SFU CMPT-212 2008-1 Topic: More MFC Programming

Ján Maňuch

E-mail: jmanuch@sfu.ca

Sunday 30th March, 2008

Processing mouse messages

```
1 BEGIN_MESSAGE_MAP( CMouseWin, CFrameWnd )
2   ON_WM_LBUTTONDOWN() // left mouse button handler
3   ON_WM_RBUTTONDOWN() // right mouse button handler
4 END_MESSAGE_MAP()
```

```
1 // mouse button handlers
2 afx_msg void OnLButtonDown(UINT flags,CPoint p);
3 afx_msg void OnRButtonDown(UINT flags,CPoint p);
```

- `flags` — which button was pressed (e.g., `MK_LBUTTON`)
- `p` — coordinates of place where it was clicked

Example: mouse.cpp (modified [Deitel])

- `CClientDC dc(this)` — display context of the current window
-
```
dc.TextOut(int x,int y,LPCTSTR str, int len);
dc.TextOut(int x,int y,const CString& str);
```
  — output text at specified location
- `dc.SetTextColor(COLORREF col);` — change color `0x0FF0000` (red)
- **Problem:** if the window is covered, the custom graphics disappear
  Solution:
  - windows sends `WM_PAINT` message to our application when any part of the client area needs to be repainted
  - for custom graphics (different from controls) we should implement message handler `OnPaint()`
  - **Note:** `OnPaint()` uses `CPaintDC` instead of `CClientDC`

  Example: mouselines.cpp

Processing keyboard messages

```
1 BEGIN_MESSAGE_MAP( CMainWindow, CFrameWnd )
2   ON_WM_CHAR()
3   ON_WM_KEYDOWN()
4 END_MESSAGE_MAP()
```

```
1 //  Keyboard handler functions
2 afx_msg void OnKeyDown(UINT c,UINT cnt,UINT flags);
3 afx_msg void OnChar(UINT c,UINT cnt,UINT flags);
```

- `c` — character
- `cnt` — how many times pressed
- `flags` — which flags where present (contains the scan code, key-transition code, previous key state, and context code)

Example: key.cpp, keymenu.h, keymenu.rc

Icons

- *changing the icon in the title bar of a frame window:*

```
1  Create(
2    AfxRegisterWndClass(
3      // construct Windows class
4      NULL, // Windows class style
5      NULL, // handle to cursor
6      NULL, // handle to brush
7      // (used to repaint the client area
8      //  of the window, NULL = do not repaint)
9      AfxGetApp()->LoadIcon(IDI_MAIN_ICON)
10        // handle to icon
11        // IDI_MAIN_ICON specified in resource file
12   ),
13   // etc..
```

Example: see the `dialog` example from LN19

- *changing the icon in the title bar of a dialog:*

```
1  // in the constructor of dialog:
2  h_icon = AfxGetApp()->LoadIcon(IDI_CMPT);
3    // load and store a handle of the icon
4    // IDI_MAIN_ICON specified in the resource file
5
6  // in OnInitDialog() of dialog
7  SetIcon(h_icon,FALSE);
8    // set the title bar icon
```

Example: icondialog.rc, icondialog.h, icondialog.cpp **Notes:**

  - virtual function `BOOL OnInitDialog()` is called after all controls are constructed:
    - we can add a text into control, highlight the (part of) text, set focus or load icon
    - if we return `FALSE`, MFC will not set the focused item

- *displaying icons inside the windows:*
  - in the resource file (works for a dialog):

```
1 IDI_CMPT ICON "cmpt.ico"
2 // connect IDI_CMPT with icon in "cmpt.ico"
3 ICON IDI_CMPT, IDC_STATIC, 170, 5, 0, 0
4 // place icon with id IDI_CMPT
5 // at pos. (170,5) of the window
```

Example: icon.cpp, icon.rc, icon.h

Note: you can use the system icons by using their ids:
`IDI_ERROR`, `IDI_QUESTION`, `IDI_WARNING`,
`IDI_INFORMATION`, `IDI_APPLICATION`, `IDI_WINLOGO`

– in the .cpp file:

```
1    HICON h_icon = LoadIcon(NULL,IDI_QUESTION);
2    // handle to icon
3    DrawIcon(dc, 10, 20, h_icon);
4    // draw icon using display context dc
5    // at position (10,20)
```

Note:

we need to redraw the icon each time the window is redrawn (should be included in the `OnPaint()` message handler)

Example: icon2.cpp

Timers

- setting up a timer:

```
1 SetTimer(ID_MOVE, 100, NULL);
```

- sends message `WM_TIMER` (with parameter `ID_MOVE`) to the invoking window every 100ms
- `ID_MOVE` is the id of the timer which can be used to stop timer:

```
1 KillTimer(ID_MOVE);
```

- you have to map `WM_TIMER` to message handler:

```
1 ON_WM_TIMER()
```

  and provide message handler

```
1 afx_msg void OnTimer(UINT nIDEvent );
```

  where `nIDEvent` is the id of the timer (`ID_MOVE`)

- the third parameter could provide a callback function which would receive the above messages

Example: timer.cpp

Graphics

- **shapes:**

  Example: shapes.cpp [Deitel]

  - `CPen` — specifies the style used for painting lines/curves, can be solid, dashed, doted, etc.
  - to create a pen, call `CPen` member function `CreatePen(int style,int width,COLORREF color)`
  - `CBrush` — specifies the style used for painting shapes, can be solid, hatched or patterned
  - method `CreateSolidBrush(COLORREF)` of `CBrush` sets the color and type of brush

```
1 CPen mypen;
2 mypen.CreatePen(PS_DASHDOT,5, RGB(100,255,0));
3 CBrush mybrush;
4 mybrush.CreateSolidBrush(RGB(0,0,255));
```

