Windows Programming

Events

- One loop checking events
  - When catch event, sent to event-processing function
- Callback function
  - Triggered when some conditions are met, such as certain time passed
  - When windows receive event, we send it to the callback function

Events (cont.)

- In windows, only the main loop can receive events
  - We cannot call another function and have that function waiting for events

Standard naming prefix

- sz – character array
- s – string without a null terminator
- c – character
- b – boolean (0 or 1)
- i – integer 32 bit
- l – long
- ul – unsigned long

Standard naming prefix (cont.)

- dw – double word
- lp, lptr – long pointer
- g , g_ – global variable
- fn – function pointer
- h – handle
- v - void

Standard naming prefix example

```
int     iBytesReceived = 0;   // # of Bytes Received
int     iBytesSent = 0;       // # of Bytes Sent
char    szCommand[128];       // Chat Buffer
vServerConnection()
```

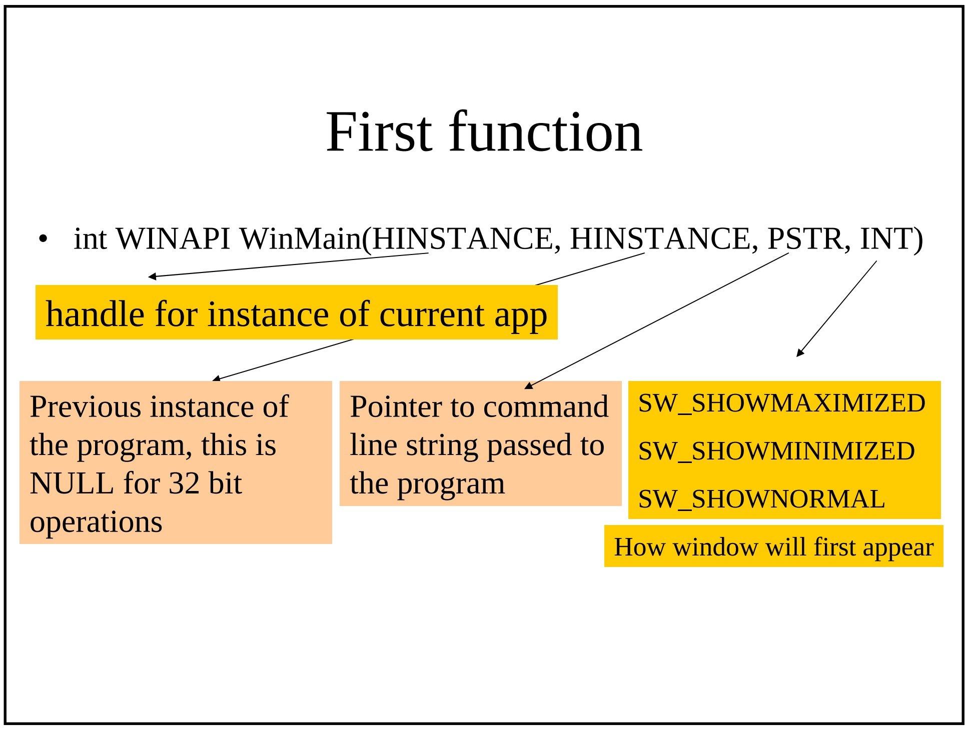
$\searrow$ เห็นชื่อก็บอก type ได้ทันที

MFC library

- MFC app wizard in Visual c++
- Bad points
  - library not written by Microsoft will not work
  - MFC programs are huge
  - Statically linked – huge size
  - Dynamically linked – read MFC dll in Windows directory, but dll changes easily
  - So MFC is not for game programming

Creating a Window

- First function
  - Defining + creating window
- Second function
  - Process all events and messages



First function

- int WINAPI WinMain(HINSTANCE, HINSTANCE, PSTR, INT)

