

Windows Programming

Processing Messages

Myung-Hun Kang
freehuni@mmlab.net

[Multimedia Lab.](#)
Dept. of Electrical and Computer Eng.
University of Seoul
Seoul, Korea

1. Contents

- What are Messages ?
- Responding to a key Press.
- Device Context
- Processing the WM_PAINT Message.
- Responding to Mouse Message.
- Generating a WM_PAINT Message.
- Generating Timer Message.

2. Main Objectives

- Win NT skeleton will be expanded to receive and process several common **messages**.

3.1 What are Message - con't

- Message is represented by a unique 32-bit integer value.
- Generally, **use the macro name**, not the actual integer value, when referring to a message.
 - WM_CHAR (contains the character code of the key that was pressed)
 - WM_PAINT (This message is sent by an application when Windows or another application makes a request to paint a portion of an application's window)
 - WM_MOVE (This message is sent after a window has been moved)
 - WM_LBUTTONDOWN (This message is posted when the user releases the left mouse button while the cursor is in the client area of a window)
 - WM_LBUTTONUP

3.1 What are Message – con't

- Two other values accompany each message and contain information related to the specific message.
 - **WPARAM** (called wParam)
 - **LPARAM** (called lParam)
 - Hold things like cursor or mouse coordinates
 - **Value of key press**
 - System related value (ex, character size)

3.1 What are Message

- TO provide easy access to each half of wParam and lParam, defines two macro called LOWORD and HIWORD
- Used like this (ex:WM_MOUSEMOVE)
 - X = LOWORD(lParam)
 - Y = HIWORD(lParam)
 - **DWORD MAKELONG(WORD wLow, WORD wHigh);**

3.2 Responding to a Key Press - con't

- Each time a key is press, a WM_CHAR message is sent to the active window.
 - Each time a WM_CHAR is sent, wParam contains the ASCII value of the key pressed
 - LOWORD(lParam) contains the number of times the key has been repeated as a result of the key being held down
 - HIWORD(lParam) are encoded as shown

15: Set if the key being release; cleared if the key had been released and is now being pressed

14: Set if the key was pressed before the message was sent;cleared if it was not press

.....

3.2 Responding to a Key Press – con't

- To process a WM_CHAR message, add it the switch statement inside program's window function.
- Before performing output, your program must first acquire permission.
 - This is calling **GetDC()**
 - This obtain a **device context**
 - This function **retrieves a handle** to a display device context (DC) for the client area of the specified window.
 - The device context is released using ReleaseDC()

3.2 Responding to a Key Press – con't

- **Its prototype**
 - `HDC GetDC(HWND hwnd);`
- **Parameter**
 - `hWnd`
 - Handle to the window whose device context is to be retrieved
 - If this value is `NULL`, `GetDC` retrieves the device context for the entire screen.
 - **Return Values**
 - Successful : Handle the device context
 - Failure : `NULL`

2006-10-25

Myung-Hun Kang

CHP_3-9

3.2 Responding to a Key Press – con't

- Release the device context when it is done with it
- Using `ReleaseDC()`
- **Its prototype is**
 - `int ReleaseDC(HWND hwnd,HDC hdc);`
- **Parameters**
 - `hWnd`
 - Handle to the window whose device context is to be released.
 - `hDC`
 - Handle to the device context to be released.
- **Return Values**
 - 1 indicates that the device context is released.
 - 0 indicates that the device context is not released.

2006-10-25

Myung-Hun Kang

CHP_3-10

3.2 Responding to a Key Press – con't

- The function that actually outputs the character is the API function **TextOut()**
- Its prototype is
 - BOOL TextOut (
 - HDC DC, // handle to DC
 - int X, // x-coordinate of starting position
 - int Y, // y-coordinate of starting position
 - LPCTSTR lpStr, // Pointer to the string to be drawn.
 - int ncount // Specifies the length of the string
 -);
- Return Values
 - succeeds : nonzero
 - Fails : zero

3.3 DEVICE CONTEXT

- Output path from your Windows NT application, through the appropriate device driver, to the client area of window
- Also fully defines the state of the device driver.
- Until a device context obtained, there is no linkage between program and the window relative to output.
- A [device context](#) is a **structure** that defines a set of graphic objects and their associated attributes, as well as the graphic modes that affect output.
 - [Ref. MSDN Library](#)

3.3 Pressing the WM_PAINT Message-con't

- Character is not redisplay
 - Minimize or overwritten
- Windows NT **does not keep a record** of what a window contains.
- It is program's job to maintain the contents of windows
- Each time the contents of a window must be redisplayed, program will be sent a WM_PAINT message
 - This message will be also be sent when window is first display.

