

Windows Programming

Lecture 13

Super-classing

- Super-classing defines a class that adds new functionality to a predefined window class, such as the button or list box controls.
- Defines a class with partly modified behavior of a pre-defined window class.

Super-classing

```
BOOL GetClassInfo  
(  
    HINSTANCE hInstance,  
    LPCTSTR lpClassName,  
    LPWNDCLASS lpWndClass,  
)
```

Class Name

Class Data

GetClassLong()

```
LONG GetClassLong (
```

```
    HWND hWnd,           // handle to window
```

```
    int nIndex           // offset of value to retrieve
```

```
);
```

Super-classing

```
DLGPROC oldWindowProc;           // Global variable
WNDCLASS wndClass;

GetClassInfo(hInstance, "BUTTON", &wndClass);
wndClass.hInstance = hInstance;
wndClass.lpszClassName = "BEEPBUTTON";
OldWindowProc = wndClass.lpfnWndProc;
wndClass.lpfnWndProc = myWindowProc;
RegisterClass( &wndClass );

hWnd = CreateWindow("BEEPBUTTON", "Virtual University",
                   WS_VISIBLE | WS_OVERLAPPEDWINDOW,
                   50, 50, 200, 100,
                   NULL, NULL, hInstance, NULL);

oldWindowProc = (WNDPROC)SetWindowLong(hWnd, GWL_WNDPROC,
                                         (LONG)myWindowProc);

while(GetMessage(&msg, NULL, 0, 0) > 0)
{
    if(msg.message == WM_LBUTTONDOWN)
        DispatchMessage(&msg);
}
return msg.wParam;
```

Super-classing

```
LRESULT CALLBACK myWindowProc(HWND hWnd,
                                UINT message,
                                WPARAM wParam,
                                LPARAM lParam)
{
    switch (message)
    {
        case WM_LBUTTONDOWN:
            MessageBeep(0xFFFFFFFF) ;
        default:
            return CallWindowProc(oldWindowProc,
                                   hWnd, message,
                                   wParam, lParam) ;
    }
    return 0;
}
```

GDI

(Graphics Device Interface)

The Windows subsystem responsible for displaying text and images on display devices and printers. The GDI processes graphical function calls from a Windows-based application. It then passes those calls to the appropriate device driver, which generates the output on the display hardware.

By acting as a buffer between applications and output devices, the GDI presents a device-independent view of the world for the application while interacting in a device-dependent format with the device. Because of the smaller memory footprint of Windows CE-based devices, Windows CE supports only a subset of the standard Win32 GDI.

Device Context

A GDI structure containing information that governs the display of text and graphics on a particular output device. A device context stores, retrieves, and modifies the attributes of graphic objects and specifies graphic modes. The graphic objects stored in a device context include a pen for line drawing, a brush for painting and filling, a font for text output, a bitmap for copying or scrolling, a palette for defining the available colors, and a region for clipping.

Device context is released, when Painting is finished.

GDI manages

- Text printing/drawing
- Color: foreground and background
- Font size
- Font face

Steps involved in output of a text string in the client area of the application

1. Get the handle to the Device Context for the window's client area from the GDI.
2. Use this Device Context for writing / painting in the client area of the window.
3. Release the Device context.

Printing a text string

```
HDC =hDC;  
hDC = GetDC (hWnd) ;  
TextOut (hDC, 0, 0,  
         "Our First GDI Call",18) ;  
ReleaseDC (hWnd, hDC) ;  
  
while (GetMessage (&msg, NULL, 0, 0) > 0)  
{  
    ... ..  
}
```

GetDC ()

The **GetDC ()** function retrieves a handle to a display device context (DC) for the client area of a specified window or for the entire screen. You can use the returned handle in subsequent GDI functions to draw in the DC.

```
hDC = GetDC ( hWnd ) ;
```

TextOut()

The **TextOut()** function writes a character string at the specified location, using the currently selected font, background color, and text color.

TextOut ()

Parameters

hdc

[in] Handle to the device context.

nXStart

[in] Specifies the logical x-coordinate.

nYStart

[in] Specifies the logical y-coordinate.

lpString

[in] Pointer to the string to be drawn.

cbString

[in] Specifies the length of the string. For the ANSI function it is a BYTE count and for the Unicode function it is a WORD count.

ReleaseDC ()

The **ReleaseDC ()** function releases a device context (DC), freeing it for use by other applications.

```
int ReleaseDC(  
    HWND hWnd,           // handle to window  
    HDC hDC              // handle to DC  
);
```

WM_PAINT

- When a minimized window is maximized, Windows requests the application to repaint the client area.
- Windows sends a **WM_PAINT** message for repainting a window.

WM_PAINT

The **WM_PAINT** message is sent when the system or another application makes a request to paint a portion of an application's window. The message is sent when the **UpdateWindow()** or **RedrawWindow()** function is called, or by the **DispatchMessage()** function when the application obtains a **WM_PAINT** message by using the **GetMessage()** or **PeekMessage()** function. A window receives this message through its **Window Procedure** function.

WM_PAINT

- The **WM_PAINT** message is generated by the system and should not be sent by an application
- The **DefWindowProc()** function validates the update region
- The system sends this message when there are no other messages in the application's message queue.

WM_PAINT

A WM_PAINT message is sent when:

- Some hidden part of a window becomes visible
- A window is resized and the window class style has the CS_REDRAW and CS_VREDRAW bits set.
- Programm scrolls its window
- InvalidateRect() is called to invalidate some part of a window.

BeginPaint()

- The **BeginPaint()** function prepares the specified window for painting and fills a **PAINTSTRUCT** structure with information about the painting.
- **BeginPaint()** first erases the background of window's client area by sending **WM_ERASEBKGND** message.
- **BeginPaint()** reserves a device context whose handle is returned to the application.

BeginPaint()

WM_PAINT message.

```
HDC BeginPaint(  
    HWND hwnd,                // handle to window  
    LPPAINTSTRUCT lpPaint // paint information  
);
```

- If the function succeeds, the return value is the handle to a display device context for the specified window.

EndPaint()

- **EndPaint()** is used to free the system resources reserved by the **BeginPaint()**.
- This function is required for each call to the **BeginPaint()** function, but only after painting is complete.

EndPoint()

```
BOOL EndPaint(  
    HWND hWnd,                // handle to window  
    CONST PAINTSTRUCT *lpPaint // paint data  
);
```

WM_SIZING

Repainting when the window is resized

Whenever a window is resized, system sends **WM_SIZING** message to the application that owns the window.

WM_SIZING

```
case WM_SIZING:
```

```
    hDC = GetDC (hWnd) ;
```

```
    TextOut (hDC, 0, 0,  
            "Our First GDI Call", 18) ;
```

```
    ReleaseDC (hWnd, hDC) ;
```

```
break;
```

DefWindowProc() erases background of the window using class brush specified while registering the window class.

WM_PAINT

To send **WM_PAINT** message whenever a window is resized, we specify **CS_HREDRAW**, **CS_VREDRAW** class styles in **WNDCLASS** structure while registering the class.