Title : Lập trình Win32

Author : Vu Hong Viet

Date : 07/09/2014

Lập trình Win32 API

1. Giới thiệu

Đây là Tutorial cho lập trình Windows API. Tutorial này sẽ hướng dẫn bạn những kiến thức cơ bản và nâng cao cho phần lập trình trong Windows API với ngôn ngữ C. Ở đây không bao hàm phần MFC (Microsoft Foundation Classes sử dụng thư viện C++ để phát triển các ứng dụng C++ trên Windows)
Tutorial này sẽ hướng dẫn các bạn tạo và test 1 ứng dụng Win32 trên Windows OS. Các bài viết trong Tutorial này được viết và complier trên Visual C++ (Visual Studio 2008).

2. Windows API

Windows API là các hàm được sử dụng để tạo các ứng dụng Windows. Windows SDK(Software Development Kit) bao gồm header file, library (Windows API) , sample, documentation và các tool (Windows SDK đã được tích hợp trong Visual Studio).

Widows API có thể chia ra thành các loại sau :
Base services
Security
Graphics
User Interface
Multimedia
Windows Shell
Networking
- Base services cung cấp các resource cơ bản trên Windows. Chúng bao gồm file system, devices, processes, threads, registry hoặc là xử lý error.
- Security cung cấp các function, interface, object và các programming element cho việc authentication, Cryptography, Security
- Graphics bao gồm các GDI (Graphic Device Interface),GDI+, DirectX hoặc OpenGL.
- User Interface cung cấp các function để tạo ra các window, các control.
- Multimedia cung cấp các tool để làm việc với video, sound, và các thiết bị đầu vào.

3.MSDN

MSDN (Microsoft Developer Network) là nơi cung cấp cho bạn các tool, môi trường phát triển ứng dụng, library. Bạn có thể truy cập theo địa chỉ sau:
http://msdn.microsoft.com/vi-vn/
Các bạn có thể tham khảo các API cho phần Tutorial trên link MSDN :
http://msdn.microsoft.com/library/default.aspx

Cấu trúc chương trình trong Windows

Chú ý: từ bây giờ cho đến hết Tutorial này. Chúng ta sẽ dùng windows : cửa sổ, còn Windows : hệ điều hành Windows.

1. windows
Theo quan điểm của người lập trình, mọi thứ trong ứng dụng đều là cửa sổ (windows)

2. Cấu trúc chương trình trong Windows
Mọi chương trình ứng dụng trong Windows bắt buộc phải có 2 hàm:
- WinMain()
- Window procedure

2.1 WinMain(): tương tự như hàm main() (trong Dos hoặc UNIX) khởi tạo chương trình ứng dụng. Có 2 nhiệm vụ chính:
- Hiển thị cửa sổ ứng dụng lên màn hình
- Tiến hành vòng lặp Message

Khai báo hàm Winmain()
Code:

```
int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,LPSTR
lpCmdLine, int nCmdShow );
```

hInstance là một thể hiện của chương trình. Nó là số nguyên 32bit, số nguyên này sẽ được cho bởi Windows khi chương trình ứng dụng bắt đầu thực hiện.
hPrevInstance : là thông số để NULL
lpCmdLine:
nCmdShow: chỉ ra cửa sổ sẽ được hiển thị như thế nào (Minimumized, maximized, Hidden).

Hàm WinMain() sẽ kết thúc khi nó nhận được bản tin WM_QUIT.
Hàm WinMain() được xây dựng theo các bước sau:

Bước 1: Đăng kí cửa sổ (Register windows)
Trước khi chúng ta tạo cửa sổ, chúng ta cần phải đăng kí cửa sổ đó với HĐH Windows. Tất cả các cửa sổ cần phải được đăng kí.
HĐH Windows định nghĩa cửa sổ dưới dạng một cấu trúc WNDCLASS.Cấu trúc này chứa các thông số quy định các đặc tính cho cửa sổ(tên cửa sổ, màu background,..). Chúng ta sẽ xét cụ thể trong ví dụ.
Cuối cùng ta gọi hàm RegisterClass() để đăng kí cửa sổ với HĐH Windows.

Bước 2: Tạo cửa sổ
Để tạo một cửa sổ ta gọi hàm CreateWindow()

Code:

```
HWND CreateWindow( LPCTSTR lpClassName, LPCTSTR lpWindowName, DWORD dwStyle,
                                        int x, int y, int nWidth, int
nHeight, HWND hWndParent, HMENU hMenu,
                                        HINSTANCE hInstance, LPVOID lpParam
);
```

Parameter :
lpClassName: tên đăng kí với HĐH Windows
lpWindowName: tên sẽ hiển thị lên cửa sổ
dwStyle:
x,y: hoành độ, tung độ để hiển thị cửa sổ
nWidth: độ rộng của cửa sổ
nHeight: độ cao của cửa sổ
hWndParent: handle tới cửa sổ cha
hMenu: handle tới các menu
hInstance: handle tới các thể hiện của chương trình
lpParam:
Return :Handle tới cửa sổ vừa mới được tạo.

2.2 Message
- Hàm WinMain() tạo ra 1 vòng lặp thông điệp(message loop). Nó là vòng lặp vô hạn, chạy trong suốt vòng đời của ứng dụng. Message loop là 1 cấu trúc đợi và phát các sự kiện hoặc các message trong chương trình. HĐH Windows giao tiếp sử dụng các message.

- Message là giá trị integer chỉ ra một sự kiện cụ thể. VD: Khi chúng ta click vào button, thay đổi kích thước cửa sổ hoặc đóng ứng dụng,…v.v. Thì sẽ có rất nhiều message được tạo ra. Các message này có thể không được xử lí đồng thời, mà các message này sẽ được đưa vào 1 hàng đợi thông điệp (message queue) và đợi để xử lí lần lượt từng message một.

- Hàm GetMessage() được sử dụng để lấy các bản tin từ message queue.
- Hàm TranslateMessage() translate virtual-key message thành character message.
(HĐH Windows tạo ra các Virtual-key message khi người dùng ấn các phím trên key-board (nhưng không phải là giá trị character). Ứng dụng muốn lấy được message này thì cần phải có hàm để translate virtual-key message thành character message).
- Hàm DispatchMessage() dùng để phát message tới window produce.

2.3 Window Procedure
Code:

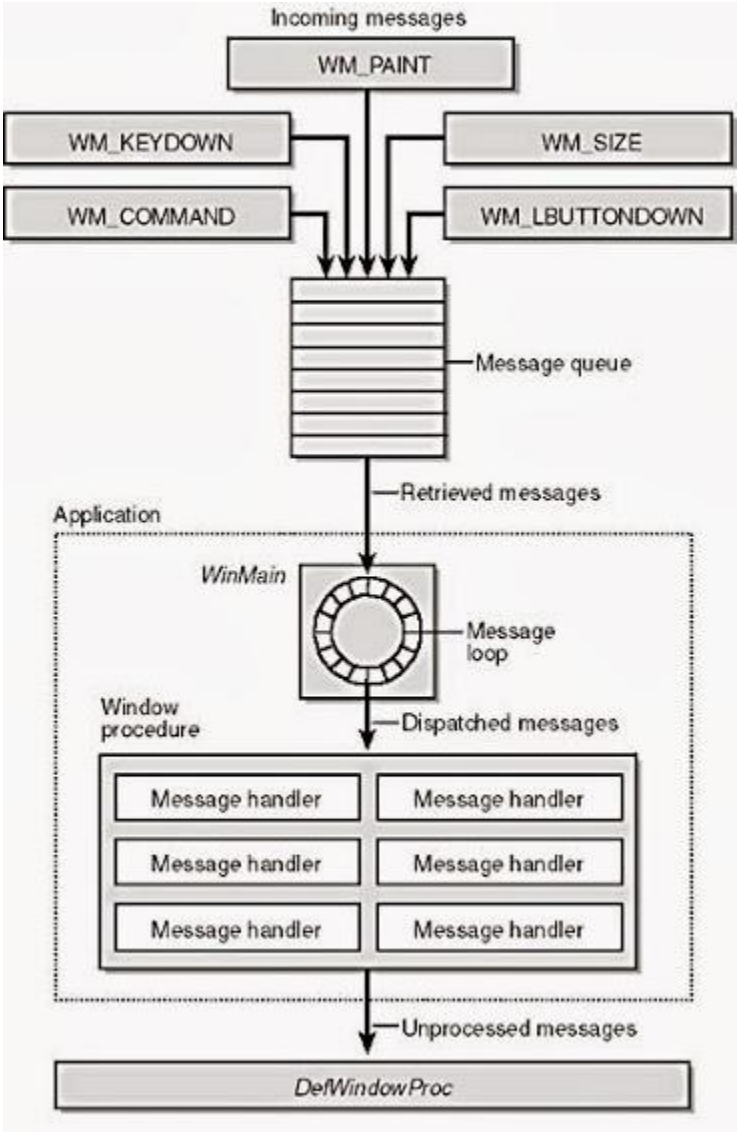
```
LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM
lParam);
```

- Hàm WindowProc() sẽ nhận và xử lí các message gửi đến. Các message nào không được xử lí trong hàm này sẽ được xử lí trong hàm DefWindowProc() của HĐH Windows.
- hwnd: handle to the windows
- uMsg:
- wParam, lParam: chứa các thông tin về message.

2.4 Các Message cơ bản
WM_CHAR Khi nhập 1 kí tự từ bàn phím
WM_COMMAND Khi lựa chọn các item trong popup menu
WM_CREAT Khi windows được tạo
WM_DESTROY Khi windows bị destroy
WM_LBUTTONDOWN Khi click chuột trái
WM_RBUTTONDOWN Khi click chuột phải
WM_MOUSEMOVE Khi di chuyển con trỏ chuột
WM_PAINT Khi windows được vẽ lại

WM_QUIT Khi close windows

Dưới đây là hình vẽ mô tả quá trình nhận và xử lí message của ứng dụng Win32



Ứng dụng win32 cơ bản

Tutorial này sẽ giúp bạn tạo ra 1 ứng dụng đơn giản là hiển thị 1 cửa sổ ra màn hình.

Code:

```
#include "windows.h"
// Gọi hàm xử lí message
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
// Hàm Winmain()
int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow )
```

```
{
  MSG  msg ;
  HWND hwnd;
  WNDCLASS wc;
  wc.style         = CS HREDRAW | CS VREDRAW;
  wc.cbClsExtra    = 0;
  wc.cbWndExtra    = 0;
  wc.lpszClassName = TEXT( "Window" );
  wc.hInstance     = hInstance ;
  wc.hbrBackground = GetSysColorBrush(COLOR 3DFACE);
  wc.lpszMenuName  = NULL;
  wc.lpfnWndProc   = WndProc;
  wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
  wc.hIcon         = LoadIcon(NULL, IDI_APPLICATION);
  RegisterClass(&wc);
  hwnd = CreateWindow( wc.lpszClassName, TEXT("Window"),
                       WS_OVERLAPPEDWINDOW | WS_VISIBLE,
                       100, 100, 250, 150, NULL, NULL, hInstance, NULL);
  ShowWindow(hwnd, nCmdShow);   // Hiển thị windows
  UpdateWindow(hwnd);   // Update windows

  while( GetMessage(&msg, NULL, 0, 0)) {
    DispatchMessage(&msg);
  }
  return (int) msg.wParam;
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam )
{
  switch(msg)
  {
    case WM_DESTROY:
      {
        PostQuitMessage(0);
        return 0;
      }
  }
  return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Giải thích:

Code:

```
#include "windows.h"
```

là header của của chương trình C.
Nó chứa việc gọi các hàm API, các macro và tất cả dữ liệu cơ bản.

Code:

```
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
```

Khai báo hàm xử lí message.

Code:

```
wc.style         =  CS_HREDRAW | CS_VREDRAW;
```

Đây là style của cửa sổ. CS_HREDRAW và CS_VREDRAW được thiết lập. Khi người dùng thay đổi kích thước cửa sổ thì cửa sổ sẽ được vẽ lại.

Code:

```
wc.cbClsExtra    = 0;
wc.cbWndExtra    = 0;
```

Ta không sử dụng các byte bổ sung (additional bytes). Nên ta đặt chúng bằng 0.

Code:

```
wc.lpszClassName = TEXT( "Window" );
```

Tên của class. Chúng ta sẽ dùng tên này để tạo cửa sổ. (Các bạn có thể thay đổi tên khác tùy theo ý các bạn).

Code:

```
wc.hInstance     = hInstance
wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
```

Đặt màu cho nền cửa sổ chính.

Code:

```
wc.lpszMenuName  = NULL;
```

Trong ví dụ này, chúng ta không thiết kế menu cho ứng dụng.

Code:

```
wc.lpfnWndProc   = WndProc;
```

Khai báo thủ tục xử lí message cho class.

Code:

```
wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
wc.hIcon         = LoadIcon(NULL, IDI_APPLICATION);
```

Đặt biểu tượng con trỏ và Icon trong ứng dụng.

Code:

```
RegisterClass(&wc);
```

Đăng kí lớp cửa sổ với Windows

Code:

```
ShowWindow(hwnd, nCmdShow);
UpdateWindow(hwnd);
```

Hiển thị và update cửa sổ.

Code:

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

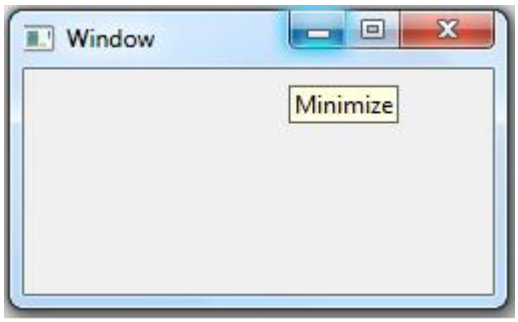
Đây là vòng lặp message. Sử dụng hàm GetMessage() để lấy message từ hàng đợi và gửi các message này cho các thủ tục xử lí message bằng hàm DispatchMessage().

Code:

```
switch(msg)
  {
    case WM_DESTROY:
      {
        PostQuitMessage(0);
        return 0;
      }
  }
  return DefWindowProc(hwnd, msg, wParam, lParam);
```

Đoạn code xử lý các message.
Khi người dùng close ứng dụng, bản tin WM_DESTROY gửi tới win procedure.
Hàm PostQuitMessage() sẽ gửi bản tin WM_QUIT tới hàng đợi thông điệp để chờ xử lí. Bản tin WM_QUIT sẽ được xử lí trong tại thủ tục mặc định của Windows.

Demo



Menu

Menubar được thấy rất nhiều trong các ứng dụng. Menubar được đặt bên dưới phần tên ứng dụng. Menubar có thể bao gồm rất nhiều các menu nhỏ (submenu hoặc menu items). Ví dụ dưới đây sẽ hướng dẫn bạn tạo một menubar gồm các submenu.

Code:

```
#include "windows.h"
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
void AddMenus(HWND);
//Define ID
#define IDM_FILE_NEW 1
#define IDM_FILE_OPEN 2
#define IDM_FILE_QUIT 3

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                        LPSTR lpCmdLine, int nCmdShow )
{
  MSG  msg ;
  WNDCLASS wc = {0};
  wc.lpszClassName = TEXT("Menu");
  wc.hInstance     = hInstance ;
  wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
  wc.lpfnWndProc   = WndProc ;
  wc.hCursor       = LoadCursor(0, IDC_ARROW);

  RegisterClass(&wc);
  CreateWindow( wc.lpszClassName, TEXT("Menu"),
                WS_OVERLAPPEDWINDOW | WS_VISIBLE,
                100, 100, 200, 150, 0, 0, hInstance, 0);

  while( GetMessage(&msg, NULL, 0, 0))
  {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }
  return (int) msg.wParam;
}
LRESULT CALLBACK WndProc( HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam )
```

```
{
  switch(msg)
  {
      case WM_CREATE:
          AddMenus(hwnd);
          break;

      case WM_COMMAND:
          switch(LOWORD(wParam))
           {
              case IDM_FILE_NEW:
              case IDM_FILE_OPEN:
                  Beep(50, 100);
                  break;
              case IDM FILE QUIT:
                  SendMessage(hwnd, WM_CLOSE, 0, 0);
                  break;
           }
           break;
     case WM_DESTROY:
          PostQuitMessage(0);
          break;

  }
  return DefWindowProc(hwnd, msg, wParam, lParam);
}
void AddMenus(HWND hwnd) {
  HMENU hMenubar;
  HMENU hMenu;

  hMenubar = CreateMenu();
  hMenu = CreateMenu();

  AppendMenu(hMenu, MF_STRING, IDM_FILE_NEW, TEXT("New"));
  AppendMenu(hMenu, MF_STRING, IDM_FILE_OPEN, TEXT("Open"));
  AppendMenu(hMenu, MF_SEPARATOR, 0, NULL);
  AppendMenu(hMenu, MF_STRING, IDM_FILE_QUIT, TEXT("Quit"));

  AppendMenu(hMenubar, MF_POPUP, (UINT_PTR)hMenu, TEXT("File"));
  SetMenu(hwnd, hMenubar);
}
```

Giải thích:

Menubar và các submenu được tạo bởi hàm CreateMenu(). Nếu tạo thành công hàm này sẽ trả về handle tới menu vừa mới được tạo.Để tạo ra từng menu item hoặc submenu ta sử dụng hàm

Code:

```
AppendMenu(hMenu, MF_STRING, IDM_FILE_NEW, TEXT("New"));
AppendMenu(hMenu, MF_STRING, IDM_FILE_OPEN, TEXT("Open"));
AppendMenu(hMenu, MF_SEPARATOR, 0, NULL);
AppendMenu(hMenu, MF_STRING, IDM_FILE_QUIT, TEXT("Quit"));
```

Đoạn code trên tạo menu gồm 3 item menu New,Open,Quit. Giữa Item Open và Quit có đường kẻ phân chia

Code:

```
AppendMenu(hMenu, MF_SEPARATOR, 0, NULL);
AppendMenu(hMenubar, MF_POPUP, (UINT_PTR)hMenu, TEXT("File"));
```

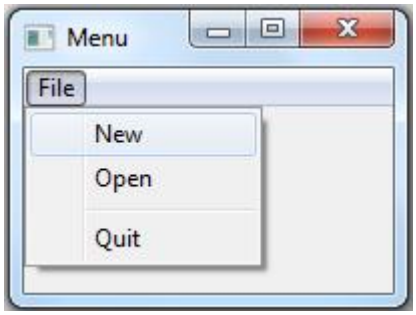
Gắn các menu item lên thanh menubar File.

Code:

```
SetMenu(hwnd, hMenubar);
```

Cuối cùng,ta dùng hàm SetMenu() để gắn menubar lên cửa sổ chính.

Demo



Popup Menu

Trong một số ứng dụng chúng ta thấy khi click chuột phải vào ứng dụng ta có thấy popup menu được hiển thị.

Code:

```
#include "windows.h"
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
#define IDM_FILE_NEW 1
#define IDM_FILE_OPEN 2
#define IDM_FILE_QUIT 3
int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                        LPSTR lpCmdLine, int nCmdShow )
{
  MSG  msg ;
  WNDCLASS wc = {0};
  wc.lpszClassName = TEXT("Application");
  wc.hInstance     = hInstance;
  wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
  wc.lpfnWndProc   = WndProc;
  wc.hCursor       = LoadCursor(0, IDC_ARROW);
```

```
  RegisterClass(&wc);
  CreateWindow( wc.lpszClassName, TEXT("Popup Menu"),
                WS_OVERLAPPEDWINDOW | WS_VISIBLE,
                100, 100, 200, 150, 0, 0, hInstance, 0);

  while( GetMessage(&msg, NULL, 0, 0))
  {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }
  return (int) msg.wParam;
}
LRESULT CALLBACK WndProc( HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam )
{
  HMENU hMenu;
  POINT point;

  switch(msg)
  {
    case WM_COMMAND:
        switch(LOWORD(wParam)) {
            case IDM_FILE_NEW:
            case IDM_FILE_OPEN:
                Beep(50, 100);
                break;
            case IDM_FILE_QUIT:
                SendMessage(hwnd, WM_CLOSE, 0, 0);
                break;
         }
        break;

    case WM_RBUTTONUP:
        point.x = LOWORD(lParam);
        point.y = HIWORD(lParam);
        hMenu = CreatePopupMenu();
        ClientToScreen(hwnd, &point);

        AppendMenu(hMenu, MF_STRING, IDM_FILE_NEW, TEXT("&New"));
        AppendMenu(hMenu, MF_STRING, IDM_FILE_OPEN, TEXT("&Open"));
        AppendMenu(hMenu, MF_SEPARATOR, 0, NULL);
        AppendMenu(hMenu, MF_STRING, IDM_FILE_QUIT, TEXT("&Quit"));

        TrackPopupMenu(hMenu, TPM_RIGHTBUTTON, point.x, point.y, 0, hwnd,
NULL);
        DestroyMenu(hMenu);
        break;

    case WM_DESTROY:
        PostQuitMessage(0);
        break;
  }
  return DefWindowProc(hwnd, msg, wParam, lParam);

}
```

Giải thích:
Khi chúng ta click chuột phải vào cửa sổ ứng dụng, message **WM_RBUTTONUP** được tạo ra.
Code:

```
POINT point;
point.x = LOWORD(lParam);
point.y = HIWORD(lParam);
```

point.x và point.y lấy tọa độ của con trỏ chuột.

Code:

```
hMenu = CreatePopupMenu();
```

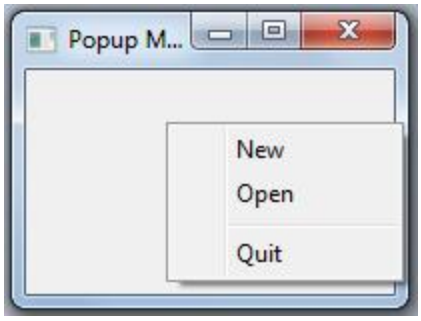
Để tạo một popup menu, chúng ta dùng hàm CreatePopupMenu()

Code:

```
TrackPopupMenu(hMenu, TPM_RIGHTBUTTON, point.x, point.y, 0, hwnd, NULL);
```

Hàm này hiển thị popup menu tại vị trí chúng ta click chuột phải trên ứng dụng.

Demo



Dialog

Cửa sổ Dialog hoặc các dialog là phần quan trọng của GUI. Dialog được sử dụng để lấy dữ liệu, chỉnh sửa dữ liệu, thay đổi cài đặt ứng dụng,..

VD1 : Dialog Box
Code:

```
#include "windows.h"

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK DialogProc(HWND, UINT, WPARAM, LPARAM);

void CreateDialogBox(HWND);
```

```
void RegisterDialogClass(HWND);

HINSTANCE ghInstance;
int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow )
{
  MSG  msg ;
  HWND hwnd;

  WNDCLASS wc = {0};
  wc.lpszClassName = TEXT( "Window" );
  wc.hInstance     = hInstance ;
  wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
  wc.lpfnWndProc   = WndProc;

