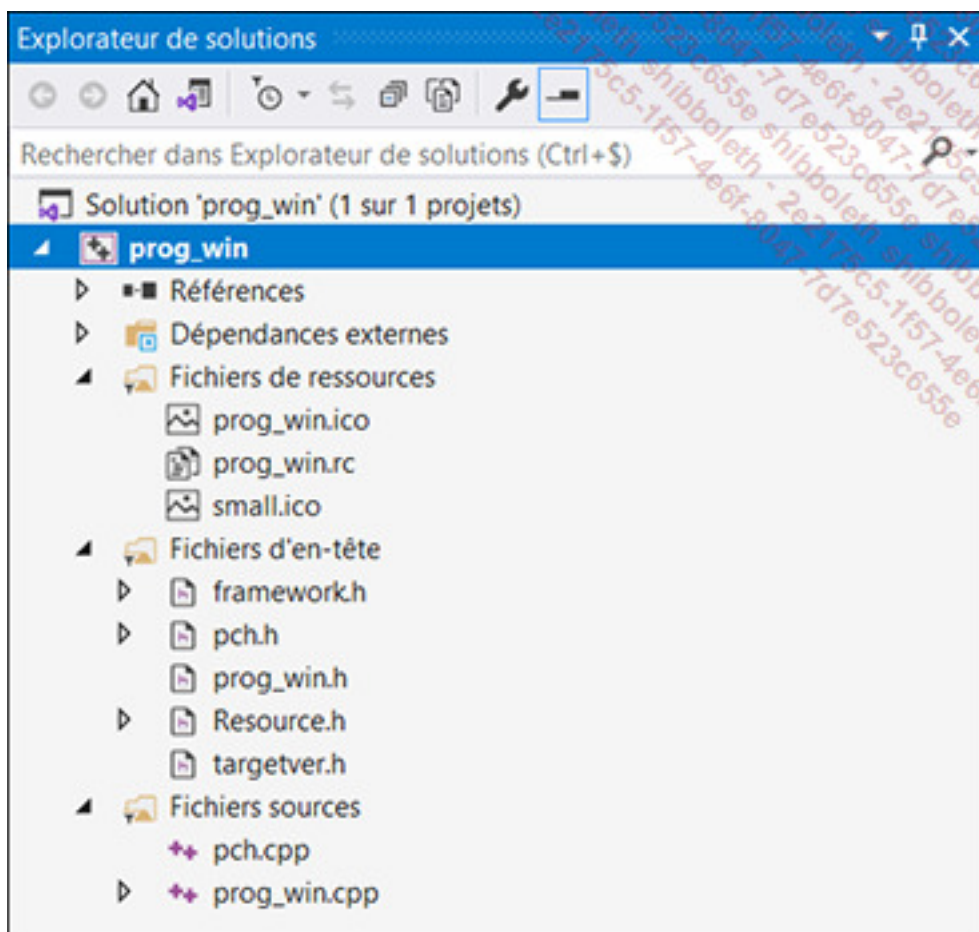
Après avoir donné un nom au projet, `prog_win` dans notre exemple, l'assistant demande choisir le type d'application :



