Lecture

16

Review of Last Lecture

- Keyboard
- Mouse
- Timer

Keyboard Input

```
getch();
```

Keyboard Input

Extended Keyboard Characters

Extended Keyboard Characters

1st byte: 0

2nd byte: scan code

Status Polling

vs

Message Driven Programming

Keyboard Messages

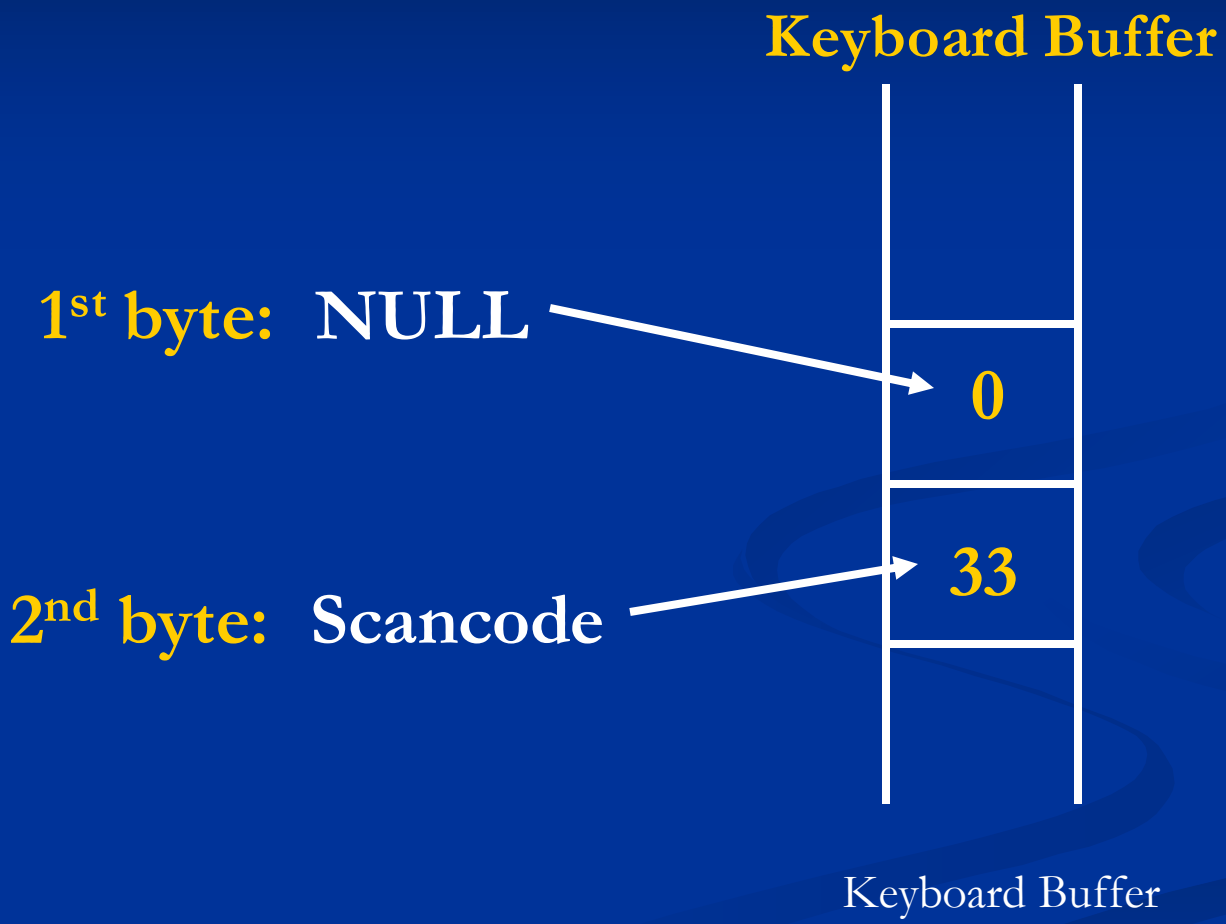
WM_KEYDOWN

WM_KEYUP

WM_SYSKEYDOWN

WM_SYSKEYUP

Pressing ALT+F in DOS



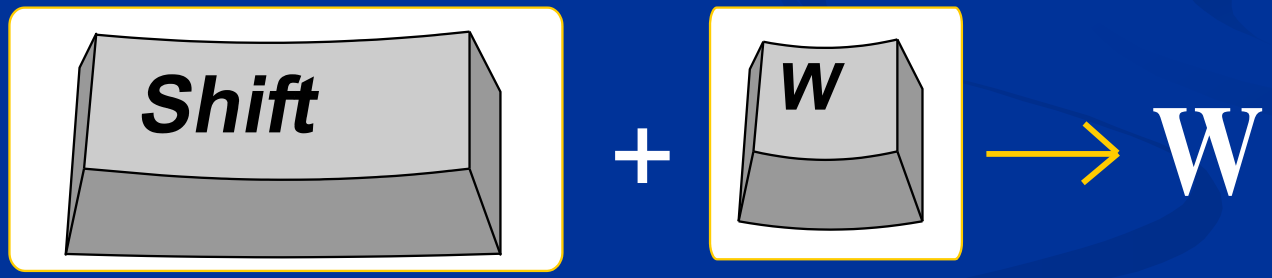
Combinations of keys with ALT generate WM_SYSKEYDOWN

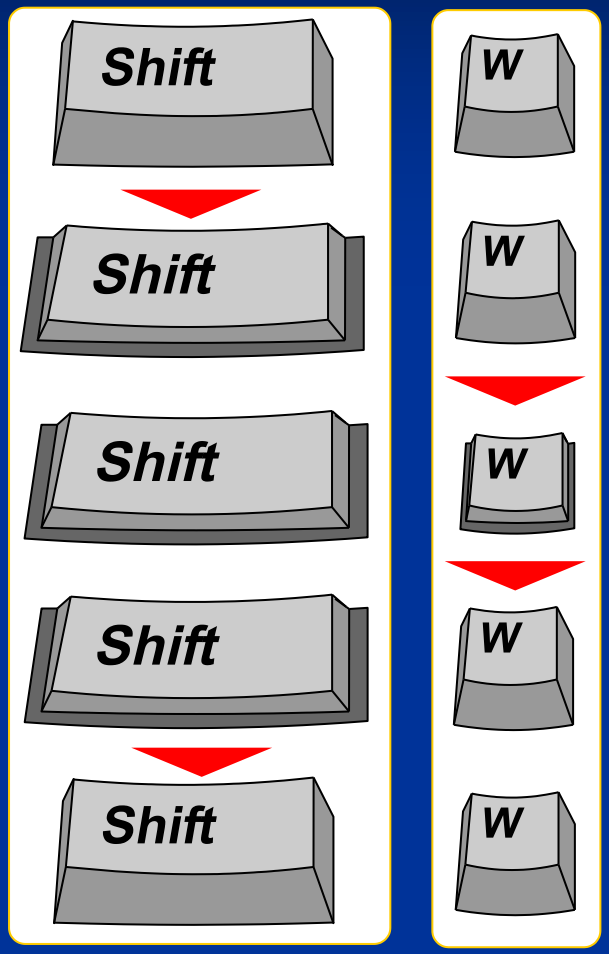
Keyboard Messages

Keystroke Messages

Character Messages

DOS





Initial State

WM_KEYDOWN
(Shift Pressed)

WM_KEYDOWN
(W Pressed)

WM_KEYUP
(W Released)

WM_KEYUP
(Shift Released)

Keystroke Messages Format

WM_KEYDOWN **WM_KEYUP**
WM_SYSKEYDOWN **WM_SYSKEYUP**

wParam: Virtual-key code

lParam: *Additional message information*

wParam	WINUSER.H identifier	Keyboard key
8	VK_BACK	Backspace
9	VK_TAB	Tab
12	VK_CLEAR	Numeric Keyboard 5 with Num Lock OFF
13	VK_RETURN	Enter *(either one)*
16	VK_SHIFT	Shift *(either one)*
17	VK_CONTROL	Ctrl *(either one)*
33	VK_PRIOR	Page Up