  RegisterClass(&wc);
  hwnd = CreateWindow( wc.lpszClassName, TEXT("Window"),
               WS_OVERLAPPEDWINDOW | WS_VISIBLE,
               100, 100, 250, 150, NULL, NULL, hInstance, NULL);

  ghInstance = hInstance;
  while( GetMessage(&msg, NULL, 0, 0))
  {
    DispatchMessage(&msg);
  }
  return (int) msg.wParam;
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam )
{
  switch(msg)
  {
      case WM_CREATE:
          RegisterDialogClass(hwnd);
          CreateWindow(TEXT("button"), TEXT("Show dialog"),
              WS_VISIBLE | WS_CHILD ,
              20, 50, 95, 25,
              hwnd, (HMENU) 1, NULL, NULL);
          break;

      case WM_COMMAND:
          CreateDialogBox(hwnd);
          break;

      case WM_DESTROY:
      {
          PostQuitMessage(0);
          return 0;
      }
  }
  return DefWindowProc(hwnd, msg, wParam, lParam);
}
LRESULT CALLBACK DialogProc( HWND hwnd, UINT msg, WPARAM wParam, LPARAM
lParam )
{
  switch(msg)
```

```
  {
    case WM_CREATE:
        CreateWindow(TEXT("button"), TEXT("Ok"),
              WS_VISIBLE | WS_CHILD ,
              50, 50, 80, 25,
              hwnd, (HMENU) 1, NULL, NULL);
        break;
    case WM_COMMAND:
        DestroyWindow(hwnd);
        break;
    case WM_CLOSE:
        DestroyWindow(hwnd);
        break;
  }
  return (DefWindowProc(hwnd, msg, wParam, lParam));
}

void RegisterDialogClass(HWND hwnd)
{
  WNDCLASSEX wc = {0};
  wc.cbSize           = sizeof(WNDCLASSEX);
  wc.lpfnWndProc      = (WNDPROC) DialogProc;
  wc.hInstance        = ghInstance;
  wc.hbrBackground    = GetSysColorBrush(COLOR_3DFACE);
  wc.lpszClassName    = TEXT("DialogClass");
  RegisterClassEx(&wc);
}
void CreateDialogBox(HWND hwnd)
{
  CreateWindowEx(WS_EX_DLGMODALFRAME | WS_EX_TOPMOST,  TEXT("DialogClass"),
TEXT("Dialog Box"),
        WS_VISIBLE | WS_SYSMENU | WS_CAPTION , 100, 100, 200, 150,
        NULL, NULL, ghInstance,  NULL);
}
```

Giải thích:

Trong ví dụ trên, chúng ta khai báo 2 hàm xử lí message.

Code:

```
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK DialogProc(HWND, UINT, WPARAM, LPARAM);
```

- Hàm thứ nhất, xử lí các message phát sinh tại cửa sổ "Window" được tạo ban đầu ngay khi chạy ứng dụng.
- Khi ta click vào button "Show dialog" chương trình sẽ tạo ra cửa sổ thứ hai. Nếu người sử dụng thao tác trên cửa sổ "Dialog Box" thì hàm xử lí message thứ hai sẽ được gọi.

Demo

VD2:Colour Dialog Box

Code:

```
#include "windows.h"

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK PanelProc(HWND, UINT, WPARAM, LPARAM);

void RegisterPanel(void);
COLORREF ShowColorDialog(HWND);

COLORREF gColor = RGB(255, 255, 255);

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow )
{
  MSG  msg ;
  WNDCLASS wc = {0};
  wc.lpszClassName = TEXT( "Color dialog box" );
  wc.hInstance     = hInstance ;
  wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
  wc.lpfnWndProc   = WndProc ;

  RegisterClass(&wc);
  CreateWindow( wc.lpszClassName, TEXT("Color dialog box"),
                WS_OVERLAPPEDWINDOW | WS_VISIBLE,
                150, 150, 250, 200, 0, 0, hInstance, 0);

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

```
  return (int) msg.wParam;
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam )
{

  static HWND hwndPanel;


  switch(msg)
  {
        case WM_CREATE:
        {
          CreateWindow(TEXT("button"), TEXT("Color"),
               WS VISIBLE | WS CHILD ,
               20, 30, 80, 25,
               hwnd, (HMENU) 1, NULL, NULL);

          RegisterPanel();
          hwndPanel = CreateWindow(TEXT("Panel"), NULL,
                          WS_CHILD | WS_VISIBLE,
                          130, 30, 80, 80,
                          hwnd, (HMENU) 2, NULL, NULL);
          break;
        }

        case WM_COMMAND:
        {
           gColor = ShowColorDialog(hwnd);
           InvalidateRect(hwndPanel, NULL, TRUE);
           break;
        }

       case WM_DESTROY:
       {
          PostQuitMessage(0);
          break;
       }
  }

  return DefWindowProc(hwnd, msg, wParam, lParam);
}

LRESULT CALLBACK PanelProc( HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam
)
{
  HDC hdc;
  PAINTSTRUCT ps;
  RECT rect;

  switch(msg)
  {
    case WM_PAINT:
    {
        GetClientRect(hwnd, &rect);
        hdc = BeginPaint(hwnd, &ps);
```

```
        SetBkColor(hdc, gColor);
        ExtTextOut(hdc, 0, 0, ETO_OPAQUE, &rect, TEXT(""), 0, NULL);
        EndPaint(hwnd, &ps);
        break;
    }
  }
  return DefWindowProc(hwnd, msg, wParam, lParam);
}


COLORREF ShowColorDialog(HWND hwnd) {

  CHOOSECOLOR cc;
  static COLORREF crCustClr[16];

  ZeroMemory(&cc, sizeof(cc));
  cc.lStructSize = sizeof(cc);
  cc.hwndOwner = hwnd;
  cc.lpCustColors = (LPDWORD) crCustClr;
  cc.rgbResult = RGB(0, 255, 0);
  cc.Flags = CC_FULLOPEN | CC_RGBINIT;
  ChooseColor(&cc);

  return cc.rgbResult;
}

void RegisterPanel(void) {

  WNDCLASS rwc = {0};
  rwc.lpszClassName = TEXT( "Panel" );
  rwc.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
  rwc.lpfnWndProc   = PanelProc;
  RegisterClass(&rwc);
}
```

Giải thích:
Trong ví dụ này, chúng ta có 1 button và 1 child window. Màu của child window được khởi tạo là màu trắng. Chúng ta có thể thay đổi màu của child window bằng cách click vào button "Colour" và chọn màu tùy ý.

Code:

```
COLORREF gColor = RGB(255, 255, 255);
```

Sử dụng macro RGB(255, 255, 255) để khởi tạo màu cho child window.

Code:

```
gColor = ShowColorDialog(hwnd);
```

Hàm này trả về giá trị màu được chọn.

Code:

```
CHOOSECOLOR cc;
```

Để tạo Color Dialog Box ta cần định nghĩa cấu trúc CHOOSECOLOR.

Code:

```
cc.rgbResult = RGB(0, 255, 0);
cc.Flags = CC_FULLOPEN | CC_RGBINIT;
```

Khởi tạo màu trên Dialog Box.

Code:

```
ChooseColor(&cc);
```

Sau khi người dùng chọn màu trên Dialog Box và nhấn OK. Hàm ChooseColor(&cc) sẽ trả về giá trị khác không và đồng thời giá trị màu được chọn được gán cho thông số rgbResult.

Hàm ShowColorDialog(HWND hwnd) sẽ trả về giá trị màu đã được chọn.

Code:

```
return cc.rgbResult;
```

Code:

```
InvalidateRect (hwndPanel, NULL, TRUE);
```

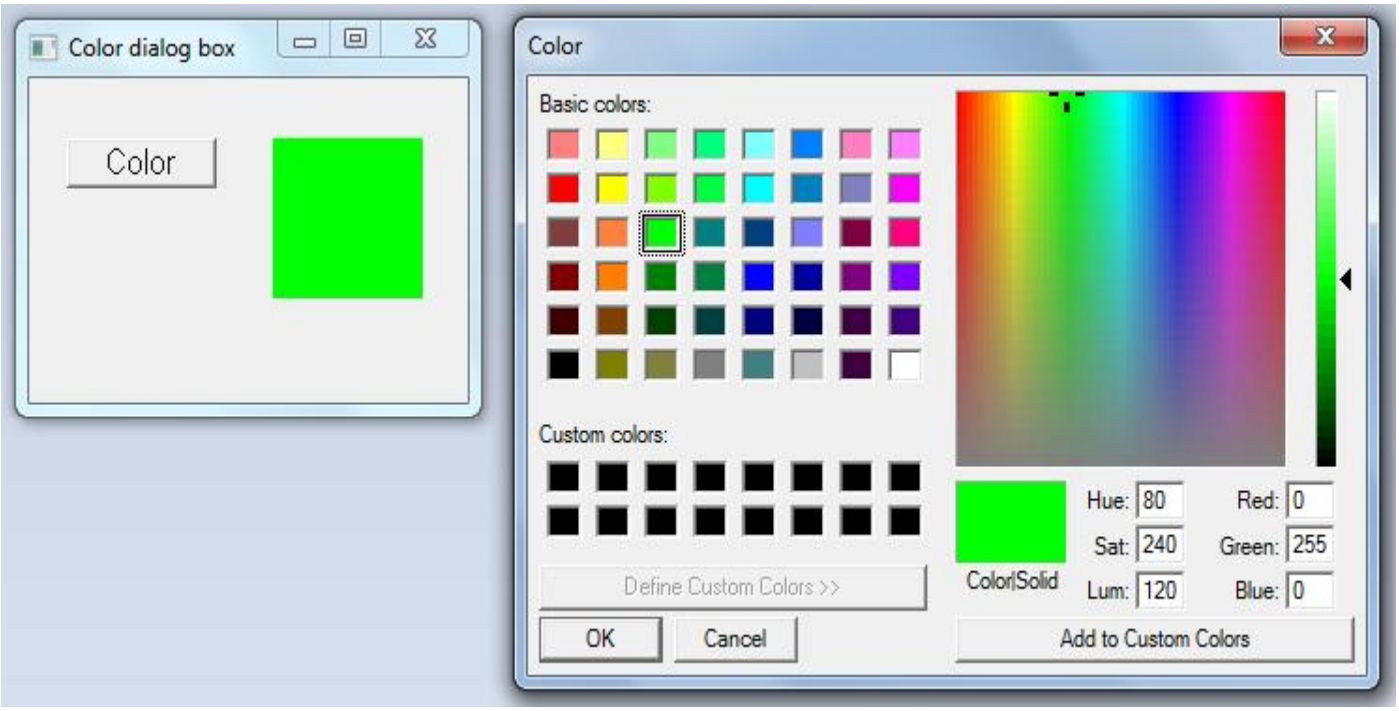
Sau khi lấy được màu đã chọn. Chúng ta gọi hàm InvalidateRect(), hàm này sẽ gửi tới child window message **WM_PAINT**.

Code:

```
SetBkColor(hdc, gColor);
ExtTextOut(hdc, 0, 0, ETO_OPAQUE, &rect, TEXT(""), 0, NULL);
```

Hàm SetBkColor() cài đặt màu của background. Hàm ExtTextOut() có chức năng hiển thị văn bản và màu lên device context. Ở đây ETO_OPAQUE là hằng số đã được định nghĩa, khia giá trị này được khai báo thì hàm ExtTextOut() chỉ hiển thị màu lên background.

