More Win32 API

- More Mouse Stuff
- GetSystemMetrics()
- WM_PAINT Messages
- GetClientRect()
- TextOut()
- wsprintf()
- Using Fonts
- GetTextMetrics()
- Example Program

Mouse Messages

- **Client Area Mouse Messages—**
  - Mouse msgs generated when mouse moves over the window's client area
  - or when pressed/released within window's client area
  - 21 messages in all
  - WM_MOUSEMOVE: Sent to window under cursor when mouse moved
    - lParam = mouse cursor X,Y position
    - wParam: Mouse notification code
      - MK_LBUTTON, MK_RBUTTON, MK_SHIFT, MK_CONTROL

- **WM_*BUTTON# :**
- *** = L, M, R**
- **# = DOWN, UP, DBLCLK**
- **DBLCLK message sent only if wndclass.style contains CS_DBLCLKS**

Input Focus

- **Window whose caption line is highlighted has "input focus"**
- **Only this window will receive keyboard input**
- **Run 2 instances of Winapp2**
  - **Note: keyboard accelerators only work with instance that has input focus**
- **Input focus not significant for mouse input**
- **Good since mouse is used to activate a window**

- **When a window gains (loses) input focus:**
  - **WM_SETFOCUS (WM_KILLFOCUS) message sent to window**
- **Common responses:**
  - **highlight an edit area, change a caption, etc.**
- ***SetFocus(hWnd)***
  - **Give a window (or a control) the input focus**
- **Response to receiving input focus depends on window style**

Nonclient Area Mouse Messages

- **Mouse actions in other parts of window➔**
  - **WM_NC* messages sent**
    - **(* = MOUSEMOVE, etc.)**
  - **wParam: HT* hit test code➔**
    - **non-client area where action occurred**
  - **lParam: mouse cursor position**
  - **Usually not processed by applications**
  - **Could use to generate other messages**
    - **e.g., WM_NCLBUTTONDOWN + coordinates ➔ WM_COMMAND**

Capturing the Mouse

- **To limit mouse to interacting with just one program**
- **e.g., screen capture program**
- **Application that does this has "captured" the mouse**
- **Only it will receive mouse messages.**
- **Use: *SetCapture(hWnd);***
- **Release with: *ReleaseCapture(void);***

Getting information on user interface items

- **Use: GetSystemMetrics(nIndex)**
  - **nIndex specifes which item**
  - **See online help**

WM_PAINT Messages

- **Sent any time client area is invalidated (exposed)**
- **Should redraw everything in exposed area**
- **Use BeginPaint(hWnd,&ps) to get a DC**
- **ps is a pointer to a PAINTSTRUCT**
  - **contains info about area to be redrawn**
- **Use EndPaint(hWnd,&ps) to release the DC**

PAINTSTRUCT

```
typedef struct tagPAINTSTRUCT
 {
  HDC     hdc;        // device context handle
  BOOL    fErase;   // should bkgnd be redrawn? T/F
  RECT    rcPaint;  // rectangular area to update
  BOOL    fRestore;  // reserved for use by Windows
  BOOL    fIncUpdate;          // reserved
  BYTE    rgbReserved[16];  // reserved
 } PAINTSTRUCT;
```

WM_PAINT Message

- If you want to keep stuff already drawn in your window after it's exposed:
  - You need to keep track of everything drawn
  - Then redraw in response to WM_PAINT

Forcing a WM_PAINT

- **InvalidateRect(hWnd,&rect,bErase);**
  - **parameters:**
    - **window to be invalidated**
    - **rectangular area (NULL ==> entire client area)**
    - **background erased (TRUE/FALSE)**
- **Causes a WM_PAINT message to be placed on the queue**
- **This could be done in response to mouse & other messages**

Determining Client Area

- **GetClientRect(hWnd,&rect)**
  - **rect pointer will contain (0,0,width,height)**
  - **You may need to know this**
    - **for animations**

Displaying Text

- **TextOut(hDC,x,y,lpTxt,cbTxt);**
  - **x,y: position on client area of window**
  - **lpTxt: string to be displayed**
  - **cbTxt: length of the string**
  - **current DC text color & bkgnd color used**
  - **current DC font is used**
  - **can use lstrlen() to get cbTxt**
    - **for example:**

```
char  cBuf [] = “Hello, World”;
TextOut (hDC, 0, 0, cBuf, lstrlen(cBuf)) ;
```

Displaying Numeric Values

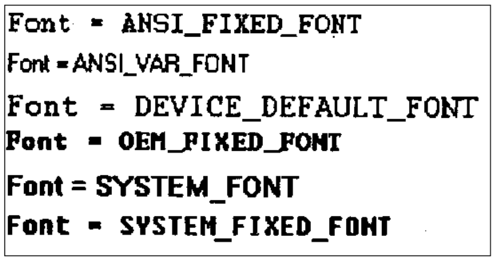
- Must format values into a string
- Can use wsprintf()
- See online help
- Example:

```
char cBuf[50];
int num = 19;
wsprintf(cBuf, “The number is: %d ", num);
TextOut(hDC, 10, 10, cBuf, lstrlen(cBuf) );
```

Using and Changing Fonts

- FONT: Typeface, style, size of characters in a character set
- Three basic kinds of fonts--
  - Stock fonts--built into Windows, always available
  - Logical or GDI fonts--defined in separate .fon (stroke or raster) or .fot/.ttf (TrueType) font resource files in \windows\system and stored on disk
  - Device fonts--native to the output device (e.g., built-in printer fonts).

