Cấu trúc chương trình trong Windows

hú 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()

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

Paramter

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()

```
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

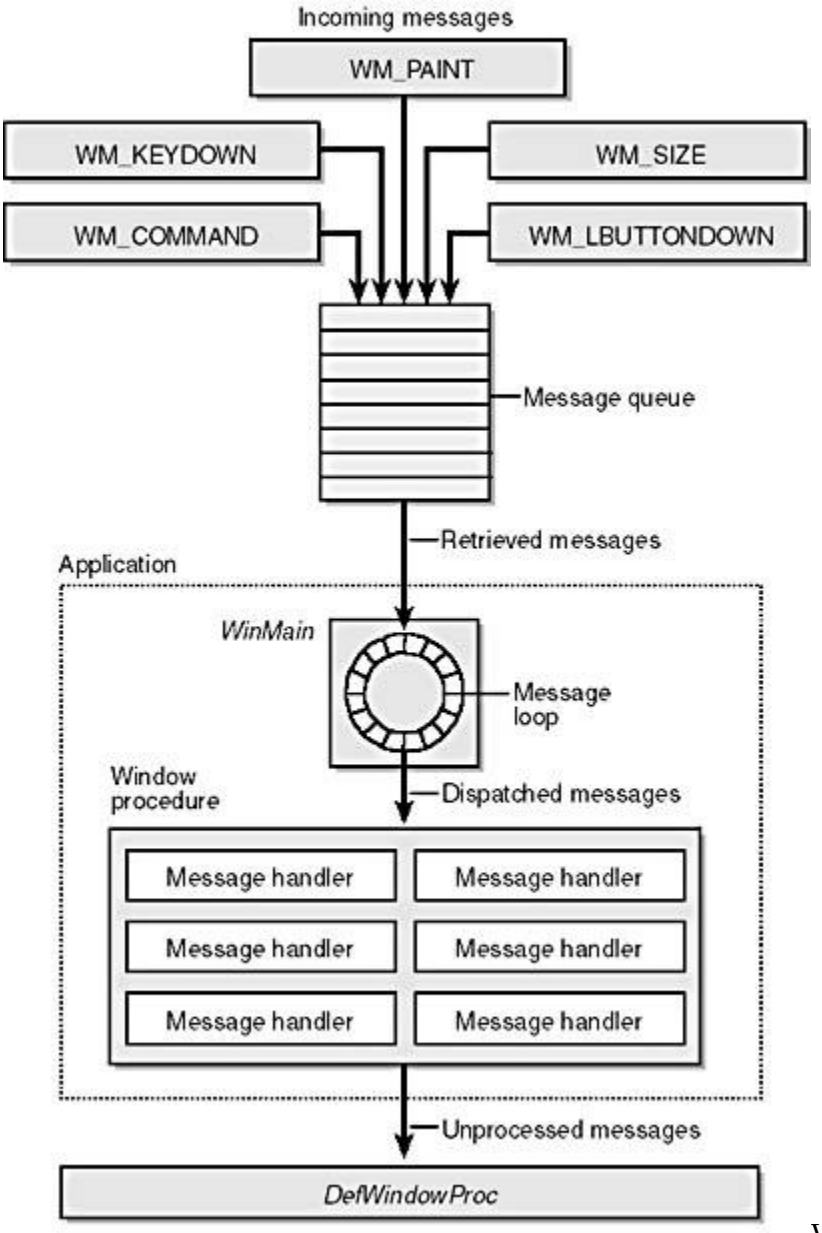
```
LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
```

– 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_CREATE: 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



Win32 flow

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.

```
1#include "windows.h"
2
```

```
// 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);   // Display 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;
        }

```

```
49    }
50    return DefWindowProc(hwnd, msg, wParam, lParam);
51}
52
53
```

Giải thích:

```
1#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.

```
1LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
```

Khai báo hàm xử lí message.

```
1wc.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.

```
1wc.cbClsExtra    = 0;
2wc.cbWndExtra    = 0;
```

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

```
1wc.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).

```
1wc.hInstance     = hInstance
2wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
```

Đặt màu cho nền cửa sổ chính.

```
1wc.lpszMenuName  = NULL;
```

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

```
1wc.lpfnWndProc   = WndProc;
```

Khai báo thủ tục xử lí message cho class.

```
1wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
```

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

Đặt biểu tượng con trỏ và Icon trong ứng dụng.

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

Đăng kí lớp cửa sổ với Windows

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

Hiển thị và update cửa sổ.

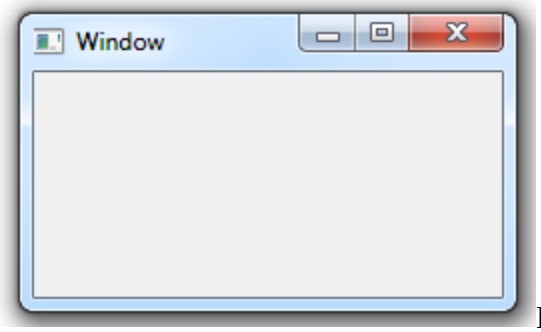
```
while( GetMessage(&msg, NULL, 0, 0))
{
    DispatchMessage(&msg);
}
```

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().

```

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

Kết quả:



Basic windows

Menu

Thanh **Menu** được sử dụng 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.

```
#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, 20

    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

```
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

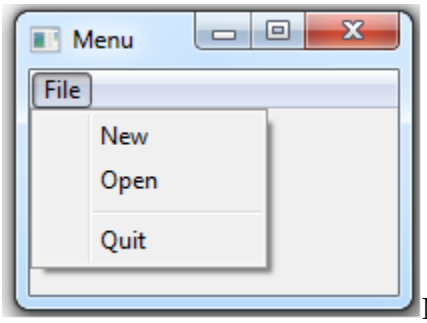
```
AppendMenu(hMenu, MF_SEPARATOR, 0, NULL);
AppendMenu(hMenubar, MF_POPUP, (UINT_PTR)hMenu, TEXT("File"));
```

Gắn các menu item lên thanh menubar File.

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

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

Kết quả:



Menu

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.

```
#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, 1

    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.

```
POINT point;

point.x = LOWORD(lParam);

point.y = HIWORD(lParam);
```

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

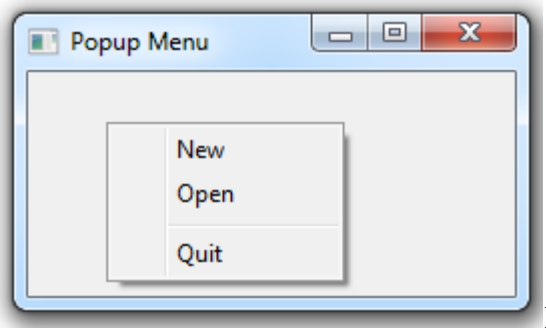
```
hMenu = CreatePopupMenu();
```

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

```
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.

Kết quả:



Popup menu

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,..

Dialog Box

```
#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 lpCmdLin
{
    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 |

    ghInstance = hInstance;
    while( GetMessage(&msg, NULL, 0, 0))
```

```
25      {
26          DispatchMessage(&msg);
27      }
28      return (int) msg.wParam;
29  }
30
31  LRESULT CALLBACK WndProc( HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam )
32  {
33      switch(msg)
34      {
35      case WM_CREATE:
36          RegisterDialogClass(hwnd);
37          CreateWindow(TEXT("button"), TEXT("Show dialog"), WS_VISIBLE | WS_CHILD
38          break;
39       case WM_COMMAND:
40          CreateDialogBox(hwnd);
41          break;
42       case WM_DESTROY:
43          PostQuitMessage(0);
44          return 0;
45      }
46      return DefWindowProc(hwnd, msg, wParam, lParam);
47  }
48
49  LRESULT CALLBACK DialogProc( HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam )
50  {
51      switch(msg)
52      {
53      case WM_CREATE:
54          CreateWindow(TEXT("button"), TEXT("Ok"), WS_VISIBLE | WS_CHILD , 50, 50,
55          break;
56      case WM_COMMAND:
57          DestroyWindow(hwnd);
58          break;
59      case WM_CLOSE:
60          DestroyWindow(hwnd);
61          break;
62      }
63      return (DefWindowProc(hwnd, msg, wParam, lParam));
64  }
65
66  void RegisterDialogClass(HWND hwnd)
67  {
68      WNDCLASSEX wc = {0};
69      wc.cbSize = sizeof(WNDCLASSEX);
70      wc.lpfnWndProc = (WNDPROC) DialogProc;
```

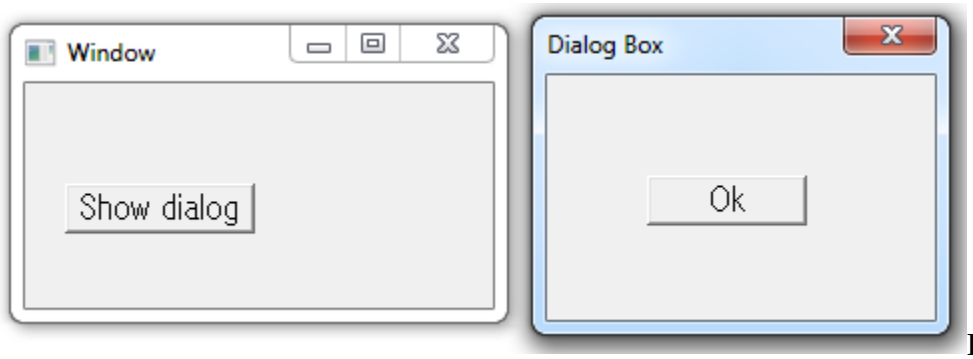
```
71      wc.hInstance = ghInstance;
72      wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
73      wc.lpszClassName = TEXT("DialogClass");
74      RegisterClassEx(&wc);
75  }
76
77  void CreateDialogBox(HWND hwnd)
78  {
79      CreateWindowEx(WS_EX_DLGMODALFRAME | WS_EX_TOPMOST, TEXT("DialogClass"), TEX
80  NULL, ghInstance, NULL);
    }
```

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

```
1  LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
2  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.

Kết quả:



Dialog Box

Colour Dialog Box

```
1  #include "windows.h"
2
3  LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
4  LRESULT CALLBACK PanelProc(HWND, UINT, WPARAM, LPARAM);
5
6  void RegisterPanel(void);
7  COLORREF ShowColorDialog(HWND);
8
```

```
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_OVERLAPPEDWIND
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,
        RegisterPanel();
        hwndPanel = CreateWindow(TEXT("Panel"), NULL, WS_CHILD | WS_VISIBLE, 13
        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);
```

```
101     rwc.lpfnWndProc = PanelProc;
102     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.

```
1COLORREF gColor = RGB(255, 255, 255);
```

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

```
1gColor = ShowColorDialog(hwnd);
```

Hàm này trả về giá trị màu được chọn.

```
1CHOOSECOLOR cc;
```

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

```
1cc.rgbResult = RGB(0, 255, 0);
2cc.Flags = CC_FULLOPEN | CC_RGBINIT;
```

Khởi tạo màu trên Dialog Box.

```
1ChooseColor(&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.

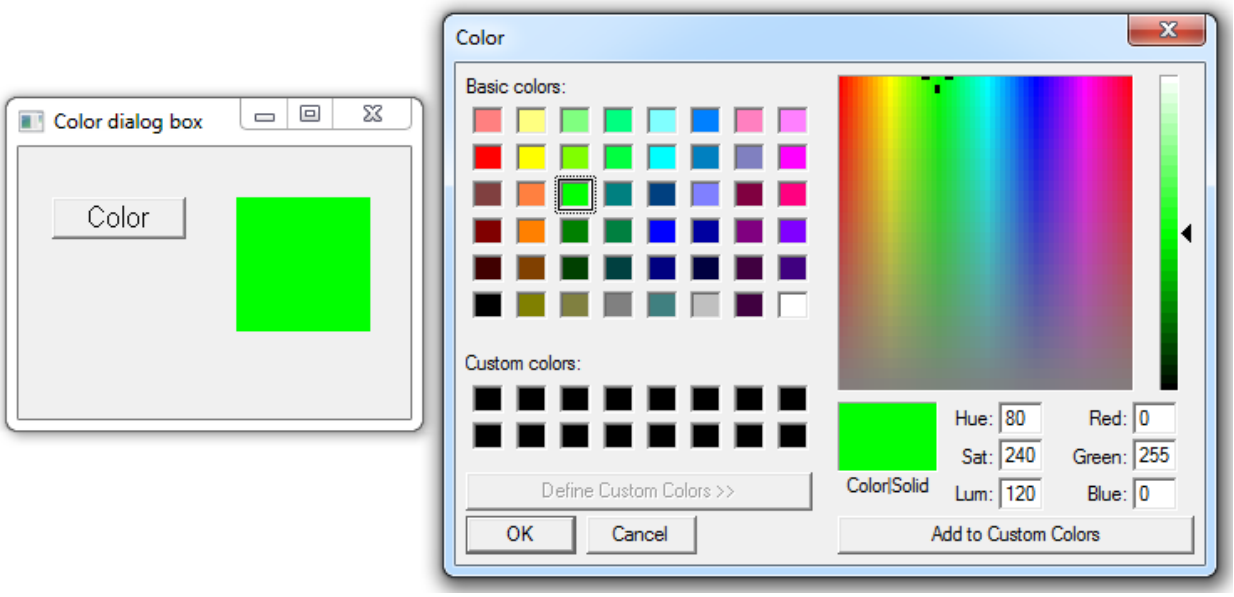
```
1return cc.rgbResult;
1InvalidateRect (hwndPanel, NULL, TRUE);
```

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

```
1SetBkColor(hdc, gColor);
2ExtTextOut(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, khi giá trị này được khai báo thì hàm ExtTextOut() chỉ hiển thị màu lên background.

Kết quả:



Color Dialog

Button

```
#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,

    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,
        CreateWindow(TEXT("button"), TEXT("Quit"), WS_VISIBLE | WS_CHILD, 120, 50, 80, 25, hwnd,
        break;
    }
    case WM_COMMAND:
    {
        if (LOWORD(wParam) == 1)
        {
            Beep(400, 500);
        }
        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.

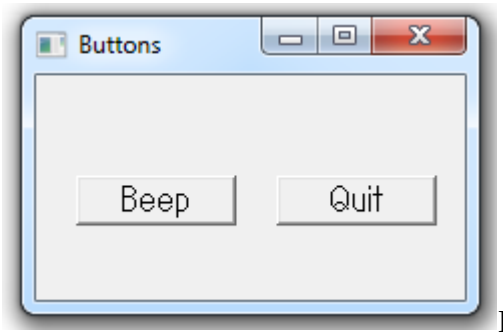
```
CreateWindow(TEXT("button"), TEXT("Beep"), WS_VISIBLE | WS_CHILD, 20, 50, 80, 25, hwnd, (HMENU) 1
```

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

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

```
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(400, 500) được gọi (tiếng Beep).
Khi bấm vào button “Quit” hàm **PostQuitMessage**(0) sẽ gửi message tới Window yêu cầu đóng ứng dụng.
Kết quả:



Button

CheckBox

Radio buttons dùng để chọn chỉ một cái trong nhiều lựa chọn còn checkbox thì ngược lại, nó cho phép bạn không chọn lựa cái nào hoặc chọn nhiều cái (từ một cho đến tất cả)

```
#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,
    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
        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.

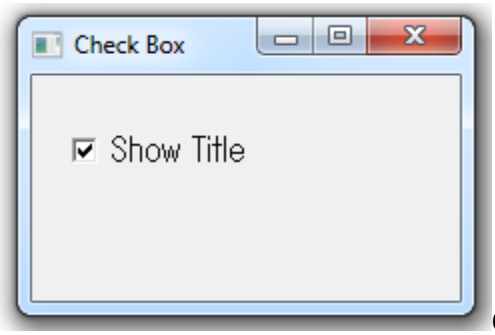
```
CreateWindow(TEXT("button"), TEXT("Show Title"), WS_VISIBLE | WS_CHILD | BS_CHECKBOX, 20, 20, 185
```

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

```
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.

Kết quả:



Check Box

Control Static

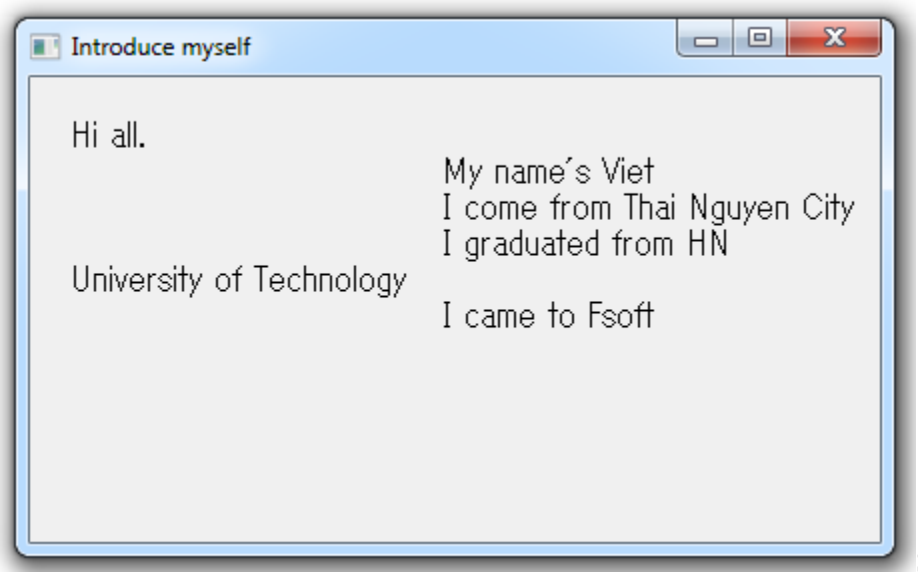
```
1
2
3 #include "windows.h"
4
5 LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
6 int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR lpCmdLine, int nCmdShow
7 {
8     MSG  msg;
9     WNDCLASSW wc = {0};
10    wc.lpszClassName = L"Static Control";
11    wc.hInstance     = hInstance;
12    wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
13    wc.lpfnWndProc   = WndProc;
14    wc.hCursor       = LoadCursor(0,IDC_ARROW);
15
16    RegisterClassW(&wc);
17    CreateWindowW(wc.lpszClassName, L"Introduce myself", WS_OVERLAPPEDWINDOW | WS_VISIBLE, 100,
18
19    while( GetMessage(&msg, NULL, 0, 0))
20    {
21        TranslateMessage(&msg);
22        DispatchMessage(&msg);
23    }
24    return (int) msg.wParam;
25 }
26
27 LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
28 {
29    static wchar_t *lyrics =  L"Hi all.\n\
30                               My name's Viet\n\
31                               I come from Thai Nguyen City\n\
32                               I graduated from HN University of Technology \n\
33                               I came to Fsoft";
34    switch(msg)
35    {
36    case WM_CREATE:
37        CreateWindowW(L"STATIC", lyrics, WS_CHILD | WS_VISIBLE | SS_LEFT, 20, 20, 400, 230, hwnd
38        break;
39    case WM_DESTROY:
40        PostQuitMessage(0);
41        break;
42    }
43    return DefWindowProcW(hwnd, msg, wParam, lParam);
44 }
```

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

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

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

Kết quả:



Static control

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.

```
#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, 10

    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,

        CreateWindow(TEXT("button"), TEXT("Blue"), WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON, 2

        CreateWindow(TEXT("button"), TEXT("Yellow"), WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON,

        CreateWindow(TEXT("button"), TEXT("Orange"), WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON,
        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.

```
CreateWindow(TEXT("button"), TEXT("Choose Color"), WS_CHILD | WS_VISIBLE | BS_GROUPBOX, 10, 10, 1
```

Group box được tạo với thông số BS_GROUPBOX.

```
CreateWindow(TEXT("button"), TEXT("Blue"), WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON, 20, 30, 10
```

Radio button “Blue” được tạo với thông số **BS_AUTORADIOBUTTON**.

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

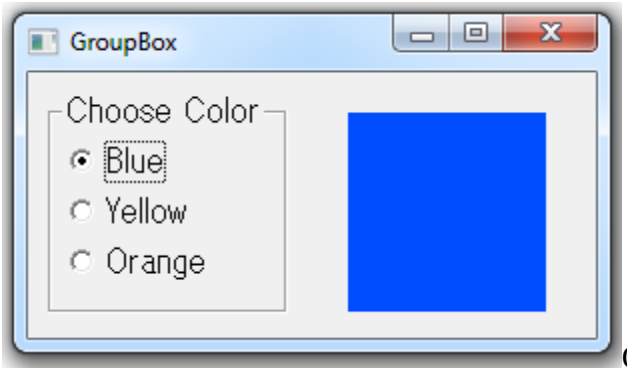
```
case ID_BLUE:
    g_color = RGB(0, 76, 255);
    break;
```

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.

```
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.

Kết quả:



Group Box

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.