Virtual-key codes for Windows keys

wParam	WINUSER.H identifier	Keyboard key
91	VK_LWIN	Left Windows key
92	VK_RWIN	Right Windows key
93	VK_APPS	Application key

lParam of WM_KEYDOWN

Bits	
0 - 15	Repeat count
16 - 23	Scan code
24	1, if an *Extended Key* 0, otherwise

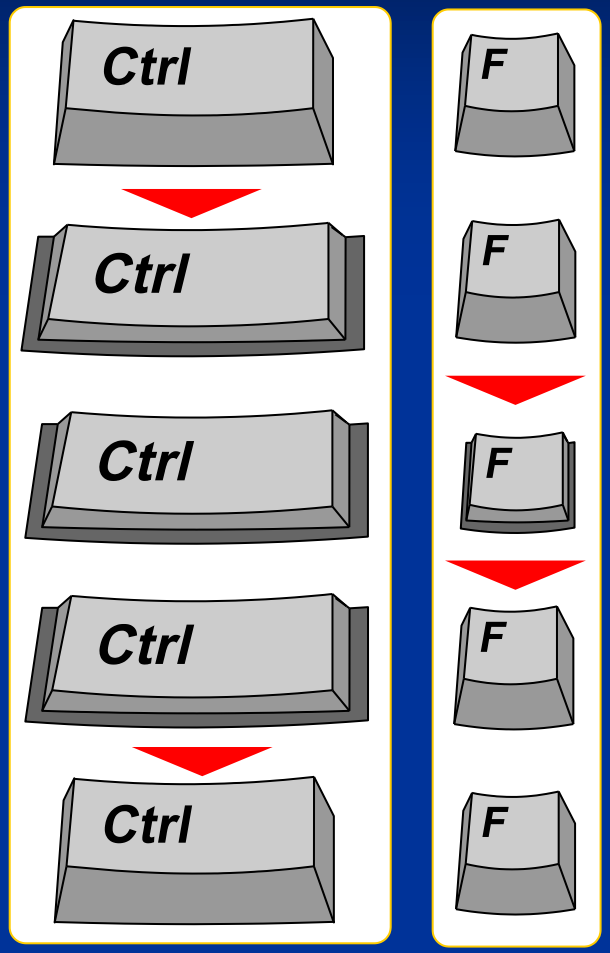
Extended Keys *were not part of the standard IBM keyboard.*
These appeared on IBM 101- and 102-keys Enhanced Keyboards later.

Holding down a key on the keyboard

WM_KEYDOWN
WM_KEYDOWN
WM_KEYDOWN
... ...
... ...
WM_KEYUP

OR

WM_SYSKEYDOWN
WM_SYSKEYDOWN
WM_SYSKEYDOWN
... ...
... ...
WM_SYSKEYUP



Initial State

WM_KEYDOWN
(Ctrl Pressed)

WM_KEYDOWN
(F Pressed)

WM_KEYUP
(F Released)

WM_KEYUP
(Ctrl Released)

