An Introduction to Windows Programming

Two approaches:
- Win32 API
  - Most basic
- MFC
  - Encapsulates API functions into classes
  - For most apps, easiest to use

Windows Programming

- Event-driven, graphics oriented
- Example: User clicks mouse over a program's window area--
  - Windows decodes HW signals from mouse
  - figures out which window user has selected
  - sends a message to that window's pgm:
    - "User has clicked over (X,Y)"
    - "Do something and return control to me"
  - Program reads message data, does what's needed, returns control to Windows

Overview of Win32 API

Program Structure--2 main tasks:
- Initial activities
- Process messages from Windows (the message loop)

Pseudocode

- Initialize variables, memory space
- Create & show program's Window
- Loop
  - Fetch any msg sent from Windows to this pgm
  - If message is QUIT
    - terminate program, return control to Windows
  - If message is something else
    - take actions based on msg & parameters
    - return control to Windows
- End Loop

Essential Parts of a Windows Program

1. The source program (.c/.cpp file):
   - A. WinMain() function
     - 0. declarations, initialization, etc.
     - 1. register window class
     - 2. create a window based on a registered class
     - 3. show window, make it update its client area
     - 4. the message loop
       - get messages from Windows and forward to callback message-processing function
B. WndProc(): the message-processing function
  • a big switch/case statement
  • handles messages of interest

Under Win32 API, programmer must write WinMain() and the WndProc()
  Under MFC, Wizards do most of the work
  WinMain() and WndProc() are buried in the framework
  Write “message mapped handler functions” instead

II. The resource script (.rc file):
  • contains resource (Windows static) data
  • determine “look and feel” of the application
  • separate from code and dynamic data
  • compiled by a separate “Resource Compiler”
  • Examples:
    • Keyboard Accelerators, Bitmaps, Cursors, Dialog Box, Fonts, Icons, Menus, String Tables
  • Separation of resources and program code
  • separates tasks of programmer & designer
  • can change user interface without touching code

The WinMain() Function
  • WinMain() starts first
  • integer exit code returned to Windows
  • 4 parameters passed in from Windows
    • hInstance: a handle, identifies current pgm instance
    • lpszCmdLine: string containing command line arguments
    • nCmdShow: how window is to appear when shown

RegisterClass(&wndclass);
  • Give general properties of the “window class” from which the application’s window is derived
  • Specify them in a WNDCLASS structure
    • See online help for details

CreateWindow() arguments:
  • Create the window in memory giving it specific properties:
    • window class name
    • window caption
    • window style (Boolean OR of style masks)
    • initial x , y position in pixels
    • initial width, height
    • parent window handle (if main window, NULL)
    • window menu handle (NULL if class menu used)
    • program instance handle (passed in from Windows)
    • creation parameters (for extra data, usually NULL)
  • Returns a ‘window handle’

ShowWindow (hWnd,nCmdShow);
  • makes window visible on screen
  • hWnd: which window to make visible
  • nCmdShow: how (normal, minimized, etc.)
    • set by Windows environment when program is started;
    • value is passed in from Windows;
    • “normal” can be overridden
UpdateWindow (hWnd);
- Causes client area to be updated
- Painted with background brush

The Message Loop
- User interaction---> a msg sent to a window
- Lots of other actions---> messages
- A message structure:
  - HWND hwnd; // target window handle
  - UINT message; // msg ID value--WM_***
  - WPARAM wParam; // data passed in msg
  - LPARAM lParam; // more data in msg
  - DWORD time; // time msg was sent
  - POINT pt; // mouse cursor position (x,y)

GetMessage()
- Program must keep checking for messages
- Use message loop w/ GetMessage()
- BOOL GetMessage(
  LPMSG lpMsg, // ptr to msg struct
  HWND hWnd, // target window
  UINT wMsg1, // 1st msg in range
  UINT wMsg2, // last msg in range
)

The Message Loop
- GetMessage() reads next msg from app's msg queue
- Fills MSG struct pointed to by first param
- Place in a loop:
  while (GetMessage(&msg, NULL, 0, 0))
  {
    ... }
  return((int)msg.wParam);
- Returns non-0, except for WM_QUIT msg
  -- Terminates msg loop & returns control to Windows

