Introduction À Windows

Par Moon Coder
www.chez.com/mooncoder
mooncoder@gmx.net

Dernière mise à jour, 21 juin 2002

Introduction

Ah… la programmation Windows, je me rappelle bien! Quand, j'ai commencé, j'ai du abandonner les concepts de programmation du vieux dos, et la programmation séquentielle, là où toutes les opérations été permises, sans détour ni appel à une quelconque fonction tierce… ce temps est bien révolu, place à la programmation évènementielle, et les nouveaux concepts qui sont apparus.

Dans un programme Windows type, on voit apparaître la notion de fenêtre, qui n'est que la concrétisation d'un espace réservé à vos sorties sur l'écran, un espace où on peut dessiner, écrire, recevoir des clicks de souris de la part de l'utilisateur (Fig 1).

Bien sur, cette définition est un peu générale, et c'est simplement pour faciliter la compréhension. Là, ou un programme DOS, accapare toutes les ressource du système, envoyant des sorties sur n'importe quelle région de l'écran, bloquant le clavier en attendant que l'utilisateur tape une touche pour prendre une décision, un programme Windows, doit cohabiter avec plusieurs programmes qui eux aussi ont besoin de l'écran et du clavier pour leur traitement. C'est la contrainte de la multi-tâche.

Pour coordonner tous ça, Windows (ou je sais pas qui) à mis au point la notion d'Evénement. Dès qu'un événement surgit, Windows envoie à l'application un message qui n'est qu'une structure lui indiquant l'action qui vient de passer. Cette structure contient entre autre, le type d'événement qui vient de surgir (un long), et le temps d'apparition.

```
typedef struct tagMSG {
  HWND   hwnd;
  UINT   message;
  WPARAM wParam;
  LPARAM lParam;
  DWORD  time;
  POINT  pt;
} MSG
```

A la création de chaque fenêtre, Windows associe une queue où seront stockés tous les messages qui lui sont destinés, en attendant d'être traités par une fonction, appelée généralement une procédure de fenêtre.

Une fois la fenêtre crée, et la procédure de fenêtre fixée, il ne reste plus qu'à faire appel au nombreux API[1], et faire le traitement selon les résultats renvoyés.

→ La programmation séquentielle est toujours possible sous Windows, on appelle généralement les applications qui adoptent ce principe, Programme Console.

La Création d'une application Console, ne diffère en rien du vieux temps, un programme avec une fonction principale, appelée main().

```
#include <iostream.h>
Void main ()
{
     cout << "bonjour le monde ! \n" ;
}
```

Bien sûr, on peut appeler des fonctions de l'Api dans un programme console

[1] Application Programming interface, des fonctions mises à disposition par le système d'exploitation pour accomplir certaines tâches.

Les Fenêtres

Un programme Windows, avec une interface utilisateur graphique (appelle GUI), comporte généralement une fonction ***WinMain()***, et une fonction WinProc(), appelée procédure de fenêtre..

La première fonction, est responsable de remplir les tâche suivante:
- Enregistrement de la classe de fenêtre.
- Création de la fenêtre proprement dite.
- Récupération des messages puis Distribution vers les fenêtres filles.

C'est à WinProc() qui incombe la lourde tâche de bien traiter les messages reçus de la part de Windows - à coups de switch et de case.

C'est toujours le même schéma, vous pouvez vous en passer d'apprendre par coeur les étapes qui suivent, juste en tirer le principe, et avoir à porté de main, un template (squelette) comme le fait gracieusement Visual C++ 6.

Enregistrement de la classe

Chaque fenêtre possède une Classe qui indique entre autre, le style, l'icône, le curseur, le menu, et le plus important de tous : la procédure de fenêtre, elle est indiquée dans le membre `lpfnWndProc`, cette procédure est appelée une procédure de rappel, (callback function).