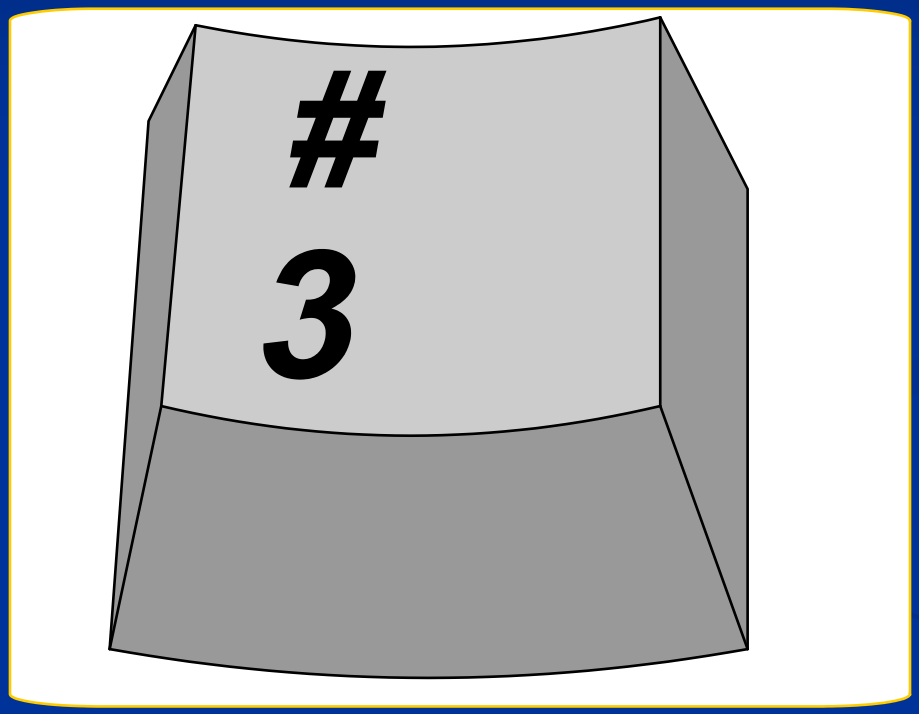
GetKeyState()

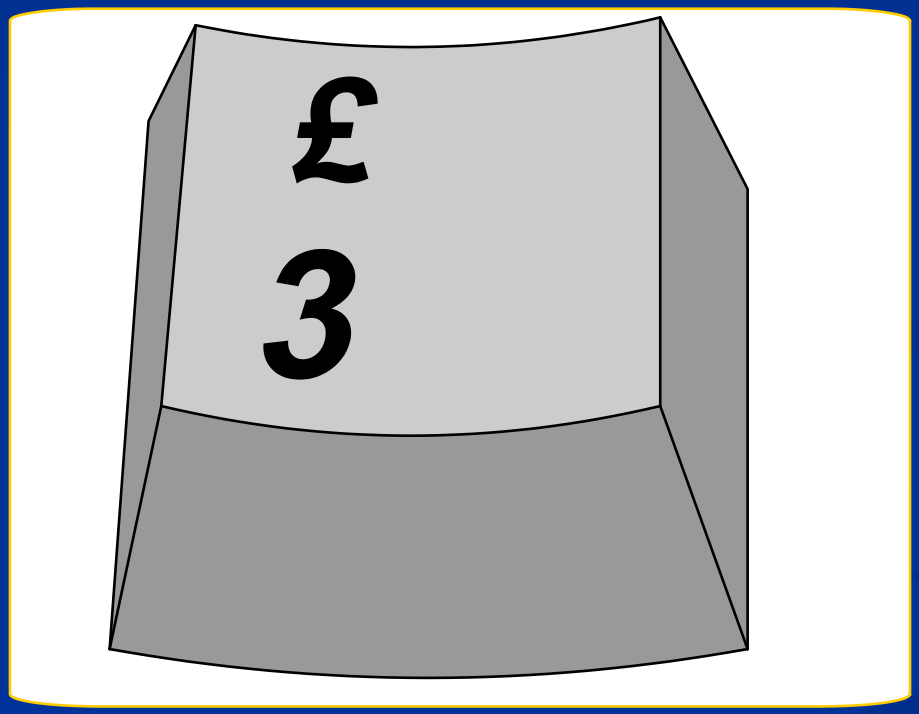
```
SHORT GetKeyState(
    int nVirtKey
);
```

virtual-key code

GetAsyncKeyState()

```
SHORT GetAsyncKeyState(
    int nVirtKey        virtual-key code
);
```


#
3


£
3

TranslateMessage()

```
BOOL TranslateMessage(
   CONST MSG *lpMsg    message information
);
```