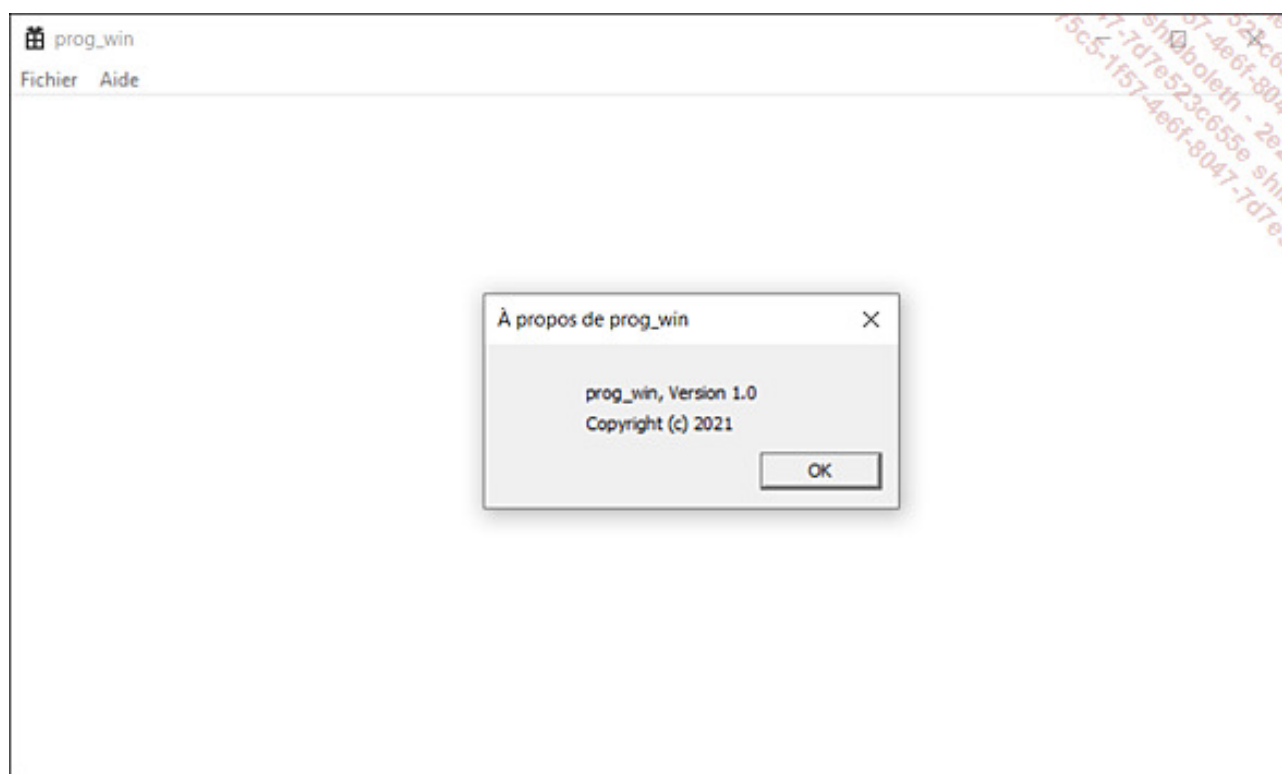
Pour réaliser des applications fenêtrées, les types **Application de bureau (.exe)** et **Bibliothèque de liens dynamiques (.dll)** sont les plus directs. Compte tenu de la taille des fichiers d'en-tête, il est utile d'activer le support des en-têtes précompilés :



Visual Studio initialise le projet C++ à partir de fichiers .h et .cpp mais aussi de fichiers ressources .rc et .ico.



Le programme est déjà fonctionnel, la commande [F5] lance sa compilation et son exécution :



## b. Les types de données Win32

Rappelons-nous que l'interface de programmation Win32 a été conçue de façon assez proche du matériel, autrement dit c'est un mélange d'assembleur et de langage C. Les premières applications étaient écrites en langage C et son empreinte se manifeste encore dans la désignation des types de données.

Le support de C++ est apparu après coup pour faciliter le développement d'applications complexes telles que MS Office à l'aide de puissants frameworks publiés par Microsoft tels que MFC, ATL, COM ou encore OWL édité par Borland.

Dans l'univers natif Windows, ces frameworks ont moins la cote, les environnements managés tels que .NET ont pris le relais. Mais la programmation C++ sous Win32 reste une référence incontournable pour la mise au point d'applications performantes.

Au fil des versions de Windows, en partant des premières versions 1.0, 2.0, 3.0 et 3.11 avec le réseau, en passant par les révolutionnaires NT 4, Windows 2000 et plus récemment Windows 7 à Windows 10, un élément matériel a considérablement évolué : la mémoire, qui bien plus que le processeur a vu sa capacité s'étendre selon un facteur exponentiel. Les PC sont passés d'un espace mémoire adressable de 640 Ko à plusieurs dizaines de



Go.

Quel impact cela peut-il avoir sur la conception du système d'exploitation Windows et de son interface de programmation d'application API Win32 ? En réalité, cela change constamment, les "adresses" des fonctions système évoluent au gré du grossissement du code de base, tandis que les pointeurs vers les données passées en argument de ces fonctions se font de plus en plus longs.

Il était de plus évident que la richesse d'un système d'exploitation fenêtré rendrait tôt ou tard le langage C assez limité pour réaliser des applications d'envergure et que d'autres langages comme C++, VB, Power Builder ou C# viendraient compléter l'offre aux développeurs.

En fin de compte, les concepteurs de Windows ont choisi des termes génériques pour encapsuler les structures C dans des objets de plus haut niveau, capables de s'affranchir des particularités de l'adressage mémoire et de s'adapter aux différents langages de programmation cibles.