handle for instance of current app

Previous instance of the program, this is NULL for 32 bit operations

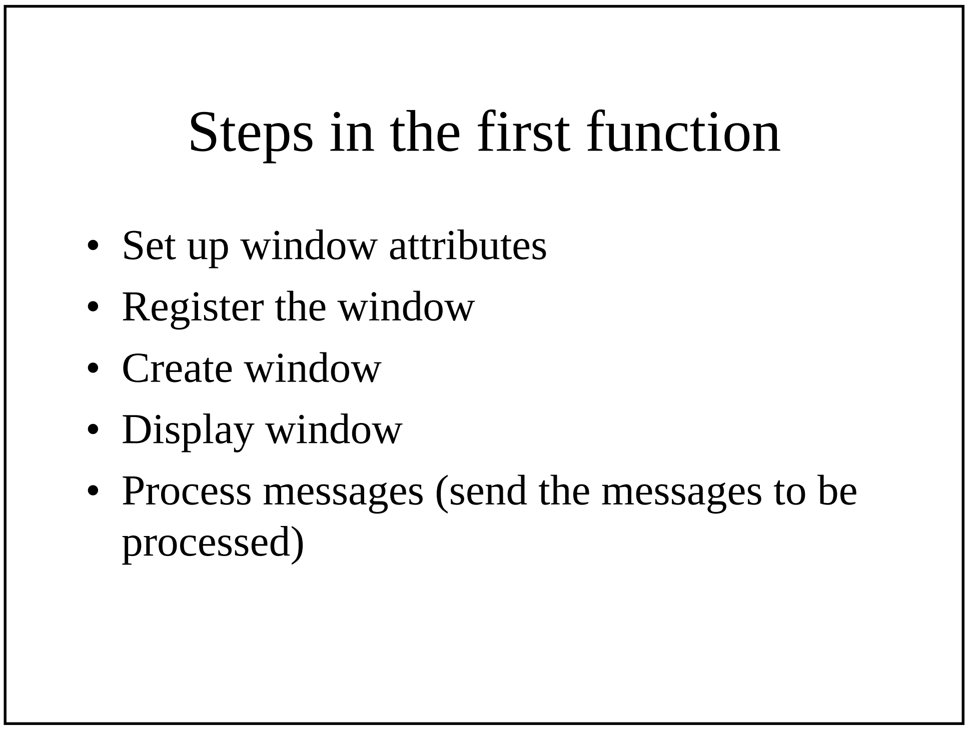
Pointer to command line string passed to the program

SW_SHOWMAXIMIZED

SW_SHOWMINIMIZED

SW_SHOWNORMAL

How window will first appear



Steps in the first function

- Set up window attributes
- Register the window
- Create window
- Display window
- Process messages (send the messages to be processed)

```
// Standard windows include
#include <windows.h>
```

$\longrightarrow$ ทุกวินโดวส์ต้องมี

```
// Message Loop CallBack Function prototype ( REQUIRED FOR ALL
    //WINDOWS PROGRAMS )
LRESULT CALLBACK fnMessageProcessor (HWND, UINT,
    WPARAM, LPARAM);

// Function to Create the Window and Display it ( REQUIRED FOR ALL
    //WINDOWS PROGRAMS )

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE
    hPrevInstance, PSTR szCmdLine, int iCmdShow)
{
    HWND              hWnd;
    MSG               msg;
    WNDCLASSEX        wndclass;
```

```
// Set up window attributes
wndclass.cbSize             = sizeof(wndclass);
wndclass.style              = CS_HREDRAW | CS_VREDRAW;
wndclass.lpfnWndProc        = fnMessageProcessor; //the message get sent here
wndclass.cbClsExtra         = 0;
wndclass.cbWndExtra         = 0;
wndclass.hInstance          = hInstance;
wndclass.hIcon              = LoadIcon( NULL, IDI_APPLICATION );
wndclass.hCursor            = LoadCursor( NULL, IDC_ARROW );
wndclass.hbrBackground      = (HBRUSH) GetStockObject (WHITE_BRUSH);
wndclass.lpszMenuName       = NULL;
wndclass.lpszClassName      = "Simple Window Class";//Registered Class Name
wndclass.hIconSm            = LoadIcon( NULL, IDI_APPLICATION );

if( RegisterClassEx( &wndclass ) == 0 ) // tell windows so we can use class by name
{
      // Do error logic here
      exit(1);
}
```