The Main Message Loop

Message Processing
- What goes inside the message loop:
  TranslateMessage (&msg)--
  "Cooks" keyboard input
  Converts raw key codes to ANSI codes
  DispatchMessage (&msg)--
  Sends message on to Windows, which forwards it to pgm's "Window Procedure":
  WndProc()--
  2nd member of WNDCLASS structure
  Programmer must write this function
The Window Procedure

- “Callback” function (called by Windows)
  - Programmer writes this function
- Should contain a switch/case statement:
  - Looks at message ID of current message
  - Acts appropriately on “interesting” messages
  - Forwards other messages to default Window procedure--DefWindowProc()

WndProc()

LRESULT CALLBACK WndProc (>
    HWND hWnd, UINT wMessage,>
    WPARAM wParam, LPARAM lParam)

Parameters--
Same as first four fields of MSG structure:
- window associated with message
- message ID (what message is)
- msg data (wParam & lParam)

Some Important Messages

- WM_DESTROY -- User performed action to kill window
  -- window is being destroyed
- WM_COMMAND -- User clicked on a menu item (menu item ID provided)
- WM_*BUTTONDOWN -- left/right mouse button pressed
  (* = L or R; x,y coordinates provided)
- WM_MOUSEMOVE -- mouse moved
- WM_CHAR -- User pressed valid ANSI code character
  -- keyboard key combination (ANSI code provided)
- WM_PAINT -- Part of window was exposed & should be
  -- redrawn
- WM_KEYDOWN -- keyboard key pressed (virtual key code provided)

Simple Example Win32 API Program

- Does very little except close/size itself
- On CS-560 Sample Programs web page
  http://www.cs.binghamton.edu/~reckert/460/460pgms.htm
- See: “Simple Windows Program”

Text and Graphics Output

- Displaying something in a window
- Text & graphics done one pixel at a time
- Any size/shape/position possible
- Design goal: Device Independence
Device Independent Graphics Interface

- Windows programs don’t access hardware devices directly
- Make calls to generic drawing functions within the Windows ‘Graphics Device Interface’ (GDI) -- a DLL
- The GDI translates these into HW commands

Program → GDI → Hardware

Device Context

- Windows programs don’t draw directly on the hardware
- Draw on “Device Context” (DC)
  - Abstracts the device it represents
  - Like a painter’s canvas
  - Specifies drawing attributes
    - e.g., text color
  - Contains drawing objects
    - e.g., pens, brushes, bitmaps, fonts

The DC and the GDI

Windows Drawing Using the GDI and the DC

Application → GDI Functions → GDI Interface (GDI) → Drawing Commands → Hardware

Some GDI Attributes

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>DEFAULT</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background color</td>
<td>white</td>
<td>SetBkColor()</td>
</tr>
<tr>
<td>Background mode</td>
<td>OPAQUE</td>
<td>SetBkMode()</td>
</tr>
<tr>
<td>Clipping Region</td>
<td>whole surface</td>
<td>SelectClipRgn()</td>
</tr>
<tr>
<td>Current Position</td>
<td>(0,0)</td>
<td>MoveTo()</td>
</tr>
<tr>
<td>Drawing Mode</td>
<td>R2COPYPEN</td>
<td>SetROP2()</td>
</tr>
<tr>
<td>Mapping Mode</td>
<td>MM_TEXT</td>
<td>SetMapMode()</td>
</tr>
<tr>
<td>Text Color</td>
<td>Black</td>
<td>SetTextColor()</td>
</tr>
</tbody>
</table>

Some GDI Drawing Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Default</th>
<th>What it is</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitmap</td>
<td>none</td>
<td>image object</td>
</tr>
<tr>
<td>Brush</td>
<td>WHITE_BRUSH</td>
<td>area fill object</td>
</tr>
<tr>
<td>Font</td>
<td>SYSTEM_FONT</td>
<td>text font object</td>
</tr>
<tr>
<td>Pen</td>
<td>BLACK_PEN</td>
<td>line-drawing object</td>
</tr>
<tr>
<td>Color Palette</td>
<td>DEFAULT_PALETTE</td>
<td>color combinations</td>
</tr>
</tbody>
</table>