On remplit les champs de la structure WNDCLASS, puis on fait appel à la fonction RegisterClassEx(), avec comme paramètre l'adresse de cette structure.

```
wc.lpszClassName = "NomClass"; // le paramètre à spécifier dans CreateWindowEx()
wc.lpfnWndProc = WndProc;
wc.style = CS_VREDRAW | CS_HREDRAW;
wc.hInstance = hInstance;
wc.hIcon = LoadIcon( NULL, IDI_APPLICATION );  // icône de l'application
wc.hCursor = LoadCursor( NULL, IDC_ARROW );  // curseur de souris
wc.hbrBackground = (HBRUSH)( COLOR_WINDOW+1 ); // couleur de la fenêtre
wc.lpszMenuName = NULL ; // pas de menu
wc.cbClsExtra = 0;
wc.cbWndExtra = 0;

RegisterClass (&wc);   // enregistrer la classe.
```

Création

On se doute pourquoi, il y a séparation en deux étapes de la procédure de création d'une fenêtre. Comme, il y parait clairement dans la section réservée à l'enregistrement, la fenêtre possède plusieurs attributs qu'il faut fixer. On peut considérer cette étape comme la construction d'un moule. Ensuite on crée des fenêtres en se basant sur ce moule. C'est pour cette raison que les contrôles EDIT, BUTTON, n'ont pas à être enregistrés, ils le sont déjà. Il suffit simplement de faire appel à la fonction ***CreateWindowEx()***, qui est nécessaire dans la mesure où il faut indiquer la position de la fenêtre, les styles, le titre qui apparaîtra en haut, et l'instance qui a enregistré la classe (dans le cas de Windows 2000 et XP, ce paramètre est ignoré).

```
hwndMain = CreateWindowEx(
      0,         // pas de style extravagant
      "NotreClass",   // le nom de la classe, enregistré par RegisterClassEx()
      "tire de Fenêtre ", // Le nom de la fenêtre.
      WS_OVERLAPPEDWINDOW | // style :
         WS_HSCROLL | // scroll bar horizontale
         WS_VSCROLL,  // vertical scroll bar
      CW_USEDEFAULT,  // laisse Windows décider des coordonnées
```

```
            CW_USEDEFAULT,  //
            CW_USEDEFAULT,  // laisse Windows décider de la taille
            CW_USEDEFAULT,  //
            (HWND) NULL,     // pas de fenêtre parent
            (HMENU) NULL,   // pas de menu
            hInstance,       // l'instance, passée en WinMain()
            NULL);           // paramètre a passer à WndPorc().
```

Affichage

On peut s'en passer de l'appel de ShowWindow() qui fait apparaître la fenêtre à l'écran et spécifier le Style WS_VISIBLE dans l'appel de CreateWindow(). La principale raison pour faire appel à ShowWindow(), c'est pour faire des traitement avant d'afficher à l'écran la fenêtre, e. g. centrage de la fenêtr,…

Pompage des messages

Une fois la classe enregistrée et la fenêtre affichée, on commence par récupérer les messages et les distribuer vers nos fenêtres (car on peut en avoir plusieurs).

```
     while( GetMessage( &msg, NULL, 0, 0 ) ) {
          TranslateMessage( &msg );
          DispatchMessage( &msg );
     }
```

On fait appel à la fonction GetMessage() en lui indiquant une structure de type MSG, et l'intervalle des message (dans l'exemple, ci-dessus, on veut récupérer tous les messages). On fait ensuite appel à TranslateMessage() pour transformer les messages de clavier en simples caractères (WM_KEYDOWN & WM_KEYUP en WM_CHAR)

A la fin, DispatchMessage() envoie le message vers la fenêtre concernée. On sort de cette boucle à la réception du message WM_QUIT, qui est transmis par la fonction PostQuitMessage().

Procédure de Fenêtre

Chaque fenêtre doit posséder une procédure qui traite les message que Windows lui envoi (e. g. Un click de souris ou de clavier, fermeture de l'application). Le squelette de cette fonction est toujours le même.

```
LRESULT WINAPI WndProc( HWND hwnd, UINT Msg, WPARAM wParam, LPARAM lParam)
{
switch( uMsg )
{
     case WM_PAINT:
          ...
          break ;
     case WM_DESTROY:
          PostQuitMessage(0) ; // la boucle se termine après l'envoi de ce message
          break ;
     default:
          Return (DefWindowProc (hWnd, msg, wParam, lParam));
}
```

Pour tous les messages non traités, on fait appel à la fonction DefWindowProc(). C'est une fonction mise à la disposition du programmeur par Windows, pour lui faciliter la tâche et faire en sorte qu'on ne traître que les messages intéressant.

Le paramètre hWnd, indique la fenêtre pour qui, le message a été destiné, wParam et lParam quand à eux donne plus d'information pour le message Msg.