Pointer to our class object

Register allows- calling class name, creating window

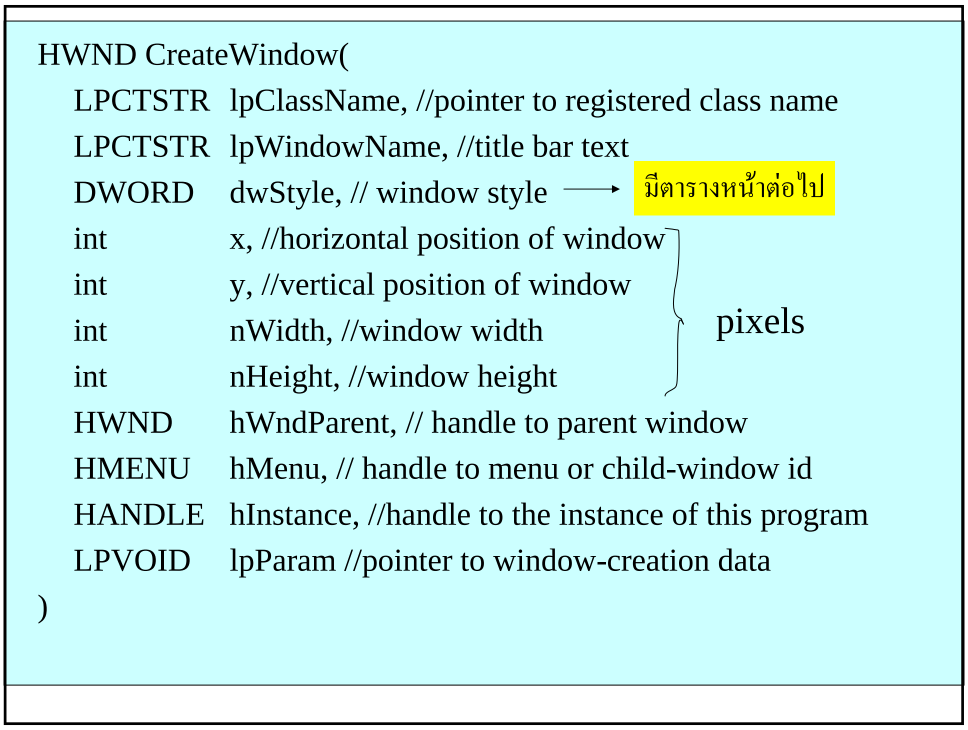
- cbSize - size of WNDCLASSEX data structure
- style - various styles can be combined with |
- lpfnWndProc - pointer to message-processing function (function name)
- cbClsExtra – number of bytes following the class structure (always 0)
- cbWndExtra – number of bytes following the Windows instance (always 0)
- hInstance – handle to the instance of of the window where class will run
- hIcon – handle to the icon (if null, the application must draw its own icon)
- hCursor - handle to the cursor (if null, the application must draw its own cursor)
- hbrBackground – handle to the background brush
- lpszMenuName - pointer to menu name (games don’t have one, so NULL)
- lpszClassName - pointer to class name
- hIconSm – handle to an icon used when the program is represented by a small icon

```
// Create the window
hWnd = CreateWindow("Simple Window Class", // Application Name
                    "Simple Window Program",   // Name Displayed on Title Bar

                    WS_OVERLAPPEDWINDOW, //style
                    CW_USEDEFAULT,
                    CW_USEDEFAULT,
                    CW_USEDEFAULT,
                    CW_USEDEFAULT,
                    NULL,
                    NULL,
                    hInstance,
                    NULL );

// Display the window
ShowWindow( hWnd, iCmdShow ); //return BOOL
UpdateWindow( hWnd ); //นี่คือการ repaint
```