Message Loop

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

Character Messages

WM_CHAR
WM_SYSCHAR
WM_DEADCHAR

wParam

character code

lParam

additional message information

Enter key pressed

↓

`WM_KEYDOWN` message generated

↓

`TranslateMessage()` called

↓

`WM_CHAR` message generated

`wParam` *contains* *`'\n'`* *OR*

`0x0A` *OR*

`10`

Pressing **W** on the keyboard

Message	wParam
WM_KEYDOWN	Virtual key code for 'W' (0x57)
WM_CHAR	Character code for 'w' (0x77)
WM_KEYUP	Virtual key code for 'W' (0x57)

Pressing **Shift+W** on the keyboard

Message	wParam
WM_KEYDOWN	Virtual key code for VK_SHIFT(0x10)
WM_KEYDOWN	Virtual key code for 'W' (0x57)
WM_CHAR	Character code for 'W' (0x57)
WM_KEYUP	Virtual key code for 'W' (0x57)
WM_KEYUP	Virtual key code for VK_SHIFT(0x10)

Holding down **W** on the keyboard

Message	wParam
WM_KEYDOWN	Virtual key code for 'W' (0x57)
WM_CHAR	Character code for 'w' (0x77)
WM_KEYDOWN	Virtual key code for 'W' (0x57)
WM_CHAR	Character code for 'w' (0x77)
⋮	⋮
WM_KEYUP	Virtual key code for 'W' (0x57)

Caret

mouse *cursor*

keyboard *caret*

Caret Functions

`CreateCaret()`	*creates a new caret and associates it with the window*
`DestroyCaret()`	*destroys the caret*
`ShowCaret()`	*shows the caret*
`HideCaret()`	*hides the caret*
`SetCaretPos()`	*sets the caret position*
`GetCaretPos()`	*gets the caret position*

Mouse

Mouse Handling in DOS

Driver installation

(e.g. mouse.sys, mouse.com etc.)

↓

INT 33h services used to manipulate the mouse

```
int GetSystemMetrics(int nIndex);
```

Values of nIndex

SM_MOUSEPRESENT

TRUE or nonzero if a mouse is installed;
FALSE or zero otherwise.

SM_CMOUSEBUTTONS

Number of buttons on mouse, *or*
zero if no mouse is installed.

Client-Area Mouse Messages

Left	WM_LBUTTONDOWN	*pressed*
	WM_LBUTTONUP	*released*
	WM_LBUTTONDBCLK	*double-click*
Middle	WM_MBUTTONDOWN	*pressed*
	WM_MBUTTONUP	*released*
	WM_MBUTTONDBCLK	*double-click*
Right	WM_RBUTTONDOWN	*pressed*
	WM_RBUTTONUP	*released*
	WM_RBUTTONDBCLK	*double-click*

Mouse double-clicks

CS_DBLCLKS:

Double-click messages are sent to the window if this class-style *is specified*

Mouse Double-Click Messages
CS_DBLCLKS class style

Not specified	Specified
WM_LBUTTONDOWN	WM_LBUTTONDOWN
WM_LBUTTONUP	WM_LBUTTONUP
WM_LBUTTONDOWN	WM_LBUTTONDBLCLK
WM_LBUTTONUP	WM_LBUTTONUP

LONG GetMessageTime(VOID);

Retrieves the message time for the last message retrieved by the **GetMessage** *function.*

Client area messages

`WM_LBUTTONDOWN, WM_RBUTTONDOWN,`
`WM_LBUTTONUP, WM_RBUTTONUP etc.`

wParam: status of a few virtual keys

`MK_LBUTTON`	*The left mouse button is down.*
`MK_CONTROL`	*The CTRL key is down.*
`MK_SHIFT`	*The SHIFT key is down. etc.*

lParam:

Low word: x-coordinate of cursor

High word: y-coordinate of cursor

Nonclient-Area Mouse Messages

Left	WM_NCLBUTTONDOWN	*pressed*
	WM_NCLBUTTONUP	*released*
	WM_NCLBUTTONDBCLK	*double-click*
Middle	WM_NCMBUTTONDOWN	*pressed*
	WM_NCMBUTTONUP	*released*
	WM_NCMBUTTONDBCLK	*double-click*
Right	WM_NCRBUTTONDOWN	*pressed*
	WM_NCRBUTTONUP	*released*
	WM_NCRBUTTONDBCLK	*double-click*