2006-10-25

Myung-Hun Kang

CHP_3-13

3.3 Pressing the WM_PAINT Message-con't

- Processing a **WM_PAINT** message is to add its case to the switch statement inside the window function.
 - case WM_PAINT: //process a repaint request
 - hdc = BeginPaint(hwnd, &paintstruct); // get DC
 - TextOut(hdc, 1, 1, str, strlen(str));
 - EndPaint(hwnd, &paintstruct);
 - break;

2006-10-25

Myung-Hun Kang

CHP_3-14

3.3 Pressing the WM_PAINT Message

- Device Context is obtained using call to **BeginPaint()** instead of **GetDC()**
 - Prototype
 - HDC `BeginPaint(HWND hwnd, LPPAINTSTRUCT lpPaint);`
 - Parameter
 - `hwnd`
 - Handle to the window to be repainted.
 - `lpPaint`
 - Long pointer to the `PAINTSTRUCT` structure that will receive painting information.
- Released with a call to **EndPaint()**

3.4 Responding to Mouse Message-con't

- Mouse-based operation System
- This section examine the two most common.
 - **WM_LBUTTONDOWN**
 - Generated when the left button is pressed
 - **WM_RBUTTONDOWN**
 - Generated when the left button is pressed
 - To begin , add the responses to the two mouse messages to the switch statement in the window function.

3.4 Responding to Mouse Message-con't

- Shown here

```
- case WM_LBUTTONDOWN:  
    hdc = GetDC(hwnd);  
    strcpy(str, "Left Button is down");  
    TextOut(hdc, LOWORD(IParam), HIWORD(IParam), str,  
           strlen(str));  
    ReleaseDC(hwnd, hdc);  
    break;
```

- **Mouse's current X,Y location** is specified in **LOWORD(IParam)** and **HIWORD(IParam)**

3.4 Responding to Mouse Message-con't

- A Closer Look at the Mouse Message

- Each time a **WM_LBUTTONDOWN** or a **WM_RBUTTONDOWN** message is generated, several pieces of information are also supplied in the **wParam** parameter.

- **WM_LBUTTONDOWN**

- **fwKeys = wParam;** // key flags
- **xPos = LOWORD(IParam);** // horizontal position of cursor
- **yPos = HIWORD(IParam);** // vertical position of cursor

3.5 Generating a WM_PAINT Message

- Win NT is multitasking system
- Program has information to output, it simply requests that a WM_PAINT message be sent when windows NT is ready to do so.
- To cause windows NT to send WM_PAINT message, program will call [the InvalidateRect\(\)](#) API function.
 - When InvalidateRect() is called, it tells windows NT that the window is invalid and must be redrawn.
 - Causes Windows NT to send a WM_PAINT message to the window.

3.5 Generating a WM_PAINT Message

- InvalidateRect()
 - prototype
 - `BOOL InvalidateRect(HWND hwnd, CONST RECT *lpRect, BOOL bErase);`
 - parameters
 - hwnd
 - Handle to the window whose update region has changed
 - NULL : the system invalidates and redraws all windows
 - lpRect
 - Pointer to a [RECT](#) structure that contains the client coordinates of the rectangle to be added to the update region
 - NULL : the entire client area is added to the update region.
 - bErase
 - TRUE : the background is erased when the [BeginPaint](#) function is called
 - FALSE : the background remains unchanged.

3.6 Generating Timer Messages - con't

- It is possible to establish a timer that will interrupt program at periodic intervals.
- Each time the timer goes off, it sends a `WM_TIMER` message to window function.
- To start timer, use the `SetTimer()` API function
- Terminate the application or program executes a call to the `KillTimer()` API function.
- Its [prototype](#)
 - Ref. MSDN Library

2006-10-25

Myung-Hun Kang

CHP_3-21

3.6 Generating Timer Messages - con't

- `SetTimer()`
 - prototype
 - `UINT SetTimer(HWND hWnd, UINT_PTR nIDEvent, UINT uElapse, TIMERPROC lpTimerFunc);`
 - parameters
 - `hWnd`
 - Handle to the window to be associated with the timer
 - `nIDEvent`
 - Specifies a nonzero timer identifier
 - `uElapse`
 - Specifies the time-out value, in milliseconds.
 - `lpTimerFunc`
 - Pointer to the function to be notified when the time-out value elapsed
 - `NULL`: the system posts a [WM_TIMER](#) message to the application queue.

2006-10-25

Myung-Hun Kang

CHP_3-22

3.6 Generating Timer Messages - con't

- KillTimer()
 - prototype
 - **BOOL KillTimer(HWND *hWnd*, UINT_PTR *uIDEvent*);**
 - parameter
 - *hwnd*
 - Handle to the window associated with the specified timer
 - *uIDEvent*
 - Specifies the timer to be destroyed.

3.6 Generating Timer Messages

- **WM_TIMER**
 - *wTimerID* = *wParam*;
 - *tmprc* = (**TIMERPROC ***) *lParam*;
- **Parameters**
 - *wTimerID*(*wParam*): Specifies the timer identifier.
 - *Tmprc* (*lParam*): contains the time that the event occurred

4. Summary