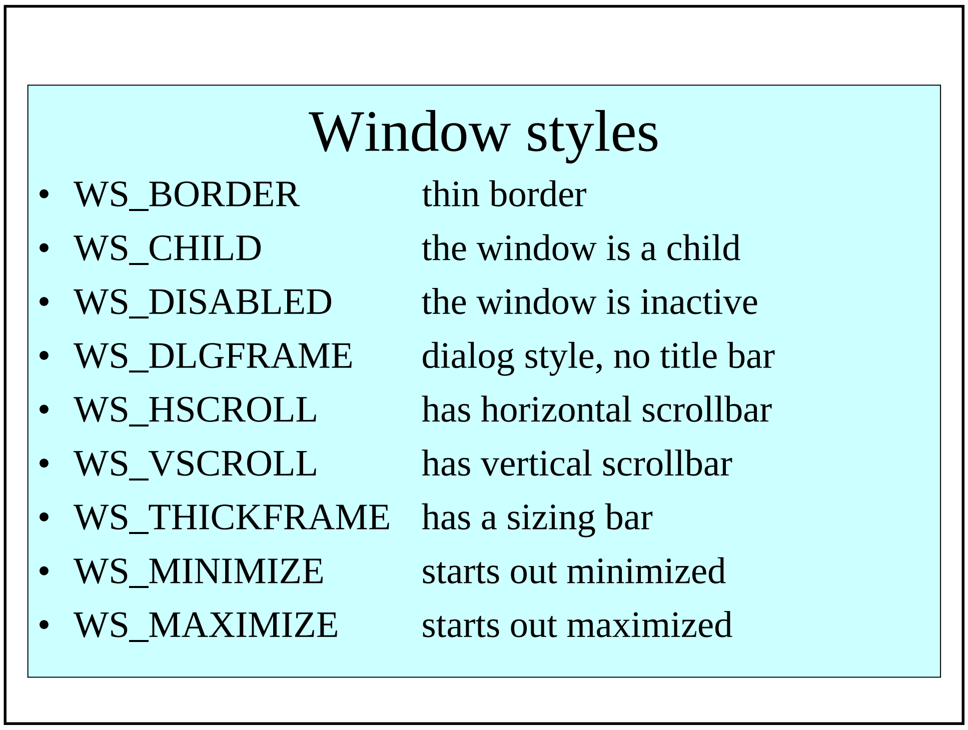
Use class name

HWND - Handle to the window we want to show

Integer –display state value (มีตาราง)



```
HWND CreateWindow(
    LPCTSTR  lpClassName, //pointer to registered class name
    LPCTSTR  lpWindowName, //title bar text
    DWORD    dwStyle, // window style  →  มีตารางหน้าต่อไป
    int      x, //horizontal position of window  ┐
    int      y, //vertical position of window    │
    int      nWidth, //window width              ├  pixels
    int      nHeight, //window height            ┘
    HWND     hWndParent, // handle to parent window
    HMENU    hMenu, // handle to menu or child-window id
    HANDLE   hInstance, //handle to the instance of this program
    LPVOID   lpParam //pointer to window-creation data
)
```



Window styles

• WS_BORDER	thin border
• WS_CHILD	the window is a child
• WS_DISABLED	the window is inactive
• WS_DLGFRAME	dialog style, no title bar
• WS_HSCROLL	has horizontal scrollbar
• WS_VSCROLL	has vertical scrollbar
• WS_THICKFRAME	has a sizing bar
• WS_MINIMIZE	starts out minimized
• WS_MAXIMIZE	starts out maximized

Display state

SW_SHOW	show window in current state
SW_HIDE	hide the window, activate another
SW_SHOWNORMAL	activate and display default state
SW_SHOWNA	show window in its current state but leaves the currently active window active
SW_RESTORE	restores from min or max
SW_MINIMIZE	minimize window and activate the next one
SW_MAXIMIZE	display maximize window
SW_SHOWMAXIMIZED	activate window and display it maximized
SW_SHOWMINIMIZED	activate window and display it minimized

```
    // Process messages until the program is terminated
    while( GetMessage ( &msg, NULL, 0, 0 ) )
    {
          TranslateMessage( &msg );
          DispatchMessage( &msg );
    }

    return ( msg.wParam );
}
```

GetMessage()

```
BOOL GetMessage( //return non-zero if success, or zero if
                 //WM_QUIT message is present, or –1 if fails
    LPMSG lpMsg, //pointer to msg
    HWND hWnd, //handle to window to get message from
    UINT wMsgFilterMin, //limit message range , just set to 0
    UINT wMsgFilterMax
)
MSG{
    HWND      hwnd; // window that receives message
    UINT      message; //message number
    WPARAM    wParam;
    LPARAM    lParam;
    DWORD     time; //the time the message was generated
    POINT     pt; //screen coordinate of the cursor when the
              //message was generated}
```

TranstaleMessage()

BOOL TranslateMessage(CONST MSG *lpMsg)

Pointer to msg that we just got from GetMessage()

Translate virtual key message to character message, then put it to be read by the next GetMessage() i.e. check for keyboard events and post them to the queue

DispatchMessage()

LONG DispatchMessage(CONST MSG *lpMsg)

This message is sent to the event procedure of this window

- In the example, it goes to fnMessageProcessor()

Second function – processing message

```
LRESULT CALLBACK fnMessageProcessor ( HWND hWnd, UINT iMsg,
    WPARAM wParam, LPARAM lParam ){
    switch( iMsg )
    {
        // Called when window is first created
        case WM_CREATE:
                return( 0 );
        // Called when the window is refreshed
        case WM_PAINT:
                return( 0 );
        // Called when the user closes the window or terminates the application
        case WM_DESTROY:
                PostQuitMessage( 0 );
                return( 0 );
    }
  // clean up with default processor
    return DefWindowProc( hWnd, iMsg, wParam, lParam );}
```