```
#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, 1

    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);
}
```

```
47
48void CreateControls(HWND hwnd)
49{
50    HWND hLeftLabel = CreateWindowW(L"STATIC", L"0", WS_CHILD | WS_VISIBLE, 0, 0, 10, 30, hwnd,
51    HWND hRightLabel = CreateWindowW(L"STATIC", L"100", WS_CHILD | WS_VISIBLE, 0, 0, 30, 30, hwn
52    hlbl = CreateWindowW(L"STATIC", L"0", WS_CHILD | WS_VISIBLE, 270, 20, 30, 30, hwnd, (HMENU)3
53    INITCOMMONCONTROLSEX icex;
54    icex.dwSize = sizeof(INITCOMMONCONTROLSEX);
55    icex.dwICC  = ICC_LISTVIEW_CLASSES;
56    InitCommonControlsEx(&icex);
57
58    hTrack = CreateWindowW(L"msctls_trackbar32", L"Trackbar Control", WS_CHILD | WS_VISIBLE | TB
20, 20, 170, 30, hwnd, (HMENU) 3, NULL, NULL);
59    SendMessageW(hTrack, TBM_SETRANGE,  TRUE, MAKELONG(0, 100));
60    SendMessageW(hTrack, TBM_SETPAGESIZE, 0,  10);
61    SendMessageW(hTrack, TBM_SETTICFREQ, 10, 0);
62    SendMessageW(hTrack, TBM_SETPOS, FALSE, 0);
63    SendMessageW(hTrack, TBM_SETBUDDY, TRUE, (LPARAM) hLeftLabel);
64    SendMessageW(hTrack, TBM_SETBUDDY, FALSE, (LPARAM) hRightLabel);
65}
66
67void UpdateLabel(void)
68{
69    LRESULT pos = SendMessageW(hTrack, TBM_GETPOS, 0, 0);
70    wchar_t buf[4];
71    wsprintfW(buf, L"%ld", pos);
72    SetWindowTextW(hlbl, buf);
73}
74
75
76
```

Giải thích:

```
1HWND hLeftLabel = CreateWindowW(L"STATIC", L"0", WS_CHILD | WS_VISIBLE, 0, 0, 10, 30, hwnd, (HMEN
2HWND hRightLabel = CreateWindowW(L"STATIC", L"100", WS_CHILD | WS_VISIBLE, 0, 0, 30, 30, hwnd, (H
3hlbl = CreateWindowW(L"STATIC", L"0", WS_CHILD | WS_VISIBLE, 270, 20, 30, 30, hwnd, (HMENU)3, NUL
```

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.

```
1INITCOMMONCONTROLSEX icex;
2icex.dwSize = sizeof(INITCOMMONCONTROLSEX);
3icex.dwICC  = ICC_LISTVIEW_CLASSES;
4InitCommonControlsEx(&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.

```
hTrack = CreateWindowW(L"msctls_trackbar32", L"Trackbar Control", WS_CHILD | WS_VISIBLE | TBS_AUT
```

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

```
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.

```
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.

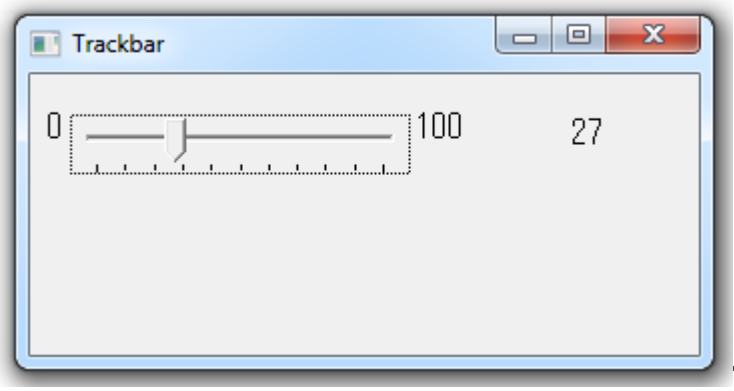
```
case WM_HSCROLL:
    UpdateLabel();
    break;
```

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

```
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).

Kết quả:



Trackbar

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 %.

```
#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
    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,
        CreateWindow(TEXT("button"), TEXT("Start"), WS_CHILD | WS_VISIBLE, 85, 90, 80, 25, hwnd,
        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.

```
hwndPrgBar = CreateWindowEx(0, PROGRESS_CLASS, NULL, WS_CHILD | WS_VISIBLE | PBS_SMOOTH, 30, 20,
```

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

```
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.

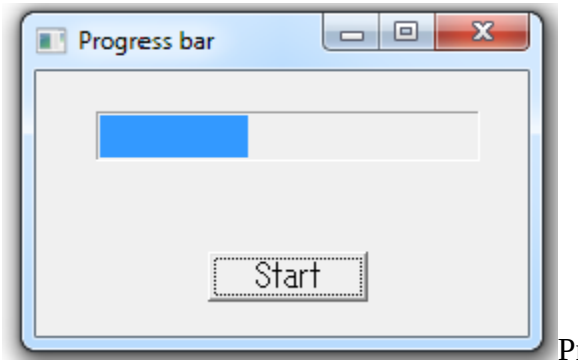
```
i = 1;
SendMessage( hwndPrgBar, PBM_SETPOS, 0, 0 );
SetTimer(hwnd, ID_TIMER, 5, NULL);
```

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.

```
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.

Kết quả:

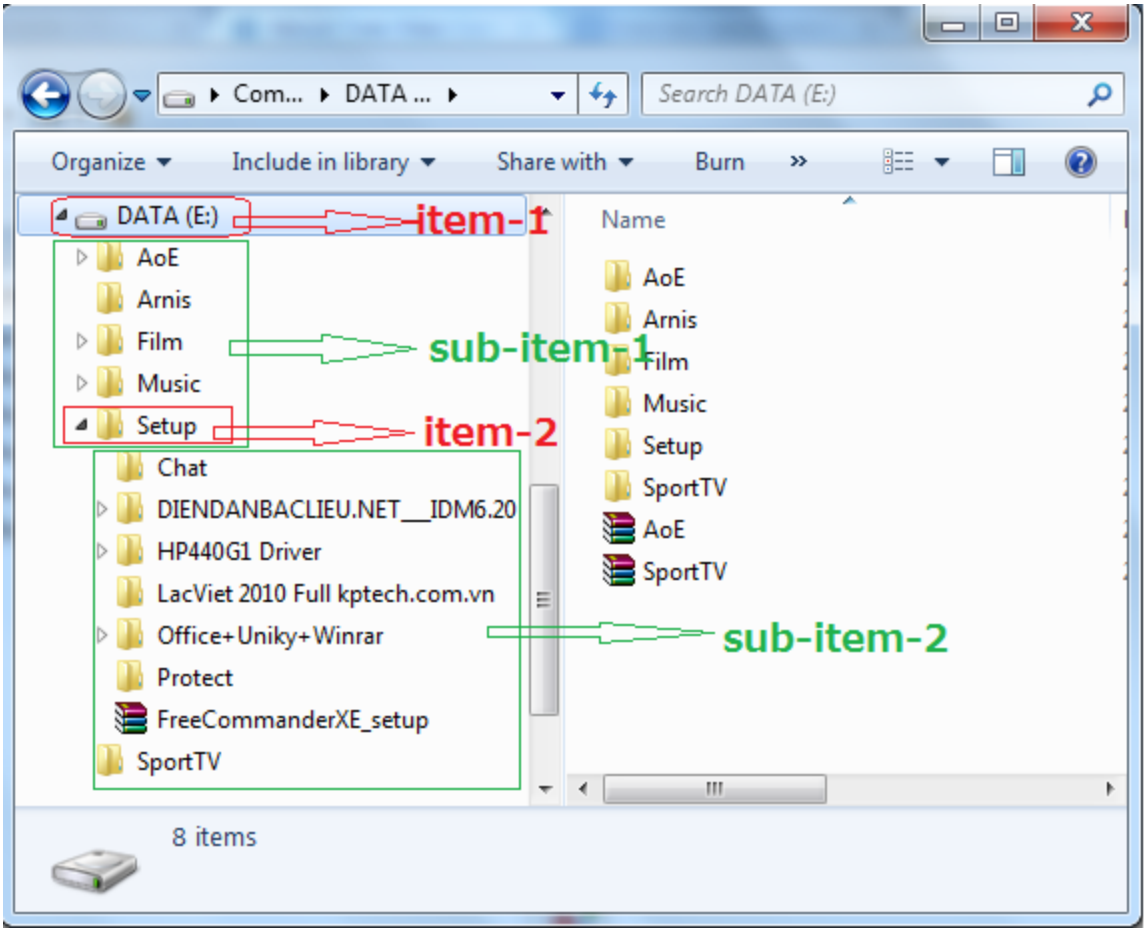


Progress bar

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.



Tree view

```
#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 nCmdSho
{
    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,
    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_BORDE
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

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

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

```
hwnd = CreateWindow( wc.lpszClassName, TEXT("Menu"), WS_OVERLAPPEDWINDOW | WS_VISIBLE, 100, 100,
```

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.

```
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.

```
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.

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

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

```
hwndTV = CreateWindowEx(0, WC_TREEVIEW, TEXT("Tree View"), WS_VISIBLE | WS_CHILD | WS_BORDER | TV
```

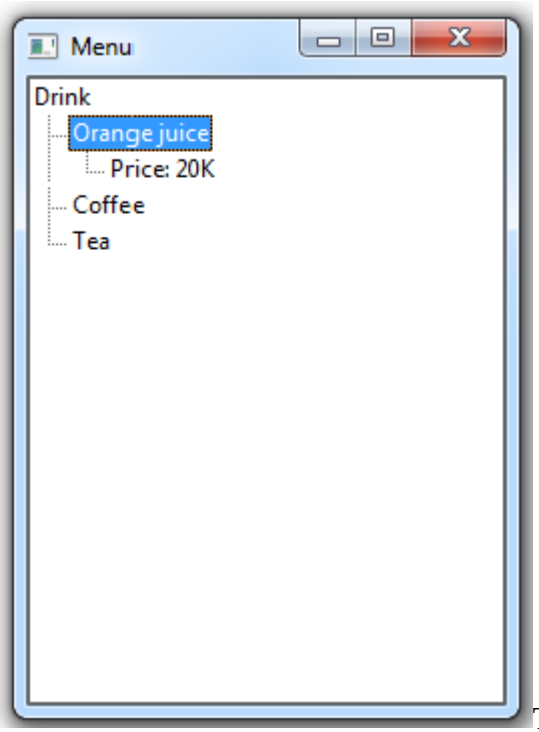
Để tạo treeview ta dùng hàm CreateWindowEx() với hằng số **WC_TREEVIEW** được truyền cho tham số lpClassName.
Trong đó:
– rcClient.right : là chiều rộng của cửa sổ cha (parent)
– rcClient.bottom : là chiều cao của cửa sổ cha (parent)
– ID_TREEVIEW : là ID của treeview (là hằng số nguyên, người lập trình 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).

```
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.
...
Kết quả:



Tree view

List box

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

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

#define IDC_LIST 1
#define IDC_STATIC 2

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

HINSTANCE g_hinst;
```

```
10
11typedef struct
12{
13    TCHAR name[30];
14    TCHAR role[20];
15} Friends;
16
17Friends friends[] =
18{
19    {TEXT("vncoding"), TEXT("Admin")},
20    {TEXT("vhnhan01"), TEXT("member")},
21    {TEXT("thanhx175"), TEXT("member")},
22    {TEXT("tienle"), TEXT("member")},
23    {TEXT("hoanghoa"), TEXT("member")},
24};
25
26int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow
27{
28    MSG msg;
29    WNDCLASS wc = {0};
30    wc.lpszClassName = TEXT( "Application" );
31    wc.hInstance = hInstance;
32    wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
33    wc.lpfnWndProc = WndProc;
34    wc.hCursor = LoadCursor(0, IDC_ARROW);
35    g_hinst = hInstance;
36
37    RegisterClass(&wc);
38    CreateWindow( wc.lpszClassName, TEXT("List Box"), WS_OVERLAPPEDWINDOW | WS_VISIBLE, 100, 100
39
40   while( GetMessage(&msg, NULL, 0, 0))
41   {
42       TranslateMessage(&msg);
43       DispatchMessage(&msg);
44   }
45   return (int) msg.wParam;
46}
47
48
49LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
50{
51    static HWND hwndList, hwndStatic;
52    int i, sel;
53    TCHAR buff[100];
54
55    switch(msg)
    {
    case WM_CREATE:
```

```
        hwndList = CreateWindow(TEXT("listbox") , NULL, WS_CHILD | WS_VISIBLE | LBS_NOTIFY, 10,

        hwndStatic = CreateWindow(TEXT("static") , NULL, WS_CHILD | WS_VISIBLE, 200, 10, 120, 45

        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:

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

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

```
hwndList = CreateWindow(TEXT("listbox") , NULL, WS_CHILD | WS_VISIBLE | LBS_NOTIFY, 10, 10, 150,

hwndStatic = CreateWindow(TEXT("static") , NULL, WS_CHILD | WS_VISIBLE, 200, 10, 120, 45, hwnd,(H
```

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

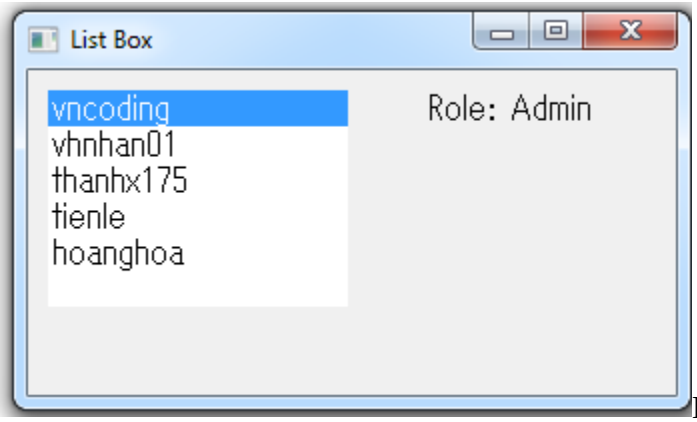
```
for (i = 0; i < ARRAYSIZE(friends); i++)
{
    SendMessage(hwndList, LB_ADDSTRING, 0, (LPARAM) friends[i].name);
}
```

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

```
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.

Kết quả:



List box

ComboBox

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

```
#include <windows.h>

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

HINSTANCE g_hinst;
```

```
6
7int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
8                    LPSTR lpCmdLine, int nCmdShow)
9{
10    HWND hwnd;
11    MSG msg ;
12    WNDCLASS wc = {0};
13    wc.lpszClassName = TEXT("Application");
14    wc.hInstance = hInstance;
15    wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
16    wc.lpfnWndProc = WndProc;
17    wc.hCursor = LoadCursor(0,IDC_ARROW);
18
19    g_hinst = hInstance;
20
21    RegisterClass(&wc);
22    hwnd = CreateWindow(wc.lpszClassName, TEXT("Combo Box"),
23                        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
24                        100, 100, 270, 170, 0, 0, hInstance, 0);
25
26
27    while( GetMessage(&msg, NULL, 0, 0))
28    {
29        DispatchMessage(&msg);
30    }
31    return (int) msg.wParam;
32}
33
34LRESULT CALLBACK WndProc( HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam )
35{
36    static HWND hwndCombo, hwndStatic;
37    const TCHAR *items[] = { TEXT("C/C++"), TEXT("C#"), TEXT("Java"), TEXT("Python") };
38    int i;
39    LRESULT sel = 0;
40
41    switch(msg)
42    {
43    case WM_CREATE:
44        hwndCombo = CreateWindow(TEXT("combobox"), NULL,
45                                 WS_CHILD | WS_VISIBLE | CBS_DROPDOWN,
46                                 10, 10, 120, 110, hwnd, NULL, g_hinst, NULL);
47
48        CreateWindow(TEXT("button"), TEXT("Drop down"),
49                           WS_CHILD | WS_VISIBLE,
50                           150, 10, 90, 25, hwnd, (HMENU)1, g_hinst, NULL);
51
52        hwndStatic = CreateWindow(TEXT("static"), TEXT(""),
53                                  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.

```
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.

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

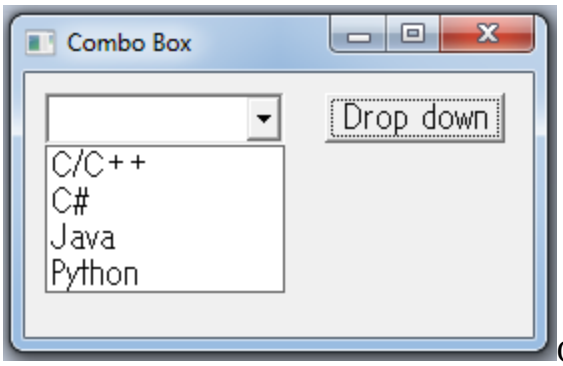
Ta fill combo box với các item. Để add 1 string tới combo box, ta gửi message CB_ADDSTRING tới combo box.

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.

```
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).

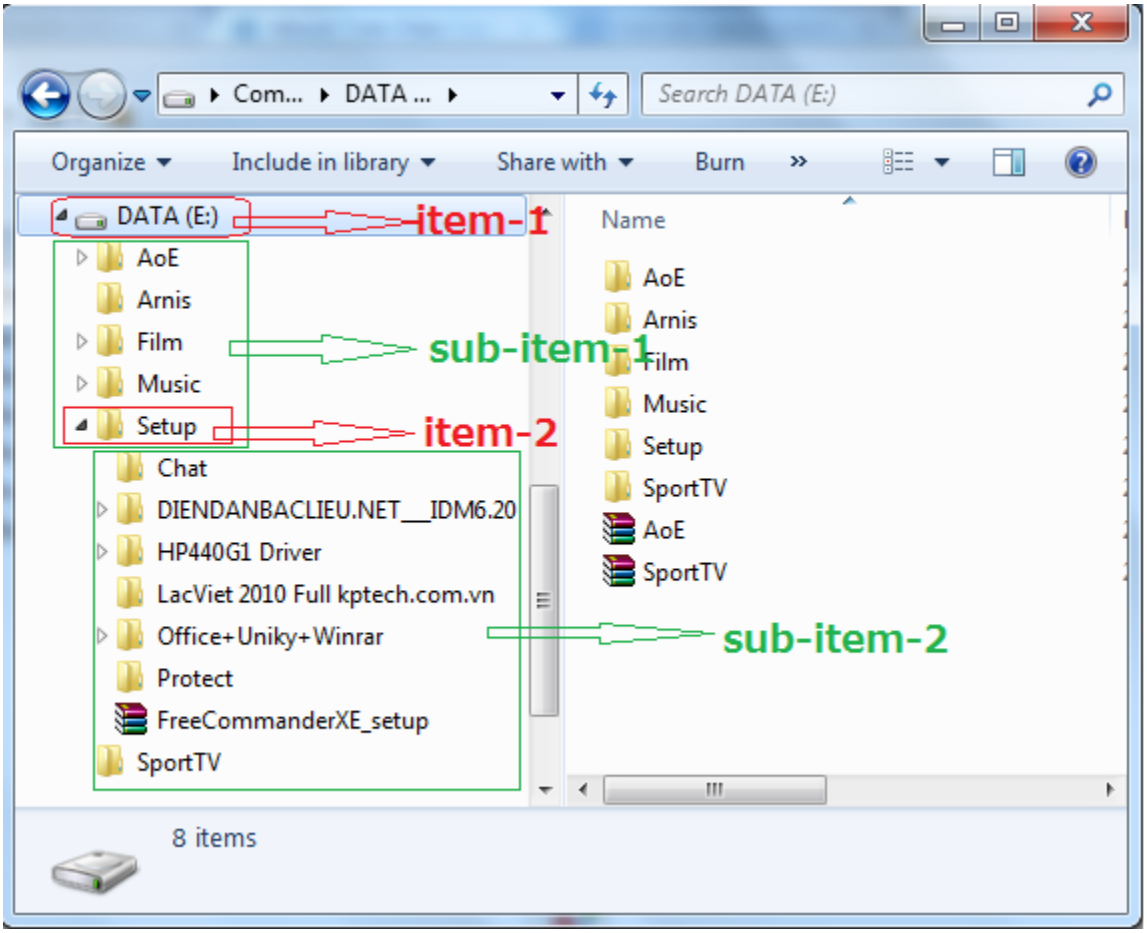
Kết quả:



Combobox

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.



Treeview

```
#include <windows.h>
#include <commctrl.h>

#pragma comment( lib, "comctl32.lib" )

#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:

```
#include <commctrl.h>
```

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

```
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.

```
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.

```
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.

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

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

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)

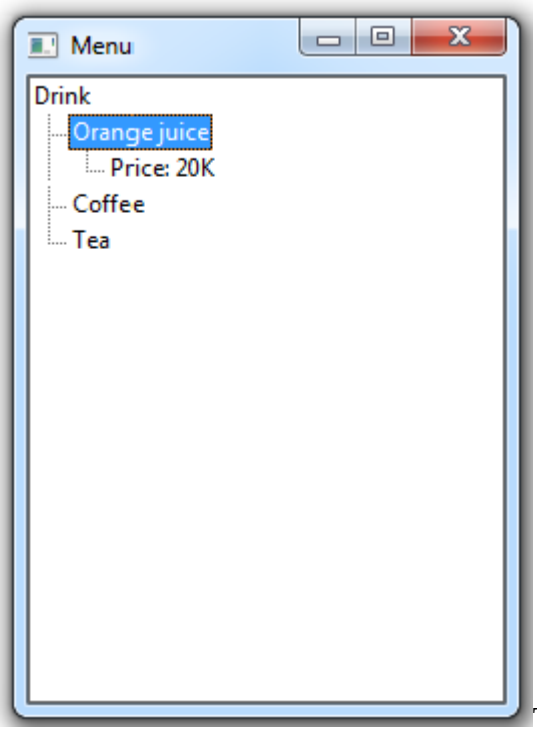
– rcClient.bottom : là chiều cao của cửa sổ cha (parent)
– ID_TREEVIEW : là ID của treeview (là hằng số nguyên, người lập trình tự định nghĩa)
– g_hInst: là biến instant global. (Trong bài này, biến này không có nhiều ý nghĩa. Tôi sẽ giới thiệu trong các bài viết sắp tới).

```
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.
...

Kết quả:



Treeview

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.