Création Des Contrôles

Les contrôles standards

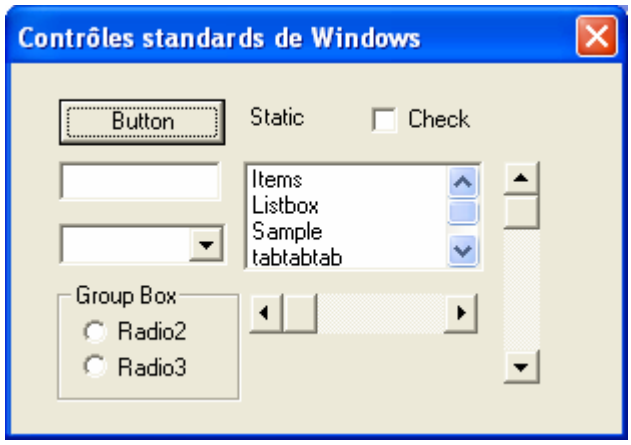
Les nombreux buttons, champs de texte, et autres que Windows met à la disposition de l'utilisateur pour standardiser les interfaces, sont appelés fenêtre fille (Child Window), ils ont tous un fonctionnement précis.

Ces contrôles ne sont rien d'autre que des fenêtres. Mais là où vous enregistrer la classe de fenêtre, Windows le fait pour vous, et fournie la procédure de traitement des messages, ce qui fait que il reste simplement à créer le contrôle, en spécifiant le nom de la classe dans la fonction CreateWindowEx().

```
//Exemple de création d'un button

hwndButton = CreateWindowEx(
      0,
      "BUTTON",                    // le type de contrôle
      "tire de Fenêtre ",          // le texte qui apparaît sur le button
      WS_CHILD | WS_VISIBLE,       //
      10,10,20,20,                 // la positon et la taille du button
      hWnd,                        // fenêtre contenant le button.
      (HMENU) ID_BUTTON,           // l'identificateur du button.
      hInstance,                   // l'instance, passée en WinMain()
      NULL);
```

Voici une fenêtre représentant tous les contrôles standard de Windows



Sur cette image, on voit bien les contrôles, Bouton, Static, Check box, Edit, List Box, Combo Box. Ensuite dans un Group Box, il y deux radio buttons, enfin une Horizontal et vertical scroll bar control.

Windows à deux façons d'identifier un contrôle, soit par son handle, obtenu en le créant, soit un ID (nombre) passé à la fonction CreateWindowEx() dans le paramètre du Menu.

A la réception d'un Click de souris de la part de l'utilisateur pour un Bouton, ou clavier dans un champs de texte (Edit), le contrôle envoie le message WM_COMMAND pour la fenêtre parent, pour lui indiquer un changement dans son état. Le paramètre lParam indique le handle du contrôle, alors que wParam indique son ID dans l'entier du poids fort et le type de message (click de la souris avec la droite ou double click) dans l'entier du poids faible. On se sert des Macro HIWORD() et LOWORD() pour les récupérer.

```
switch( uMsg )
{
    case WM_COMMAND:
          switch(LOWORD(wParam))
          {
              case ID_BUTTON :
                ...
              break ;
              case ID_EDIT :
              ...
              break ;
          }
    break ;
```

```
    }
    ...
```

Les Contrôles Communs

Avec l'introduction de Windows© 95, et l'amélioration de son interface, plusieurs contrôles font leur apparition, parmi eux on cite, les TreeView, ListView, Image List, etc. leur création ne diffère en rien des précédents, sinon le fait qu'il faut charger la librairie (DLL) comctrl32.dll responsable de leur enregistrement avec l'insertion quelque part dans votre code de cette fonction InitCommonControlsEx().à l'exception du RichEdit contrôle qui constitue à lui seul une DLL.

Mais, là où la création d'un contrôle standard nécessite l'appel d'une seule fonction, les contrôles communs ont besoin de plusieurs lignes de code. Cela est dû à leur complexité Certains contrôles sont créer à l'aide de fonctions dites wrapper, comme CreateToolbarEx() pour la toolbar et CreateStatusWindow() pour Status Window.

Windows subclassing et superclassing

On désigne par ces termes le fait de modifier le comportement des contrôles de Windows. Soit pour changer une caractéristique, ou lui ajouter de nouvelles.