Demo



Control Static

Code:

```
#include "windows.h"

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);


int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PWSTR lpCmdLine, int nCmdShow)
{
  MSG  msg;
  WNDCLASSW wc = {0};
  wc.lpszClassName = L"Static Control";
  wc.hInstance     = hInstance;
  wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
  wc.lpfnWndProc   = WndProc;
  wc.hCursor       = LoadCursor(0,IDC_ARROW);


  RegisterClassW(&wc);
  CreateWindowW(wc.lpszClassName, L"Introduce myself",
                WS_OVERLAPPEDWINDOW | WS_VISIBLE,
                100, 100, 420, 270, 0, 0, hInstance, 0);
```

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

  return (int) msg.wParam;
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
    WPARAM wParam, LPARAM lParam)
{
  static wchar_t *lyrics =  L"Hi all.\n\
My name's Viet\n\
I come from Thai Nguyen City\n\
06/2012 : I graduated from HN University of Technology \n\
08/2012 : I came to Fsoft";

  switch(msg)
  {
      case WM_CREATE:

        CreateWindowW(L"STATIC", lyrics,
            WS_CHILD | WS_VISIBLE | SS_LEFT,
            20, 20, 400, 230,
            hwnd, (HMENU) 1, NULL, NULL);
        break;

      case WM_DESTROY:

        PostQuitMessage(0);
        break;
  }

  return DefWindowProcW(hwnd, msg, wParam, lParam);
}
```

Giải thích:

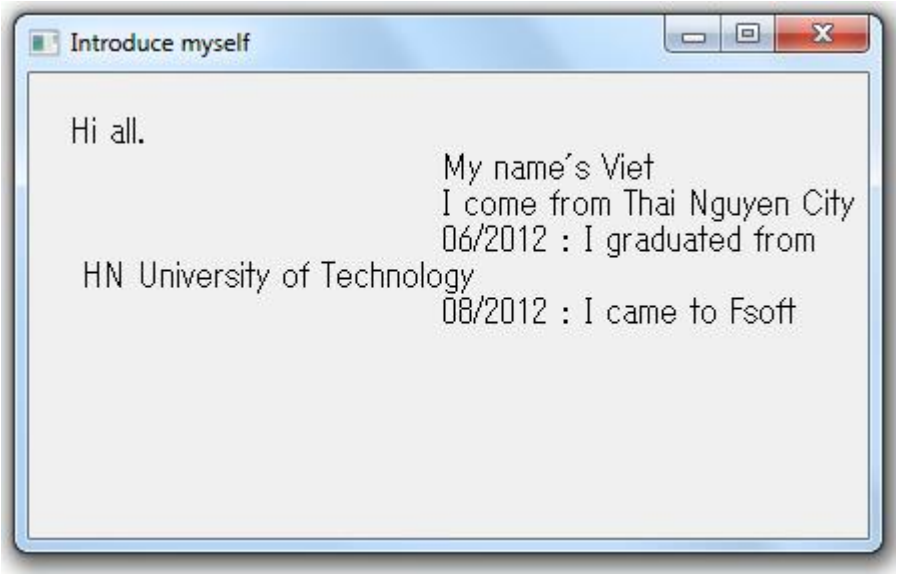
Bài viết này giúp các bạn tạo ra 1 cửa sổ static để hiển thị text lên đó.

Code:

```
CreateWindowW(L"STATIC", lyrics,
    WS_CHILD | WS_VISIBLE | SS_LEFT,
    20, 20, 300, 230,
    hwnd, (HMENU) 1, NULL, NULL);
```

Hàm CreateWindowW giúp tạo cửa sổ (đối tượng) static. Nếu thay "STATIC" bằng "Button" thì chương trình sẽ tạo ra 1 nút bấm.

Demo



Button

Code:

```
#include "windows.h"

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);


int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                        LPSTR lpCmdLine, int nCmdShow )
{
  MSG  msg ;
  WNDCLASS wc = {0};
  wc.lpszClassName = TEXT( "Buttons" );
  wc.hInstance     = hInstance ;
  wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
  wc.lpfnWndProc   = WndProc ;
  wc.hCursor       = LoadCursor(0, IDC_ARROW);


  RegisterClass(&wc);
  CreateWindow( wc.lpszClassName, TEXT("Buttons"),
                WS_OVERLAPPEDWINDOW | WS_VISIBLE,
                150, 150, 230, 150, 0, 0, hInstance, 0);

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

```
  return (int) msg.wParam;
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam )
{


  switch(msg)
  {
        case WM CREATE:
        {
            CreateWindow(TEXT("button"), TEXT("Beep"),
                            WS_VISIBLE | WS_CHILD ,
                            20, 50, 80, 25,
                            hwnd, (HMENU) 1, NULL, NULL);

             CreateWindow(TEXT("button"), TEXT("Quit"),
                            WS_VISIBLE | WS_CHILD ,
                            120, 50, 80, 25,
                            hwnd, (HMENU) 2, NULL, NULL);
             break;
        }

       case WM_COMMAND:
       {
            if (LOWORD(wParam) == 1)
            {
                Beep(40, 50);
            }

            if (LOWORD(wParam) == 2) {
               PostQuitMessage(0);
            }

            break;
        }

       case WM_DESTROY:
       {
          PostQuitMessage(0);
          break;
       }
  }
  return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Giải thích:

Trong VD trên chúng ta xây dựng cửa sổ gồm 2 button. Ta sử dụng hàm CreateWindow() để tạo Button.

Code:

```
CreateWindow(TEXT("button"), TEXT("Beep"),
                            WS_VISIBLE | WS_CHILD ,
                            20, 50, 80, 25,
                            hwnd, (HMENU) 1, NULL, NULL);
```

```
CreateWindow(TEXT("button"), TEXT("Quit"),
                             WS_VISIBLE | WS_CHILD ,
                             120, 50, 80, 25,
                             hwnd, (HMENU) 2, NULL, NULL);
```

Button thứ nhất có nhãn là "Beep" và ID=1.
Button thứ hai có nhãn là "Quit" và ID=2.
Khi ta bấm button thì message WM_COMMAND được phát đi. Hàm WndProc() sẽ xử lí message này
Code:

```
case WM_COMMAND:
      {
           if (LOWORD(wParam) == 1)
           {
               Beep(40, 50);
           }

           if (LOWORD(wParam) == 2)
           {
              PostQuitMessage(0);
           }

           break;
       }
```

LOWORD(wParam) lấy word thấp của tham số wParam. LOWORD(wParam)= ID của button. Nhờ đó, mà chương trình phân biệt được khi người sử dụng bấm vào button nào.
Khi bấm vào button "Beep" hàm Beep(40, 50) được gọi (tiếng Beep).
Khi bấm vào button "Quit" hàm PostQuitMessage(0) sẽ gửi message tới Window yêu cầu đóng ứng dụng.

Demo

CheckBox

Code:

```
#include "windows.h"

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

TCHAR title[] = TEXT("Check Box");

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                        LPSTR lpCmdLine, int nCmdShow )
{
  MSG  msg ;
  WNDCLASS wc = {0};
  wc.lpszClassName = TEXT( "Check Box" );
  wc.hInstance     = hInstance ;
  wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
  wc.lpfnWndProc   = WndProc ;
  wc.hCursor       = LoadCursor(0, IDC_ARROW);


  RegisterClass(&wc);
  CreateWindow( wc.lpszClassName, title,
                WS_OVERLAPPEDWINDOW | WS_VISIBLE,
                150, 150, 230, 150, 0, 0, hInstance, 0);

  while( GetMessage(&msg, NULL, 0, 0))
  {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }
  return (int) msg.wParam;
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam )
{

  switch(msg)
  {
      case WM_CREATE:
      {
            CreateWindow(TEXT("button"), TEXT("Show Title"),
                      WS_VISIBLE | WS_CHILD | BS_CHECKBOX,
                      20, 20, 185, 35,
                      hwnd, (HMENU) 1, ((LPCREATESTRUCT)lParam)->hInstance,
NULL);
            CheckDlgButton(hwnd, 1, BST_CHECKED);
            break;
      }

      case WM_COMMAND:
      {
           BOOL checked = IsDlgButtonChecked(hwnd, 1);
            if (checked)
            {
                CheckDlgButton(hwnd, 1, BST_UNCHECKED);
                 SetWindowText(hwnd, TEXT(""));
            }
```

```
            else
             {
                  CheckDlgButton(hwnd, 1, BST_CHECKED);
                  SetWindowText(hwnd, title);
             }
             break;
       }

      case WM_DESTROY:
      {
            PostQuitMessage(0);
            break;
      }
  }
  return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Giải thích:
Trong bài viết này, tilte của ứng dụng sẽ được ẩn hoặc hiện phụ thuộc vào trạng thái của checkbox.
Code:

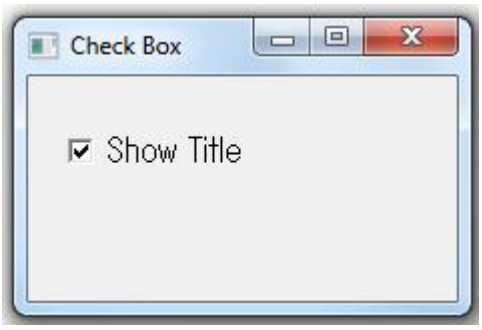
```
CreateWindow(TEXT("button"), TEXT("Show Title"),
                     WS_VISIBLE | WS_CHILD | BS_CHECKBOX,
                     20, 20, 185, 35,
                     hwnd, (HMENU) 1, ((LPCREATESTRUCT)lParam)->hInstance,
NULL);
```

Để tạo checkbox ta sử dụng hàm CreateWindow() để tạo button loại checkbox. Ta chọn hằng số BS_CHECKBOX.
Code:

```
CheckDlgButton(hwnd, 1, BST_CHECKED);
```

Hàm CheckDlgButton() khởi tạo trạng thái ban đầu của checkbox. Hằng số BST_CHECKED chỉ ra checkbox đã được check.

Demo



Group box và radio button

Ở bài này,chúng ta sẽ đi tìm hiểu về group box và radio button.
• Group box là 1 đường bao hình chữ nhật bao quanh các 1 bộ các control. Control thường các các radio button. Group box được đánh nhãn để mô tả về control này. Group control được dùng để nhóm các control cùng loại.
• Radio button là 1 loại button đặc biệt có thể lựa chọn nhưng không xóa được. Nó cho phép người dùng lựa chọn 1 trong nhóm các option.
Code:

```
#include "windows.h"

#define ID_BLUE 1
#define ID_YELLOW 2
#define ID_ORANGE 3

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

HINSTANCE g_hinst;
COLORREF g_color;


int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow)
{
  HWND hwnd;
  MSG  msg ;
  WNDCLASS wc = {0};
  wc.lpszClassName = TEXT("GroupBox");
  wc.hInstance     = hInstance ;
  wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
  wc.lpfnWndProc   = WndProc ;
  wc.hCursor       = LoadCursor(0,IDC_ARROW);

  g_hinst = hInstance;

  RegisterClass(&wc);
  hwnd = CreateWindow(wc.lpszClassName, TEXT("GroupBox"),
               WS_OVERLAPPEDWINDOW | WS_VISIBLE,
               100, 100, 300, 170, 0, 0, hInstance, 0);


  while( GetMessage(&msg, NULL, 0, 0)) {
    DispatchMessage(&msg);
  }
  return (int) msg.wParam;
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam )
{
  HDC hdc;
  PAINTSTRUCT ps;
  HBRUSH hBrush, holdBrush;
  HPEN hPen, holdPen;

  switch(msg)
```

```
  {
      case WM_CREATE:
          CreateWindow(TEXT("button"), TEXT("Choose Color"),
                WS_CHILD | WS_VISIBLE | BS_GROUPBOX,
                10, 10, 120, 110, hwnd, (HMENU) 0, g hinst, NULL);
          CreateWindow(TEXT("button"), TEXT("Blue"),
                WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON,
                20, 30, 100, 30, hwnd, (HMENU)ID_BLUE , g_hinst, NULL);
          CreateWindow(TEXT("button"), TEXT("Yellow"),
                WS CHILD | WS VISIBLE | BS AUTORADIOBUTTON,
                20, 55, 100, 30, hwnd, (HMENU)ID_YELLOW , g_hinst, NULL);
          CreateWindow(TEXT("button"), TEXT("Orange"),
                WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON,
                20, 80, 100, 30, hwnd, (HMENU)ID_ORANGE , g_hinst, NULL);

          break;

      case WM_COMMAND:
           if (HIWORD(wParam) == BN_CLICKED) {
               switch (LOWORD(wParam)) {
                    case ID_BLUE:
                        g_color = RGB(0, 76, 255);
                        break;
                    case ID_YELLOW:
                        g_color = RGB(255, 255, 0);
                        break;
                    case ID_ORANGE:
                        g_color = RGB(255, 123, 0);
                        break;
                }
               InvalidateRect(hwnd, NULL, TRUE);
           }
          break;

      case WM_PAINT:
          hdc = BeginPaint(hwnd, &ps);
          hBrush = CreateSolidBrush(g_color);
          hPen = CreatePen(PS_NULL, 1, RGB(0, 0, 0));
          holdPen = (HPEN)SelectObject(hdc, hPen);
          holdBrush = (HBRUSH) SelectObject(hdc, hBrush);

          Rectangle(hdc, 160, 20, 260, 120);

          SelectObject(hdc, holdBrush);
          SelectObject(hdc, holdPen);
          DeleteObject(hPen);
          DeleteObject(hBrush);
          EndPaint(hwnd, &ps);
          break;

      case WM_DESTROY:
          PostQuitMessage(0);
          break;
  }
  return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Giải thích :
Trong bài viết này, chúng ta có 1 group box gồm 3 radio button. Ta có thể lựa chọn background cho hình chữ nhật bên phải bằng cách click vào radio button.
Code:

```
CreateWindow(TEXT("button"), TEXT("Choose Color"),
       WS_CHILD | WS_VISIBLE | BS_GROUPBOX,
       10, 10, 120, 110, hwnd, (HMENU) 0, g_hinst, NULL);
```

Group box được tạo với thông số BS_GROUPBOX.

Code:

```
CreateWindow(TEXT("button"), TEXT("Blue"),
       WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON,
       20, 30, 100, 30, hwnd, (HMENU)ID_BLUE , g_hinst, NULL);
```

Radio button "Blue" được tạo với thông số BS_AUTORADIOBUTTON.

Tương tự các hàm tiếp theo tạo ra các radio button "Yellow" và "Orange".

Code:

```
case ID_BLUE:
     g_color = RGB(0, 76, 255);
     break;
```

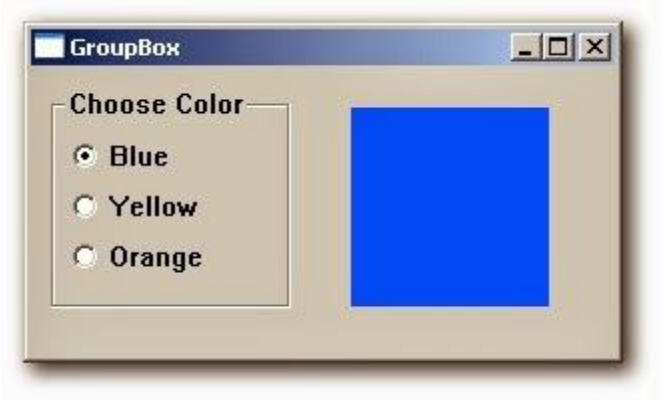
Nếu button "Blue" được chọn thì biến g_color sẽ được fill màu blue. Biến g_color được dùng để tạo brush để fill màu cho hình chữ nhật bên phải.

Code:

```
InvalidateRect(hwnd, NULL, TRUE);
```

Hàm này invalidate hình chữ nhật (trong trường hợp này là toàn bộ window), khiến cho toàn bộ window được vẽ lại.
Trong case WM_PAINT sẽ vẽ hình chữ nhật.

Demo



Trackbar

Trackbar bao gồm 1 slider + 1 tick mark. Chúng ta có thể sử dụng chuột hoặc bàn phím để di chuyển thanh slider. Trackbar được dùng để lựa chọn 1 giá trị xác định trong 1 dải giá trị liên tục.
Chúng ta tạo trackbar bằng 3 static text control. 2 đối tượng static được đặt ở bên trái và bên phải của slider. Khi ta kéo slider trượt thì text của đối tượng static còn lại sẽ thay đổi.
Code:

```
#include "windows.h"
#include "commctrl.h"

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
void CreateControls(HWND hwnd);
void UpdateLabel(void);

HWND hTrack;
HWND hlbl;

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
  PWSTR lpCmdLine, int nCmdShow)
{
  HWND hwnd;
  MSG  msg ;

  WNDCLASSW wc = {0};
  wc.lpszClassName = L"Trackbar";
  wc.hInstance     = hInstance ;
  wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
  wc.lpfnWndProc   = WndProc ;
  wc.hCursor       = LoadCursor(0,IDC_ARROW);

  RegisterClassW(&wc);
  hwnd = CreateWindowW(wc.lpszClassName, L"Trackbar",
    WS_OVERLAPPEDWINDOW | WS_VISIBLE, 100, 100, 350, 180, 0, 0, hInstance,
0);

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

  return (int) msg.wParam;
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
  WPARAM wParam, LPARAM lParam )
{
  switch(msg)
  {
     case WM_CREATE:
       CreateControls(hwnd);
       break;
```

```
    case WM_HSCROLL:
      UpdateLabel();
      break;

    case WM DESTROY:
      PostQuitMessage(0);
      break;
  }

  return DefWindowProcW(hwnd, msg, wParam, lParam);
}

void CreateControls(HWND hwnd)
{
  HWND hLeftLabel = CreateWindowW(L"STATIC", L"0",
    WS_CHILD | WS_VISIBLE, 0, 0, 10, 30, hwnd, (HMENU)1, NULL, NULL);

  HWND hRightLabel = CreateWindowW(L"STATIC", L"100",
    WS_CHILD | WS_VISIBLE, 0, 0, 30, 30, hwnd, (HMENU)2, NULL, NULL);

  hlbl = CreateWindowW(L"STATIC", L"0", WS_CHILD | WS_VISIBLE,
    270, 20, 30, 30, hwnd, (HMENU)3, NULL, NULL);

  INITCOMMONCONTROLSEX icex;

  icex.dwSize = sizeof(INITCOMMONCONTROLSEX);
  icex.dwICC  = ICC_LISTVIEW_CLASSES;
  InitCommonControlsEx(&icex);

  hTrack = CreateWindowW(L"msctls_trackbar32", L"Trackbar Control",
      WS_CHILD | WS_VISIBLE | TBS_AUTOTICKS,
      20, 20, 170, 30, hwnd, (HMENU) 3, NULL, NULL);

  SendMessageW(hTrack, TBM_SETRANGE,  TRUE, MAKELONG(0, 100));
  SendMessageW(hTrack, TBM_SETPAGESIZE, 0,  10);
  SendMessageW(hTrack, TBM_SETTICFREQ, 10, 0);
  SendMessageW(hTrack, TBM_SETPOS, FALSE, 0);
  SendMessageW(hTrack, TBM_SETBUDDY, TRUE, (LPARAM) hLeftLabel);
  SendMessageW(hTrack, TBM_SETBUDDY, FALSE, (LPARAM) hRightLabel);
}

void UpdateLabel(void)
{
  LRESULT pos = SendMessageW(hTrack, TBM_GETPOS, 0, 0);
  wchar_t buf[4];
  wsprintfW(buf, L"%ld", pos);

  SetWindowTextW(hlbl, buf);
}
```

Giải thích:

Code:

```
HWND hLeftLabel = CreateWindowW(L"STATIC", L"0",
  WS_CHILD | WS_VISIBLE, 0, 0, 10, 30, hwnd, (HMENU)1, NULL, NULL);
```

```
HWND hRightLabel = CreateWindowW(L"STATIC", L"100",
  WS_CHILD | WS_VISIBLE, 0, 0, 30, 30, hwnd, (HMENU)2, NULL, NULL);

hlbl = CreateWindowW(L"STATIC", L"0", WS_CHILD | WS_VISIBLE,
  270, 20, 30, 30, hwnd, (HMENU)3, NULL, NULL);
```

3 static control sẽ được tạo. 2 static đầu tiên hiển thị giá trị min,max của trackbar. Static còn lại sẽ hiển thị giá trị được chọn.

Code:

```
INITCOMMONCONTROLSEX icex;
icex.dwSize = sizeof(INITCOMMONCONTROLSEX);
icex.dwICC  = ICC_LISTVIEW_CLASSES;
InitCommonControlsEx(&icex);
```

Nếu chúng ta muốn sử dụng 1 trong những common control, thì cần phải load common control DLL (Comctl32.dll) và đăng kí common control class lấy từ DLL.
Hàm InitCommonControlsEx() phải được khai báo trước khi tạo common control.

Code:

```
hTrack = CreateWindowW(L"msctls_trackbar32", L"Trackbar Control",
    WS_CHILD | WS_VISIBLE | TBS_AUTOTICKS,
    20, 20, 170, 30, hwnd, (HMENU) 3, NULL, NULL);
```

Trackbar đã được tạo, TBS_AUTOTICKS tạo ra thang chia độ trên trackbar.

Code:

```
SendMessageW(hTrack, TBM_SETRANGE,  TRUE, MAKELONG(0, 100));
SendMessageW(hTrack, TBM_SETPAGESIZE, 0,  10);
SendMessageW(hTrack, TBM_SETTICFREQ, 10, 0);
SendMessageW(hTrack, TBM_SETPOS, FALSE, 0);
```

Ta cần gửi message đến trackbar để tạo thành 1 trackbar hoàn chỉnh.
TBM_SETRANGE dùng để đặt dải giá trị cho trackbar (ở đây là 100).
TBM_SETPAGESIZE dùng để đặt bước nhảy khi ta click chuột, thanh slider sẽ trượt đi 1 khoảng là 10.
TBM_SETTICFREQ dùng để đặt độ dài mỗi vạch trên trackbar.
TBM_SETPOS dùng để thiết lập vị trí ban đầu cho trackbar.

Code:

```
SendMessageW(hTrack, TBM_SETBUDDY, TRUE, (LPARAM) hLeftLabel);
SendMessageW(hTrack, TBM_SETBUDDY, FALSE, (LPARAM) hRightLabel);
```

TBM_SETBUDDY dùng để đặt control static "0" và "100" vào bên trái và bên phải của trackbar. Nếu TRUE : bên trái, FALSE : bên phải.

Code:

```
case WM_HSCROLL:
  UpdateLabel();
  break;
```

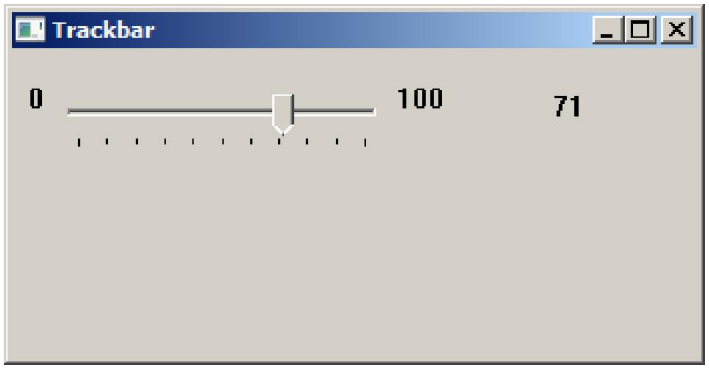
Khi ta di chuyển thanh slider thì hàm WndProc() sẽ nhận được message WM_HSCROLL (trong trường hợp trackbar nằm ngang).

Code:

```
void UpdateLabel(void)
{
  LRESULT pos = SendMessageW(hTrack, TBM_GETPOS, 0, 0);
  wchar_t buf[4];
  wsprintfW(buf, L"%ld", pos);
  SetWindowTextW(hlbl, buf);
}
```

Trong hàm UpdateLabel(), ta lấy vị trí hiện tại của slider bằng cách gửi message TMB_GETPOS tới trackbar. Giá trị trả về của hàm SendMessageW() sẽ được convert sang dạng text và được set lên trackbar bằng hàm SetWindowTextW(hlbl, buf).

Demo



ComboBox

Combo box là tổ hợp của edit box hoặc satic text và list box. Combo box được dùng để lựa chọn 1 item từ 1 list option.

Code:

```
#include "windows.h"

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
```

```
HINSTANCE g_hinst;