- to get the device context to use your own pen or brush, you have to use `SelectObject()` method:

```
1 CPaintDC dc(this); // device context of this window
2 CPen *oldpen=dc.SelectObject(&mypen);
3 CBrush *oldbrush=dc.SelectObject(&mybrush);
```

- before returning from the method, we should change back to original pen and brush, as our pen and brush are defined as local variables

```
1 dc.SelectObject(oldpen);
2 dc.SelectObject(oldbrush);
```

- shapes:

```
1 dc.Rectangle(left,top,right,bottom);
2 dc.Ellipse(left,top,right, bottom);
```

  it will use current pen to draw the boundary and current brush to fill interior of the shape

- **lines**:

  Example: lines.cpp [Deitel]

  – for drawing single lines:

```
1  dc.MoveTo( x, y );          // set starting point
2  dc.LineTo( 3 * x, y );      // draw top line
```

  – for drawing a collection of lines:

```
1     dc.Polygon( alpPoints, POLY_POINTS );
```

  where `alpPoints` is an array of `CPoint`s and `POLY_POINTS` is the number of points in the array

  * the polygon will be automatically closed by connecting the last vertex to the first vertex
  * interior of polygon will be filled using current brush

- **pixels:**

  Example: colors.cpp [Deitel]

  – *creating own* `CDC` (base class of `CClientDC` and `CPaintDC`):

```
1 CClientDC dc(this); // device context of window
2 CDC mydc;
3 mydc.CreateCompatibleDC(&dc);
4 // memory device context compatible with dc
```

  but it needs some block of memory where to write drawing operations, i.e., a `CBitmap` object:

```
1 CBitmap mybitmap;
2 mybitmap.CreateCompatibleBitmap( &dc, xMax, yMax );
3 // make it compatible with dc and set size
4 mydc.SelectObject(&mybitmap);
```

  – to copy content of our own `CDC` to other device context:

```
1 dc.BitBlt(x, y,         // destination
2           256, 256,     // width, height
3           &mydc, 0, 0, // source, x, y
4           SRCCOPY );    // straight copy (overwrite)
```

  it can also merge (“and”, “or” or “xor”), invert etc..

- `SetPixel(x,y,color)` — draw one pixel
- `void Draw3dRect(rectangle,color, color_shadow)` — draws shaded rectangle
- displaying text:

```
1 dc.SetTextColor( RGB( red, green, blue ));
2 dc.SetBkColor( RGB( 255, 255, 255 )); // white
3 dc.TextOut( x - 60, y / 2, text, length );
```

- to force redraw:

```
1 InvalidateRect( NULL, FALSE );
2 // redraw all, no erase
```

the first parameter is the area which should be redrawn and the second whether to erase current content

- in real applications, instead of repainting the custom graphics on each redraw, operations are performed in a memory DC and repaint draws memory DC to actual window
  Example: mouselines2.cpp
  - `GetSystemMetrics(int which)` — obtain from Windows information about sizes of different display elements/components (usually in pixels)

```
1 // get phys. screen limits of the primary monitor
2  int xMax = GetSystemMetrics( SM_CXSCREEN );
3  int yMax = GetSystemMetrics( SM_CYSCREEN );
```

  `SM_CMOUSEBUTTONS` — gets the number of mouse buttons

- **images:**
  Example: image.cpp, image.rc [Deitel]
  - to load an image specified in the resource file to a bitmap:

```
1 mybitmap.LoadBitmap( "COOL_BMP" );
```

Scrolling

- to add scroll bars to the windows, specify `WS_HSCROLL` and `WS_VSCROLL` in the description of the style of the window:

```
1    Create(NULL, "CMPT-212 Scrolling",
2         WS_OVERLAPPEDWINDOW | WS_HSCROLL | WS_VSCROLL );
```

- next, we need to specify some information for each scroll bar using `SetScrollInfo()` method. Here, we would specify which values the scroll bar should represent. For instance,

```
1   SCROLLINFO si;
2   si.nMin=0;        // the minimum value the scroll bar represents
3   si.nMax=maxvsize; // the maximum value the scroll bar represents
4   // this is set to the vertical size of our virtual area
5   si.nPage=clientrect.bottom; // the physical size of
6     // the displayed area -  important to set properly
7   SetScrollInfo(SB_VERT, &si, TRUE);
```

Note: To guarantee a completely correct behavior, one should update this information each time the window is resized (i.e., implement message handler for `WM_SIZE` message).

- Windows API sends messages `WM_VSCROLL` and `WM_HSCROLL` to our application, when the scroll is changed — we need to provide message handlers

  Question: What should a message handler do?

  – Recognize what kind of scroll message was received. For instance, if the user drags the thumb of the scroll bar to a new position, type of the message will be `SB_THUMBPOSITION`.

  – Update a position of thumb in the scroll bar using `SetScrollPos()` or `SetScrollInfo()` method.

  – Physically scroll the visible content of the main window using method `ScrollWindowEx()`. This method takes how many pixels it should scroll the x and y coordinates of the displayed area, i.e, we have to calculate the difference between old and new positions represented by the scroll bar.

Example: scroll.cpp

References

Good tutorial for MFC:

- Beginning Visual C++ 6.0" by Ivor Horton

Online sources:

- `http://devcentral.iticentral.com/articles/MFC/default.php`
- `http://visualcpp.net/`