Voici quelques exemples de types Win32 :

Type	Forme pointeur	Commentaire
CHAR	LPCHAR	Un char, un pointeur long vers un char
WCHAR	LPWCHAR	Un caractère sur deux octets ( <i>wide</i> ) Un pointeur vers un double char.
CST	LPCST	Une chaîne asciiz constante, un pointeur long vers une chaîne asciiz
RECT	LPRECT	Un rectangle, un pointeur long vers un rectangle
INSTANCE	HINSTANCE	Une instance d'application, un handle d'instance d'application
PARAM, LPARAM		Paramètre (apparenté d'entier), paramètre long
WPARAM, LPARAM		Paramètre mot (apparenté int32), paramètre double mot
RESULT, LRESULT		Résultat (apparenté entier), résultat long

### c. Les handles et les messages

Nous avons déjà rencontré les **handles** à l'occasion de l'étude des fichiers. Un handle est

une sorte de référence système, un pointeur qui s'interprète comme un index ou un itérateur au sens STL.

Les handles sont les véritables pointeurs vers les structures système importantes. Les pointeurs longs (LPxxx) sont plutôt destinés à l'échange rapide de petits objets au sein de l'espace mémoire du processus.

Une application Windows dispose d'un handle d'instance et le système Windows adresse à cette instance des messages pour la piloter.

Il existe plusieurs types de messages pour chaque domaine du système, **WM\_PAINT** et **WM\_INVALIDATE** pour l'interface graphique, **WM\_COMMAND** pour les menus, **WM\_INITDIALOG** et **WM\_DESTROY** pour la gestion des fenêtres.

#### d. La boucle de message

Une application Win32 s'articule autour de sa boucle de message. Après une phase d'initialisation, une fonction **WndProc** (*Window Procedure*) est appelée (*callback*) par Windows à chaque message. Elle est chargée de dispatcher le traitement selon les paramètres du message :

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_COMMAND:
        {
            int wmlId = LOWORD(wParam);
            // Analyse les sélections de menu :
            switch (wmlId)
            {
                case IDM_ABOUT:
                    DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                    break;
                case IDM_EXIT:
                    DestroyWindow(hWnd);
                    break;
                default:
                    return DefWindowProc(hWnd, message, wParam, lParam);
            }
        }
    }
}
```

```

    }
}

break;

case WM_PAINT:
{
    PAINTSTRUCT ps;
    HDC hdc = BeginPaint(hWnd, &ps);
    // TODO: Ajoutez ici le code de dessin qui utilise hdc...
    switch (affichage)
    {
        case Affichage::Surface:
            OnDrawG(hWnd, hdc, ps);
            break;
        case Affichage::Fractale:
            OnDrawF(hWnd, hdc, ps);
            break;
    }

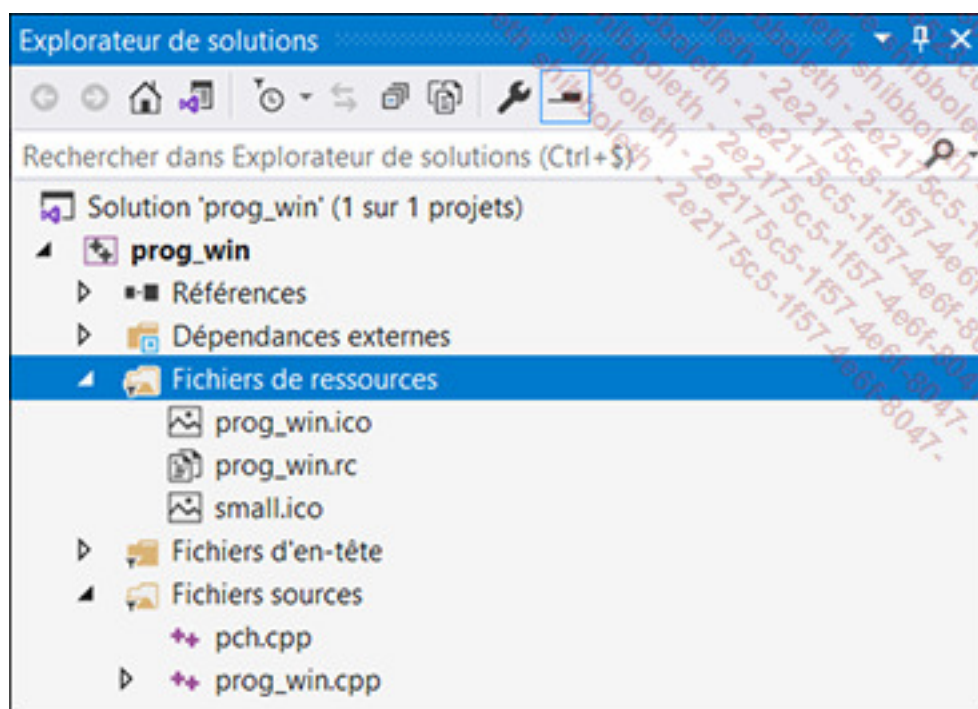
    EndPaint(hWnd, &ps);
}
break;
case WM_DESTROY:
    PostQuitMessage(0);
    break;
default:
    return DefWindowProc(hWnd, message, wParam, lParam);
}