```
#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,

    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;
```

```
38  TCHAR text[250];
39  LRESULT count, id;
40  INITCOMMONCONTROLSEX icex;
41
42  switch(msg)
43  {
44  case WM_CREATE:
45      icex.dwSize = sizeof(INITCOMMONCONTROLSEX);
46      icex.dwICC = ICC_TAB_CLASSES;
47      InitCommonControlsEx(&icex);
48      hTab = CreateWindow(WC_TABCONTROL, NULL, WS_CHILD | WS_VISIBLE, 0, 0, 200, 150, hwnd,(HM
49      hEdit = CreateWindow(L"edit",NULL,WS_CHILD | WS_VISIBLE | WS_BORDER, 250, 20, 100, 25, h
50
51      CreateWindow(L"button",L"Add", WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 250, 50, 100, 25,
52
53      CreateWindow(L"button", L"Del", WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 250, 80, 100, 25,
54
55      CreateWindow(L"button",L"Delall", WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 250, 110, 100,
56      break;
57  case WM_COMMAND:
58      switch(LOWORD(wParam))
59      {
60          case BTN_ADD:
61              GetWindowText(hEdit, text, 250);
62              if (lstrlen(text) !=0 )
63              {
64                  tie.mask = TCIF_TEXT;
65                  tie.pszText = text;
66                  count = SendMessage(hTab, TCM_GETITEMCOUNT, 0, 0);
67                  SendMessage(hTab, TCM_INSERTITEM, count, (LPARAM) (LPTCITEM) &tie);
68              }
69              break;
70          case BTN_DEL:
71              id = SendMessage(hTab, TCM_GETCURSEL, 0, 0);
72              if (id != -1)
73              {
74                  SendMessage(hTab, TCM_DELETEITEM, 0, id);
75              }
76              break;
77          case BTN_DELALL:
78              SendMessage(hTab, TCM_DELETEALLITEMS, 0, 0);
79              break;
80      }
81      break;
82  case WM_DESTROY:
83      PostQuitMessage(0);
84      break;
85  }
```

```
84      return(DefWindowProc(hwnd, msg, wParam, lParam));
85  }
86
87
88
89
```

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.

```
1 hTab = CreateWindow(WC_TABCONTROL, NULL, WS_CHILD | WS_VISIBLE, 0, 0, 200, 150, hwnd,(HMENU) ID_T
```

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

```
1 if (lstrlen(text) !=0 )
2 {
3     tie.mask = TCIF_TEXT;
4     tie.pszText = text;
5     count = SendMessage(hTab, TCM_GETITEMCOUNT, 0, 0);
6     SendMessage(hTab, TCM_INSERTITEM, count, (LPARAM) (LPTCITEM) &tie);
7 }
```

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.

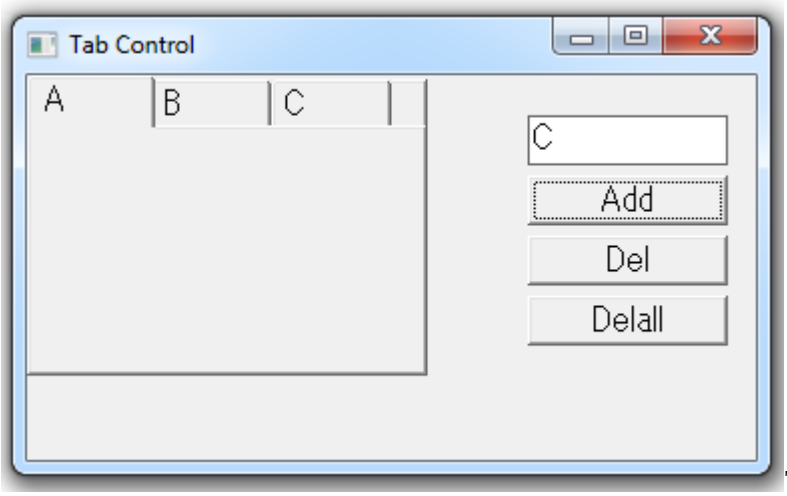
```
1 id = SendMessage(hTab, TCM_GETCURSEL, 0, 0);
2 if (id != -1)
3 {
4     SendMessage(hTab, TCM_DELETEITEM, 0, id);
5 }
```

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.

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

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

Kết quả:



Tab control

Tooltip

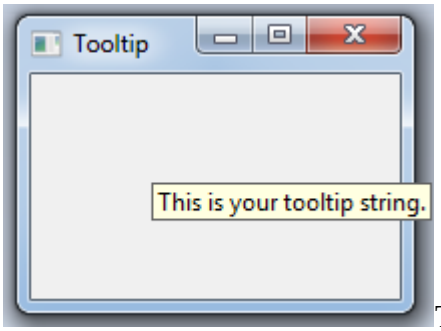
Tooltip được ẩn đi trong hầu hết thời gian. Nó là 1 box nhỏ xuất hiện gần GUI khi con trỏ chuột di chuyển qua nó. Nó sẽ hiển thị message giải thích cho GUI đó. Tooltip được sử dụng chính trong phần help của ứng dụng.

Code:

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

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
void CreateMyTooltip(HWND);
HINSTANCE g_hInst;
int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow
{
    MSG msg;
    WNDCLASS wc = {0};
    wc.lpszClassName = L"Tooltip" ;
    wc.hInstance = hInstance ;
    wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
    wc.lpfnWndProc = WndProc ;
    wc.hCursor = LoadCursor(0, IDC_ARROW);

    RegisterClass(&wc);
    CreateWindow( wc.lpszClassName, L"Tooltip", WS_OVERLAPPEDWINDOW | WS_VISIBLE, 100, 100, 200,

```

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

void CreateMyTooltip (HWND hwnd)
{
    HWND hwndTT = CreateWindowEx(WS_EX_TOPMOST, TOOLTIPS_CLASS, NULL, WS_POPUP | TTS_NOPREFIX |
g_hInst,NULL);

    SetWindowPos(hwndTT, HWND_TOPMOST, 0, 0, 0, 0, SWP_NOMOVE | SWP_NOSIZE | SWP_NOACTIVATE);

    // Set up "tool" information. In this case, the "tool" is the entire parent window.
    TOOLINFO ti = { 0 };
    ti.cbSize = sizeof(TOOLINFO) - 4;
    ti.uFlags = TTF_SUBCLASS;
    ti.hwnd = hwnd;
    ti.hinst = g_hInst;
    ti.lpszText = TEXT("This is your tooltip string.");;

    GetClientRect (hwnd, &ti.rect);

    // Associate the tooltip with the "tool" window.
    SendMessage(hwndTT, TTM_ADDTOOL, 0, (LPARAM) (LPTOOLINFO) &ti);
}
```

Kết quả:



ToolTip

Burning control

Burning control là cửa sổ rất hay gặp trong các ứng dụng như: Nero,...

Code:

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

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

HINSTANCE g_hinst;
LRESULT g_pos = 150;

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
 PWSTR lpCmdLine, int nCmdShow)
{
    HWND hwnd;
    MSG msg;
    WNDCLASSW wc = {0};

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

    g_hinst = hInstance;

    RegisterClassW(&wc);
```

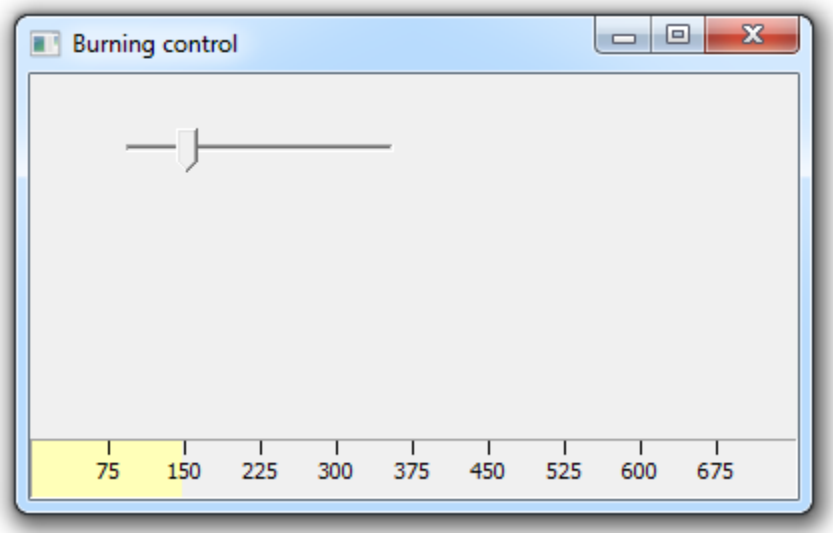
```
    hwnd = CreateWindowW(wc.lpszClassName, L"Burning control", WS_OVERLAPPEDWINDOW | WS_VISIBLE
    while( GetMessage(&msg, NULL, 0, 0))
    {
        DispatchMessage(&msg);
    }
    return (int) msg.wParam;
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    static HWND hwndTrack, hwndBurn;
    WNDCLASSW rwc = {0};

    INITCOMMONCONTROLSEX InitCtrlEx;

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

    switch(msg)
    {
    case WM_CREATE:
        rwc.lpszClassName = L"BurningControl";
        rwc.hbrBackground = GetSysColorBrush(COLOR_BTNFACE);
        rwc.style = CS_HREDRAW;
        rwc.lpfnWndProc = PanelProc;
        rwc.hCursor = LoadCursor(0, IDC_ARROW);
        RegisterClassW(&rwc);

        hwndBurn = CreateWindowExW(WS_EX_STATICEDGE , L"BurningControl", NULL, WS_CHILD | WS_VI

        hwndTrack = CreateWindowExW(0, TRACKBAR_CLASSW, NULL, WS_CHILD | WS_VISIBLE | TBS_FIXED

        SendMessage(hwndTrack, TBM_SETRANGE, TRUE, MAKELONG(0, 750));
        SendMessage(hwndTrack, TBM_SETPAGESIZE, 0, 20);
        SendMessage(hwndTrack, TBM_SETTICFREQ, 20, 0);
        SendMessage(hwndTrack, TBM_SETPOS, TRUE, 150);
        break;

    case WM_SIZE:
        SetWindowPos(hwndBurn, NULL, 0, HIWORD(lParam)-30, LOWORD(lParam), 30, SWP_NOZORDER);
        break;
    case WM_HSCROLL:
        g_pos = SendMessage(hwndTrack, TBM_GETPOS, 0, 0);
        InvalidateRect(hwndBurn, NULL, TRUE);
        break;
    case WM_DESTROY:
```

```
72          PostQuitMessage(0);
73          break;
74      }
75      return DefWindowProcW(hwnd, msg, wParam, lParam);
76  }
77
78
79  LRESULT CALLBACK PanelProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
80  {
81      HBRUSH hBrushYellow, hBrushRed, holdBrush;
82      HPEN hPen, holdPen;
83      HFONT hFont, holdFont;
84      PAINTSTRUCT ps;
85      RECT rect, rect2;
86
87      wchar_t *cap[] = { L"75", L"150", L"225", L"300", L"375", L"450", L"525", L"600", L"675"};
88
89      HDC hdc;
90      int till;
91      int step, full;
92      int i;
93
94      switch(msg)
95      {
96      case WM_PAINT:
97          hdc = BeginPaint(hwnd, &ps);
98          GetClientRect(hwnd, &rect);
99          till = (rect.right / 750.0) * g_pos;
100         step = rect.right / 10.0;
101         full = (rect.right / 750.0) * 700;
102
103         hBrushYellow = CreateSolidBrush(RGB(255, 255, 184));
104         hBrushRed = CreateSolidBrush(RGB(255, 110, 110));
105
106         hPen = CreatePen(PS_NULL, 1, RGB(0, 0, 0));
107         holdPen = (HPEN)SelectObject(hdc, hPen);
108
109         hFont = CreateFontW(13, 0, 0, 0, FW_MEDIUM, 0, 0, 0, 0, 0, 0, 0, 0, L"Tahoma");
110         holdFont = (HFONT)SelectObject(hdc, hFont);
111
112         if(till > full)
113         {
114             SelectObject(hdc, hBrushYellow);
115             Rectangle(hdc, 0, 0, full, 30);
116             holdBrush = (HBRUSH)SelectObject(hdc, hBrushRed);
117             Rectangle(hdc, full, 0, till, 30);
        }
        else
```

```
118         {
119             holdBrush = (HBRUSH)SelectObject(hdc, hBrushYellow);
120             Rectangle(hdc, 0, 0, till, 30);
121         }
122         SelectObject(hdc, holdPen);
123 
124         for ( i = 1; i < 10; i++)
125         {
126             MoveToEx(hdc, i*step, 0, NULL);
127             LineTo(hdc, i*step, 7);
128             rect2.bottom = 28;
129             rect2.top = 8;
130             rect2.left = i*step-10;
131             rect2.right = i*step+10;
132 
133             SetBkMode(hdc, TRANSPARENT) ;
134             DrawTextW(hdc, cap[i-1], wcslen(cap[i-1]), &rect2, DT_CENTER);
135         }
136 
137         SelectObject(hdc, holdBrush);
138         DeleteObject(hBrushYellow);
139         DeleteObject(hBrushRed);
140         DeleteObject(hPen);
141 
142         SelectObject(hdc, holdFont);
143         DeleteObject(hFont);
144 
145         EndPaint(hwnd, &ps);
146         break;
147     }
148 
149     return DefWindowProcW(hwnd, msg, wParam, lParam);
150 }
151 
152 
153 
154 
155 
```

Giải thích:
Trong ví dụ này, ứng dụng bao gồm trackbar và burning control. Burning control được vẽ trong WM_PAINT message. Text, line,background trên burning control được vẽ bằng các hàm GDI

Kết quả:



Burning Control

Graphic Device Interface (GDI)

GDI là giao diện làm việc đồ họa. GDI được sử dụng để tương tác với các thiết bị đồ họa như màn hình, máy in hoặc file. GDI cho phép người lập trình hiển thị dữ liệu lên màn hình hoặc máy in mà không cần quan tâm tới đặc tính kĩ thuật của thiết bị (màn hình, máy in,...). Theo quan điểm người lập trình, GDI là 1 thư viện các API function làm việc với đồ họa. GDI bao gồm các vecto đồ họa 2D, Fonts và Image. Để bắt đầu vẽ đồ họa, chúng ta phải lấy được đối tượng "ngữ cảnh thiết bị" (device context).

Pixel

```
#include "windows.h"

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

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR lpCmdLine
{
    MSG msg;
```

```
    WNDCLASSW wc = {0};

    wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.lpszClassName = L"Pixels";
    wc.hInstance = hInstance;
    wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
    wc.lpfnWndProc = WndProc;
    wc.hCursor = LoadCursor(0, IDC_ARROW);

    RegisterClassW(&wc);
    CreateWindowW(wc.lpszClassName, L"Pixels", WS_OVERLAPPEDWINDOW | WS_VISIBLE,
hInstance, NULL);

    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_PAINT:
        DrawPixels(hwnd);
        break;
    case WM_DESTROY:
```

```
        PostQuitMessage(0);
        return 0;
    }
    return DefWindowProcW(hwnd, msg, wParam, lParam);
}

void DrawPixels(HWND hwnd)
{
    PAINTSTRUCT ps;
    RECT r;

    HDC hdc = BeginPaint(hwnd, &ps);
    GetClientRect(hwnd, &r);
    for (int i=0; i<1000; i++)
    {
        int x = (rand() % r.right - r.left);
        int y = (rand() % r.bottom - r.top);
        SetPixel(hdc, x, y, RGB(255, 0, 0));
    }
    EndPaint(hwnd, &ps);
}
```

Giải thích:
Hàm SetPixel() là hàm để vẽ 1 pixel lên cửa sổ ứng dụng.
Trong ví dụ này, chúng ta sử dung hàm SetPixel() để vẽ 1000 pixel màu đỏ lên cửa sổ ứng dụng.

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

Với 2 cờ CS_HREDRAW và CS_VREDRAW, cửa sổ sẽ được vẽ lại khi thay đổi kích thước

```
case WM_PAINT:
    DrawPixels(hwnd);
```

```
3       break;
```

Việc vẽ pixel được thực hiện trong message WM_PAINT. Chương trình sẽ gọi DrawPixels() để vẽ 1000 pixel.

```
1HDC hdc = BeginPaint(hwnd, &ps);
```

Hàm BeginPaint() chuẩn bị vùng cửa sổ để vẽ pixel. Hàm này sẽ điền thông tin vào struct PAINTSTRUCT. Và hàm trả về handle tới device context(thiết bị ngữ cảnh) của ứng dụng.

```
1GetClientRect(hwnd, &r);
```

Hàm GetClientRect() kích thước (tọa độ) vùng cửa sổ cần vẽ pixel. Thông tin về tọa độ cửa sổ được lưu trong struct RECT r.

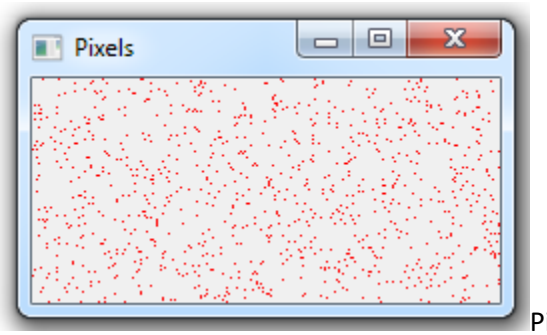
```
1for (int i=0; i<1000; i++)
2{
3    int x = (rand() % r.right - r.left);
4    int y = (rand() % r.bottom - r.top);
5    SetPixel(hdc, x, y, RGB(255, 0, 0));
6}
```

Hàm SetPixel vẽ pixel đỏ (RGB(255, 0, 0)) lên cửa sổ.

```
1EndPaint(hwnd, &ps);
```

Khi vẽ xong, chúng ta gọi hàm EndPaint() để giải phóng bộ nhớ chứa thiết bị ngữ cảnh (device context) mà hàm BeginPaint() đã lấy.

Kết quả:



Pixels

Lập Trình Win32 API - Phần 1: Tổng Quan

 NGUYÊN PHAN

18/09/2020•3 MIN READ

Khởi tạo project dụng Win32 API xây dựng ứng dụng giao diện Windows Form với C++.

Win32 API là gì?

API viết tắt của Application Programming Interface - Giao diện lập trình ứng dụng. Là các phương thức, giao thức kết nối với các thư viện và ứng dụng.

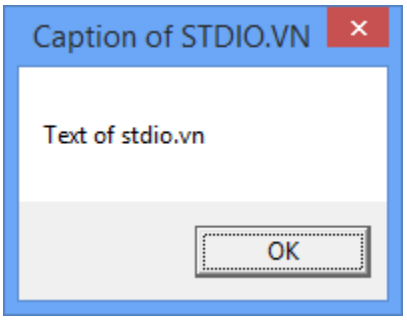
Win32 viết tắt của Windows 32-bit, cung cấp một số hàm, thủ tục giúp tạo ra một chương trình.

Các thuật ngữ

- Handle: quản lý một đối tượng cụ thể, thao tác với đối tượng đó thông qua handle.
- Window: giao diện cửa sổ của một "Control", trong các ứng dụng khác có thể là khái niệm Control (button, label, ...), với Win32, Window chính là tên gọi chung.
- Message: thông điệp gửi đến máy tính qua các thao tác trên máy như click chuột, nhấn bàn phím,...
- Menu: thể hiện một danh sách đối tượng
- Queue: cấu trúc dữ liệu tổ chức theo nguyên tắc First-In-First-Out.