- Can be created with GDI functions
- Must be “selected” into a DC to be used
Colors in Windows

- Uses 4-byte numbers to represent colors
- Simplest method—direct color:
  - typedef DWORD COLORREF;

<table>
<thead>
<tr>
<th>0</th>
<th>Blue (0-255)</th>
<th>Green (0-255)</th>
<th>Red (0-255)</th>
</tr>
</thead>
</table>
- MSB=0:
  - => RGB direct color used (default)
  - Other bytes specify R, G, B intensities

RGB() Macro

- Specify Red, Green, Blue intensities
- RGB() generates a COLORREF value
- Can be used in color-setting fns), e.g.
  - COLORREF cr;
  - cr = RGB (0,0,255);  // blue */

Example usage in a program
SetTextColor(RGB(255,0,0)); //red text
SetBkColor(RGB(0,0,255)); //blue bkgnd

(These are member functions of CDC)

Some GDI Drawing Primitives

- Arc(x1,y1,x2,y2,xStart,yStart,xEnd,yEnd);
- Ellipse (x1,y1,x2,y2);
- MoveTo (x1,y1);
- LineTo (x1,y1);
- Polygon (points_array,nCount);
- Polyline (points_array,nCount);
- Rectangle (x1,y1,x2,y2);
- SetPixel (x1,y1,colorref);
- Many more (see on-line help on CDC)

Example Win32 API Program

- Draws a rectangle or ellipse in response to menu item clicked
- On CS-560 Sample Programs web page
- See: "First Real Windows Program"
  - .cpp, .rc, .h files

Creating a Win32 API Application with Visual Studio

- Startup
  - click 'Start' on Task Bar – 'All Programs'
  - 'Microsoft Visual Studio .NET' – 'Microsoft Visual Studio .NET'
- Creating a new solution
  - 'File' – 'New' – 'Project' from menu bar
  - In 'New Project' box, select 'Visual C++ Project' from 'Project Types': & click on 'Win32 Project' in 'Templates'
  - Set the 'Location' to a convenient directory & name the project (e.g. win32app1)
- Click 'Application Settings' in resulting 'Application Wizard' Box
  - Choose 'Windows Application' from 'Application Type' radio buttons
  - Select 'An Empty Project'
  - Click 'Finish'
- Click 'OK' in 'New Project Information' window
**Inserting source files into project:**
- Open a new C++ file & type or copy/paste the code into the program:
  - `File | New | File` from menu
  - Choose ‘Visual C++’ from ‘Categories’, C++ file (.cpp) from ‘Templates’, & click ‘Open’
  - Type or paste source code in the resulting Edit window
  - Save the file as a C++ source file, giving it an appropriate name (e.g., Win32App1)
- Add the source file to the project:
  - Choose ‘Project’ / ‘Add Existing Item’ from the menu
  - Select the .cpp file & click ‘Open’
  - Confirm that it was added to the project by expanding ‘Source Files’ in the Solution Explorer Window
  - If Solution Explorer is not visible, select ‘View – Solution Explorer’ from the menu

**Alternative Way of Adding a Source File to a Project:**
- Copy the file into the project’s directory
- Choose ‘Project’ / ‘Add Existing Item’ from the menu
- Select the .cpp file & click ‘Open’

**Building the Solution:**
- ‘Build’ – ‘Build Solution’ from menu
  - Shortcut key: F7
- Project will be compiled/linked
- Messages/errors will appear in Output Window

**Running the Program:**
- ‘Debug’ – ‘Start’ from menu
  - Shortcut key: F5
- Or ‘Debug’ – ‘Start Without Debugging’ from menu
  - Shortcut key: Ctrl-F5
  - or click exclamation point

**Cleanup:**
- Copy solution, project, source, header, resource files to disk
- Copy .exe file from project’s Debug directory
- Best: Delete all temporary files & copy entire solution (project directory) to floppy
- Delete project directory from hard drive

**Exiting Developer Studio:**
- ‘File | Exit’ from menu