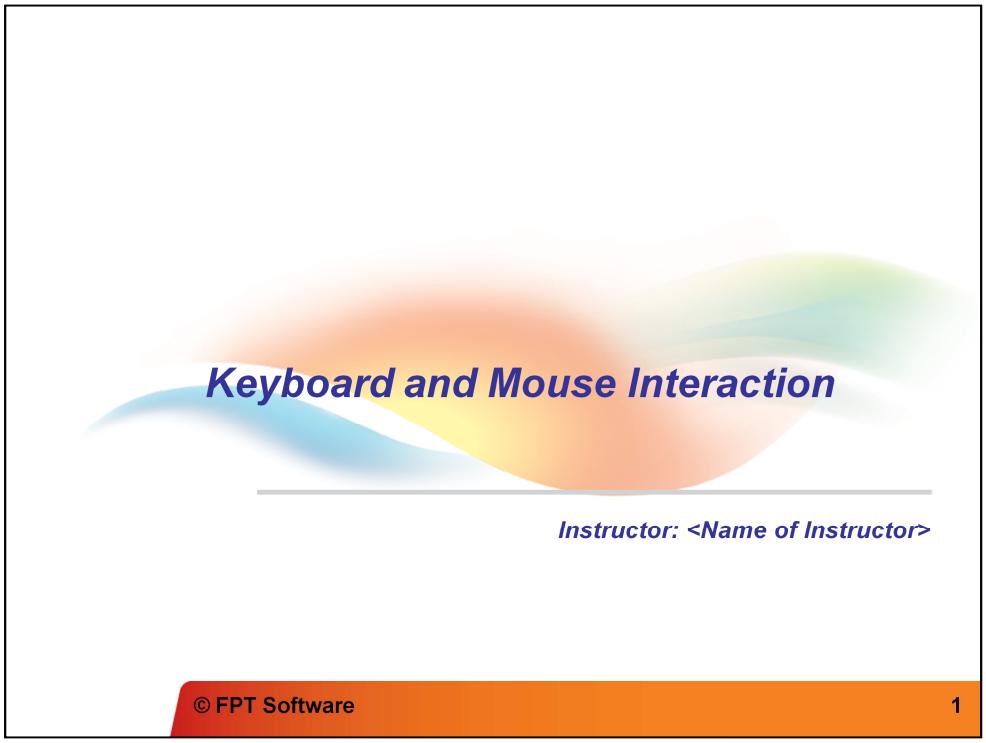


Latest updated by: HanhTT1

Agenda

- Keyboard Interaction
- Mouse Interaction
- Child Window Controls

Keyboard Interaction

Keyboard Basics

- Keyboard input is delivered to program's window procedure in the form of messages
- Handling the keyboard is knowing which messages are important and which are not
- Program does not need to act on every keyboard messages it receives, WINDOWS handles many keyboard functions itself

Keyboard Basics

- Ignoring the keyboard :
  - ALT key
  - Keyboard accelerators (for example : Ctrl – S)
  - Edit controls on Dialogs
  - Child controls
  - . . .

Keyboard Basics

- Queues and Synchronization :
  - User presses/releases keys on keyboard
  - Windows and keyboard device driver translates hardware scan code into formatted messages
  - Windows stores these messages in system message queue
  - Application has finished processing a previous user input message in application's message queue
  - Windows takes message from system message queue and places into application's message queue

Keyboard Basics

- Keyboard messages distinguish between :
  - Keystrokes
  - Characters
- For keystrokes combinations that result in displayable characters, Windows sends to program both keystroke messages and character messages
- Some keys do not generate characters, Windows generates and sends to program only keystroke messages

Keystroke Messages

- When user presses a key, Windows places either a WM_KEYDOWN or WM_SYSKEYDOWN message in the application's message queue that is set focus by Windows
- When user releases a key, Windows places either a WM_KEYUP or WM_SYSKEYUP message in the application's message queue that is set focus by Windows
- Usually, the up and down messages occur in pairs

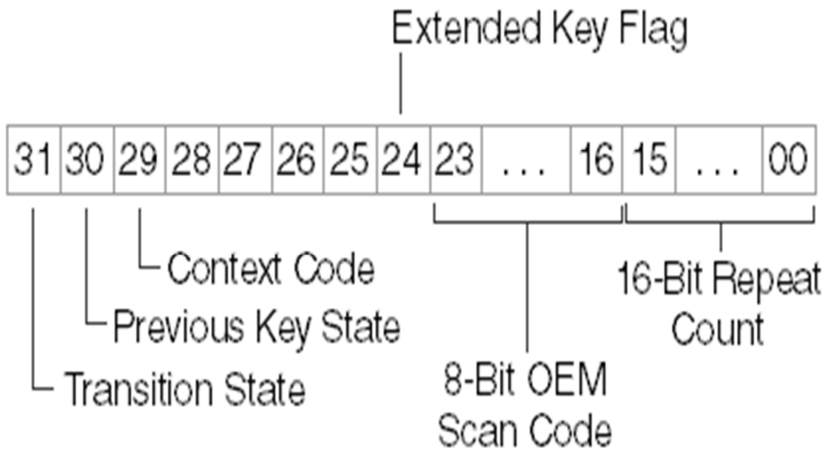
Keystroke Messages

- For all four keystroke messages :
  - **wParam** is a virtual key code that identifies the key is being pressed or released
  - **lParam** contains information about status of key
- Virtual key code is a representation for hardware scan code of keys for processing keyboard in a device independent manner