Hướng dẫn

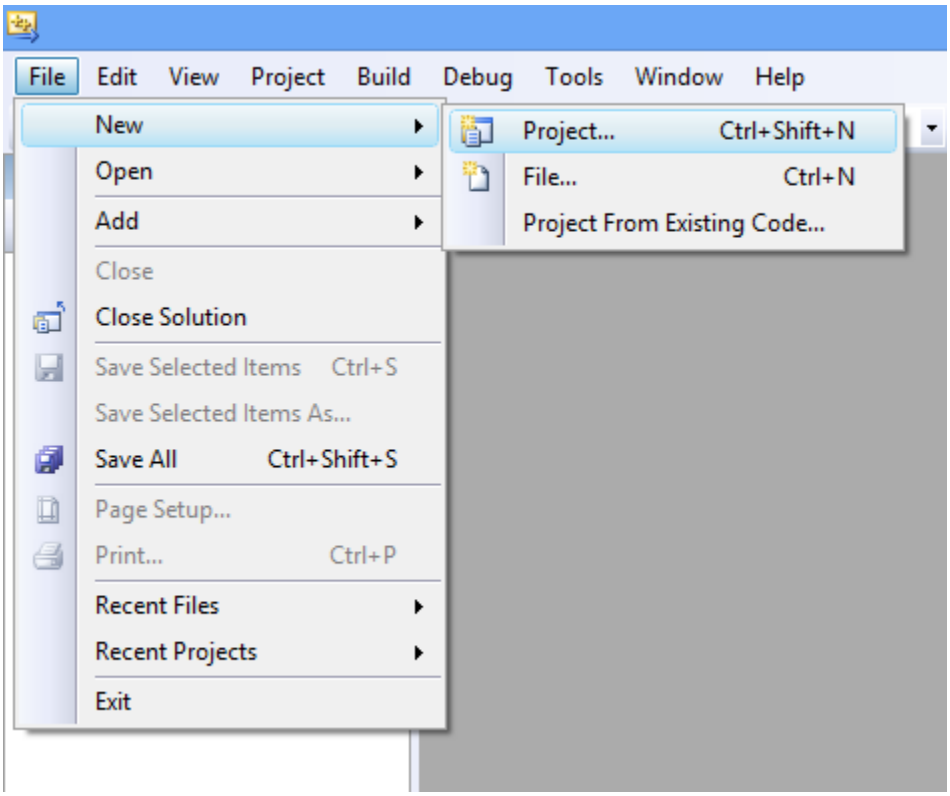
Bài viết hướng dẫn tạo một cửa sổ như sau



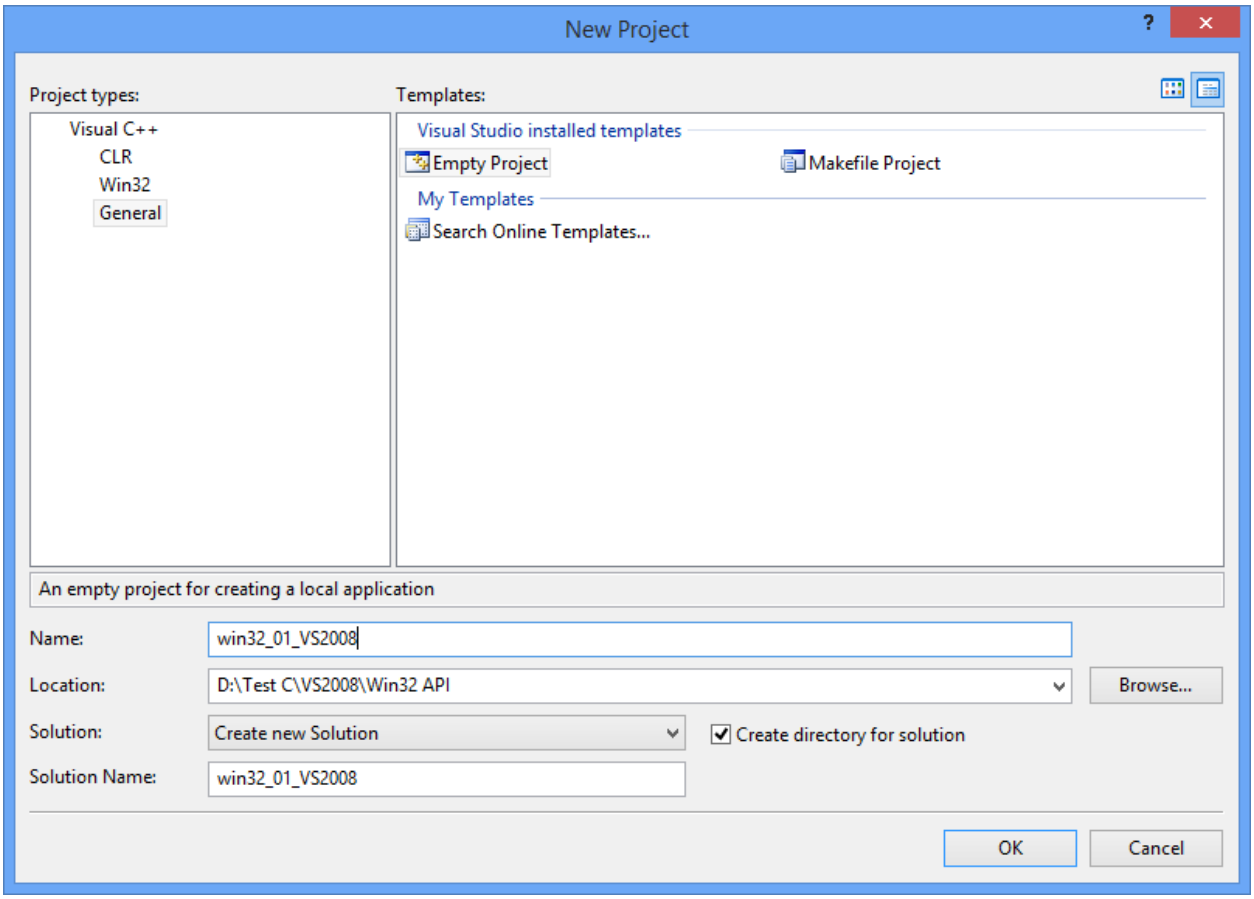
Tạo project

Tạo một Empty Project, các hình ảnh sau sử dụng **Visual Studio 2008**, có thể làm tương tự với **Visual Studio 2019** hoặc các phiên bản khác.

Vào `File -> New -> Project` hoặc dùng tổ hợp phím `Ctrl + Shift + N`

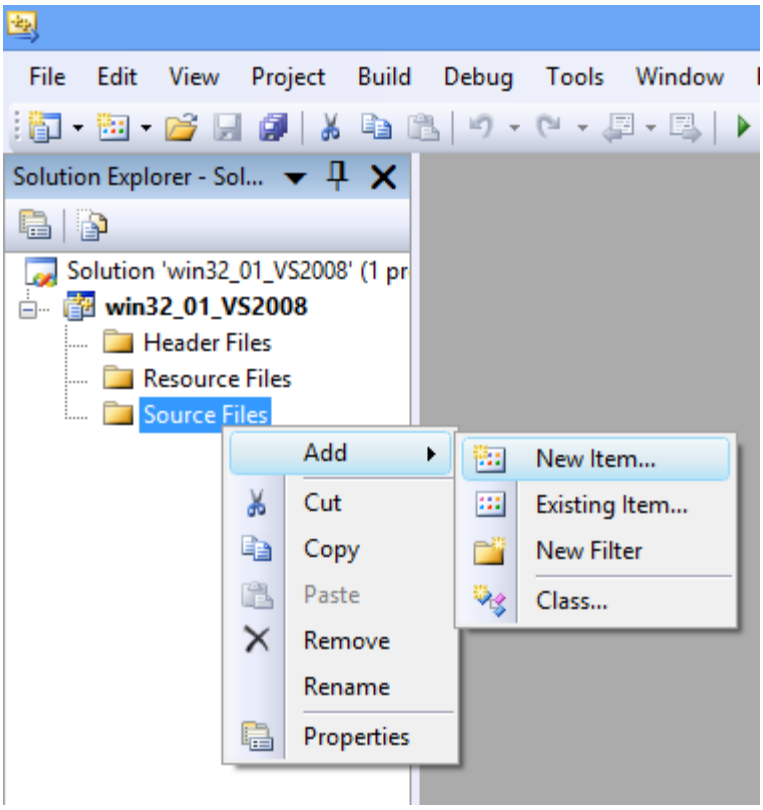


Vào mục `General`, chọn `Empty project` (với **VS2013**, chọn `Visual C++ -> General -> Empty project`). Sau khi chọn xong, đặt tên cho project, cụ thể ở đây là **win32_01_VS2008**.

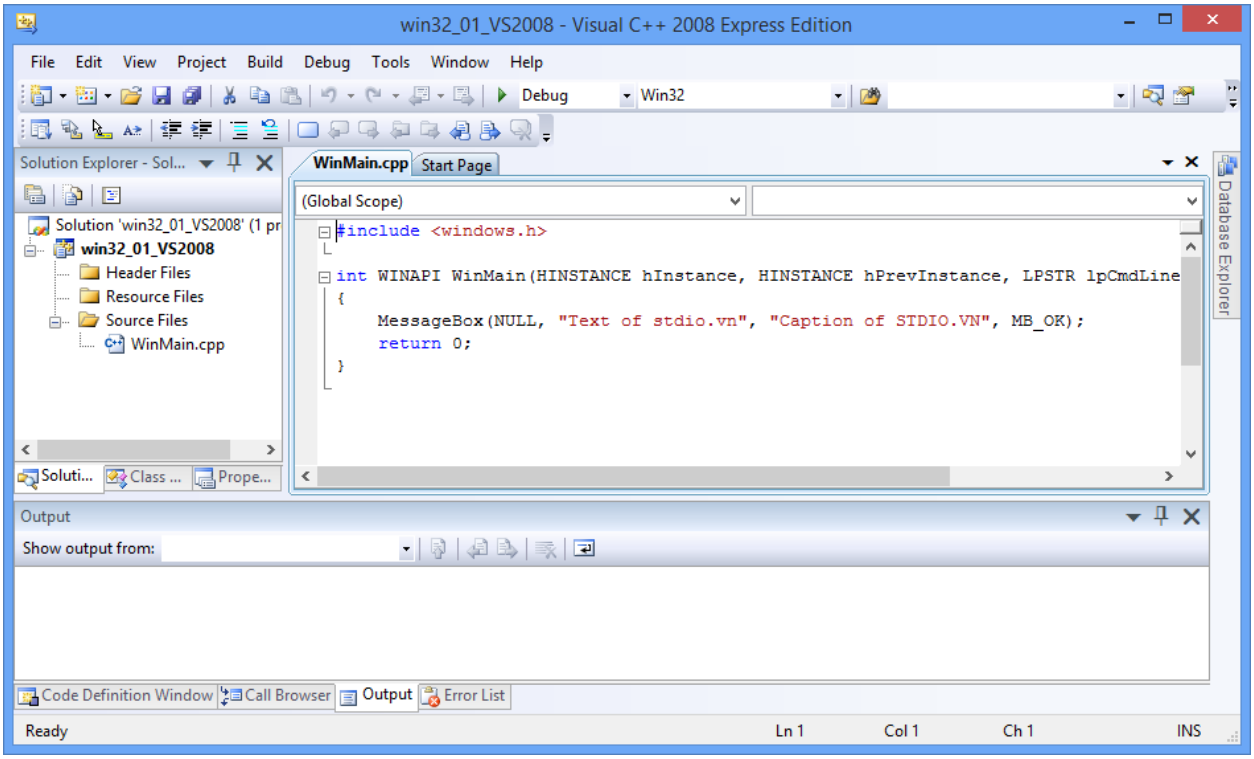


Trên thanh **Solution Explorer** nằm bên tay trái (hoặc chọn `View -> Solution Explorer`).

Nhấn chuột phải vào `Source Files -> Add -> New Item`.



Ở Categories, chọn `Visual C++ -> Code -> C++ file (.cpp)` và đặt tên cho nó. Ở đây là `WinMain`, nhằm phân biệt với `main.cpp` khi thao tác trên ứng dụng Console.



Thao tác trên WinMain

Tạo file `WinMain.cpp` với nội dung như sau.

```
#include <Windows.h>

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdL
ine, int nCmdShow)
{
        MessageBox(NULL, "Text of stdio.vn", "Caption of STDIO.VN", MB_OK);
        return 0;
}
```

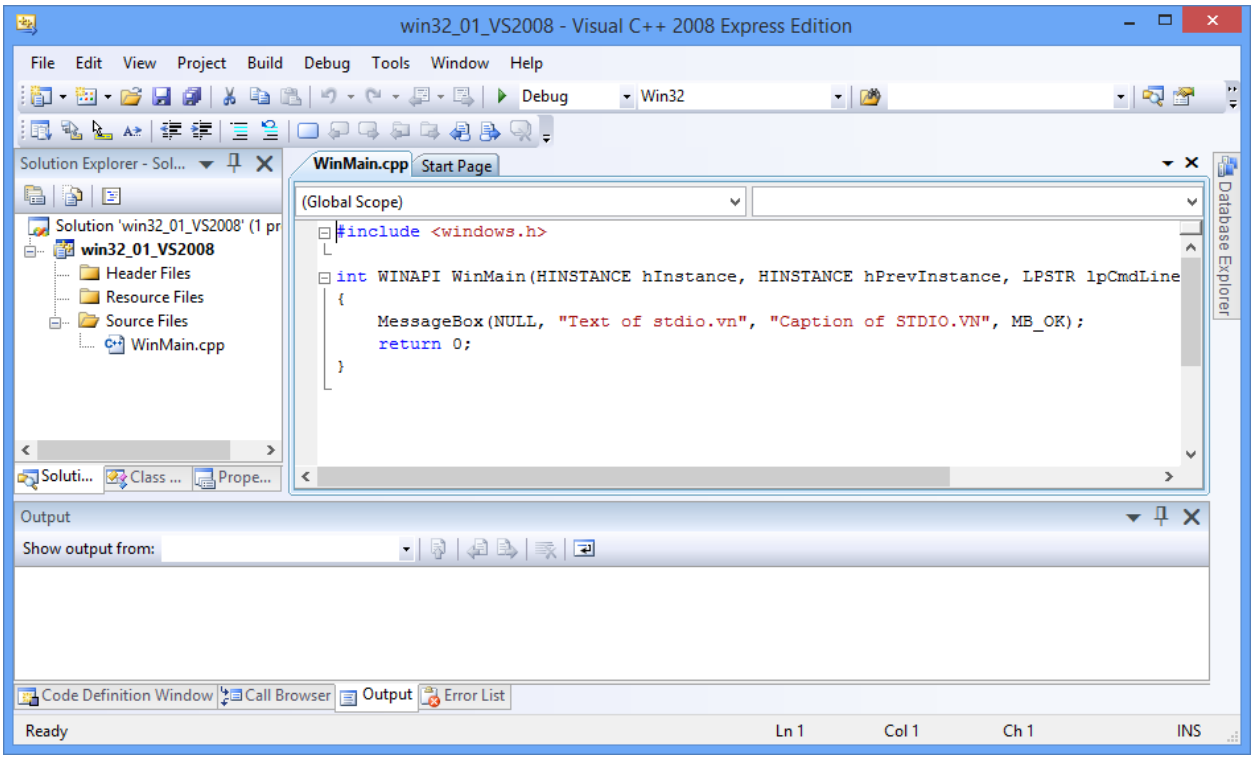
Trong đoạn code trên:

- `Windows.h`: Thư viện Win32 API mà windows cung cấp.
- `int WinMain()`: Tương tự như hàm `main()` trước đây, nhưng thể hiện ra window.

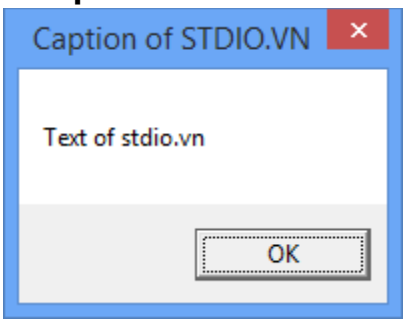
Phân tích các thành phần trong `WinMain`:

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

- `HINSTANCE hInstance`: Quản lý việc thực thi của chương trình, nó quản lý tài nguyên của chương trình đang cài đặt, cụ thể là file .exe trong hệ thống (tìm hiểu sau)
- `HINSTANCE hPrevInstance`: Con trỏ trỏ đến việc thực thi trước đó.
- `LPSTR lpCmdLine`: Command line arguments, chuỗi ký tự bình thường, không được chứa tên chương trình
- `int nCmdShow`: 1 số nguyên để cung cấp cho hàm `ShowWindow()`.
- Hàm `MessageBox(HWND hwnd, LPCSTR lpText, LPCSTR lpCaption, UINT uType)`: Hộp tin nhắn từ hệ thống, thông tin hệ thống gửi đến người dùng:
    - `hwnd`: Con trỏ trỏ vào một window nào đó.
    - `lpText`: Đoạn văn bản thể hiện ở thân `MessageBox`.
    - `lpCaption`: Tương tự lpText nhưng là văn bản nằm ở trên thanh trên cùng.
    - `uType`: Thể hiện kiểu của MessageBox, ví dụ như ở đây là `MB_OK`. Trỏ vào `MB_OK`, đây là một con số (`0x00000000L`). Trong thư viện windows, thư viện định nghĩa `MB_OK` giúp dễ nhớ hơn.



Kết quả



Khi theo dõi loạt bài viết sau đây, sẽ bắt gặp vài loại biến khác lạ như `UINT` định nghĩa của kiểu `unsigned int` – số nguyên không dấu hay `LPSTR` cho chuỗi `char*`, ...

Lập Trình Win32 API - Phần 2: Tạo Cửa Sổ Cơ Bản

NGUYÊN PHAN

18/09/2020•7 MIN READ

Hệ điều hành Windows cung cấp các cửa sổ Window. Có thể sử dụng Win32 API để tạo các cửa sổ này, là nền tảng cho ứng dụng về mặt UI.

Hướng dẫn

Cài đặt ban đầu

Tạo một project tương tự như ở phần tạo project.

Thao tác trên file `WinMain` (file `WinMain` đang rỗng):

```
#include <windows.h>

const char* ClassName = "Win32 API from stdio.vn";

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

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdL
ine, int nCmdShow)
{
}
```

Các cài đặt bao gồm:

- `ClassName`: tên chương trình, đặt tên ClassName là vì sau này khi thao tác với nhiều "đối tượng" khác nhau thì mỗi đối tượng thường có các window khác nhau, giống như một chương trình có nhiều cửa sổ, nhấn nút này sẽ nhảy ra cửa sổ khác.
- `WndProc`: thủ tục tạo window.
- `WinMain`: đã giải thích ở bài viết trước.

Bước 1: tạo một Window Class

Trong hàm `WinMain()`

```
WNDCLASSEX wc;

wc.cbSize       = sizeof(WNDCLASSEX);
wc.style        = 0;
wc.lpfnWndProc  = WndProc;
wc.cbClsExtra   = 0;
wc.cbWndExtra   = 0;
wc.hInstance    = hInstance;
wc.hIcon        = LoadIcon(NULL, IDI_APPLICATION);
wc.hCursor      = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground        = (HBRUSH)(COLOR_WINDOW + 1);
wc.lpszMenuName= NULL;
wc.lpszClassName        = ClassName;
wc.hIconSm      = LoadIcon(NULL, IDI_APPLICATION);

if (!RegisterClassEx(&wc))
{
        MessageBox(NULL, "Cannot Register window", "Error", MB_ICONEXCLAMATION
| MB_OK);
        return 0;
}
```

Trong các hàm trên:

- `wc.cbSize`: kích thước trên bộ nhớ của window class.
- `wc.style`: kiểu của Class, khác với kiểu của window (window style), thường có giá trị 0.
- `wc.lpfnWndProc`: thủ tục của window (window procedure) là con trỏ trỏ tới `WinProc`.
- `wc.cbClsExtra`, `wc.cbWndExtra`: số lượng dữ liệu tối đa được cài đặt cho class, thường có giá trị 0.

- `wc.hInstance`: quản lý thông tin của cửa sổ, tương đương với giá trị khai báo ở `WinMain()`.
- `wc.hIcon`: icon lớn của class, như đoạn code ở trên khai báo là icon có sẵn trong hệ thống.
- `wc.hCursor`: con trỏ.
- `wc.hbrBackground`: màu nền.
- `wc.lpszMenuName`: con trỏ trỏ về dữ liệu của các thanh menu.
- `wc.lpszClassName`: tên của class, như trên là cài đặt tên có sẵn khai báo ở đầu bài.
- `wc.hIconSm`: tương tự như icon nhưng là icon nhỏ, hiện ở bên trái cùng chương trình.

`if` để kiểm tra việc đăng ký `Window Class` có thành công không, nếu không thành công thì hiện một cửa sổ thông báo với nội dung như trên.

Bước 2: sử dụng Window Class ở trên để tạo cửa sổ

```
HWND hwnd;
hwnd = CreateWindowEx(
                WS_EX_CLIENTEDGE,
                ClassName,
                "The title of stdio.vn",
                WS_OVERLAPPEDWINDOW,
                CW_USEDEFAULT,
                CW_USEDEFAULT,
                240,
                120,
                NULL,
                NULL,
                hInstance,
                NULL
                );
if (hwnd == NULL)
{
        MessageBox(NULL, "Window Creation Failed!", "Error!", MB_ICONEXCLAMATI
ON | MB_OK);
        return 0;
}

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

Hàm trên dùng để đăng ký một `handle` cho `Window Class` ta vừa dựng. Đồng nghĩa việc tạo một `handle` mà khi tương tác đến `handle` này thì cũng có nghĩa ta đang tương tác với chính `window class` đó.

Trong các hàm trên:

- Vế 1: 1 kiểu mở rộng của window style.
- Vế 2: Tên của Class, để hệ thống hiểu được ta đang thao tác với Class nào.
- Vế 3: Tiêu đề (Title Bar hoặc Caption ở bài viết 1).
- Vế 4: 1 kiểu của window style (tìm hiểu sau).
- Vế 5, vế 6: Tọa độ X, Y khi phần mềm bắt đầu (góc bên trái trên cùng), giá trị `CW_USEDEFAULT` là để hệ thống tự chọn giá trị cho nó.
- Vế 7, vế 8: Chiều rộng và chiều cao của window.
- Vế 9: Cửa sổ "cha" của cửa sổ này. Khi đi sâu vào loạt bài viết này thì sẽ có khái niệm cha và con, hiểu đơn giản, trong một cửa sổ có nhiều thành phần. Ví dụ : 1 cửa sổ, trong cửa sổ đó có 1 nút thì nút đó là "con" của cửa sổ còn cửa sổ đó là "cha" của nút. Ở đây vì đây là cửa sổ đầu tiên (Top level) nên giá trị này là `NULL`.
- Vế 10: Handle của menu chương trình, đặt `NULL` vì chưa có.
- Vế 11: Instance của chương trình.
- Vế 12: Con trỏ dùng để gửi các thông tin bổ sung.

Tương tự bước 2, `if` để kiểm tra việc đăng ký có thành công không.

Hàm `ShowWindow()` để hiện thị cửa sổ, `UpdateWindow()` để cập nhật, làm mới cho cửa sổ vừa tạo.

Bước 3: tạo vòng lặp message

Sau khi mọi thứ ở trên được khởi tạo. Vòng lặp Message sẽ "bắt" các thông tin "đối xử" với window và thực hiện nó.

Sử dụng vòng lặp vì khi thao tác trên console, bạn nhập lệnh hoặc giá trị rồi Enter, giá trị sẽ được đưa vào hệ thống ta gọi đó là lệnh. Với Win32 API, các lệnh này xảy ra liên tục nên nó được đưa vào hàng đợi (message queue). Hệ thống sẽ lấy những lệnh kế tiếp để thực thi và vòng lặp message sẽ làm việc này.