Nonclient-Area Messages

`WM_NCLBUTTONDOWN, WM_NCLBUTTONUP etc.`

wParam: hit-test value

`HTCAPTION` *cursor on the title bar*

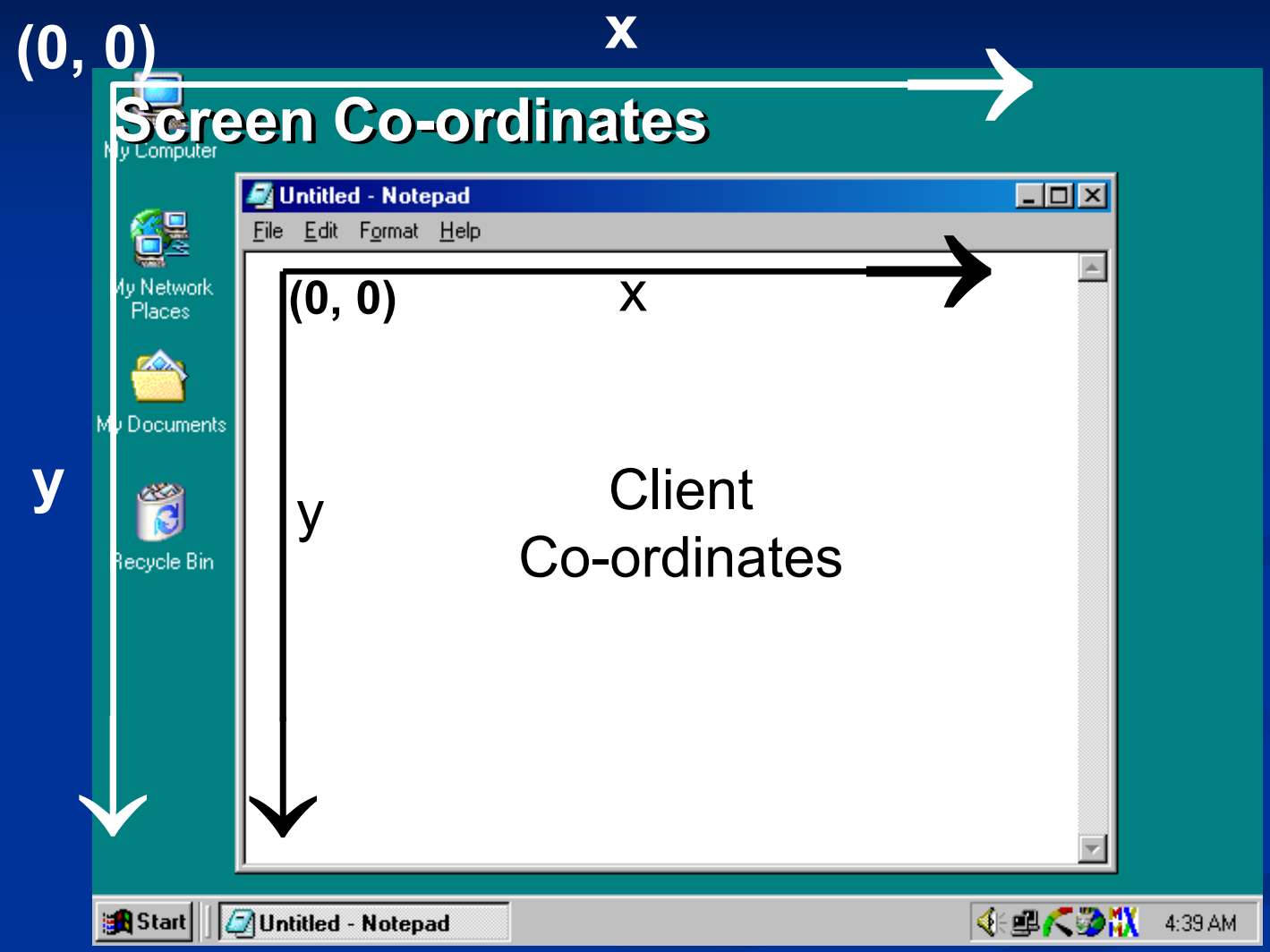
`HTCLOSE` *cursor in a **Close** button.*

etc.