Window we are processing message

Message ID
มีตาราง

It processes any messages you left behind

Message ID

- WM_CREATE วินโดวส์ถูกสร้างจาก CreateWindow()
- WM_PAINT ต้องวาดวินโดวส์นี้ใหม่
- WM_QUIT ผู้ใช้ปิดวินโดวส์ไปแล้ว
- WM_TIMER เวลาหมด
- WM_KEYDOWN ผู้ใช้กดคีย์
- WM_KEYUP ผู้ใช้ปล่อยคีย์
- WM_MOUSEMOVE เมาส์ถูกเลื่อน
- WM_ACTIVATE วินโดวส์ถูก activate หรือ deactivate
- WM_SIZE วินโดวส์ถูกเปลี่ยนขนาด

Message ID (ต่อ)

- WM_LBUTTONDOWN ปุ่มเมาส์ซ้ายถูกกด
- WM_LBUTTONUP ปุ่มเมาส์ซ้ายถูกปล่อย
- WM_RBUTTONDOWN ปุ่มเมาส์ขวาถูกกด
- WM_RBUTTONUP ปุ่มเมาส์ขวาถูกปล่อย
- WM_SETFOCUS วินโดวส์ได้คีย์บอร์ดโฟกัส
- WM_DESTROY ได้เวลาปิดวินโดวส์แล้ว

Multithreading

- Each thread needs stack space (static memory space)
- Communicate by shared variable
  - Main can set value, and the thread will see
  - But what about writing to the shared space at the same time
  - Use mutex

Thread creation

```
// Memory shared with thread
struct SharedMemory
{
    int iQuit;
    int   iCredits;
};

// Our global shared memory
SharedMemory g_SharedMem;
// Thread function prototype
void thrCreditsThread(SharedMemory *sm);
```

```
int WINAPI WinMain (HINSTANCE hInstance,
   HINSTANCE hPrevInstance, PSTR szCmdLine, int
   iCmdShow)
{
   HWND              hWnd;
   MSG               msg;
   WNDCLASSEX   wndclass;
   int                    iStartCredits = 0;// Keep track of last
                                               //known credits

   // Thread Variables
   DWORD    dwCreditsID;
   HANDLE   hCreditsThreadHandle;
```

```
   hCreditsThreadHandle = CreateThread(
        NULL,  // Security, default is ok
        NULL,  // Initial stack size (in bytes), default (1MB)

        (LPTHREAD_START_ROUTINE )
   &thrCreditsThread,    // The thread function

        &g_SharedMem,  // Data to pass to the function
        NULL,              // Creation flags, keep NULL
        &dwCreditsID // Identifier (we don’t use this)
   );
```

Closing a thread

```
// Wait on the thread to finish (wait forever if necessary)
WaitForSingleObject( hCreditsThreadHandle, INFINITE );
```

Handle to thread we want to wait

Time to wait (millisec.)

```
// Free the credits thread
CloseHandle( hCreditsThreadHandle );
```

Setting Up Visual C++

- File-> new
- Under the projects tab, select win32 application
- Name your project (say, HelloWorld), then click ok
- choose the empty project

Setting Up Visual C++ (cont.)

- Now in the project window, go up to the menu and select
  - Project -> Add to Project ->C++ Source File
- Let’s create HelloWorld

Creating HelloWorld

- Try typing this code in the blank window

This will be HelloWorld.cpp

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

LPCTSTR         lpszApplicationName = "HelloWorld";
LPCTSTR         lpszTitle           = "Hello World Program";

// Message Loop CallBack Function prototype ( REQUIRED FOR ALL
   //WINDOWS PROGRAMS )
LRESULT CALLBACK fnMessageProcessor (HWND, UINT,
   WPARAM, LPARAM);
```

$\downarrow$

```
// Function to Create the Window and Display it ( REQUIRED FOR ALL
    //WINDOWS PROGRAMS )