return 0;
}

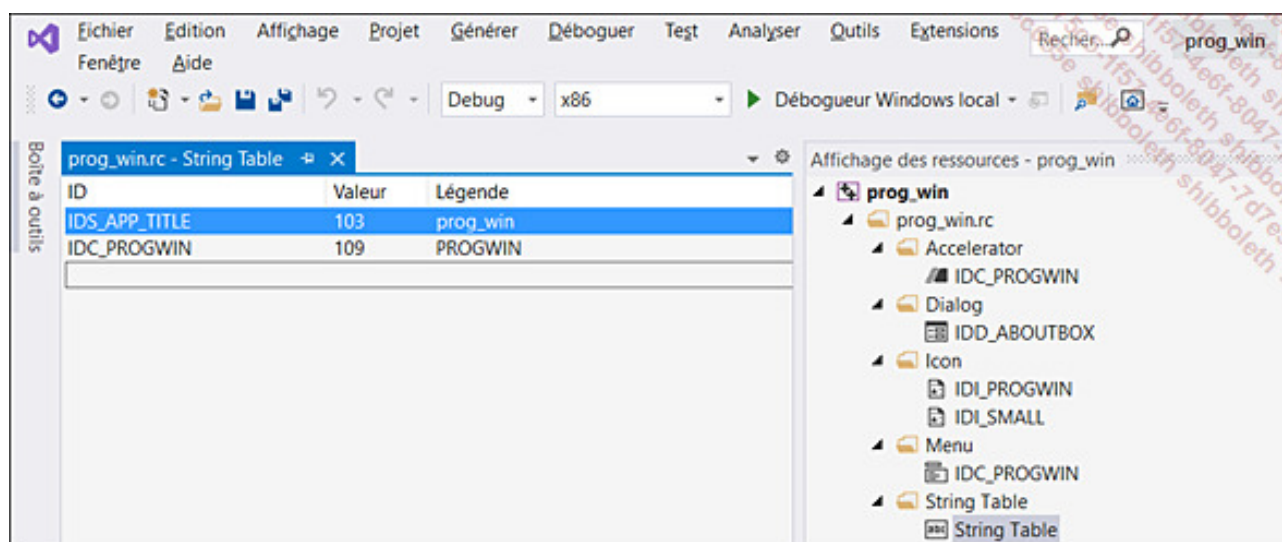
```

## e. Les fichiers de ressource

Les ressources sont des éléments annexes qui ne sont pas traités par le compilateur C++. Il s'agit des menus, des icônes, des tables de chaînes, des images, des types de fichiers, des raccourcis claviers, des boîtes de dialogue... venant en appui du programme C++.



Les fichiers de ressources disposent de leur propre compilateur (**rc** pour *resource compiler*) invoqué par Visual Studio à la génération du projet.



Chaque ressource dispose d'un identifiant apparenté entier. Par convention ces identifiants sont préfixés par **ID\_** et sont définis à la fois dans le fichier de ressource et dans le programme C++ chargé de récupérer l'objet ressource au cours de l'exécution du programme.

```

#define IDS_APP_TITLE          103

#define IDR_MAINFRAME          128
#define IDD_PROGWIN_DIALOG    102
#define IDD_ABOUTBOX          103
#define IDM_ABOUT              104
#define IDM_EXIT              105
#define IDI_PROGWIN            107
#define IDI_SMALL              108
#define IDC_PROGWIN            109
#define IDC_MYICON             2
#ifndef IDC_STATIC
#define IDC_STATIC             -1
#endif
// Valeurs par défaut suivantes des nouveaux objets
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS

#define _APS_NO_MFC              130
#define _APS_NEXT_RESOURCE_VALUE 129
#define _APS_NEXT_COMMAND_VALUE 32771
#define _APS_NEXT_CONTROL_VALUE 1000
#define _APS_NEXT_SYMED_VALUE  110
#endif
#endif

```

Les ressources permettent de séparer la logique de programmation C++ avec la construction d'éléments d'interface graphique tels que des boîtes de dialogue. Visual Studio propose des outils prêts à l'emploi et une fenêtre d'édition des propriétés très pratique pour modifier le contenu des boîtes de dialogue.