lParam:

Low word: x-coordinate of cursor

High word: y-coordinate of cursor


(0, 0)
x
Screen Co-ordinates
y
My Computer
My Network Places
My Documents
Recycle Bin
Untitled - Notepad
File Edit Format Help
(0, 0)
x
y
Client
Co-ordinates
Start
Untitled - Notepad
4:39 AM

Converstion between Screen and Client-area coordinates

```
BOOL ScreenToClient(
  HWND hWnd,          handle to window
  LPPOINT lpPoint     screen coordinates
);

BOOL ClientToScreen(
  HWND hWnd,          handle to window
  LPPOINT lpPoint     client coordinates
);
```

WM_NCHITTEST

wPraram

Not used

lParam

Low word: x-coordinate of cursor

High word: y-coordinate of cursor

WM_NCHITTEST

↓

DefWindowProc()
examines where does the cursor position lie

↓

DefWindowProc() returns a hit-test value
(*HTCAPTION, HTCLOSE, etc.*) to the system

↓

System generates further messages

WM_LBUTTONDOWN **WM_NCLBUTTONDOWN**
WM_LBUTTONUP **WM_NCLBUTTONUP**

Capturing A Mouse

```
HWND SetCapture(
    HWND hWnd       handle to window
);
```

Limitation of mouse capture in Win32

- If the mouse is captured, and
- no mouse button is down ***outside*** your window

Mouse messages **will be** *directed to other windows underneath the cursor.*

Timers

Original IBM PC microprocessor clock

4.772720 MHz = 4772720 Hz

4.772720 MHz divided by 2^{18}

4772720 / 262144 ≈ 18.2 Hz

or **18.2 times per second**

Timer BIOS interrupt is called

18.2 times per second

or **once every 54.925 msec**

Setting a timer

```
UINT_PTR SetTimer(
  HWND hWnd,              handle to window
  UINT_PTR nIDEvent,      timer identifier
  UINT uElapse,           timer interval or
                          time-out value
  TIMERPROC lpTimerFunc
                          timer procedure
);
```

Setting timer interval to

1000 milliseconds (1 seconds)

causes

WM_TIMER messages to be posted

every second

WM_TIMER

`wParam` *timer identifier*

Destroying a timer

```
BOOL KillTimer(
   HWND hWnd,              handle to window
   UINT_PTR uIDEvent       timer identifier
);
```

Timer set to

500 milliseconds

WM_TIMER

↓

passed to **DefwindowProc()**

↓

TIMERPROC called

Prototype of TimerProc:

```
VOID CALLBACK TimerProc(
  HWND hwnd,              handle to window
  UINT uMsg,          WM_TIMER message
  UINT_PTR idEvent,         timer identifier
  DWORD dwTime          current system time
);
```

WM_TIMER

Asynchronous Message

System timer in hardware operates at

18.2 ticks per second, or
Once per 54.925 milliseconds

Hence,
Practically we can ***not*** set a timer of higer resolution. i.e.

One that works at less than 54.925 milliseconds intervals

Setting a timer without a window handle

```
UINT_PTR SetTimer(
   HWND hWnd,
```

if set to NULL, timer is associated with the application queue

```
   UINT_PTR nIDEvent,
```

ignored if the hWnd *parameter is NULL*

```
   ..
   ..
);
```

Returns a timer identifier

Processing WM_TIMER message of a timer not associated with any window

```
while(GetMessage(&msg,NULL,0,0) > 0)
{
   if(msg.wParam == WM_TIMER)
   {
      … … …
   }
   else
   {
     DispatchMessage(&msg);
   }
}
```