```
MSG Msg;

while (GetMessage(&Msg, NULL, 0, 0) > 0)
{
        TranslateMessage(&Msg);
```

```
        DispatchMessage(&Msg);
}
return Msg.wParam;
```

Hàm trên gồm:

- `GetMessage()` là hàm lấy thông tin từ message queue, khi tương tác với hệ thống, các tương tác được máy hiểu là các message và lưu vào một queue trong hệ thống, vòng lặp sẽ bắt các message này từ hệ thống.
- `TranslateMessage()` – Hàm này sẽ dịch message, hiểu đơn giản nếu bạn gõ chữ thì chắc chắn bạn sẽ gõ phím, việc dịch sẽ tương tự như vậy.
- `DispatchMessage()` – Hàm này sẽ xác định message này được gửi đến cửa sổ nào và thực thi nó, ví dụ như bạn có 2 cái cửa sổ A và B thì khi bạn tương tác lên cửa sổ A, A thì hàm sẽ thực hiện nó với A còn B thì không.

Việc trả về `Msg.wParam` là để phục vụ cho hệ thống hiểu được tại sao lại dừng vòng lặp.

Đây làm bước cuối cùng thao tác trên `WinMain()`, sau khi hết hàm, đóng khung lại (' } ').

Bước 4: thủ tục window – Window Procedure

Thủ tục window giống như bộ não của chương trình vậy, sau khi bắt thông tin ở vòng lặp, `DispatchMessage()` sẽ gửi tin này đến window procedure của window ta đang tương tác.

Hàm bên dưới là để hiện thực khai báo `WndProc` ở đầu đề bài:

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

Hàm trên có:

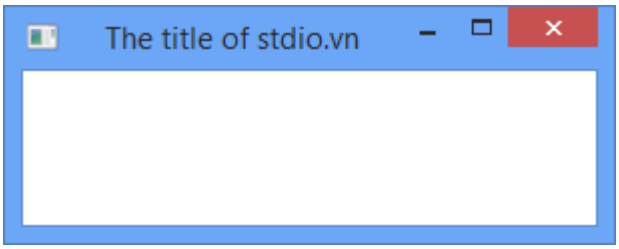
- Hàm switch là để xác định `Message` này là message như thế nào để xử lý. Như ví dụ ở trên nếu là message `Alt + F4` thì sẽ thực hiện như bên dưới.
- `DestroyWindow()` Hàm sẽ gửi câu lệnh `WM_DESTROY` đến `message queue`.
- Khi `WM_DESTROY` được thực hiện, nó sẽ xóa các "con" của cửa sổ này trước khi thực hiện các thao tác kế tiếp.
- Hàm `PostQuitMessage(0)` – Gửi lệnh `WM_QUIT` đến hệ thống.
- Hàm `DefWindowProc()` – hàm thực hiện các tác vụ mà ta đối xử với window, hiểu đơn giản, ta chỉ xử lý các lệnh `message` đặc biệt còn các `message` còn lại ta để thư viện `Win32 API` tự xử lý.

Tổng kết

```
#include <windows.h>

const char* ClassName = "Win32 API form stdio.vn";

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

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

    // Bước 1: Đăng ký 1 window class
    ...

    // Bước 2: Tạo window
    ...

    // Bước 3: Vòng lặp Message
    ...
}

// Bước 4: Tạo thủ tục window
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    ...
}
```

Kết quả

Nhấn phím `F5` hoặc chọn `Debug -> Start Debugging` để thấy kết quả.



Tại cửa sổ này, khi thao tác ví dụ như kéo mở rộng hoặc di chuyển, tất cả những hành động tương tác với window này sẽ được coi là message và đưa vào vòng lặp message ở bên trên. Khi vào vòng lặp thì `DispatchMessage()` sẽ thực hiện hành động này và thể hiện ra window.

Lưu ý

Thử thay đổi vài giá trị để thấy sự khác biệt. Ví dụ như ở Bước 2 – vế 4: đổi lại là `WS_CAPTION` rồi tiến hành build.

Lập Trình Win32 API - Phần 3: Tạo Và Sử Dụng Resource

 NGUYÊN PHAN

18/09/2020•5 MIN READ

Resource - tài nguyên là một sức mạnh lớn trong việc lập trình bao gồm ứng dụng. Bài viết sau hướng dẫn phương pháp tạo và sử dụng resource - icon, menu, dialog với Win32 API.

Khái niệm

Resource: tài nguyên được đưa từ bên ngoài vào chương trình, có thể là file ảnh, file âm thanh...

Resource Editor: công cụ của Microsoft, hỗ trợ đắc lực cho phép thêm các resource bên ngoài vào để quản lý, chỉnh sửa, xem trước, ... và dịch toàn bộ quá trình thực hiện ra 1 file .rc viết dưới ngôn ngữ resource script, công cụ được tích hợp sẵn trên các bản **Visual Studio** trừ bản **Express**.

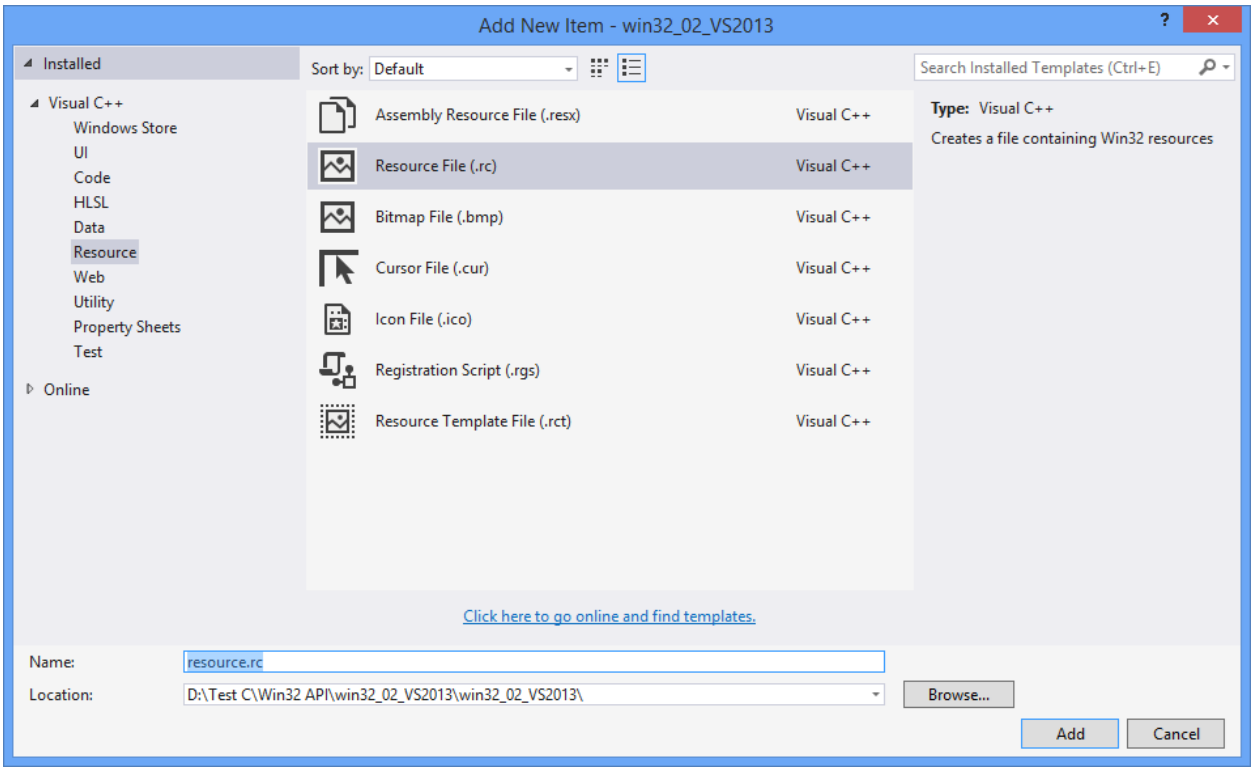
Bài viết hướng dẫn cách sử dụng Resource Editor chèn các resource vào phần mềm.

Hướng dẫn

Tạo 1 project tương tự như project ở bài viết tạo cửa số căn bản.

Bước 1: tạo file resource

Chọn `Solution Explorer` (hoặc `View -> Solution Explorer`) `-> Chuột phải` vào mục `Resource Files -> Add -> New Item -> Resource -> Resource Files`. Sau đó đặt tên file tùy ý, thông thường là **resource.rc**.

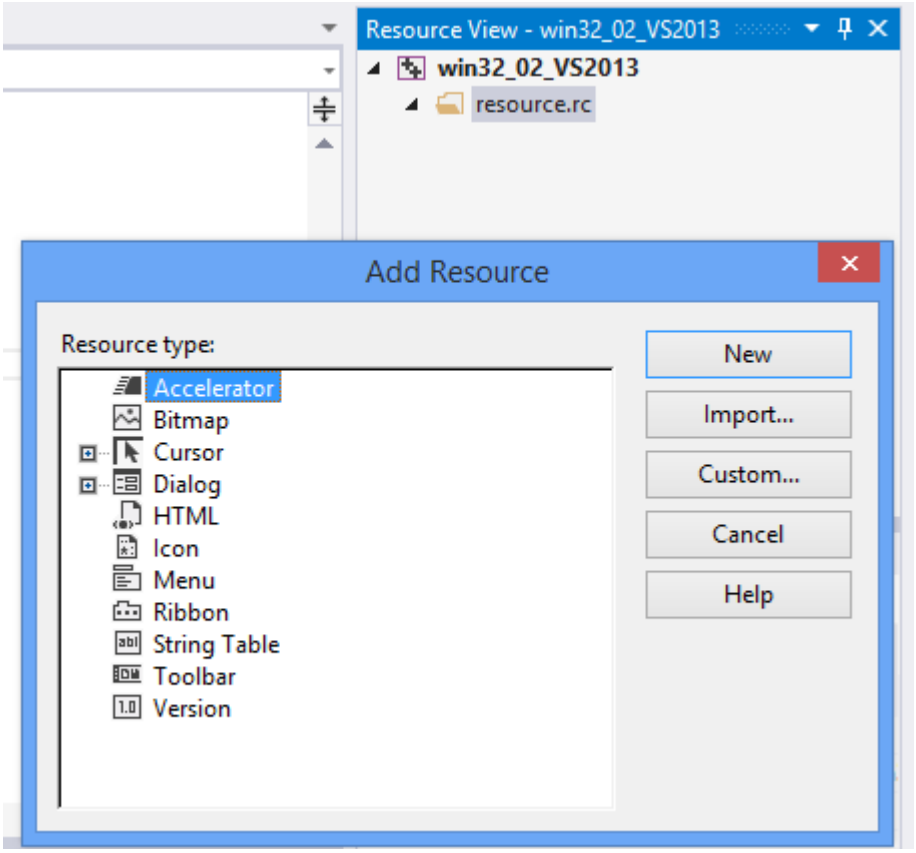


Sau khi tạo xong, **Visual Studio** tự động chuyển sang cửa sổ Resource View, tạo chuyển lại cửa sổ Solution Explorer. Hệ thống tự tạo cho 1 file là `resource.h`.

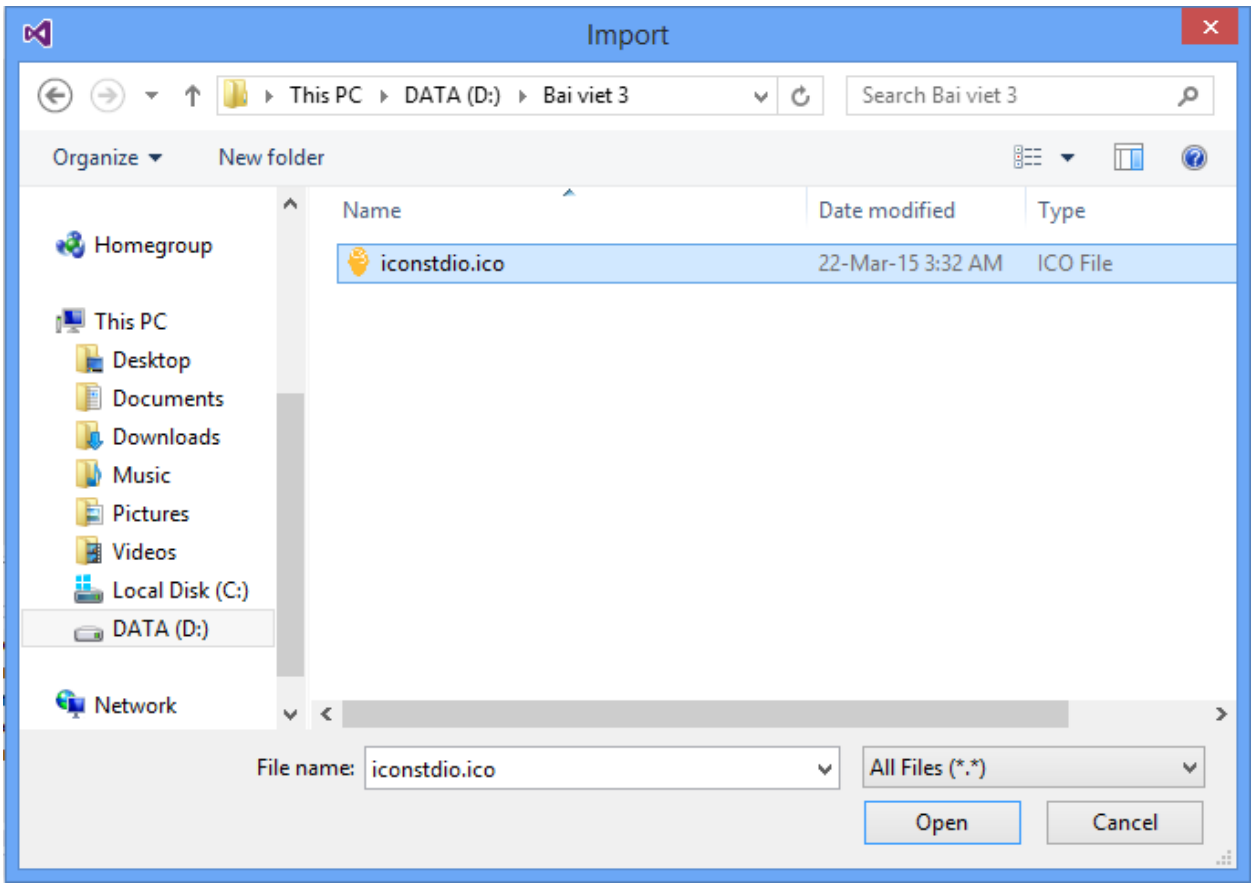
Bước 2: chèn icon

Icon là một hình ảnh có kích cỡ nhỏ, đại diện cho chương trình.

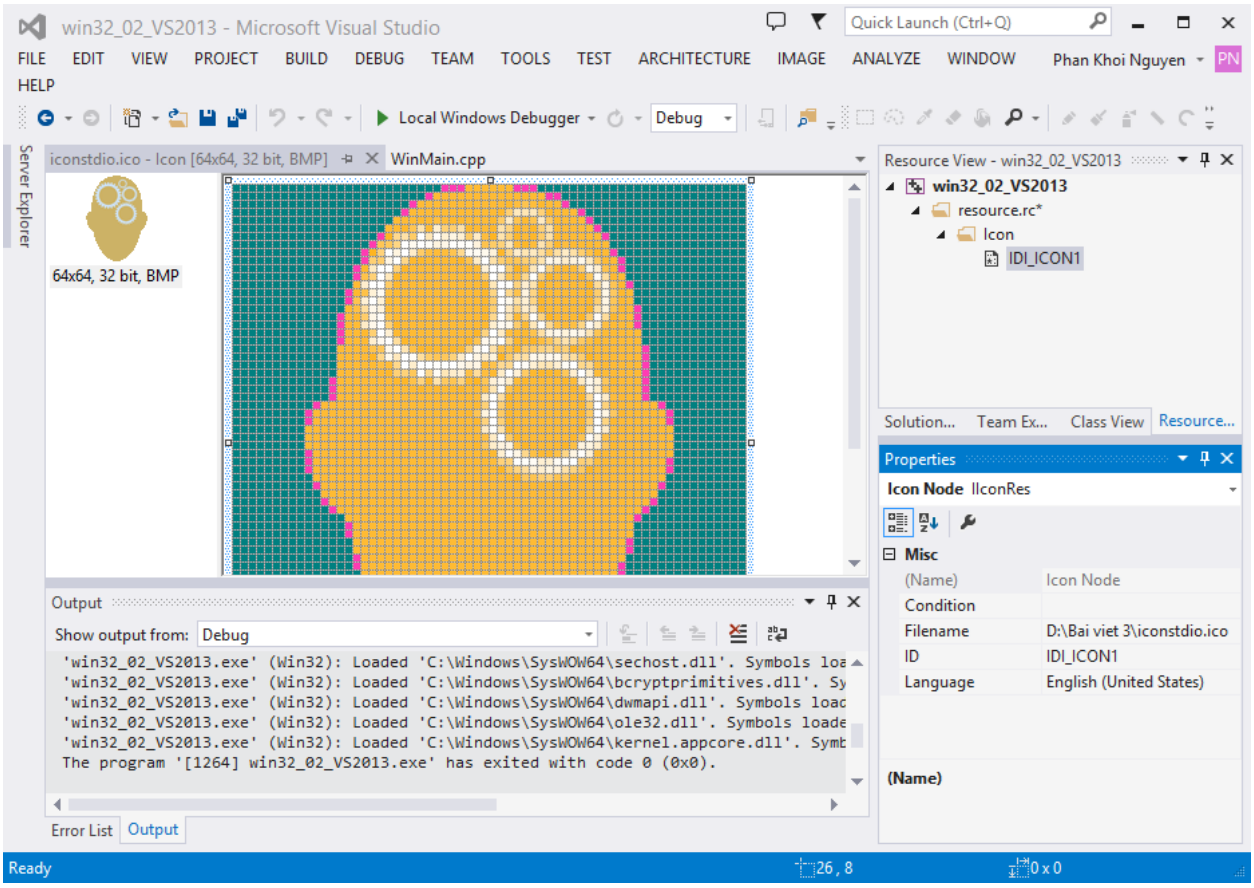
Double click vào file `resource.rc` sẽ được chuyển đến cửa sổ Resource Editor. Nhấp huột phải vào `resource.rc` chọn Add Resource và xuất hiện bảng sau.



Chọn nút Import



Chọn đường dẫn đến thư mục chứa icon.



Icon được tải lên sẽ lên sẽ có tên là `IDI_ICON1` đây là tên hệ thống đặt.

Nếu cần đổi tên, bên dưới phần `Properties (hoặc chuột phải -> Properties)` sửa tên tại phần ID.

Thao tác trên file `WinMain.cpp` để tải được icon lên:

Thêm file header mà hệ thống tự tạo lúc tạo resource:

```
#include "resource.h"
```

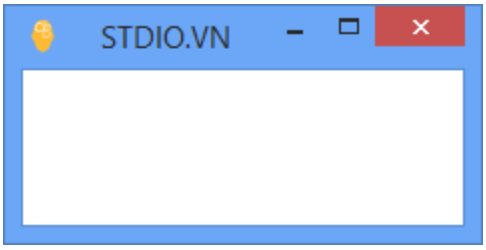
Ở đoạn khởi tạo window class, các biến `wc.hIcon` và `wc.hIconSm` sửa lại như sau:

```
wc.hIcon = LoadIcon(GetModuleHandle(NULL), MAKEINTRESOURCE(IDI_ICON1));
wc.hIconSm = LoadIcon(GetModuleHandle(NULL), MAKEINTRESOURCE(IDI_ICON1));
```

Giải thích:

- `GetModuleHandle(NULL)`: hàm này trả về con trỏ trỏ tới hInstance tức là tiến trình của phần mềm.
- `MAKEINTRESOURCE(IDI_ICON1)`: hàm này truy xuất vào resource mà đã tạo ra để lấy `IDI_ICON1` mà đã thao tác ở trên.

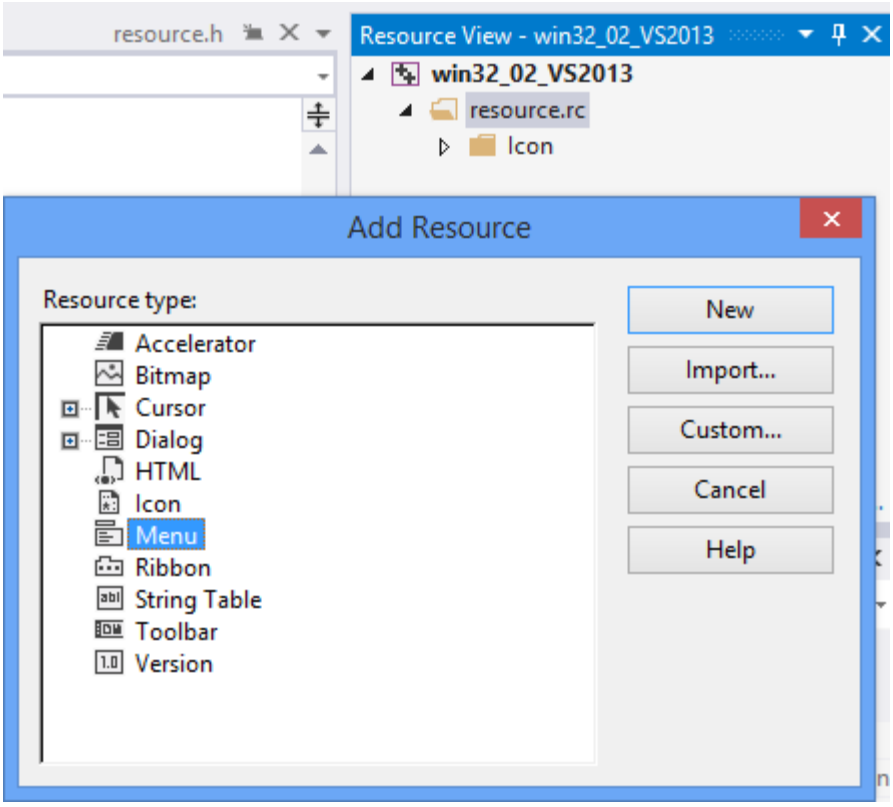
Chạy thử chương trình



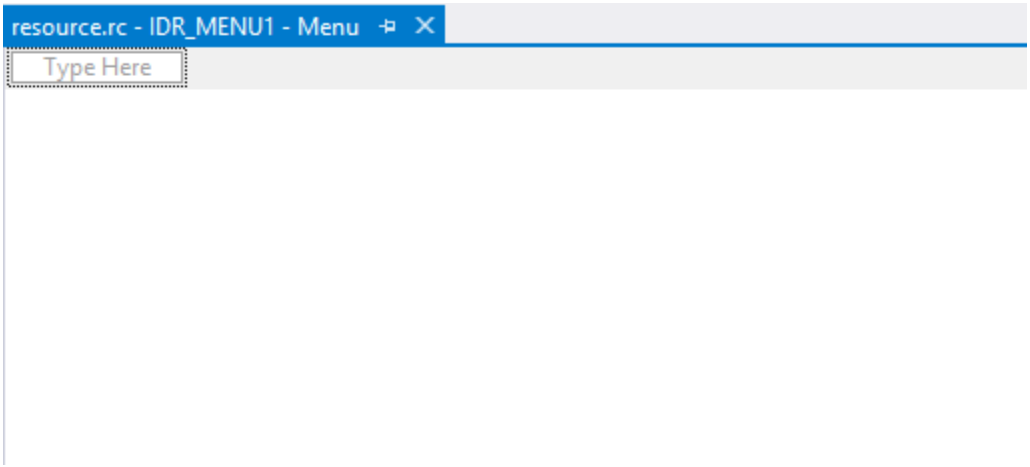
Bước 3: tạo menu

Menu là một liệt kê các thao tác mà lập trình viên đưa ra cho người dùng sử dụng.

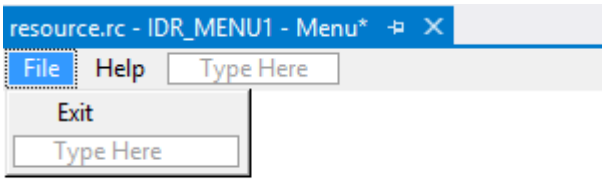
Tạo 2 lệnh cơ bản nhất là thoát chương trình và một lệnh sẽ đưa ra thông tin phần mềm đang tương tác.



Trong cửa sổ Resource View, double click vào Menu / IDR_MENU1.



Tạo menu như sau: File / Exit và Help / About



Thêm lệnh cho File / Exit. Chuột phải vào ô `Exit -> Properties -> ID` của Exit là `ID_FILE_EXIT`.

Trở lại với file `WinMain.cpp` ở đoạn code window class, sửa biến `wc.lpszMenuName` như sau:

```
wc.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);
```

`WndProc`, thêm trường hợp:

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
        switch (msg)
        {
        case WM_COMMAND:
                switch (LOWORD(wParam))
                {
                case ID_FILE_EXIT:
                        PostQuitMessage(0);
                        break;
                case ID_HELP_ABOUT:
                        break;
                }
        // ...
        }
        return 0;
}
```

Giải thích:

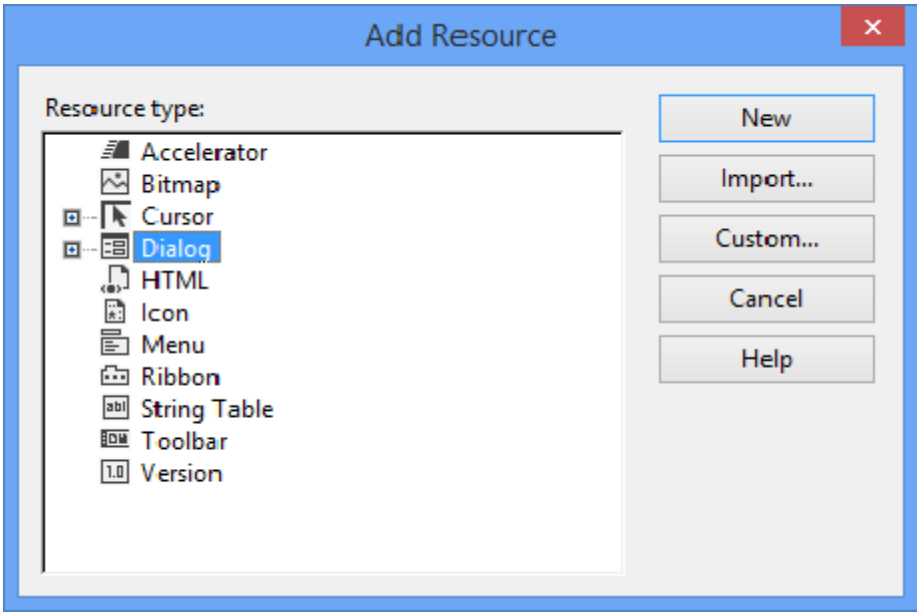
- `WM_COMMAND`: Hiểu đơn giản ở đây, là message tương ứng với việc tương tác trên thanh menu, các nút button (tìm hiểu sau). Trong đó với các lệnh như `WM_COMMAND` có các biến hỗ trợ là `wParam` và `lParam`.
- `LOWORD(wParam)`: Thể hiện cho con số mà các trường hợp ví dụ cụ thể là `ID_FILE_EXIT` được định nghĩa ở `resource.h`, khi chọn File / Exit thì `WM_COMMAND` sẽ được gửi kèm với `wParam` và `lParam` để phần mềm hiểu đang chọn lệnh nào.

Chạy thử chương trình và có thể thoát chương trình thông qua File / Exit.

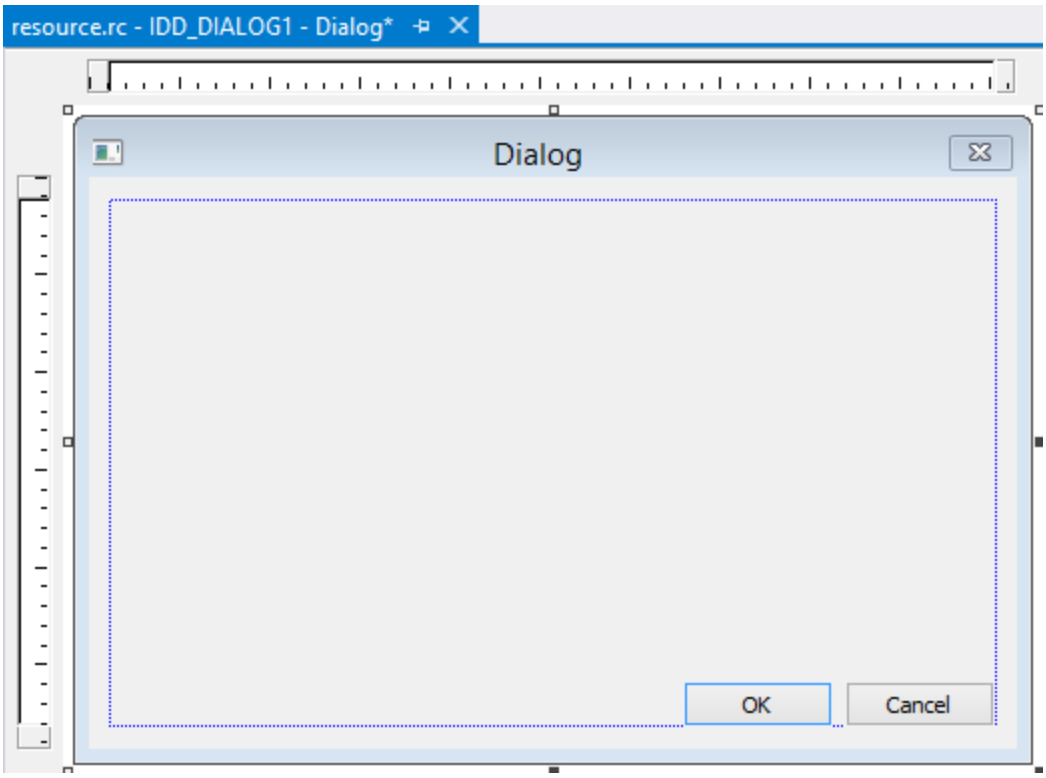
Bước 4: tạo Dialog Box

Dialog Box là công cụ mà lập trình viên dùng để tương tác với người dùng của mình.

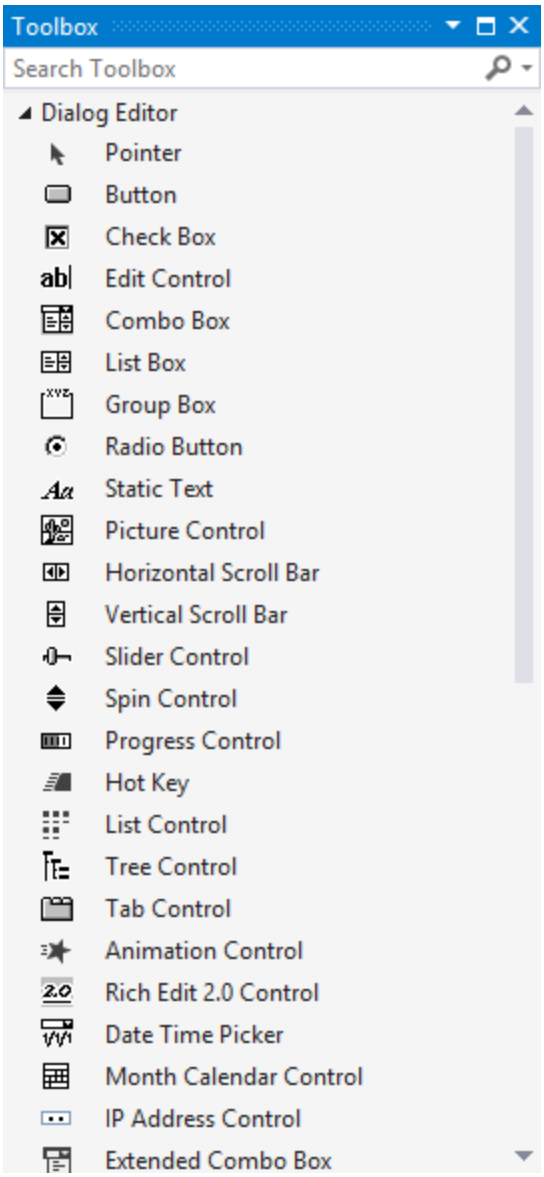
Tạo Dialog Box bằng các thao tác ở phần 2, 3 mà tạo Dialog Box.



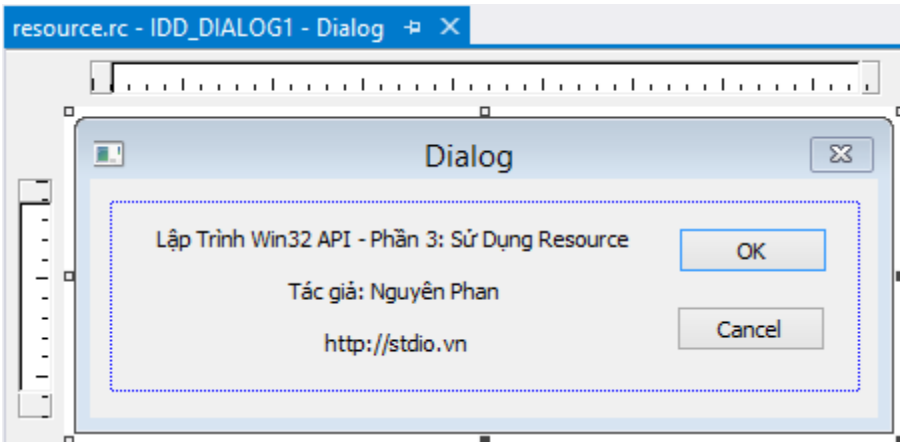
Double click vào Dialog:



Các bạn chọn `View -> Tool Box`, các bạn sẽ có được Dialog Editor như sau:



Với các công cụ trên, tạo 1 Dialog như sau:



Gợi ý: Kéo thả các button OK, Cancel, sử dụng Static Text để thêm đoạn text trên. Chọn Static Text, vào `Properties -> Caption` để chỉnh sửa văn bản, `Properties -> Align Text -> Center` để căn giữa.

Sau khi tạo được một Dialog như trên, đưa Dialog này vào chương trình.

`Dialog Box` là 1 cửa sổ, có các button là OK và Cancel vì vậy nó cũng cần một thủ tục riêng để vận hành, tương tự như `WndProc`.

```
BOOL CALLBACK AboutBox(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
	switch (msg)
	{
	case WM_INITDIALOG:
		return 1;
	case WM_COMMAND:
		switch (LOWORD(wParam))
		{
		case IDOK:
			EndDialog(hwnd, IDOK);
			break;
		case IDCANCEL:
			EndDialog(hwnd, IDCANCEL);
			break;
		}
		break;
	default:
		return 0;
	}
	return 1;
}
```

Ý nghĩa của hàm trên tương tự với hàm `WndProc` và hàm `WM_COMMAND` .

Thêm trường hợp sau Help / About lên `WndProc`.

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
        switch (msg)
        {
        case WM_COMMAND:
                switch (LOWORD(wParam))
                {
                // ..
                case ID_HELP_ABOUT:
                        DialogBox(GetModuleHandle(NULL), MAKEINTRESOURCE(IDD_DI
ALOG1), hwnd, AboutBox);
                        break;
                }
        // ..
        }
        return 0;
}
```

Giải thích:

- `DialogBox()`: là hàm đưa Dialog đã định nghĩa trên resource ra màn hình.
- `EndDialog()`: hàm này sẽ kết thúc Dialog Box và trả về hàm `DialogBox` giá trị ở vế 2.

DLL và cấu trúc của Windows

Các hộp thoại thông dụng

Dưới đây là một hộp thoại mở tập tin

Hộp thoại này không chỉ tiết kiệm thời gian cho chúng ta, mà còn tạo nên một giao diện người sử dụng gần gũi và thống nhất với môi trường Windows

Hộp thoại này do Windows cung cấp, nằm trong thư viện \Windows\System\comdlg32.Dll. Tập tin này chứa các đoạn chương trình tạo ra các hộp thoại thông dụng khác nhau. Do tập hợp lại trong một thư mục dùng chung là Windows\system, nó cho phép mọi ứng dụng Windows có quyền truy cập và thậm chí bản thân Windows cũng vậy.

Các tập tin DLL này được biên dịch với C/C++.

Hộp thoại mở tập tin là một trong những hộp thoại thông dụng của Windows

Thư viện liên kết động

Đối với các ngông ngữ lập trình cổ điển như C, khi biên dịch chương trình, ta có một tập tin EXE duy nhất có thể được thi hành mà không cần bất cứ tập tin nào khác (tập tin.vbp của Visual Basic không phải trường hợp này, vì nó chỉ chạy trong môi trường Visual Basic). Toàn bộ chương trình cần thiết được chứa hết trong một tập tin EXE. Tuy nhiên, cũng có rất nhiều thư viện C được dùng rộng rãi. Vấn đề là làm sao sử dụng lại các đoạn chương trình viết sẵn trong chương trình của ta. Đó chính là *liên kết* (*link*). Có hai loại liên kết : *liên kết tĩnh* (*static link*) và *liên kết động* (*dynalic link*).

Liên kết tĩnh

Cung cấp một kết nối nền vững giữa chương trình và module viết sẵn ngay lúc thiết kế; tương tự ta tạo module trong Visual Basic và gọi thủ tục trong đó, chỉ khác là liên kết tĩnh thì chứa bên ngoài Visual Basic. Tuy nhiên, để sử dụng liên kết tĩnh, ta cần copy phần chương trình viết sẵn của thư viện vào tập tin chương trình khi biên dịch. Từ đó trở đi, chúng trở thành một phần của chương trình và bị khoá chặt với chương trình.

Liên kết động

Là giải pháp linh hoạt hơn liên kết tĩnh. Tập tin thư viện bên ngoài không bị ràng buộc với bên ngoài. Nó chứa ở một nơi sao cho tập tin EXE có thể tìm ra và gửi thông điệp cho nó. Khi thi hành, các thông điệp này là những cuộc gọi đến các hàm/thủ tục, yêu cầu phần chương trình nào đó của DLL được thi hành.

Các DLL của Visual Basic

Có lẽ Visual Basic là một minh hoạ cho việc sử dụng DLL. Vào thư mục *Windows**system*, ta sẽ thấy một loạt các tập tin cấu tạo nên cơ chế thi hành của VB.

Ví dụ, *VB5DB.DLL* chứa chương trình kiên kết với DAO (*đối tượngtruycậpdữ liệu*) lúc thi hành để ứng dụng có thể tìm kiếm các cơ sở dữ liệu cục bộ.

Khi ta xây dựng một ứng dụng cơ sở dữ liệu và biên dịch nó, tập tin EXE không biết

Gì về cơ sở dữ liệu. Thay vào đó, nó sử dụng một số đoạn chương trình của vb cho phép nạp thư viện *vb5db.dll* lúc thi hành và gọi các hàm trong đó.

Thế mạnh của DLL

- *Nhất quán*: người sử dụng ưa chuộng Windows vì nó không ít thì nhiều có một giao diện người sử dụng phổ biến cho mọi ứng dụng. Ví dụ các hộp thoại thông dụng, các menu, thanh công cụ của *Office97*… Nghĩa là có những đoạn chương trình chung để tạo ra chúng.

- *Dễ bảo trì*: Những thay đổl hoặc bổ sung nếu có sẽ thể hiện trên mọl ứng dụng.

- *Tập tin EXE nhỏ hơn*: Do một phần công việc chứa ở nơi khác, và không gắn kết “cứng nhắc” như liên kết tĩnh, kích cỡ tập tin EXE được giảm nhỏ.

Chỉ có điều là DLL còn chứa nhiều phần khác, không chỉ là những gì chương trình của ta cần.

Cấu trúc Windows

DLL là nền tảng của thiết kế Windows. Windows thực chất là tập hợp các *DLL* để các ứng dụng khác nhau có thể dùng chung. Bên trong các *DLL* này là hàng trăm hàm/thủ tục. Ta gọi chúng là *WindowsAPI*.

WIN API

Giao diện lập trình ứng dụng (*Application Programmer’s Interface*) là tập hợp các hàm/thủ tục có sẵn của Windows. Chúng gẫn gũi với ngôn ngữ C/C++ hơn.

Visual Basic được thiết kế theo kiểu che bớt các công việc bên dướl hệ thống. Phần lớn các cuộc goi đến các hàm API được lồng trong các dạng lệnh Visual Basic, từ khoá, phương thức và thuộc tính. Chúng sẽ được thông dịch thành *WinAPI*bên trong của Visual Basic.

Tuy nhiên, vẫn có một số hàm API mà Visual Basic không có phần tương đương.

Ví dụ, Visual Basic chuẩn không có cách nào cho người lập trình điều khiển hệ thống *multimedia* của Windows, nhưng với *winapi*, ta có thể đạt được kết quả. Hiểu rõ W*INAPI*, ta có thể khám phá những năng lực tiềm tàng của chúng.

Lớp bọc API và các điều khiển hiệu chỉnh

Điều khiển hiệu chỉnh (OCX hay activex) bản thân chúng là những lớp bọc API, chúng chuyển giao các chức năng theo kiểu Visual Basic một cách thân thiện.

Điều khiển *ActiveX*và *OLEAutomationServers*đưa chương trình vào các đề án mà không cần phảl có một DLL thực sự.

Ta cũng có thể gọl API trong các module lớp, nghĩa là đưa năng lực API vào đốl tượng Visual Basic.

Sử dụng API

Tìm kiếm API

Ta có thể tìm các API thông qua tập tin *Trợgiúp*(*Help*) của Visual Basic, qua sách tra cứu

Trình duyệt API (TextAPI Viewer)

Được cung cấp sẵn khi cài Visual Basic. Khi ta cần tra cứu cú pháp chính xác của hàm API, ta dùng *TextAPIViewer*. Tuy nhiên, để có thông tin chi tiết hơn như hàm API làm gì, truyền tham số gì, trả về giá trị gì, ta cần có quyển sách tra cứu.

Ngoài ra, chương trình này còn cho phép copy nội dung API đến *clipboard*để dán vào chương trình.

Các DLL của Windows

Các API được tổ chức trong bốn DLL chính của Windows:

a. **KERNEL32**:

Là DLL chính, đảm nhiệm quản lý bộ nhớ, thực hiện chức năng đa nhiệm và những hàm ảnh hưởng trực tiếp đến hoạt động của Windows.

b. **USER32**:

Thư viện quản lý Windows. Thư viện này chứa các hàm xử lý menu, định giờ, truyền tin, tập tin và nhiều phần không được hiển thị khác của Windows.

c. **GDI32**:

Giao diện thiết bị đồ hoạ (Graphics Device Interface). Thư viện này cung cấp các hàm vẽ trên màn hình, cũng như kiểm tra phần biểu mẫu nào cần vẽ lại.

d. **WINNM**:

Cung cấp các hàm multimedia để xử lý âm thanh, nhạc, video thời gian thực, lấy mẫu, v.v… Nó là DLL 32 bit. (Thư viện 16 bit tên là *MMSYSTEM*)

Ta có thể tìm các tập tin này trong thư mục *Windows**system*. Ngoài ra, còn có các DLL nhỏ hơn, cũng được dùng phổ biến để cung cấp các dịch vụ đặc biệt cho ứng dụng.

Trên đây là các tên DLL 32 bit. Phiên bản VB4 là bản cuối cùng còn hỗ trợ 16 bit.

Gọi API

Gọi API không khác gì với gọi hàm/ thủ tục trong module của đề án. Ví dụ ta có thủ tục:

```
Public sub FindText(obiDataControl as Control, _ SFilename as String)
    ‘ Code to implement function here
    End sub
```

Để gọi thủ tục ta dùng :

```
FindText datTitles, “Titles”
```

Chỉ có điều API là một thủ tục không chỉ nằm ngoài module mà còn nằm ngoài Visual Basic.

Khai báo một cuộc gọI API:

Trước khi dùng hàm của DLL, ta cần khai báo hàm đó. Visual Basic cần biết:

— Tên hàm / thủ tục.

— Tập tin DLL chứa nó.

— Tham số truyền.

— Kiểu dữ liệu truyền về nếu là hàm

Khai báo API tương tự khai báo hằng/ thủ tục thông thường. Ta vẫn bắt đầu bằng từ khoá *Sub/Function*, chỉ khác là trước đó phảI có từ khoá *Declare.*

Ví dụ mẫu - Tạo cửa sổ nhấp nháy bằng cách gọi API

1. Tạo đề án chuẩn mớI

2. Vẽ điều khiển định giờ (timer) trên biểu mẫu và định thuộc tính

Interval là 10. Nó sẽ gây ra một sự kiện timer mỗI 10 mi-li-giây.

Biểu tượng điều khiển Timer trên hộp công cụ

3. Nhấn đúp lên cửa sổ này để mở *Cửa sổ Code*

```
Private Sub Timer1_Timer()
    Dim nReturnValue As Integer
    nReturnValue = Flash(Form1.hWnd, True)
    End Sub
```

4. Khai báo hàm *Flash* trong *GeneralDeclarations*:

```
Private Declare Function Flash Lib "User32"_ Alias "FlashWindow"_
    (ByVal hWnd As Long,_
    ByVal bInvert As Long) As Long
```

5. Thi hành chương trình. Khi biểu mẫu xuất hiện, tiêu đề của nó nhấp nháy.

Mặc dù ta thấy chương trình này rất đơn giản, nhưng nếu viết bằng các hàm Visual Basic thông thường, nó rất phức tạp và tốn rất nhiều chương trình.

Từ khoá *Declare*báo VB biết đây là khai báo một hàm của DLL.

Sau *Declare*là từ khoá *Sub*hay *Function*, cho biết đây là thủ tục hay hàm. Ta chỉ có một trong hai lựa chọn.

Từ khoá Lib cho biết tên DLL đang chứa hàm/ thủ tục đó. Ở đây là thư viện User32. Từ khoá Alias cho biết tên thực sự của thủ tục / hàm trong thư viện. Nó có thể khác vớI tên ta khai báo trước từ khoá Lib.

Cuối cùng là khai báo các tham sổ truyền, cùng vớI kiểu dữ liệu hàm trả về.

Ở đây tham số được truyền là :

```
(ByVal hWnd As Long, ByVal bInvert As Long) As Long
```

Tham số đầu, *hWnd,*là “handle”, xác định cửa sổ cần nhấp nháy. Tham số thứ hai, *bInvert*là giá trị *Boolean*. Nếu *bInvert*được truyền vào có giá trị *True*, thanh tiêu đề sẽ nhấp nháy. Để trả về trạng thái đầu, ta phảI gọI lạI lần nữa, vớI *bInvert*mang giá trị *False*.

Với nhiều hàm API, tên Alias trùng với tên thực. Khi đó Visual Basic sẽ tự động loại bỏ phần Alias. Ví dụ:

```
Private Declare Function FlashWindow Lib "User32"_ Alias "FlashWindow"_
    (ByVal hWnd As Long,_
    ByVal bInvert As Long) As Long
```

Visual Basic sẽ đổi thành:

```
Private Declare Function FlashWindow Lib "User32"_ (ByVal hWnd As Long,_
    ByVal bInvert As Long) As Long
```

Tuy nhiên một số có tên không hợp lệ đốI vớI Visual Basic, như *_lopen*, một số khác có nhiều phiên bản, ví dụ có ký tự **A** và **W** ở cuối tên. Nói chung, tốt nhất nên dùng tên thực của API. Một số lập trình viên dùng *Alias*để thay thế tên hàm, hoặc thậm chí khai báo hai tên cho hai phiên bản hàm để nhận các tham số truyền khác nhau.

```
nReturnValue = Flash(Form1.hWnd, True)
    Sau khi khai báo hàm API, ta có thể gọI API như một hàm hoặc thủ tục Visual Basic thông thường. GọI Flash là gọI đến API trong DLL, và ta lưu giá trị trả về trong biến nReturnValue.
```

Đối với các hàm thông thường, ta có thể không cần sử dụng giá trị trả về của hàm. Tuy nhiên, ta vẫn cần chứa giá trị trả về vào một biến dù ta không có ý định sử dụng nó. Phần lớn API trả về mã lỗi kiểu số, và ta có thể dùng nó để kiểm tra mọi việc có hoạt động chính xác hay không.

Trong thực tế, bỏ qua giá trị trả về không chỉ là lườI biếng mà còn thực sự nguy hiểm nếu ta đang gọI nhiều API.

Sử dụng API sai có thể dẫn đến treo Windows, nếu không nói là treo máy. Khi làm việc vớI các API phức tạp, như những hàm cần cấp phát nhiều vùng nhớ và tài nguyên hệ thống. Không nên bắt chiếc các lập trình viên cẩu thả bỏ qua các giá trị trả về. Vì hàm DLL nằm ngoài ứng dụng, chúng tự kiểm tra lỗi – ta chỉ biết có sai sót thông qua giá trị trả về.

Handle

Lấy biểu mẫu làm ví dụ. Windows dùng một cấu trúc để lưu giữ thông tin của biểu mẫu. Thông tin này đồng nhất với thông tin chứa trong cửa sổ *Properties*. Windows chứa cấu trúc của từng cửa sổ trong một danh sách dài gồm các cấu trúc dữ liệu liên quan đến mọi cửa sổ của mọi chương trình đang chạy. Để xác định cấu trúc nào thuộc cửa sổ nào, nó dùng *handle*. Nó không dùng tên biểu mẫu vì tên cũng là một thuộc tính của biểu mẫu. *Handle* chính là số ID của một đối tượng trong Windows.

Khi ta bắt đầu dùng API, nhất là những API có xử lý với biểu mẫu, ta sẽ thường xuyên làm việc với handle. Visual Basic chứa handle như một thuộc tính chỉ được đọc, có thể dùng làm tham số truyền cho những hàm của Windows khi cần.

Thuộc tính này gọi là *hWnd*(*handleđếnmộtcửasổ*), chỉ có thể truy cập lúc thi hành. Mặc dù nó không mang ý nghĩa trong chương trình, nhưng nó có thể được đọc, và truyền như một tham số đến API. Các API có liên quan hiển thị cửa sổ sẽ cần tham số *hWnd*để biết chính xác cửa sổ mà nó cần xử lý.

Khai báo tham số truyền

Điểm quan trọng trong khai báo tham số truyền cho API là từ khoá *Byval*.

Với chương trình thông thường, nếu truyền giá trị cho hàm, Visual Basic biết rằng nó chỉ xử lý với bản sao của tham số.

```
Function Square(Byval Number as Double) as Double
```

Một cách khác để truyền tham số là truyền tham chiếu. tham số truyền là biến chứ không phải là bản sao của nó. Do đó nếu hàm thay đổi tham số, các thay đổi này sẽ ảnh hưởng lên biến truyền vào. Nếu không chỉ rõ *Byval*, VB sẽ tự động xem đó là truyền tham chiếu.

Nếu là hàm hoặc thủ tục do ta viết, nếu có sai sót dothiếu *Byval*, hậu quả không nghiêm trọng, Windows không bị treo.

Tuy nhiên, với các DLL, tình hình nguy hiểm hơn nhiều. Nếu ta quên *Byval*, VB tự động truyền một con trỏ đến biến. Nó cho biết địa chỉ của biến trên vùng nhớ. Sau đó hàm này đến địa chỉ đó và lấy giá trị về.

Nếu một hàm của DLL chờ một kết quả trong khoảng từ 0 đến 3, và ta truyền một biến tham chiếu, giá trị thực sự truyền vào có thể là 1002342, là địa chỉ vùng nhớ của biến. Hàm này sẽ xử lý số 1002342 thay vì số thuộc khoảng (0-3), kết quả là hệ thống treo.

Không hề có thông báo lỗi ở đây; ta chỉ biết được API bị lỗi khi hệ thống rối loạn ,và treo cứng. Một trong những kinh nghiệm khi làm việc với API là lưu lại. Vì chúng ,ta đang mạo hiểm ra ngoài vùng an toàn của Visual Basic, khi bị lỗi, hệ thống treo và ta mất hết dữ liệu. Luôn luôn lưu đề án trước khi chạy đoạn chương trình goin API.

Từ menu *Tools*, chọn *Options* để mở hộp thoại *Options*. Chọn tab *Environment*, đánh dấu vào tuỳ chọn *SaveChanges*.

Sử dụng lớp với API

Sử dụng riêng lẽ từng hàm API sẽ gây khó khăn cho những người đọc chương trình nếu họ không phải là người lập trình ban đầu, nhất là đối với các ứng dụng lớn.

Giải pháp của Visual Basic 6 là chuyển các API thành các lớp (các điều khiển ActiveX). Từng API có thể xếp vào những nhóm tuỳ thuộc lĩnh vực nó xử lý. Các nhóm này có thể chuyển thành các lớp của Visual Basic. Ví dụ, tạo một lớp có các chức năng về multimedia của các API về lĩnh vực này.

Dùng API khai thác khả năng Multimedia

Lớp multimedia

Lớp này chứa một bộ các lệnh multimedia thông dụng. Khi một đối tượng được tạo từ lớp, nó mang những chức năng tương tự một điều khiển – có thể xem hay quy định thuộc tính, các phương thức. Nó che đi các lệnh gọi API.

Các phương thức mà lớp này hỗ trợ:

Phương t hức	**Mô tả**
MmOpen	Mở tập tin (video, âm thanh, nhạc, v.v...) chuẩn bị *Play*
MmClose	Đóng tập tin đang mở, ngăn cấm hoạt động *Play*
MmPause	Dừng *Play* trên tập tin hiện hành
MmStop	Dừng hẳn *Play*
MmSeek	Tìm một vị trí trong tập tin
MmPlay	*Play* tập tin đang mở, phát ra âm thanh trong loa

Các phương thức này là những hàm riêng rẽ trong lớp MMedia.cls và cho phép sử dụng các API theo nhiều cách.

Sau đây là các thủ tục thuộc tính trong tập tin nguồn:

Thuộc tính	**Mô tả**
Filename	Tên của tập tin đang mở
Length	Chiều dài của tập tin đang mở
Position	Vị trí hiện hành trong tập tin – ta có thể kết hợp với thuộc tính*Length*để hiển thị trạng thái Play
Satus	Một từ cho biết trạng thái tập tin (*Play,dừngtạm,dừnghẳn,v.v...*)
Wait	Nếu là *True*, chương trình sẽ chờ đến khi *Play*xong mới làmtiếp. Nếu là *False*, nó thi hành theo kiểu *đa nhiệm*

Ví dụ mẫu - Sử dụng lớp Multimedia

1) Mở tập tin TestMM.vbp

2) Điều chỉnh kích cỡ biểu mẫu chính và vẽ một nút lệnh và một điều khiển hộp thoại thông dụng:

Thiết kế biểu mẫu

Nếu không thấy điều khiển hộp thoại thông dụng trên hộp công cụ, từ menu Project, chọn Components, và chọn vào hộp đánh dấu “Microsoft Common Dialog Control 6.0”.

3) Để hộp thoại (thông dụng) bật ra khi nhấn vào nút lệnh, ta xử lý sự kiện Click trên nút lệnh bằng cách gõ vào:

```
Private Sub Command1_Click()
    With CommonDialog1
    .Filter = "WaveAudio (*.wav)|*.wav|Midi
    (*.mid)|*.mid|Video files (*.avi)|*.avi"
    .FilterIndex = 0
    .ShowOpen
    End With
End Sub
```

4) Chạy chương trình và nhấn nút lệnh, ta sẽ thấy hộp thoại mở tập tin quen thuộc xuất hiện cho ta chọn tập tin multimedia:

Chọn mở tập tin multimedia

5) Kế tiếp ta chuyển lớp multimedia thành một đối tượng.

```
Private Sub Command1_Click()
    Dim Multimedia As New MMedia
With CommonDialog1
    .Filter = "WaveAudio (*.wav)|*.wav|Midi
(*.mid)|*.mid|Video files (*.avi)|*.avi"
    .FilterIndex = 0
.ShowOpen
    End With
If CommonDialog1.Filename <> "" Then Multimedia.mmOpen CommonDialog1.Filename Multime
dia.mmPlay
    End If
End Sub
```

Thi hành chương trình. Tìm một tập tin multimedia trên đĩa cứng (thường chứa trong thư mục *Windows\Media*) và play.

Lưu ý rằng để play các tập tin âm thanh như *WAV*và *MID*, ta cần có *card*âm thanh trên máy .

Trong dòng đầu của sự kiện click, ta tạo một đối tượng multimedia dẫn xuất từ lớp *MMedia*. Đây là bước chuyển từ một lớp sang một đối tượng.

```
Private Sub Command1_Click()

    Dim Multimedia As New MMedia
```

Bốn dòng kế sử dụng đối tượng *multimedia* để mở tập tin dùng phương thức *mmOpen*và *Play* bằng phương thức *mmPlay*.

```
If CommonDialog1.Filename <> "" Then Multimedia.mmOpen CommonDialog1.Filename Multime
dia.mmPlay

    End If
```

Tạo lớp bao bọc các API làm vấn đề đơn giản hơn. Nếu lớp này được đem thương mại hoá, người sử dụng nó sẽ không cần phải hiểu về API, họ chỉ cần biết cách thức hoạt động của lớp mà thôi.

Tìm hiểu lớp Multimedia

Windows có nhiều phân hệ, mỗi phân hệ đảm nhiệm một chức năng nhất định. Một trong những phần này là MCI. MCI là tên gọi tắt của *Multimedia Control Interface*, cung cấp một giải pháp độc lập với thiết bị để sử dụng các tính năng của Windows thông qua chương trình.

Khi viết chương trình trò chơi trên DOS, ta phải xử lý với nhiều chuẩn card âm thanh và hình ảnh khác nhau. Tính năng độc lập với thiết bị, và các chương trình điều khiển thiết bị cung cấp bởi Windows cho phép ta làm việc với bất kỳ *card*âm thanh, hình ảnh nào với cùng chương trình, miễn là chúng được hỗ trợ bởi Windows.

MCI cung cấp lớp đệm giữa lập trình viên và các thiết bị dùng xử lý dữ liệu *multimedia*như các *card* âm thanh, hình ảnh.

MCI sẽ làm việc với các chương trình điều khiển thiết bị của Windows, và cuối cùng là phần cứng *multimedia*. Lập trình viên, yêu cầu MCI dùng hàm API *mciSendString*. Lệnh này sau đó được gọi xuống chương trình điều khiển thiết bị, ta không cần quan tâm.

MCI là một đối tượng độc lập. Nó có thể được lập trình và có ngôn ngữ lập trình riêng. Khi ta dùng *mciSendString*, ta đang lập trình MCI.

Sử dụng mciSendString

Cú pháp của **mciSendString**:

```
<ResultCode> = mciSendString(“<Command>”, _

    <ReturnString>, <ReturnLength>, <CallbackHadle>)
```

<ResultCode> là một số long integer, và thay đổi tuỳ theo dòng lệnh.

<Command> đặt trong đấu trích dẫn, phải là một từ dưới dạng chuỗi ký tự và là lệnh gửi đến MCI; như là *Play* để play một tập tin, *Open* để mơe tập tin, v.v...

Một số lệnh MCI trả về một chuỗi ký tự. Lệnh *Status* trả về một chuỗi cho biết tập tin dừng hẳn (*Stopped*), hay đang chơi (*Playing*), hay dừng tạm (*Pause*), v.v...

API cần biết bao nhiêu dữ liệu được chứa trong biến chuỗi, tham số kế tiếp là chiều dài chuỗi. Do đó, nếu ta phát lệnh đến MCI trả về một chuỗi, ta phải truyền một biến chuỗi có chiều dài nhất định và cho biết chiều dài của nó:

```
Dim sReturnString As String * 255

    Dim nReturn As Long

nReturn = mciSendString("status waveaudio mode",_

    sReturnString, 255, 0)
```

Thêm * **255** vào khai báo *sReturnString*cho biết chiều dài của nó là 255.

Sử dụng hàm Callback trong Visual Basic

Hàm *Callback*thực ra chỉ áp dụng cho C/C++, Delphi, hay một số ngôn ngữ biên dịch cấp thấp, không dùng với Visual Basic. Tuy nhiên, VB6 cho phép ta sử dụng hàm *Callback*mà không cần thêm các chương trình phụ đặc biệt như trong các phiên bản trước.

Khi ta dùng API, chương trình của ta không thể nào biết được điều gì đang xảy ra khi hàm đang chạy. Ta phải chờ đến khi nó kết thúc, và kiểm tra giá trị trả về. Ý tưởng của hàm *Callback*là một API mà khi chạy, nó có thể gọi đến một hàm hoặc thủ tục của chương trình ta đang viết.

Ta phải tạo một hàm Public ở trong một module chương trình của Visual Basic, với các tham số truyền cần thiết của API. Sau đó, khi gọi API, ta gửi một con trỏ (pointer) - địa chỉ vùng nhớ của hàm Callback. Ta phải dùng toán tử mới AddressOf:

```
nResult = someAPIFunction(ParamOne, ParamTwo, _ AddressOfMyCallback)
```

Khi API chạy, nó gọi mmọt hàm trong chương trình của chúng ta và gửi các tham ,số cần thiết. Thường nó được được dùng để cập nhật thanh trạng thái, lấy danh sách font hệ thống, và các công việc khác.

Như đã nói, chúng ta sẽ không nói thêm về các hàm *Callback*. Các hàm này làm phức tạp hơn cho chương trình và nhiều khi làm treo hệ thống. Tuy nhiên, trợ giúp của Visual Basic sẽ cung cấp một số ví dụ nếu bạn uốn tìm hiểu kỹ hơn.

Mở tập tin Media

Ta gửi tên tập tin cho lệnh **Open** để mở tập tin. Đây là tên chuẩn như: **C:\Video.avi**.

```
Open <filename> Type <typestring> Alias <aname>

    ...

    ...

    ‘Issue command to do something to the file

    ...

    ...

    Close <aname>
```

Sau từ khoá *Type*là kiểu tập tin. Kiểu chuẩn của Windows là *WaveAudio*đối với tâp tin *WAV*, *AVIVideo*đối với *AVI*, và *Sequencer* đối với *MID*.

Alias dùng để thay thế tên tạp tin mở:

```
Open c:\video.avi Type AVIVideo Alias Peter
```

Nếu ta gửi dòng lệnh này đến MCI bằng *MCISenString*, nó yêu cầu MCI mở tập tin *C:\video.avi*như một tập tin video của Microsoft, và nó sẽ dùng tên *Peter* để chỉ ra tập tin này.

Mỗi lần mở tập tin, lệnh MCI có thể dùng bí danh để chơi tập tin, dừng hẳn hay tạm dừng, hoặc hiển thị trạng thái, v.v... Ví dụ:

Play Peter Pause Peter Stop Peter

Sau đó, ta cần đóng tập tin bằng cách gửi lệnh Close, theo sau là bí danh của tập

```
nReturn = mciSendString(“Close Peter”, “”,0,0)
```

Hiển thị trạng thái và vị trí của tập tin Multimedia

1. Mở đề án TestMM.vbp

2. Chúng ta sẽ thêm một số điều khiển để xem thuộc tính Status và Position của lớp MMedia. Thêm một điều khiển thanh diễn tiến (ProgressBar), một nhãn, một điều khiển định giờ (timer):

Nếu không thấy điều khiển ProgressBar trên hộp công cụ, từ menu Project, chọn Components, chọn vào hộp đánh dấu “Microsoft Windows Common Controls 6.0 ”.

3. Mở cửa sổ Properties của điều khiển Timer, đổi thuộc tính Enabled thành False, và Interval là 500. Xoá Caption của điều khiển nhãn.

4. Nhấn đúp chuột lên nút lệnh để mở sự kiện Click:

Thiết kế biểu mẫu

```
Private Sub Command1_Click()
    ...
    ...
    If CommonDialog1.Filename <> "" Then
```

```
Multimedia.Wait = False Multimedia.mmOpen CommonDialog1.Filename ProgressBar1.Value =
0
    ProgressBar1.Max= Multimedia.Length
    Timer1.Enabled = True
    Multimedia.mmPlay
End If
    End Sub
```

5. Trở về biểu mẫu, nhấn đúp chuột lên điều khiển Timer1 để mở sự kiện Timer.