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow)
{
  HWND hwnd;
  MSG  msg ;
  WNDCLASS wc = {0};
  wc.lpszClassName = TEXT("Application");
  wc.hInstance     = hInstance ;
  wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
  wc.lpfnWndProc   = WndProc ;
  wc.hCursor       = LoadCursor(0,IDC_ARROW);

  g hinst = hInstance;

  RegisterClass(&wc);
  hwnd = CreateWindow(wc.lpszClassName, TEXT("Combo Box"),
               WS_OVERLAPPEDWINDOW | WS_VISIBLE,
               100, 100, 270, 170, 0, 0, hInstance, 0);


  while( GetMessage(&msg, NULL, 0, 0)) {
    DispatchMessage(&msg);
  }
  return (int) msg.wParam;
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam )
{

  static HWND hwndCombo, hwndStatic;
  const TCHAR *items[] = { TEXT("FreeBSD"), TEXT("OpenBSD"),
      TEXT("Ubuntu"), TEXT("Solaris") };
  int i;
  LRESULT sel = 0;

  switch(msg)
  {
      case WM_CREATE:
            hwndCombo = CreateWindow(TEXT("combobox"), NULL,
                  WS_CHILD | WS_VISIBLE | CBS_DROPDOWN,
                  10, 10, 120, 110, hwnd, NULL, g_hinst, NULL);

            CreateWindow(TEXT("button"), TEXT("Drop down"),
                  WS_CHILD | WS_VISIBLE,
                  150, 10, 90, 25, hwnd, (HMENU)1, g_hinst, NULL);

            hwndStatic = CreateWindow(TEXT("static"), TEXT(""),
                  WS_CHILD | WS_VISIBLE,
                  150, 80, 90, 25, hwnd, NULL, g_hinst, NULL);

            for ( i = 0; i < 4; i++ ) {
                SendMessage(hwndCombo, CB_ADDSTRING, 0, (LPARAM) items[i]);
            }
```

```
             break;

      case WM_COMMAND:
           if (HIWORD(wParam) == BN_CLICKED) {
                SendMessage(hwndCombo, CB SHOWDROPDOWN, (WPARAM) TRUE, 0);
           }

           if ( HIWORD(wParam) == CBN_SELCHANGE) {
                sel = SendMessage(hwndCombo, CB_GETCURSEL, 0, 0);
                SetWindowText(hwndStatic, items[sel]);
                SetFocus(hwnd);
           }

           break;

      case WM_DESTROY:
          PostQuitMessage(0);
          break;
  }
  return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Giải thích:
Trong ví dụ này, ta đặt 3 control trong window : combo box , button và static text. Static text hiển thị các item được lựa chọn từ combo box. Click button để mở combo box.
Code:

```
hwndCombo = CreateWindow(TEXT("combobox"), NULL,
     WS_CHILD | WS_VISIBLE | CBS_DROPDOWN,
     10, 10, 120, 110, hwnd, NULL, g_hinst, NULL);
```

Để tạo combo box, ta sử dụng "combobox" string. Ta sử dụng CBS_DROPDOWN flag.

Code:

```
for ( i = 0; i < 4; i++ ) {
     SendMessage(hwndCombo, CB_ADDSTRING, 0, (LPARAM) items[i]);
 }
```

Ta fill combo box với các item. Để add 1 string tới combo box, ta gửi message CB_ADDSTRING tới combo box.
Nếu ta lựa chọn 1 item từ combo box,hàm WndProc(..,..) sẽ nhận message WM_COMMAND và CBN_SELCHANGE là phần word cao của tham số wParam.

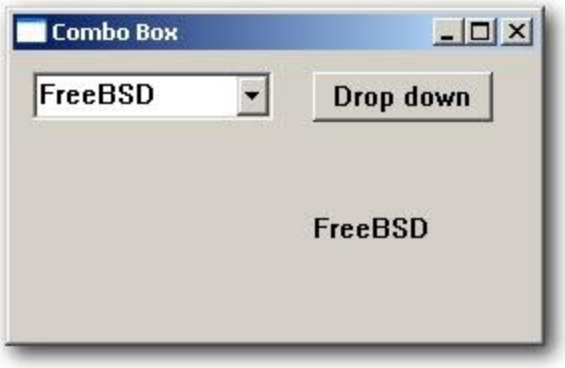
Code:

```
sel = SendMessage(hwndCombo, CB_GETCURSEL, 0, 0);
SetWindowText(hwndStatic, items[sel]);
SetFocus(hwnd);
```

Khi ta send message CB_GETCURSEL tới commbo box. Giá trị trả về của hàm SendMessage( ) là chỉ số của item được lựa chọn. Ta sẽ set string được lựa chọn từ combo box cho static

text bằng SetWindowText(hwndStatic, items[sel]); . Combo box đã được focus. Để loại bỏ focus của combo box sau khi lựa chọn item, ta dùng hàm SetFocus(hwnd).

Demo



Progress Bar

Progress bar là cửa sổ cho người dùng biết quá trình (cài đặt phần mềm, download tài liệu, quét virut,...) đã thực hiện được bao nhiêu % và còn lại bao nhiêu %.

Code:

```
#include "windows.h"
#include "commctrl.h"

#define ID_BUTTON 1
#define ID_TIMER 2

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
HINSTANCE g_hinst;


int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow)
{

  HWND hwnd;
  MSG  msg ;
  WNDCLASS wc = {0};
  wc.lpszClassName = TEXT("Application");
```

```
  wc.hInstance     = hInstance ;
  wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
  wc.lpfnWndProc   = WndProc ;
  wc.hCursor       = LoadCursor(0,IDC_ARROW);

  g_hinst = hInstance;

  RegisterClass(&wc);
  hwnd = CreateWindow(wc.lpszClassName, TEXT("Progress bar"),
               WS OVERLAPPEDWINDOW | WS VISIBLE,
               100, 100, 260, 170, 0, 0, hInstance, 0);


  while( GetMessage(&msg, NULL, 0, 0)) {
    DispatchMessage(&msg);
  }
  return (int) msg.wParam;
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam )
{

  static HWND hwndPrgBar;
  static int i = 1;

  INITCOMMONCONTROLSEX InitCtrlEx;

  InitCtrlEx.dwSize = sizeof(INITCOMMONCONTROLSEX);
  InitCtrlEx.dwICC  = ICC_PROGRESS_CLASS;
  InitCommonControlsEx(&InitCtrlEx);

  switch(msg)
  {
      case WM_CREATE:
          hwndPrgBar = CreateWindowEx(0, PROGRESS_CLASS, NULL,
                WS_CHILD | WS_VISIBLE | PBS_SMOOTH,
                30, 20, 190, 25, hwnd, NULL, g_hinst, NULL);

          CreateWindow(TEXT("button"), TEXT("Start"),
                WS_CHILD | WS_VISIBLE,
                85, 90, 80, 25, hwnd, (HMENU) 1, g_hinst, NULL);

          SendMessage(hwndPrgBar, PBM_SETRANGE, 0, MAKELPARAM(0, 150));
          SendMessage(hwndPrgBar, PBM_SETSTEP, 1, 0 );
          break;


      case WM_TIMER:
          SendMessage( hwndPrgBar, PBM_STEPIT, 0, 0 );
          i++;
          if ( i == 150 )
              KillTimer(hwnd, ID_TIMER);
          break;

      case WM_COMMAND:
          i = 1;
```

```
            SendMessage( hwndPrgBar, PBM_SETPOS, 0, 0 );
            SetTimer(hwnd, ID_TIMER, 5, NULL);
            break;

        case WM_DESTROY:
            KillTimer(hwnd, ID_TIMER);
            PostQuitMessage(0);
            break;
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Giải thích:
Trong bài viết này, chúng ta tạo 1 progress bar và 1 button. Button dùng để start progress bar. Chúng ta có sử dụng 1 bộ timer (định thời gian) để update progress bar.

Code:

```
hwndPrgBar = CreateWindowEx(0, PROGRESS_CLASS, NULL,
    WS_CHILD | WS_VISIBLE | PBS_SMOOTH,
    30, 20, 190, 25, hwnd, NULL, g_hinst, NULL);
```

Chúng ta tạo progress bar với tên cửa sổ **PROGRESS_CLASS** và với tham số **PBS_SMOOTH**

Code:

```
SendMessage(hwndPrgBar, PBM_SETRANGE, 0, MAKELPARAM(0, 150));
SendMessage(hwndPrgBar, PBM_SETSTEP, 1, 0 );
```

Hai câu lệnh này dùng để set phạm vi và bước nhảy cho progress bar.

Code:

```
i = 1;
SendMessage( hwndPrgBar, PBM_SETPOS, 0, 0 );
SetTimer(hwnd, ID_TIMER, 5, NULL);
```

Khi chúng ta ấn button "Start", chúng ta set giá trị i = 1, set vị trí ban đầu cho progress bar, khởi tạo cho bộ timer. Theo chu kì, bộ timer sẽ gửi các message **WM_TIMER** tới procedure của HĐH Windows.

Code:

```
SendMessage( hwndPrgBar, PBM_STEPIT, 0, 0 );
i++;
if ( i == 150 )
    KillTimer(hwnd, ID_TIMER);
```

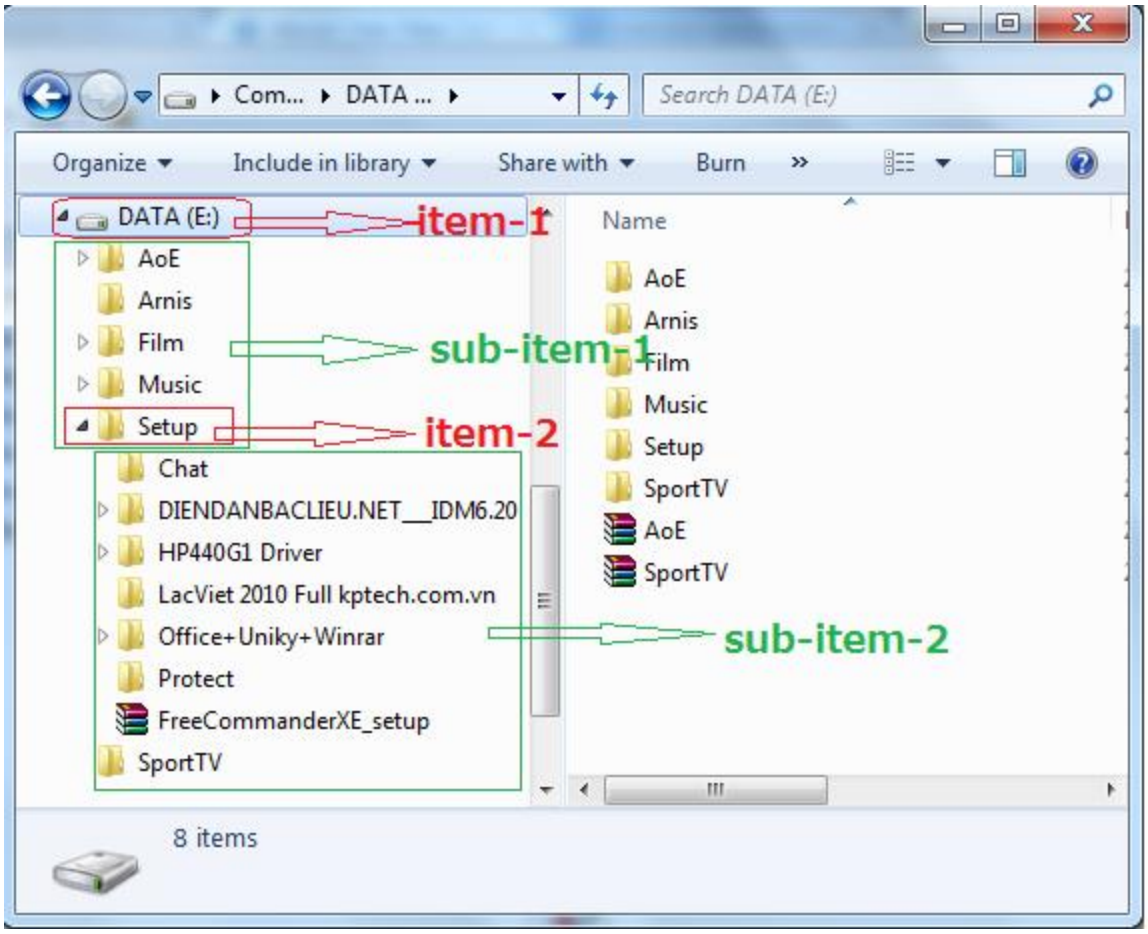
Trong suốt quá trình nhận message **WM_TIMER**, chúng ta sẽ update progress bar bằng cách gửi message **PBM_STEPIT** tới progress bar. Khi progress bar chạy hết, chúng ta sẽ

kill timer.

Demo

Treeview

Tree-view là 1 dạng cửa sổ hiển thị 1 danh sách các item dạng parent-child.
Ví dụ bạn hay gặp nhất là trình quản lý thư mục của Windows OS.



Code:

```
#include "windows.h"
#include "commctrl.h"

#define ID_TREEVIEW 100

HINSTANCE g_hInst;

HWND CreateATreeView(HWND hwndParent);
HTREEITEM AddItemToTree(HWND hwndTV, LPTSTR lpszItem, int nLevel);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

// Ham Winmain()
int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow )
{
    MSG  msg ;
    HWND hwnd;
    WNDCLASS wc;
    wc.style         = CS_HREDRAW | CS_VREDRAW;
    wc.cbClsExtra    = 0;
    wc.cbWndExtra    = 0;
    wc.lpszClassName = TEXT( "TreeView" );
```

```
    wc.hInstance     = hInstance ;
    wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
    wc.lpszMenuName  = NULL;
    wc.lpfnWndProc   = WndProc;
    wc.hCursor       = LoadCursor(NULL, IDC ARROW);
    wc.hIcon         = LoadIcon(NULL, IDI_APPLICATION);
    RegisterClass(&wc);
    // Create parent window
    hwnd = CreateWindow( wc.lpszClassName, TEXT("Menu"),
        WS OVERLAPPEDWINDOW | WS VISIBLE,
        100, 100, 250,350, NULL, NULL, hInstance, NULL);
    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);   // Update windows

    while( GetMessage(&msg, NULL, 0, 0))
    {
        DispatchMessage(&msg);
    }
    return (int) msg.wParam;
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam )
{
    switch(msg)
    {
        case WM_CREATE:
        {
            HWND hwndTreeview;
            hwndTreeview = CreateATreeView(hwnd);
            AddItemToTree(hwndTreeview, L"Drink", 1);
            AddItemToTree(hwndTreeview, L"Orange juice", 2);
            AddItemToTree(hwndTreeview, L"Price: 20K", 3);
            AddItemToTree(hwndTreeview, L"Coffee", 2);
            AddItemToTree(hwndTreeview, L"Price: 25K", 3);
            AddItemToTree(hwndTreeview, L"Tea", 2);
            AddItemToTree(hwndTreeview, L"Price: 15K", 3);
            break;
        }
        case WM_DESTROY:
        {
            PostQuitMessage(0);
            return 0;
        }
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
// Ham tao treeview
HWND CreateATreeView(HWND hwndParent)
{
    RECT rcClient;  // dimensions of client area
    HWND hwndTV;    // handle to tree-view control

    // Ensure that the common control DLL is loaded.
    InitCommonControls();

    // Get the dimensions of the parent window's client area, and create
```

```
    // the tree-view control.
    GetClientRect(hwndParent, &rcClient);
    hwndTV = CreateWindowEx(0,
        WC_TREEVIEW,
        TEXT("Tree View"),
        WS_VISIBLE | WS_CHILD | WS_BORDER | TVS_HASLINES,
        0,
        0,
        rcClient.right,
        rcClient.bottom,
        hwndParent,
        (HMENU)ID_TREEVIEW,
        g_hInst,
        NULL);
    return hwndTV;
}

HTREEITEM AddItemToTree(HWND hwndTV, LPTSTR lpszItem, int nLevel)
{
    TVITEM tvi;
    TVINSERTSTRUCT tvins;
    static HTREEITEM hPrev = (HTREEITEM)TVI_FIRST;
    static HTREEITEM hPrevRootItem = NULL;
    static HTREEITEM hPrevLev2Item = NULL;
    HTREEITEM hti;

    tvi.mask = TVIF_TEXT | TVIF_IMAGE
        | TVIF_SELECTEDIMAGE | TVIF_PARAM;

    // Set the text of the item.
    tvi.pszText = lpszItem;
    tvi.cchTextMax = sizeof(tvi.pszText)/sizeof(tvi.pszText[0]);

    // Assume the item is not a parent item, so give it a
    // document image.
    tvi.iImage = 0;
    tvi.iSelectedImage = 0;

    // Save the heading level in the item's application-defined
    // data area.
    tvi.lParam = (LPARAM)nLevel;
    tvins.item = tvi;
    tvins.hInsertAfter = hPrev;

    // Set the parent item based on the specified level.
    if (nLevel == 1)
        tvins.hParent = TVI_ROOT;
    else if (nLevel == 2)
        tvins.hParent = hPrevRootItem;
    else
        tvins.hParent = hPrevLev2Item;

    // Add the item to the tree-view control.
    hPrev = (HTREEITEM)SendMessage(hwndTV, TVM_INSERTITEM,
        0, (LPARAM)(LPTVINSERTSTRUCT)&tvins);
```

```
    if (hPrev == NULL)
        return NULL;

    // Save the handle to the item.
    if (nLevel == 1)
        hPrevRootItem = hPrev;
    else if (nLevel == 2)
        hPrevLev2Item = hPrev;

    // The new item is a child item. Give the parent item a
    // closed folder bitmap to indicate it now has child items.
    if (nLevel > 1)
    {
        hti = TreeView_GetParent(hwndTV, hPrev);
        tvi.mask = TVIF_IMAGE | TVIF_SELECTEDIMAGE;
        tvi.hItem = hti;
        tvi.iImage = 0;
        tvi.iSelectedImage = 0;
        TreeView_SetItem(hwndTV, &tvi);
    }
    return hPrev;
}
```

Giải thích:
Thư viện
Code:

```
#include "commctrl.h"
```

Chứa các function liên quan đến các lớp cửa sổ common control.

Code:

```
hwnd = CreateWindow( wc.lpszClassName, TEXT("Menu"),
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        100, 100, 250,350, NULL, NULL, hInstance, NULL);
```

Trong bài này, tôi có ý định tạo 1 cửa sổ cha (parent) với title là "Menu" với kích thước như hàm khai báo ở trên. Sau đó, tôi sẽ tạo cửa sổ treeview là cửa sổ con (child) của cửa sổ parent. Cửa sổ treeview này có chức năng hiển thị 1 menu đồ uống như: Orange juice, Coffee, Tea,... Khi người dùng click vào tên 1 đồ uống bất kì, thì giá của loại đồ uống đó sẽ được hiển thị dưới dạng cây.

Code:

```
hwndTreeview = CreateATreeView(hwnd);
```

Hàm này để tạo treeview.
Trong đó:
- Giá trị truyền vào là handler của cửa sổ cha.
- Giá trị trả về: là handler trỏ tới treeview vừa được tạo.

Code:

```
InitCommonControls();
```

Treeview là 1 trong các loại cửa sổ kiểu common control (treeview, tooltip, trackbar, ...). Do vậy, cần gọi hàm này trước khi tạo treeview.

Code:

```
GetClientRect(hwndParent, &rcClient);
```

Hàm này lấy tọa độ (left, right, top, bottom) của cửa sổ parent và lưu vào biến rcClient.

Code:

```
hwndTV = CreateWindowEx(0,
        WC_TREEVIEW,
        TEXT("Tree View"),
        WS_VISIBLE | WS_CHILD | WS_BORDER | TVS_HASLINES,
        0,
        0,
        rcClient.right,
        rcClient.bottom,
        hwndParent,
        (HMENU)ID_TREEVIEW,
        g_hInst,
        NULL);
```

Để tạo treeview ta dùng hàm CreateWindowEx() với hằng số **WC_TREEVIEW** được truyền cho tham số lpClassName.
Trong đó:
- rcClient.right : là chiều rộng của cửa sổ cha (parent)
- rcClient.bottom : là chiều cao của cửa sổ cha (parent)
- ID_TREEVIEW : là ID của treeview (là hằng số nguyên, người lập trình tự định nghĩa)
- g_hInst: là biến instant global. (Trong bài này, biến này không có nhiều ý nghĩa. Tôi sẽ giới thiệu trong các bài viết sắp tới).

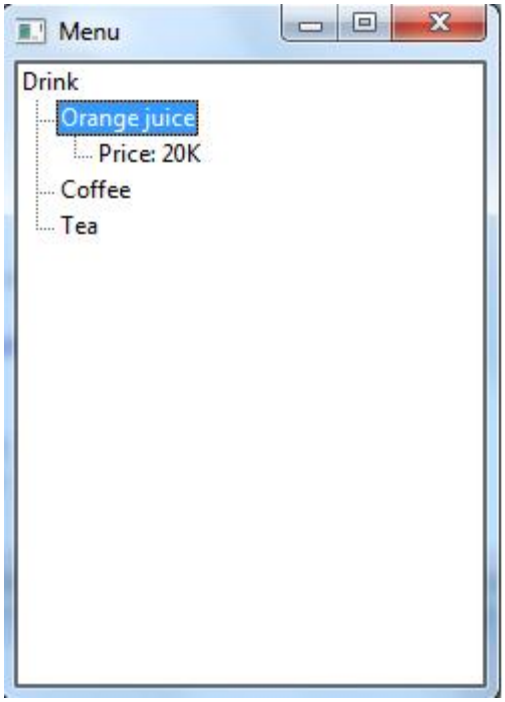
Code:

```
AddItemToTree(hwndTreeview, L"Drink", 1);
AddItemToTree(hwndTreeview, L"Orange juice", 2);
AddItemToTree(hwndTreeview, L"Price: 20K", 3);
AddItemToTree(hwndTreeview, L"Coffee", 2);
AddItemToTree(hwndTreeview, L"Price: 25K", 3);
AddItemToTree(hwndTreeview, L"Tea", 2);
AddItemToTree(hwndTreeview, L"Price: 15K", 3);
```

Các hàm trên add các item vào cửa sổ treeview vừa được tạo.
Trong đó:
- hwndTreeview: handler trỏ tới treeview được tạo.
- "Drink", "Orange juice",... là các item và sub-item của treeview.

Dưới đây, ta sẽ đi tìm hiểu hàm AddItemToTree() cụ thể.
...

Demo



Tab control

Tab control là 1 dạng cửa sổ có thể tạo nhiều tab. Ví dụ như: các trình duyệt web, các bạn có thể mở nhiều tab khác nhau, mỗi tab sẽ chứa nội dung trang web bạn muốn truy cập.

Code:

```
#include "windows.h"
#include "commctrl.h"

#define ID_TABCTRL 1
#define EDIT 2
#define BTN_ADD 3
#define BTN_DEL 4
#define BTN DELALL 5

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
HWND hTab, hEdit;
HINSTANCE g_hinst;

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow )
{
```

```
    MSG  msg ;
    WNDCLASS wc = {0};
    wc.lpszClassName = TEXT( "Application" );
    wc.hInstance     = hInstance ;
    wc.hbrBackground = GetSysColorBrush(COLOR 3DFACE);
    wc.lpfnWndProc   = WndProc ;
    wc.hCursor       = LoadCursor(0,IDC_ARROW);

    g_hinst = hInstance;