La différence entre ces deux notions, réside dans le niveau de changement, alors que le subclassing opère sur une seule instance d'un contrôle - un seul bouton de votre application -, le superclassing quand à lui, se base sur la classe d'un contrôle et par la suite sur tous les bouton qui sont créer à partir de cette classe modifiée.

Windows subclassing

Supposons le cas ou vous voulez avoir un champ de texte, où vous acceptez seulement les chiffres, la seule méthode possible pour le moment est de laisser l'utilisateur taper son texte, pour ensuite le rejeter en cas de présence de caractères.

Avec le subclassing, vous détourner la procédure de fenêtre, et à chaque réception de caractère vous saurez si l'utilisateur a tapé un caractère ou un chiffre. Cet exemple s'appliquer seulement sur la fenêtre de procédure, mais il y a d'autre

```
SetWindowLong(HWND hWnd, int nIndex, long dwNewLong) ;
```

Où nIndex peut prendre des valeurs parmi

GWL_EXSTYLE :	fixe un nouveau style étendu.
GWL_STYLE	fixe un nouveau style
GWL_WNDPROC	fixe une nouvelle fenêtre de procédure
GWL_HINSTANCE	fixe une nouvelle instance
GWL_ID	fixe une nouvel ID pour la fenêtre.

Windows superclassing

La méthode précédente suffisait dans la mesure où il fallait changer seulement le comportement d'un seul contrôle, mais imaginez que vous vouliez changer le style de tous les boutons de votre application, il ne serait pas raisonnable de subclasser tous les contrôles. Il est judicieux d'opérer au niveau de la classe.

La première des choses est de faire appel a la fonction GetClassInfoEx() pour récupérer la structure WNDCLASS.

```
BOOL GetClassInfoEx(
  HINSTANCE hinst,    // handle to application instance
  LPCTSTR lpszClass,  // class name
  LPWNDCLASSEX lpwcx  // class data
);
```

Après appel de cette fonction, vous aurez la classe que Windows a enregistré la classe avec, tous le champs sont prêts à être changé pour refléter le nouveau comportement.

Vous devez accomplir les tâches suivantes afin de compléter le subclassing:

1. Les membres hInstance et lpzClassName doivent être changées.
2. Enregistrer la nouvelle classe, comme vous le ferez pour une nouvelle classe.
3. Créer de nouveaux contrôles à partir de cette nouvelle classe.

Les Ressources

Parmi les ressources connus qu'on peut citer, sont les Boites de dialogues, les tables de chaînes de caractères, les raccourcis, etc.

Les ressources sont un moyen pour faciliter la création

Vous spécifier le tous dans un fichier de ressources, généralement avec l'extension « .rc », puis vous compiler le fichier avec un *compilateur de ressource.* Le fichier résultant doit être lié avec le fichier exécutable. Les IDE modernes comporte généralement un éditeur de ressource graphique, qui facilite grandement le travail avec les ressource, un simple click de souris sur un boutons, et toutes les étapes décrites sont effectuées.

L'utilisateur fait appel à la fonction FindRessource(), pour savoir si la ressource se trouve bien dans le module spécifiés, ensuite à LoadRessource(), pour charger la ressource en mémoire. Ou utiliser les fonctions dédiées aux ressource spécifique comme LoadBitmap().

Les Boîtes de dialogues

Les boîtes de dialogue ne sont que des simples fenêtres comme ceux que vous créiez.

Ils sont conçus pour envoyer ou récupérer plus d'information de l'utilisateur. Se sont des simples fenêtres, crées sans passer par le schéma habituel (enregistrement et création),

Il existe deux types de boîte de dialogue, modal et non modal.

```
MYDIALOG DIALOG DISCARDABLE  0, 0, 222, 188
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "titre de la fenêtre"
BEGIN
    PUSHBUTTON      "&Ok",ID_CLOSE,112,163,47,14
    PUSHBUTTON      "&Cancel",ID_CANCEL,160,163,47,14
    LTEXT           "texte statique",IDC_STATIC,18,79,50,8
    EDITTEXT        ID_BUTTONTEXT,89,128,109,14,ES_AUTOHSCROLL
END.
```

La procédure de fenêtre de la boite de dialogue est la même que celle d'une procédure de fenêtre, la différence réside dans l'appel de DefWindowProc(), alors que dans une procédure de fenêtre pour une fenêtre normale vous appelez cette fonction, dans une procédure de fenêtre pour une boite de dialogue, vous retournez seulement FALSE pour les messages non traitées.