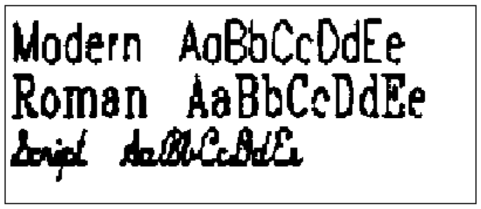
Some Stock Fonts



Font = ANSI_FIXED_FONT
Font = ANSI_VAR_FONT
Font = DEVICE_DEFAULT_FONT
Font = OEM_FIXED_FONT
Font = SYSTEM_FONT
Font = SYSTEM_FIXED_FONT

Windows Stock Fonts

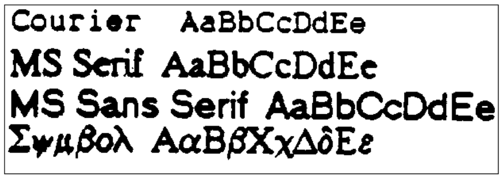
Some Stroke Fonts



Modern AaBbCcDdEe
Roman AaBbCcDdEe
Script AaBbCcDdEe

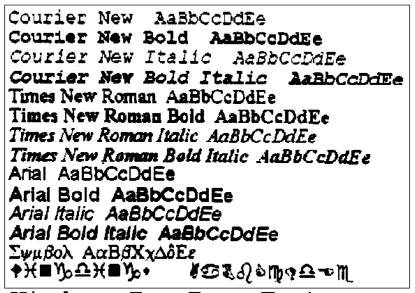
Windows Stroke Fonts

Some Raster Fonts


Courier AaBbCcDdEe
MS Serif AaBbCcDdEe
MS Sans Serif AaBbCcDdEe
ΣψμβολΑαΒβΧχΔδΕε

Windows Raster Fonts

Some True Type Fonts


Courier New AaBbCcDdEe
Courier New Bold AaBbCcDdEe
Courier New Italic AaBbCcDdEe
Courier New Bold Italic AaBbCcDdEe
Times New Roman AaBbCcDdEe
Times New Roman Bold AaBbCcDdEe
Times New Roman Italic AaBbCcDdEe
Times New Roman Bold Italic AaBbCcDdEe
Arial AaBbCcDdEe
Arial Bold AaBbCcDdEe
Arial Italic AaBbCcDdEe
Arial Bold Italic AaBbCcDdEe
ΣψμβολΑαΒβΧχΔδΕε

Windows TrueType Fonts

Using Stock Fonts

- **GetStockObject()**
  - **returns handle to the desired font**
  - **can be selected into a DC**

```
HDC      hDC;
HFONT    hFont;
hDC = GetDC(hWnd);
hFont = GetStockObject (ANSI_VAR_FONT);
SelectObject (hDC,hFont);
```

Using Logical Fonts

- Obtain a handle to the font data resource and select it into the DC
  - Just like a stock font, except it's loaded from separate file (.fon, .fot/.ttf).
  - Use CreateFont() instead of GetStockObject() to load and get a font handle.
  - CreateFont() makes new fonts by interpolating data in a font file
    - ==> New sizes, bold/underlined, rotated/distorted
    - Called logical since they come from program logic not just from a file

CreateFont()

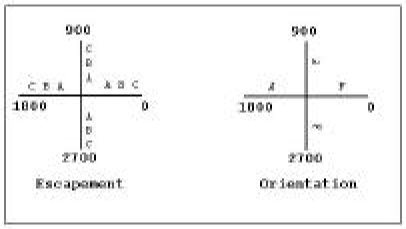
hFont = CreateFont (Ht, Width, Escapement, Orientation, Weight, Italic, Underline, StrikeOut, CharSet, OutputPrecision, ClipPrecision, Quality, PitchAndFamily, Facename);

- 14 parameters, many are often set to 0 ==> defaults
- See the on-line help on CreateFont():

Example call to CreateFont()--

```
hFont = CreateFont (36, 0, 3000, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,"Roman")
```

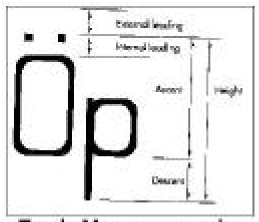
Escapement & Orientation



Character Escapement & Orientation

Determining Character Sizes

- With CreateFont(), may not get what you want
- Use GetTextMetrics(hDC,lpTextmetric)
  - See online help



Font Measurements

FONT1 Example Program

- User types ==> blue text in client area
- Can change font from menu
- Backspace editing feature
- cBuf[] builds text string as it's input
- WM_CHAR message received ==> character tested & appended to cBuf if:
  - Character is alphanumeric
    - IsCharAlphaNumeric()
  - Or character is punctuation
    - if helper function IsAnsiPunc() returns TRUE
  - And cBuf[] hasn't been filled

- To display, force a WM_PAINT message
  - InvalidateRect()
- Response: draw cBuf[] string
- Also string will be redrawn automatically after exposure (resizing, uncovering)

- WM_CHAR for printable characters
- WM_KEYDDOWN for Backspace
- IsAnsiPunc()--a helper function that tests ranges of ANSI codes for punctuation characters
- WM_CREATE when program starts -->
  - Use CreateFont() to create new Roman font & save handle in hFont
- WM_COMMAND to choose font from a popup menu (set nFontChoice variable)

WM_PAINT message:

1. Get a DC with BeginPaint()
2. Change color to blue
3. Check value of nFontChoice
4. SelectObject() to select chosen font into DC
5. TextOut() to output the cBuf[] string
6. Release DC with EndPaint()

Note use of static variables to “remember” variable values from one WndProc() callback to another