    RegisterClass(&wc);
    CreateWindow( wc.lpszClassName, TEXT("Tab Control"),
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        100, 100, 380, 230, 0, 0, hInstance, 0);

    while( GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return (int) msg.wParam;
}


LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)

{

    TCITEM tie;
    TCHAR text[250];
    LRESULT count, id;
    INITCOMMONCONTROLSEX icex;

    switch(msg)
    {

       case WM_CREATE:

            icex.dwSize = sizeof(INITCOMMONCONTROLSEX);
            icex.dwICC = ICC_TAB_CLASSES;
            InitCommonControlsEx(&icex);

            hTab = CreateWindow(WC_TABCONTROL, NULL, WS_CHILD | WS_VISIBLE,
                0, 0, 200, 150, hwnd,(HMENU) ID_TABCTRL, g_hinst, NULL);

            hEdit = CreateWindow(L"edit",NULL,WS_CHILD | WS_VISIBLE |
WS_BORDER,
                250, 20, 100, 25, hwnd, (HMENU) EDIT, g_hinst, NULL);

            CreateWindow(L"button",L"Add", WS_CHILD | WS_VISIBLE |
BS_PUSHBUTTON,
                250, 50, 100, 25, hwnd, (HMENU) BTN_ADD, g_hinst, NULL);

            CreateWindow(L"button", L"Del", WS_CHILD | WS_VISIBLE |
BS_PUSHBUTTON,
                250, 80, 100, 25, hwnd, (HMENU) BTN_DEL, g_hinst, NULL);
```

```
            CreateWindow(L"button",L"Delall", WS_CHILD | WS_VISIBLE |
BS_PUSHBUTTON,
                250, 110, 100, 25, hwnd, (HMENU) BTN_DELALL, g_hinst, NULL);
            break;

        case WM_COMMAND:

            switch(LOWORD(wParam))
            {

                 case BTN_ADD:
                     GetWindowText(hEdit, text, 250);

                     if (lstrlen(text) !=0 )
                     {
                        tie.mask = TCIF_TEXT;
                        tie.pszText = text;
                        count = SendMessage(hTab, TCM_GETITEMCOUNT, 0, 0);
                        SendMessage(hTab, TCM_INSERTITEM, count,
                            (LPARAM) (LPTCITEM) &tie);
                     }
                     break;

                 case BTN_DEL:
                     id = SendMessage(hTab, TCM_GETCURSEL, 0, 0);
                     if (id != -1)
                     {
                         SendMessage(hTab, TCM_DELETEITEM, 0, id);
                     }
                     break;

                case BTN_DELALL:
                    SendMessage(hTab, TCM_DELETEALLITEMS, 0, 0);
                    break;
            }
            break;

        case WM_DESTROY:
            PostQuitMessage(0);
            break;
    }
    return(DefWindowProc(hwnd, msg, wParam, lParam));
}
```

Giải thích:
Trong bài viết này, chúng ta sử dụng 1 tab control, 1 edit control và 3 button. Chúng ta có thể tạo mới và xóa bỏ các tab vừa tạo trên tab control.

Code:

```
hTab = CreateWindow(WC_TABCONTROL, NULL, WS_CHILD | WS_VISIBLE,
                0, 0, 200, 150, hwnd,(HMENU) ID_TABCTRL, g_hinst, NULL);
```

Để tạo cửa sổ tab control, chúng ta vẫn sử dụng hàm CreateWindow() quen thuộc với tham số **WC_TABCONTROL**.

Code:

```
if (lstrlen(text) !=0 )
{
    tie.mask = TCIF_TEXT;
    tie.pszText = text;
    count = SendMessage(hTab, TCM_GETITEMCOUNT, 0, 0);
    SendMessage(hTab, TCM_INSERTITEM, count,
        (LPARAM) (LPTCITEM) &tie);
}
```

Để tạo tab mới trên tab control, chúng ta cần gán giá trị cho struct **TCITEM**. Trong bài này, chúng ta chỉ muốn tạo tab với nội dung là text nên sử dụng **TCIF_TEXT**. Sau đó, chúng ta gửi 2 message tới tab control.
- Gửi message thứ nhất để lấy số lượng tab trên tab control hiện tại
- Gửi message thứ hai để insert tab mới vào tab control.

Code:

```
id = SendMessage(hTab, TCM_GETCURSEL, 0, 0);
if (id != -1)
{
    SendMessage(hTab, TCM_DELETEITEM, 0, id);
}
```

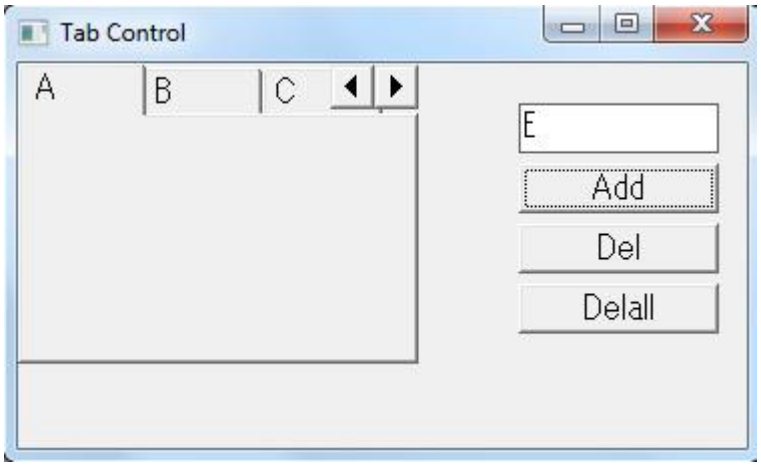
Để xóa tab được chỉ định trên tab control, chúng ta gửi message TCM_GETCURSEL tới tab control để lấy về id của tab đang được chỉ định. Sau đó, chúng ta gửi message TCM_DELETEITEM để xóa tab đó.

Code:

```
SendMessage(hTab, TCM_DELETEALLITEMS, 0, 0);
```

Để xóa toàn bộ các tab trên tab control. Chúng ta gửi message **TCM_DELETEALLITEMS** tới tab control.

Demo



ListBox

List box là cửa sổ liệt kê nhiều item, mà người dùng có thể lựa chọn 1 hoặc nhiều item.
Code:

```
#include "windows.h"
#include "strsafe.h"

#define IDC_LIST 1
#define IDC_STATIC 2

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

HINSTANCE g_hinst;

typedef struct
{
    TCHAR name[30];
    TCHAR role[20];
} Friends;

Friends friends[] =
{
    {TEXT("vncoding"), TEXT("Admin")},
    {TEXT("vhnhan01"), TEXT("member")},
    {TEXT("thanhx175"), TEXT("member")},
    {TEXT("tienle"), TEXT("member")},
    {TEXT("hoanghoa"), TEXT("member")},
};

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow )
{
    MSG  msg ;
    WNDCLASS wc = {0};
```

```
    wc.lpszClassName = TEXT( "Application" );
    wc.hInstance     = hInstance;
    wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
    wc.lpfnWndProc   = WndProc;
    wc.hCursor       = LoadCursor(0, IDC ARROW);

    g_hinst = hInstance;

    RegisterClass(&wc);
    CreateWindow( wc.lpszClassName, TEXT("List Box"),
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        100, 100, 340, 200, 0, 0, hInstance, 0);

    while( GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return (int) msg.wParam;
}


LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{

    static HWND hwndList, hwndStatic;
    int i, sel;
    TCHAR buff[100];

    switch(msg)
    {

        case WM_CREATE:

            hwndList = CreateWindow(TEXT("listbox") , NULL, WS_CHILD |
WS_VISIBLE | LBS_NOTIFY,
                10, 10, 150, 120, hwnd,(HMENU) IDC_LIST, g_hinst, NULL);

            hwndStatic = CreateWindow(TEXT("static") , NULL, WS_CHILD |
WS_VISIBLE,
                200, 10, 120, 45, hwnd,(HMENU) IDC_STATIC, g_hinst, NULL);

            for (i = 0; i < ARRAYSIZE(friends); i++)
            {
                SendMessage(hwndList, LB_ADDSTRING, 0, (LPARAM)
friends[i].name);
            }
            break;

        case WM_COMMAND:
            if (LOWORD(wParam) == IDC_LIST)
            {
                if (HIWORD(wParam) == LBN_SELCHANGE)
                {
                    sel = (int) SendMessage(hwndList, LB_GETCURSEL, 0, 0);
                    StringCbPrintf(buff, ARRAYSIZE(buff), TEXT("Role: %s"),
                        friends[sel].role);
```

```
                    SetWindowText(hwndStatic, buff);
                }
            }
            break;

        case WM_DESTROY:
            PostQuitMessage(0);
            break;
    }
    return (DefWindowProc(hwnd, msg, wParam, lParam));
}
```

Giải thích

Code:

```
CreateWindow( wc.lpszClassName, TEXT("List Box"),
    WS_OVERLAPPEDWINDOW | WS_VISIBLE,
    100, 100, 340, 200, 0, 0, hInstance, 0);
```

Trong bài viết này, chúng ta tạo 1 cửa sổ cha (parent) có tiêu đề là "List Box"

Code:

```
hwndList = CreateWindow(TEXT("listbox") , NULL, WS_CHILD | WS_VISIBLE |
LBS_NOTIFY,
    10, 10, 150, 120, hwnd,(HMENU) IDC_LIST, g_hinst, NULL);

hwndStatic = CreateWindow(TEXT("static") , NULL, WS_CHILD | WS_VISIBLE,
    200, 10, 120, 45, hwnd,(HMENU) IDC_STATIC, g_hinst, NULL);
```

Trên cửa sổ cha, chúng ta tạo 2 loại cửa sổ: list box và static box.

Code:

```
for (i = 0; i < ARRAYSIZE(friends); i++)
{
    SendMessage(hwndList, LB_ADDSTRING, 0, (LPARAM) friends[i].name);
}
```

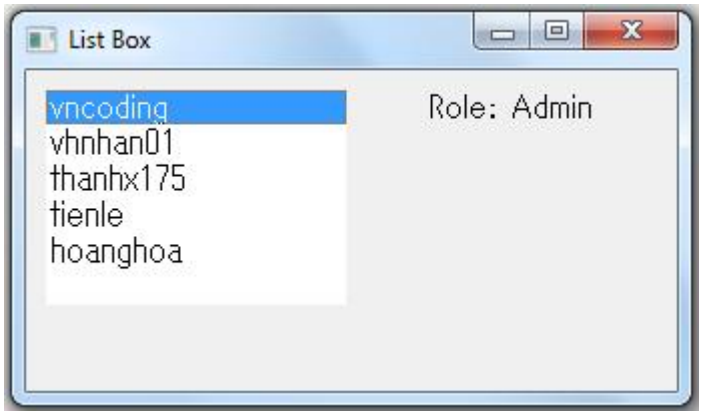
Để tạo các item trên list box, chúng ta gửi message **LB_ADDSTRING** tới list box với nội dung item là friends[i].name.

Code:

```
if (HIWORD(wParam) == LBN_SELCHANGE)
{
    sel = (int) SendMessage(hwndList, LB_GETCURSEL, 0, 0);
    StringCbPrintf(buff, ARRAYSIZE(buff), TEXT("Role: %s"),
        friends[sel].role);
    SetWindowText(hwndStatic, buff);
}
```

Nếu chúng ta chọn 1 item trên list box, hàm xử lí message WndProc() sẽ nhận message **LBN_SELCHANGE**. Tại đây, chúng ta sẽ gửi message**LB_GETCURSEL** tới list box để biết được item nào đang được lựa chọn. Sau đó, chúng ta copy nội dung role của item đang được chọn vào buff. Cuối cùng hiển thị nội dung này lên cửa sổ static box.

Demo



Các bạn có thể tham khảo thêm các bài viết về lập trình Win32 tại diễn đàn:
http://vncoding.net/forum/forumdisplay.php?f=2