Virtual Key Codes

Decimal	Hexa	Defined in WINUSER.H	IBM – Compatible Keyboard
8	08	VK_BACK	Backspace key
9	09	VK_TAB	Tab key
12	0C	VK_CLEAR	Numeric keyboar 5 with Num Lock off
13	0D	VK_RETURN	Enter key
16	10	VK_SHIFT	Shift key
17	11	VK_CONTROL	Ctrl key
18	12	VK_MENU	Alt key
19	13	VK_PAUSE	Pause key
27	1B	VK_ESCAPE	Esc key
32	20	VK_SPACE	Space bar key

lParam Information



lParam Information

- **Repeat Count** : is the number of keystrokes represented by the message. In most cases, this will be set to 1
- **OEM Scan Code** : is the code generated by the hardware of keyboard
- **Extended Key Flag** : specifies that the keystroke results from one of the additional keys on the IBM enhanced keyboard
- **Context Code** : is 1 if the ALT key is depressed during the keystroke
- **Previous Key State** : specifies status of key before generating message
- **Transition State** : is 0 if the key is being pressed and 1 if the key is being released

Shift States

- When process a keystroke message, program may need to known whether any of the shift keys (Shift, Ctrl, Alt) or toggle keys (Caps Lock, Num Lock, Scroll Lock) are pressed
- Obtain this information by use function :

  SHORT GetKeyState (int nVirtKey);

  – For **Caps Lock** button :

  ```
  int iState = 0;
  iState = GetKeyState (VK_CAPITAL);
  ```

  – For **Shift** button :

  ```
  int iState = 0;
  iState = GetKeyState( VK_SHIFT);
  ```

Handle KeyStroke Message

- Basic Logic :

```
case WM_KEYDOWN :
        switch (wParam) {
                case VK_HOME :
                        // Handling [Home] key message
                        break;
                case VK_END :
                        // Handling [End] key message
                        break;
                        ...
        }
        // Other processing
        return 0;
```

Character Messages

- The Four Character Messages
- Message Ordering
- Control Character Processing

Character Messages

- The four character messages :
  - WM_CHAR
  - WM_DEADCHAR
  - WM_SYSCHAR
  - WM_SYSDEADCHAR
- With character messages, **wParam** is not a virtual key code. Instead, it is ANSI or UNICODE character code
- Get character code in **WM_CHAR** message :

```
(TCHAR) wParam;
```

Message Ordering

- Character messages are delivered to window procedure sandwiched between keystroke messages.
- For instance, if Caps Lock is not toggled on, presses and release 'A' key, the window procedure will receive the following messages
-

Message	Key or Code
WM_KEYDOWN	Virtual key code for 'A' (0x41)
WM_CHAR	Character code for 'a' (0x61)
WM_KEYUP	Virtual key code for 'A' (0x41)

Message Ordering

- If type an uppercase A by pressing the Shift key, pressing the A key, releasing the A key, and then releasing the Shift key, the Window procedure receives five messages :

Message	Key or Code
WM_KEYDOWN	Virtual key code for VK_SHIFT (0x10)
WM_KEYDOWN	Virtual key code for 'A' (0x41)
WM_CHAR	Character code for 'A' (0x41)
WM_KEYUP	Virtual key code for 'A' (0x41)
WM_KEYUP	Virtual key code for VK_SHIFT (0x41)

Control Character Processing

- Need to read keyboard character input, program should process the WM_CHAR message
- Need to read cursors keys, function keys, Delete, Insert, Shift, Ctrl and Alt, program should process the WM_KEYDOWN message

Control Character Processing

- Basic logic of control character processing :

```
case WM_CHAR:
    [other program lines]
    switch (wParam) {
        case `\b':          // backspace
                    [processing control character]
                    break ;
        case `\t':          // tab
                    [processing control character]
                    break ;
        case `\n':          // linefeed
                    [processing control character]
                        break ;
        case `\r':          // carriage return
                    [processing control character]
                        break ;
        default:            // character codes
                    [processing control character]
                    break ;
    }
return 0 ;
```

Mouse Interaction

Mouse Basics

- Determine if a mouse is present :

  int fMouse = GetSystemMetrics (SM_MOUSEPRESENT) ;
- Determine the number of buttons on the installed mouse :

  int cButtons = GetSystemMetrics (SM_CMOUSEBUTTONS) ;
- Load cursor of mouse :

  HCURSOR hCursor = LoadCursor (NULL, IDC_ARROW);