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE
    hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
    MSG                     msg;
    HWND                    hWnd;
    WNDCLASSEX              wndclass;
    wndclass.cbSize         = sizeof(wndclass);
    wndclass.style          = CS_HREDRAW | CS_VREDRAW;
    wndclass.lpfnWndProc    = fnMessageProcessor;
    wndclass.cbClsExtra     = 0;
    wndclass.cbWndExtra     = 0;
    wndclass.hInstance      = hInstance;
    wndclass.hIcon          = LoadIcon( NULL, IDI_APPLICATION );
    wndclass.hCursor        = LoadCursor( NULL, IDC_ARROW );
    wndclass.hbrBackground          = (HBRUSH)(COLOR_WINDOW);
```

```
    wndclass.lpszMenuName           = NULL;
    wndclass.lpszClassName          = lpszApplicationName;
                                    // Registered Class Name
    wndclass.hIconSm        = LoadIcon( NULL, IDI_APPLICATION );
    if( RegisterClassEx( &wndclass ) == 0 ) {
        // Do error logic here
        exit(1);
    }
    // Create the window
    hWnd = CreateWindow(lpszApplicationName,
                lpszTitle,          // Name Displayed on Title Bar
                WS_OVERLAPPEDWINDOW,
                100,                // X-Starting Position
                100,                // Y-Starting Position
                400,                // Width
                340,                // Height
                NULL, // handle to parent window
                NULL, // handle to menu or child-window
                hInstance, //handle to the instance of this program
                NULL //pointer to window-creation data);
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    // Process messages until the program is terminated
    while( TRUE ) {
            // Check for Windows Events & Messages
            if (PeekMessage(&msg, NULL, 0, 0, PM_NOREMOVE)) {
                    if (!GetMessage(&msg, NULL, 0, 0))
                            break;
                    TranslateMessage(&msg);
                    DispatchMessage(&msg);
            }
    }

    return(msg.wParam);
}
```

```
// Windows Callback function to handle messages
LRESULT CALLBACK fnMessageProcessor ( HWND hWnd, UINT
    iMsg, WPARAM wParam, LPARAM lParam )
{
    switch (iMsg)
   {
            case WM_COMMAND:
                    break;
            case WM_CREATE:
                    break;
            case WM_DESTROY:
                    PostQuitMessage(0);
                    break;
            default:
                    return(DefWindowProc(hWnd, iMsg, wParam,
    lParam));
   }
    return(0L);
}
```

Then we can run it

- F7 to build (compile)
- Ctrl+F5 to execute
- A blank window with title bar “Hello World Program” will appear

PeekMessage

- Check for pending messages before trying to process them
- If you don’t use it, the program just waits for the next message

O if no message and non-zero otherwise

```
BOOL PeekMessage(LPMSG lpMsg,  // pointer to MSG we receive
            HWND hWnd, // handle to the window you want to
            //check, in case many windows open at the same time
```

Always 0 for game

```
            UINT wMsgFilterMin,  //starting range of message number to look at
            UINT wMsgFilterMax, // ending range of message number
            UINT wRemoveMsg // what to do after we peek
            )
```

PM_NOREMOVE -> messages stay in the queue
PM_REMOVE -> messages are removed from the queue

Adding static text to window

- See the code of this Statictext.cpp
- Almost the same as HelloWorld
- But has some additions

เลือกเลขอะไรก็ได้ที่ไม่ไปซ้ำกับ resource อื่นก็พอ

```
#include <windows.h>
#include <stdio.h>
// ID defines for static text resource object
#define         IDC_hBU_StaticText          40001

// create some global handles for our child window components
HWND            hBU_StaticText          = NULL;
LPCTSTR         lpszApplicationName     = "StaticText";
LPCTSTR         lpszTitle               = "Static Text Example";
```

Static text is actually a child window

//โค้ดเหมือนเดิมหมด จนถึงตอน createWindow เสร็จ

```
// Create the static text, it is a child window
    hBU_StaticText = CreateWindow(
            "static", // yes, the class name for static text is “static”
            "Static Text",
            WS_CHILD | SS_CENTER | WS_VISIBLE,
            120,
            130,
            135,
            28,
            hWnd, // the main window is the parent
            (HMENU)IDC_hBU_StaticText,
            hInstance,
            NULL);

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

These are based on parent window

ที่เหลือก็โค้ดเหมือนเดิมหมด

The execution result

- Window with “Static Text” word in the middle

Window class type

- STATIC
  - Text field or text box for labeling other objects
  - Does not accept input or output
- BUTTON
  - Can have text
  - Pressing it can trigger Windows message event
- SCROLLBAR
  - Clicking on it trigger Windows message event
- COMBOBOX

Window class type (cont.)

