Windows Programming

Lecture 14

WM_PAINT message

Whenever an application receives the **WM_PAINT** message, whether the entire window needs repainting?

WM_PAINT message is generated by the system only when any part of application window becomes invalid.

Windows always specifies invalid area of any window in terms of a least bounding rectangle, hence, the entire window is not repainted.

.

`WM_PAINT` message

- Only one `WM_PAINT` message for a window is stored in the message queue at one time.
- The Invalid Rectangle bounds all parts of a window that need to be redrawn.
- `WM_PAINT` is a low priority message.
- `WM_PAINT` messages processed after processing all other messages in the message queue.

Conditions under which a **WM_PAINT** message is always sent

- Any hidden part of window becomes visible
- Window is resized (and **CS_VREDRAW, CS_HREDRAW** style bits were set while registering the window class).
- Programme scrolls its window
- **InvalidateRect()** or **InvalidateRgn()** is called by the application

Conditions under which a **WM_PAINT** message may be sent

- A dialog is dismissed
- A drop-down menu disappears
- A tool tip is displayed and then it hides

Conditions under which a **WM_PAINT** message is never sent

- An icon is dragged over the window
- The mouse cursor is moved

InvalidateRect()

```
BOOL InvalidateRect(
HWND hWnd,                  // handle to window
CONST RECT *lpRect,         // rectangle coordinates
BOOL bErase                 // erase state
);
```

PAINTSTRUCT structure

```
typedef struct tagPAINTSTRUCT
{
  HDC  hdc;
  BOOL fErase;
  RECT rcPaint;
  ... ... ...
 } PAINTSTRUCT, *PPAINTSTRUCT;
```

Other GDI text output functions

```
int DrawText
(
 HDC     hDC,              // handle to DC
 LPCTSTR lpString,         // text to draw
  int    nCount,           // text length
 LPRECT  lpRect,           // formatting dimensions
  UINT   uFormat           // text drawing options

);
```

Other GDI text output functions

`TabbedTextOut()`

- Writes a character string at a specified location
- Expands tabs to the values specified in an array of tab stop positions
- Currently selected font and foreground / background colors are used

Other GDI text output functions

```
LONG TabbedTextOut(
   HDC hDC,                          // handle to DC
   int x,                            // x-coord of start
   int y,                            // y-coord of start
   LPCTSTR lpString,                 // character string
  int nCount,                        // number of characters
   int nTabPositions,                // number of tabs in array
   CONST LPINT lpnTabStopPositions,
                                     // array of tab positions
   int nTabOrigin                    // start of tab expansion
);
```

Primitive Shapes

These shapes are geometric forms that are outlined by using the current pen and filled by using the current brush.

Example

- Rectangles
- Circles
- Polygons

Primitive Shapes: `Rectangle()`

- The `Rectangle()` function draws a rectangle. The rectangle is outlined by using the current pen and filled by using the current brush.

- The rectangle is outlined using the currently selected pen and filled using the currently selected brush of the window's device context.

Rectangle()

```
BOOL Rectangle(
  HDC hdc,              // handle to DC
    int nLeftRect,      // x-coord of upper-left corner of rectangle
  int nTopRect,         // y-coord of upper-left corner of rectangle
  int nRightRect,       // x-coord of lower-right corner of rectangle
    int nBottomRect     // y-coord of lower-right corner of rectangle
);
```

Primitive shapes: `Polygon()`

The `Polygon()` function draws a polygon consisting of two or more vertices connected by straight lines. The polygon is outlined by using the current pen and filled by using the current brush and polygon fill mode.

```
BOOL Polygon(
HDC hdc,                        // handle to DC
CONST POINT *lpPoints,          // polygon vertices
int nCount                      // count of polygon vertices
);
```

Stock Objects

Pre-defined GDI objects in Windows

- Pens
- Brushes
- Fonts
- Palettes

Stock Objects: `GetStockObject()`

The `GetStockObject()` function retrieves a handle to one of the stock pens,brushes, fonts, and palettes.

```
HGDIOBJ GetStockObject
(
    int fnObject
);
```

Some stock object types

- **DKGRAY_BRUSH** Dark grey brush.
- **ANSI_FIXED_FONT** Windows fixed-pitch (monospace) system font.
- **ANSI_VAR_FONT** Windows variable-pitch (proportional space) system font.

SelectObject()

The **SelectObject()** function selects an object into the specified device context (DC). The new object replaces the previous object of the same type.

```
HGDIOBJ SelectObject
(
  HDC hdc,                  // handle to DC
  HGDIOBJ hgdiobj           // handle to object
);
```

Sent Messages

- The `SendMessage()` function sends the specified message to a window.

- It calls the window procedure for the specified window and does not return until the window procedure has returned after processing the message.

Posted Messages

- The `PostMessage()` function
- Places (posts) a message in the message queue associated with the thread that created the specified window
- Returns without waiting for the thread to process the message

Sent messages

The **SendMessage()** function sends the specified message to a window or windows. It calls the window procedure for the specified window and does not return until the window procedure has processed the message.

```
LRESULT SendMessage
(
   HWND hWnd,          // handle to destination window
   UINT Msg,           // message
  WPARAM wParam,       // first message parameter
   LPARAM lParam       // second message parameter
);
```

Posted messages

The **PostMessage()** function places (posts) a message in the message queue associated with the thread that created the specified window and returns without waiting for the thread to process the message.

```
BOOL PostMessage(
  HWND hWnd,              // handle to destination window
  UINT Msg,               // message
 WPARAM wParam,           // first message parameter
 LPARAM lParam            // second message parameter
);
```

PostQuitMessage()

- When **PostQuitMessage()** is called, then **WM_QUIT** message is sent to the application message queue.

```
VOID PostQuitMessage
(
    int nExitCode              // exit code
);
```

nExitCode

- **nExitCode** value is used as the **wParam** parameter of the **WM_QUIT** message.

- **nExitCode**

  Specifies an application exit code. This value is used as the `wParam` parameter of the `WM_QUIT` message.

PeekMessage()

- The **PeekMessage()** function dispatches incoming sent messages, checks the thread message queue for a posted message, and retrieves the message (if any exist).

```
BOOL PeekMessage(
  LPMSG lpMsg,            // message information
  HWND hWnd,              // handle to window
  UINT wMsgFilterMin,     // first message
  UINT wMsgFilterMax,     // last message
  UINT wRemoveMsg         // removal options
);
```

`PeekMessage()`
Remove flags

- **`PM_NOREMOVE`**

  Messages are not removed from the queue after processing by **`PeekMessage()`**.

- **`PM_REMOVE`**

  Messages are removed from the queue after processing by **`PeekMessage()`**.