Client-Area Mouse Message

- Window procedure receives mouse messages whenever the mouse passed over the window or is clicked within the window, event if the window is not active or does not have the input focus
- **wParam**'s value specifies the state of the mouse buttons and the Shift and Ctrl keys
- **lParam**'s value contains the position of the mouse
  - The low word is the x-coordinate
  - The high word is the y-coordinate
  - Extracted these values :

```
X = LOWORD (lParam);
Y = HIWORD (lParam);
```

Client-Area Mouse Message

Message	Sent When
WM_MOUSEMOVE	The mouse is moved over the client area of window
WM_LBUTTONDOWN	Left button is pressed
WM_MBUTTONDOWN	Middle button is pressed
WM_RBUTTONDOWN	Right button is pressed
WM_LBUTTONUP	Left button is released
WM_MBUTTONUP	Middle button is released
WM_RBUTTONUP	Right button is released
WM_LBUTTONDBLCLK	Left button is double-clicked
WM_MBUTTONDBLCLK	Middle button is double-clicked
WM_RBUTTONDBLCLK	Right button is double-clicked

Client-Area Mouse Message

- Basic logic of handling client-area mouse message :

```
switch (message) {
        case WM_LBUTTONDOWN:
                // Handling pressing left mouse button message
                        return 0 ;
        case WM_MOUSEMOVE:
                        // Handling mouse moving message
                        return 0 ;
        case WM_LBUTTONUP:
                        // Handling releasing left button message
        return 0 ;
                        . . .
}
```

Client-Area Mouse Message

- Processing Shift Keys :

```
if (wParam & MK_SHIFT) {
        if (wParam & MK_CONTROL) {
                [Shift and Ctrl keys are down]
        } else {
                [Shift key is down]
        }
} else {
        if (wParam & MK_CONTROL] {
                [Ctrl key is down]
        } else {
                [neither Shift nor Ctrl key is down]
        }
}
```

Client-Area Message

- Mouse double-click :
  - Set window's style for receiving mouse double-click message :

    wndclass.style = CS_HREDRAW | CS_VREDRAW | **CS_DBLCLKS** ;

- If include **CS_DBLCLKS** in window style, window procedure receives these follow messages for a double-click :
  - WM_LBUTTONDOWN
  - WM_LBUTTONUP
  - WM_LBUTTONDBLCLK
  - WM_LBUTTONUP

- If not include **CS_DBLCLKS** in window style, window procedure receives these follow messages for a double-click :
  - WM_LBUTTONDOWN
  - WM_LBUTTONUP
  - WM_LBUTTONDOWN
  - WM_LBUTTONUP

Non-client Area Mouse Message

- If the mouse is outside window's client area but within the window, Windows sends to window procedure a "non-client area" mouse message
- Non-client area of window includes the title bar, the menu, and the window scroll bars
- In usually, no need to process non-client are mouse messages. Instead, program simply pass them to **DefWindowProc()** so that Windows can perform system functions

Non-client Area Mouse Message

Message	Sent When
WM_NCMOUSEMOVE	The mouse is moved over the non-client area of window
WM_NCLBUTTONDOWN	Left button is pressed in non-client area of window
WM_NCMBUTTONDOWN	Middle button is pressed in non-client area of window
WM_NCRBUTTONDOWN	Right button is pressed in non-client area of window
WM_NCLBUTTONUP	Left button is released in non-client area of window
WM_NCMBUTTONUP	Middle button is released in non-client area of window
WM_NCRBUTTONUP	Right button is released in non-client area of window
WM_NCLBUTTONDBLCLK	Left button is double-clicked in non-client area of window
WM_NCMBUTTONDBLCLK	Middle button is double-clicked in non-client area of window
WM_NCRBUTTONDBLCLK	Right button is double-clicked in non-client area of window

Non-client Area Mouse Message

- The **wParam** parameter indicates the non-client area where the mouse was moved or clicked
- The **lParam** parameter contains an x-coordinate in low word and a y-coordinate in the high word
- The coordinates stored in **lParam** are screen coordinates
- Convert screen coordinates to client-area coordinates and vice versa :
  - ScreenToClient (hWnd, &pt);
  - ClientToScreen (hWnd, &pt);

The Hit-Test Message

- WM_NCHITTEST : stands for “non-client hit test”. This message precedes all other client-area and non-client area mouse messages
- The **lParam** parameters contains the x and y screen coordinates of the mouse position
- The **wParam** parameter is not used
- Windows applications generally pass this message to **DefWindowProc()**, Windows then uses WM_NCHITTEST message to generate all other mouse messages based on the position of the mouse

Capturing The Mouse

- Window procedure normally receives mouse messages when the mouse cursor is in client area or non-client area of the window
- However, a program might need to receive mouse messages when the mouse is outside the window. If so, the program could “capture” the mouse
- For capturing the mouse :
  - SetCapture (hWnd);
- For releasing capture the mouse :
  - ReleaseCapture ();


Q&A