- EDIT
  - Used to create text entry box
- LISTBOX
  - Box containing rows of text. The text can be selected
- RICHEDIT_CLASS
  - Rich edit text window

Creating Buttons

- We will also use Winmm.lib, which we will use to play sound
- Code for Button.cpp is almost the same as before

```
#include <windows.h>
#include <stdio.h>
#define         IDC_hBU_Join            40001

// create some global handles for our child window components
HWND            hBU_Join                = NULL;

LPCTSTR         lpszApplicationName     = "ButtonProg";
LPCTSTR         lpszTitle               = "Button Example";
```

↓

```
//โค้ดเหมือนเดิมหมด จนถึงตอน createWindow
// Create the window
hWnd = CreateWindow(lpszApplicationName, lpszTitle,
WS_OVERLAPPEDWINDOW, 100, 100, 400, 340, NULL, NULL,
hInstance,  NULL );

// hBU_Join Button
hBU_Join = CreateWindow(
        "BUTTON",
        "Join",
        WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
        285,
        280,
        100,
        28,
        hWnd,(HMENU)IDC_hBU_Join,hInstance,NULL);
```

ที่เหลือก็โค้ดเหมือนเดิมหมด ยกเว้นใน message processor

```
LRESULT CALLBACK fnMessageProcessor ( HWND hWnd, UINT iMsg,
    WPARAM wParam, LPARAM lParam )
{
    switch (iMsg)
   {
        case WM_COMMAND:
                // Check for child window messages
                switch(LOWORD(wParam))
                {
                        // Check if the user clicked the button
                        case IDC_hBU_Join:
                                PlaySound( "join.wav", NULL,
                                SND_FILENAME | SND_ASYNC);
                        break;
                }
                break;
        case WM_KEYDOWN:
                break;
```

40001

```
            case WM_CREATE:
                        break;

            case WM_DESTROY:
                        PostQuitMessage(0);
                        break;

            default:
                        return(DefWindowProc(hWnd, iMsg, wParam,
      lParam));
     }
     return(0L);
 }
```

PlaySound()

True if succeed and false if fail

```
BOOL PlaySound(
    LPCSTR pszSound, // sound name (file name) and location
    HMODULE hmod, //
    DWORD fdwSound // flag
)
```

```
SND_FILENAME  //The sound is loaded from a file
SND_RESOURCE //The sound is already loaded as a resource
SND_LOOP // keep playing until NULL is passed as the sound name in
    // PlaySound()
SND_ASYNC // PlaySound() does not wait for any previous sound to finish
SND_NOSTOP // old sound is not interrupted by this new sound
```

ListBox

- It is a multiple line text field, each line is an item of the list box
- Let's modify the button example to have list box

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

#define            IDC_hBU_Join            40001
#define            IDC_hLB_Output          40002

// create some global handles for our child window components
HWND               hBU_Join                = NULL;
HWND               hLB_Output              = NULL;
```

```
LPCTSTR        lpszApplicationName = "ListBoxProg";
LPCTSTR        lpszTitle           = "ListBox Example";
//โค้ดเหมือนเดิมหมด จนถึงตอน create button เสร็จ

// hLB_Output                                        เดี๋ยวมีนิยาม
    hLB_Output = CreateWindowEx(
            WS_EX_CLIENTEDGE,
            "LISTBOX",
            NULL,
            WS_CHILD | WS_VISIBLE | LBS_NOTIFY | WS_VSCROLL |
    WS_BORDER,
            5,
            47,
            380,
            230,
            hWnd,(HMENU)IDC_hLB_Output,hInstance,NULL);

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

โค้ดเหมือนเดิมหมด จนถึง message processor

```
LRESULT CALLBACK fnMessageProcessor ( HWND hWnd, UINT iMsg,
    WPARAM wParam, LPARAM lParam )
{
    char  szText[256];
    static count = 0;

    switch (iMsg)
   {
          case WM_COMMAND:
                    // Check for child window messages
                    switch(LOWORD(wParam))
                    {
                              // Check if the user clicked the button
                              case IDC_hBU_Join:
                                        sprintf(szText,"Join Number %d",count++);
                                        PlaySound("join.wav", NULL,
                                        SND_FILENAME|SND_ASYNC);
                                        vShowText(hLB_Output,szText);
                                        break;
                    }
```