```
Private Sub Timer1_Timer()
    ProgressBar1.Value = Multimedia.Position
    Label1 = "Status: " & Multimedia.Status
    If ProgressBar1.Value = ProgressBar1.Max Then
    Multimedia.mmClose
    Timer1.Enabled = False
    End If
    End Sub
```

Có một vấn đề nhỏ. Ta đã định nghĩa biến chỉ đến instance của lớp MMedia trong hàm sự kiện command1_Click(). Bây giờ chúng ta lại muốn chỉ đến nó từ trong Timer1_Timer().

GHI CHÚ Bạn sẽ được giải thích khái niêm instance trong chương 13 - Lập trình hướng đối tượng

6. Trong sự kiện Click của nút lệnh, chọn dòng khao báo biến Mutilmedia, nhấn phím Ctrl-X để cắt nó vào Clipboard và xoá nó khỏi sự kiện Command1_Click. Sau đó, chọn vào danh sách (General) trong cửa sổ Code, nhấn phím Ctrl –V để dán nó vào vùng General Declarations. Biến khai báo đặt trong vùng này sẽ là biến toàn cục đối với biểu mẫu này.

7. Thi hành chương trình. Nhấn nút “Load and Play a file”, và chọn một tậptin AVI, ví dụ tập tin video “Welcome to windows 95”.

8. Ta sẽ thấy thanh diễn tiến cho thấy bao nhiêu phần trăm của tập tin đang chơi. Khi video kết thúc ta thấy kết quả hiển thị: Stopped.

Khi ta mới nhấn nút lệnh, chương trình thiết lập các khởi tạo cho thuộc tính trước khi chơi tập tin:

```
If CommonDialog1.Filename <> "" Then
    Multimedia.Wait = False
    Multimedia.mmOpen CommonDialog1.Filename
    ProgressBar1.Value = 0
    ProgressBar1.Max = Multimedia.Length Timer1.Enabled = True Multimedia.mmPlay
```

```
    End If
```

Đối tượng multimedia có thuôc tính tên là Wait. Thuộc tính này quyết định chương trình có tiếp tục thi hành (đa nhiệm) trong khi chơi tập tin, hay phải dừng và chờ đến đến khi nó hoàn tất. Phương thức mmPlay theo dõi giá trị của biến bWait. Nếu biến này có giá trị True, nó thêm Wait vào lệnh gọi mciSendString:

```
Public Sub mmPlay()
    Dim nReturn As Long
If bWait Then
nReturn = mciSendString("Play " &_
sAlias & " wait", "", 0, 0)
    Else
    nReturn = mciSendString("Play " & sAlias,"", 0, 0) End If
    End Sub
```

Làm sao biết giá trị bWait ? Nhắc lại rằng ta có thể cung cấp các hàm thuộc tính cho phép đọc hoặc quy định giá trị của biến nội bộ:

```
Property Get Wait() As Boolean
    ' Routine to return the value of the object's wait property.
    Wait = bWait
```

End Property

```
Property Let Wait(bWaitValue As Boolean)
    'Routinetosetthevalueoftheobject's wait property
    bWait = bWaitValue
```

End Property

Bước kế là mở tập tin ta muốn chơi. Ta dùng phương thức mmOpen để mở tập tin.

Mở tập tin

Trước hết, ta khai báo một cặp biến cục bộ để giữ giá trị tạm thời.

```
Public Sub mmOpen(ByVal sTheFile As String) Dim nReturn As Long
    Dim sType As String
```

```
    If sAlias <> "" Then mmClose
    End If

    Select Case UCase$(Right$(sTheFile, 3))
    Case "WAV"
  sType = "Waveaudio"
  Case "AVI"
  sType = "AviVideo"
  Case "MID"
  sType = "Sequencer"
  Case Else
  Exit Sub
  End Select
  sAlias = Right$(sTheFile, 3) & Minute(Now)
  IfInStr(sTheFile, " ") Then
  sTheFile = Chr(34) & sTheFile & Chr(34)
  End if
  nReturn =mciSendString("Open " & sTheFile _
  & "ALIAS " &sAlias& " TYPE " & sType _
  & " wait", "", 0, 0)
  End Sub
```

Trước hết, hàm *mmOpen*kiểm tra biến ở mức module gọi là sAlias.

```
If sAlias <> “” then mmClose
    End if
```

Làm việc với MCI, ta nên dùng bí danh cho từng tập tin mở. Ở đây lớp *MMedia* thiết lập một tên cho bí danh và chứa bí danh vào biến *sAlias*. Khi ta tiếp tục mở một tập tin kế tiếp bằng *mmOpen*, hoặc chỉ ra thuộc tính tên tập tin, chương trình kiểm tra điều này và gọi một thủ tục khác để đóng tập tin thứ nhất. Đóng tập tin khi ta cần giải phóng vùng nhớ và tăng tốc độ chơi tập tin.

Cấu trúc *Select Case*kiểm tra từng loại tập tin.

Lệnh *wait*cho phép chương trình tiếp tuch chạy cho đến khi nạp thành công. Nếu không có *wait*, trên một máy nhanh với đĩa cứng chậm, có thể có vấn đề. Ta có thể cố chơi tập tin trước khi nó được nạp

xong, đơn giản bởi vì chương trình chạy nhanh hơn đĩa cứng. Lưu ý rằng nó không giống thuộc tính *Wait*trước đây để điều khiển chương trình tiếp tục chạy khi tập tin đang chơi chứ không phải đang nạp.

Lấy chiều dài tập tin

Dùng *mciSendString*để lấy hoặc quy định chiều dài. Thuộc tính *Length*của lớp *MMedia*chỉ có tính chất được phép đọc mà thôi, và ta không cung cấp hàm *Property Let*.

```
Property Get Length() As Single
    Dim nReturn As Long, nLength As Integer
    Dim sLength As String * 255

    If sAlias = "" Then
    Length = 0
    Exit Property
    End If
    nReturn = mciSendString("Status " & sAlias _
    & length",Length, 255, 0)
    nLength = InStr(sLength, Chr$(0))
    Length = Val(Left$(sLength, nLength - 1))
    End Property
```

Trước hết *sAlias*được kiểm tra xem tập tin có đang mở hay không ? Nếu chưa mở, giá trị trả về từ thủ tục thuộc tính là 0. Nếu tập tin mở rồi, lệnh *Status Length* của MCI được dùng.

Ta không lo về cách tính chiều dài tập tin, vì đơn vị đo phù hợp với thanh diễn tiến.

Lệnh *Status* là lệnh MCI đặc biệt, có thể kết nối với các từ khoá như *Length, Position,Mode*để xác định các thông tin về tập tin hiện hành. Nó trả về các thông tin này trong một chuỗi ký tự có chiều dài nhất định được truyền vào mciSendString. Trong ví dụ này chuỗi trả về là *sLength* và dài 255 ký tự.

Dĩ nhiên, nó không luôn chứa 255 ký tự trả về từ lệnh *Status*. Dùng hàm *InStr* để cắt bỏ các ký tự 0 lấp đầy khoảng trống.

Chiều dài chứa trong chuỗi được trích chuỗi và chuyển đổi sang kiểu số trước khi gán cho *Length.*

Lấy vị trí hiện hành

Lệnh *StatusPosition*có thể được gọi nhiều lần để xác định vị trí hiện hành của tập tin đang chơi:

```
Property Get Position() As Single
    Dim nReturn As Integer, nLength As Integer
    Dim sPosition As String * 255
```

```
    If sAlias = "" Then Exit Property
    nReturn = mciSendString("Status " & sAlias _
    & " position", sPosition, 255, 0)
    nLength = InStr(sPosition, Chr$(0))
    Position = Val(Left$(sPosition, nLength - 1))
    End Property
```

Thay vì gửi *Status Length*, ta gửi *Status Position*.

Lấy trạng thái hiện hà n h

Để lấy chuỗi ký tự trạng thái còn gọi là *mode*, ta truy vấn thuộc tính *Status*của lớp. Ta cũng sử dụng hàm thuộc tính *Property Get* hầu đồng nhất với thuộc tính *Position* trên đây. Chỉ khác là ta gửi *Status Mode* thay vì *Status Length* hay *Status Position*cho

mciSendString. Dĩ nhiên, không cần chuyển đổi sang kiểu số:

```
...
nReturn = mciSendString("Status " & sAlias & _
" mode", sStatus, 255, 0)
nLength = InStr(sStatus, Chr$(0)) Status = Left$(sStatus, nLength - 1)
...
```

Trở lại sự kiện *Command1_Click*. Cho tới giờ, ta đã định nghĩa thuộc tính *Wait*., mở tập tin, thiết lập thanh diễn tiến. Trước khi chơi tập tin, ta quy định *Timer*. Sau đó, ,ta chơi tập tin bằng cách gọi phương thức *mmPlay*của đối tượng *Multimedia.*

```
If CommonDialog1.Filename <> "" Then
Multimedia.Wait = False
Multimedia.mmOpen CommonDialog1.Filename
ProgressBar1.Value = 0
ProgressBar1.Max = Multimedia.Length
Timer1.Enabled = True
Multimedia.mmPlay
End If
```

Chơi trên tập tin

Trước hết, kiểm tra tập tin mở thông qua biến *sAlias*; sau đó nếu thoả điều kiện, nó thi hành lệnh *Play* của MCI.

```
Public Sub mmPlay()
    Dim nReturn As Long
If sAlias = "" Then Exit Sub
If bWait Then
    nReturn=mciSendString("Play"&_
sAlias & " wait", "", 0, 0)
Else
    nReturn = mciSendString("Play " & _
sAlias, "", 0, 0)
End If
    End Sub
```

Cập nhật thanh diễn tiến và điều khiển nhãn.

Công việc sau cùng là cập nhật thanh trạng thái và nhãn trên biểu mẫu, khi tập tin đang chơi. Trước khichơi tập tin, đặt điều khiển *Timer*với Interval là 500. Vậy nó sẽ kích hoạt đếm mỗi nửa giây. Khi đó, đoạn chương trình sau được thi hành:

```
Private Sub Timer1_Timer()
    ProgressBar1.Value = Multimedia.Position
    Label1 = "Status: " & Multimedia.Status
    If ProgressBar1.Value = ProgressBar1.Max Then
    Multimedia.mmClose
    Timer1.Enabled = False
    End If
    End Sub
```

Cuối cùng cần phải ngưng lại khi đạt đến cuối tập tin. Có thể thực hiện điều này bằng cách so sánh giá trị hiện hành và giá trị *Max* của thanh diễn tiến. Khi chúng bằng nhau, tập tin được đóng bằng phương thức *mmClose*. Sau đó, cấm Timer để ngăn hàm này chạy cho đến khi mở tập tin khác.

Tóm tắt các lệnh của MCI

Lệnh	**Mô tả**

Play	Chơi một tập tin
Pause	Tạm dừng chơi, sẵn sàng bắt đầu mọi lúc
Stop	Dừng hẳn - cần chuyển đến một vị trí nào đó để tiếp tục chơi
Seek	Theo sau là một con số, chuyển đến vị trí trong tập tin
Status Mode	Trả về một chuỗi ký tự thể hiện trạng thái tập tin(đang choi, đangmở, tạm dừng, dừng hẳn....)
Status Position	Trả về vị trí tập tin mà *playback* đã đạt đến
Status Length	Trả về chiều dài tập tin và hỗ trợ để dưa con số trả về từ StatusPosition vào một ngữ cảnh có ý nghĩa nào đó.
Close	Đóng tập tin và giải phóng vùng nhớ nó chiếm trước đó

Ngoài ra MCI còn hỗ trợ một số lệnh khác và một số lệnh đặc biệt cho mỗi định dạng tập tin.

1. Tổng quan về lập trình Win 32

Để bắt đầu với Win 32 , ta cần hiểu thế nào là API. API viết tắt cho Application Programming Interface có nghĩa là "Giao diện lập trình ứng dụng", nó cung cấp các hàm và thư viện để phục vụ cho người lập trình. Win32 là viết tắt cho cụm từ Windows 32-bit, tức ở đây, Win32 API cung cấp cho ta một số lượng hàm, thủ tục nhất định để ta dễ dàng tạo ra một chương trình theo ý muốn. Đầu tiên khi mới học Win32 có thể sẽ mất nhiều thời gian nhưng khi thuần thục, nó sẽ trở nên rất hữu dụng trong nhiều trường hợp.

2. Cách tạo ra một chương trình Win 32

Dưới đây là một đoạn code cho phép chúng ta hiển thị ra một MessageBox trong Win32 :

#include <windows.h> int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) { MessageBox(NULL, TEXT("This is my first application !"), TEXT("FirstApp"), MB_OK); return 0; }

Trước tiên chúng ta cần hiểu một số khái niệm sau :

- Handle: Một xử lí đối với chương trình của chúng ta.
- window: Cửa sổ chương trình của chúng ta.(Chú ý đây không phải là Window-Hệ điều hành).
- Message: Mỗi quá trình hay thông điệp ta sẽ gửi đến window.

Mọi chương trình trong Window đều cần có 2 hàm là :

- WinMain() :Tương tự như hàm main() , khởi tạo ứng dụng .
- Window procedure :Nhận và xử lí các meesage gửi đến.

Các thông số trong hàm WinMain() : HINSTANCE hInstance: Xử lí đối với việc thực thi của chương trình . HINSTANCE hPrevInstance: Thường là NULL trong các chương trình Win32. LPSTR lpCmdLine: Một tham số command line giống như một chuỗi nhưng không được chứa tên chương trình . int nCmdShow: Một số nguyên cung cấp cho hàm ShowWindow(). Cái này sẽ được đề cập ở sau. Ở đây mình sẽ dùng Visual Studio 2013 để minh họa cho chương trình trên. Đầu tiên ta phải tạo một Empty Project trong Visual Studio 2013 bằng cách chọn File -> New -> Project hoặc dùng tổ hợp phím Ctrl + Shift + N :

Trong Visual C++ các bạn chọn Win32 và sau đó chọn Win32Project. Trong ô name các bạn chọn tên mà mình muốn đặt.Ở đây mình chọn FirstApp:

Tiếp theo bạn chọn Next rồi chọn như hình sau và kết thúc bằng click vào button Finish để tạo project:

Trên thanh Solution Explorer nằm bên tay phải (hoặc chọn View -> Solution Explorer). Ở trong đây, các bạn chuột phải vào Source Files -> Add -> New Item:

Tiếp theo các bạn chọn Visual C++ -> Code -> C++ file (.cpp) và đặt tên cho file của mình. Ở đây mình đặt tên là FirstApp:

Cuối cùng, các bạn thêm đoạn code trên vào file FirstApp.cpp và chạy thì sẽ được kết quả như hình bên dưới:

3. Cách tạo ra cửa sổ Windows trong Win 32

Ở đây ta sẽ quy ước Window là tên hệ điều hành còn window là cửa sổ của chương trình.

3.1. Khởi tạo Windows Class

Một window class sẽ lưu trữ thông tin về kiểu của window bao gồm window procedure, cái mà sẽ điều khiển window về kích thước, vị trí cũng như các icon, foreground và background của window. Do đó bạn có thể tạo ra được một window tùy theo ý của chúng ta. Khai báo tên của window class như sau :

```
const char g_szClassName[] = "myWindowClass";
```

Tiếp theo sẽ là thông số mà chúng ta cần khởi tạo :

WNDCLASSEX wc; wc.cbSize = sizeof(WNDCLASSEX); wc.style = 0; wc.lpfnWndProc = WndProc; wc.cbClsExtra = 0; wc.cbWndExtra = 0; wc.hInstance = hInstance; wc.hIcon = LoadIcon(NULL, IDI_APPLICATION); wc.hCursor = LoadCursor(NULL, IDC_ARROW); wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1); wc.lpszMenuName = NULL; wc.lpszClassName = g_szClassName; wc.hIconSm = LoadIcon(NULL, IDI_APPLICATION); if(!RegisterClassEx(&wc)) { MessageBox(NULL, "Window Registration Failed!", "Error!", MB_ICONEXCLAMATION | MB_OK); return 0; }

Đoạn chương trình trên sẽ được sử dụng trong hàm WinMain() để khởi tạo window của chúng ta. *WNDCLASSEX* là một struct chứa các thông tin về window của chúng ta. Các thành phần của struct này như sau:

- cbSize: Cho ta biết kích thước của struct.
- style: Thường được gán về 0.
- lpfnWndProc: Trỏ tới the window procedure cho window class này.
- cbClsExtra và cbWndExtra là các byte bổ sung. Do ta không dùng đến nó nên các giá trị này được gán về 0.
- hInstance: Khai báo handle đối với chương trình của chúng ta.
- hIcon: Khai báo icon trong chương trình(thường có kích thước 32×32).
- hCursor: Biểu tượng con trỏ được hiện lên trong chương trình của chúng ta.
- hbrBackground: Khai báo màu nền cho cửa sổ chính
- lpszMenuName: Khai báo tên menu của chương trình.Nhưng ở đây ta không khởi tạo nên giá trị này bằng 0.
- lpszClassName: Tên của cửa sổ chương trình của chúng ta.
- hIconSm: Icon của chương trình khi được hiển thị trên taskbar (thường là 16×16).
- 3.2 TẠO WINDOW

Sau khi đã khởi tạo xong ,ta có thể tạo được window. Chúng ta thực hiện điều này bằng hàm CreateWindowEx().

```
HWND hwnd;

 hwnd = CreateWindowEx(

 WS_EX_CLIENTEDGE,
```

```
g_szClassName,

“The title of my window”, WS_OVERLAPPEDWINDOW,

CW_USEDEFAULT, CW_USEDEFAULT, 240, 120,

NULL, NULL, hInstance, NULL);
```

1. Thông số thứ 1 WS_EX_CLIENTEDGE là thành phần mở rộng cho window style. WS_EX_CLIENTEDGE cho ta biết được window của chúng ta có phần biên bị hạ thấp xuống.Ngoài ra các bạn có thể khai báo về 0 để xem sự khác biệt.
2. Thông số thứ 2 là class name g_szClassName như đã nói phần trên.
3. Thông số thứ 3 là “The title of my window”.Tiltle chương trình của chúng ta.
4. Thông số thứ 4 WS_OVERLAPPEDWINDOW là một thành phần của Window Style.

Bốn thông số tiếp theo CW_USEDEFAULT, CW_USEDEFAULT, 320, 240 cho ta biết được vị trí của góc trên bên trái của chương trình của chúng ta (CW_USEDEFAULT, CW_USEDEFAULT) và chiều rộng, chiều cao của window. CW_USEDEFAULT cho ta biết rằng Window sẽ chọn nơi để đặt chương trình của chúng ta. Bốn thông số cuối (NULL, NULL, g_hInst, NULL) cho ta thấy handle đối với *Parent Window*, menu, application, and xem có trỏ tới data hay không. Ta sẽ thêm vào đoạn code sau để nếu chương trình tạo window lỗi , nó sẽ hiện ra thông báo để ta có thể biết được.

```
if(hwnd == NULL)

 { MessageBox(NULL, “Window Creation Failed!”, “Error!”,

 MB_ICONEXCLAMATION | MB_OK);

 return 0;

 }
```

Sau khi đã tạo ra window ,kiểm tra hàm tạo là hợp lệ thì ta sẽ cho hiện window lên màn hình.

ShowWindow(hwnd, nCmdShow);//hiển thị window UpdateWindow(hwnd);//cập nhật lại window của chúng ta khi có các hàm message được gửi đến.

3.3. Tạo vòng lặp cho hàm GetMessage

Đây là phần khá quan trọng trong chương trình của chúng ta.

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

GetMessage sẽ lấy message từ hàng đợi trong chương trình của chúng ta. Bất kì hoạt động gì như click chuột, gõ bàn phím… , message đều được sinh ra và gửi đến hàng đợi của chương trình. Điều này được thực hiện dựa vào hàm *TranslateMessage*. Sau đó hàm *DispatchMessage* sẽ gửi message của ta tới window để hiển thị.

3.4. Windows Procedure

Nếu như vòng lặp của chúng ta là trái tim thì window procedure sẽ là bộ não.Đây chính là nơi các message sẽ được xử lí.

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

WM_CLOSE được gửi khi chúng ta ấn vào hoặc ấn Alt+F4. Chương trình của chúng ta sẽ bị tắt trước khi chương trình thực hiện xong. Khi chúng ta gọi hàm DestroyWindow(), *WM_DESTROY* sẽ được gửi tới window, trong trường hợp này, chương trình sẽ tắt các chương trình con ta gọi ra trước khi tắt chương trình chính. *PostQuitMessage* được dùng khi ta đã thực hiện xong và muốn thoát chương trình. Lúc này WM_QUIT sẽ được gửi đến vòng lặp message và ta không nhận được message nào từ hàm *GetMessage()* nữa. Hàm này sẽ trả lại giá trị FALSE. Cuối cùng kết hợp các phần trên lại với nhau ta sẽ được kết quả như sau:

Trong bài này, mình đã trình bày với các bạn về cách tạo một window bằng Win32.Trong phần tới mình sẽ giới thiệu với các bạn về các kiểu dữ liệu trong Win32.