โค้ดเหมือนเดิมหมด จนถึงเมธอด vShowText

```
// Function to display text into the edit window
//
void vShowText(HWND   hChildHandle, char *szText)
{
    int Line;

    // add string to the listbox
    SendMessage(hChildHandle,LB_ADDSTRING,0,(LPARAM)szText);

    // determine number of items in listbox
    Line = SendMessage(hChildHandle,LB_GETCOUNT,0,0);

    // flag last item as the selected item, to scroll listbox down
    SendMessage(hChildHandle,LB_SETCURSEL,Line-1,0);

    // unflag all items to eliminate negative highlight
    SendMessage(hChildHandle,LB_SETCURSEL,-1,0);
}
```

เดี๋ยวมีนิยาม

CreateWindowEx()

- Creating window, with extended style as its first parameter

```
WS_EX_CLIENTEDGE  // The window has a sunken edge
ES_EX_TRANSPARENT //child windows underneath this
                  //window are visible
WS_EX_TOPMOST // The window always stays on top
WS_EX_CONTROLPARENT // user can use Tab to switch to
                    //other child windows
WS_EX_DLGMODALFRAME //the window has double border,
                    //which can have a name if you use
                    //WS_CAPTION
```

SendMessage()

- Sends a message directly to a window
- This function calls the target window’s message procedure
- This function does not return until the sent message is processed

SendMessage() cont.

```
LRESULT SendMessage(
    HWND      hWnd,  // destination window handle
    UINT      Msg,    // Message to send
    WPARAM  wParam, //Parameter one
    LPARAM   lParam  // Parameter two
)
```

What happens in vShowText

Text to send

1

```
SendMessage(hChildHandle,LB_ADDSTRING,0,(LPARAM)szText);
```

The window does not scroll, but we want to see the last item without manually scrolling. So we need to fix it.

What happens in vShowText (cont.)

2

```
// determine number of items in listbox
Line = SendMessage(hChildHandle,LB_GETCOUNT,0,0);
```

3

```
// flag last item as the selected item, to scroll listbox down automatically
SendMessage(hChildHandle,LB_SETCURSEL,Line-1,0);
```

Set-current-selection

4

```
// unflag all items to eliminate what we highlighted highlight
SendMessage(hChildHandle,LB_SETCURSEL,-1,0);
```

Editable text box (i.e. Edit Fields)

- Let’s modify our program to have an additional edit field

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

#define         IDC_hBU_Join         40001
#define         IDC_hLB_Output       40002
#define         IDC_hEB_Name         40003

// create some global handles for our child window components
HWND            hBU_Join                    = NULL;
HWND            hEB_Name                    = NULL;
HWND            hLB_Output                  = NULL;

LPCTSTR         lpszApplicationName = "EditFieldProg";
LPCTSTR         lpszTitle           = "EditField Example";
```

```
//โค้ดเหมือนเดิมหมด จนถึงตอน create list box เสร็จ ↓

// Name
hEB_Name = CreateWindowEx(
        WS_EX_CLIENTEDGE,
        "EDIT","Your Name",
        WS_CHILD | WS_VISIBLE | WS_BORDER | ES_LEFT,
        250,
        20,
        135,
        28,
        hWnd,(HMENU)IDC_hEB_Name,hInstance,NULL);

//โค้ดเหมือนเดิมหมด จนถึง message processor ↓
```

```
LRESULT CALLBACK fnMessageProcessor ( HWND hWnd, UINT iMsg,
    WPARAM wParam, LPARAM lParam ){
    char  szText[256];
    static count = 0;
    char  szName[256];
    switch (iMsg)
   {
         case WM_COMMAND:
                    // Check for child window messages
                    switch(LOWORD(wParam))
                    {
                              // Check if the user clicked the button
                              case IDC_hBU_Join:
                              // Get the name from the name edit window
                                       GetWindowText( hEB_Name, szName,256);
                                       sprintf(szText, "<%s> Join Number %d",
                                       szName,count++);
```

ที่เหลือโค้ดเหมือนเดิมหมด

GetWindowText()

- Copies text from a window control to a specified buffer

if successful, returns the number of characters retrieved,otherwise returns 0

```
int GetWindowText(
    HWND       hWnd, // handle of the window you want to get text
    LPTSTR     lpString, // buffer
    int nMaxCount //maximum number of chars to retrieve
)
```

Exercise: create graphical chat program

The End :P