

Capítulo 1 - Programando con el API

El API de Win32 siempre se ha conocido por su complejidad y su polémica. Bueno, pues no es para tanto. Aquí vamos a ver lo que nosotros debemos saber para crear una ventana en la que podamos trabajar con DirectX. Vamos a centrarnos en el código que es como mejor se aprende:

```
#include <windows.h>
```

```
//-----
// Definiciones
//-----
#define APP_NAME "Prueba de Win32"
#define WND_WIDTH 640
#define WND_HEIGHT 480
```

En principio ponemos una serie de macros que definen el nombre de nuestra aplicación y el tamaño de nuestra ventana (ancho y alto). Esto es así por comodidad ya que de esta forma, si queremos modificar el tamaño de la ventana tan sólo deberemos hacerlo aquí y no en todos y cada uno de los lugares donde usemos estos parámetros.

Seguidamente encontramos otra macro bastante útil que sirve para averiguar si el usuario ha pulsado una tecla determinada... en principio tan sólo se utiliza para comprobar si el usuario pulsa la tecla ESC para salir del programa:

```
// Nos dice si una tecla determinada ha sido pulsada
#define GET_KEYDOWN(key) ((int)((GetAsyncKeyState(key) & (1 << 15)) >> 15))
```

A continuación vienen las declaraciones de funciones y de variables globales tal como esto:

```
//-----
// Declaración de funciones
//-----
LRESULT WINAPI MsgProc(HWND, UINT, WPARAM, LPARAM);
INT WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int);

//-----
// Declaración de variables globales
//-----
HWND g_hWnd; // Nuestra ventana
```

La función **MsgProc** será el manejador de mensajes de Windows y **WinMain** es la función principal que toda aplicación Windows debe tener.

En las variables globales nos encontramos con un handle (manejador) a nuestra ventana principal la cual utilizaremos a menudo.

Seguidamente comenzamos con la implementación de las funciones declaradas. Comenzamos con **MsgProc**:

```
//-----
// Función: MsgProc
// Propósito: Manejar los mensajes de la aplicación
//-----
LRESULT WINAPI MsgProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    switch(msg)
    {
        case WM_DESTROY:
            PostQuitMessage(0);
            return 0;

        case WM_PAINT:
            ValidateRect(hWnd, NULL);
```



```

        return 0;
    }

    return DefWindowProc(hWnd, msg, wParam, lParam);
}

```

Como podéis ver, aquí se ponen los mensajes que nos interesa controlar, es decir, si por ejemplo quisiéramos controlar el tamaño de nuestra ventana pues deberíamos poner un mensaje tipo WM_SIZE. En este caso en concreto tenemos un mensaje del tipo WM_DESTROY que Windows envía a la aplicación cuando esta se va a cerrar y un mensaje WM_PAINT en el que especificamos que la ventana se actualice...realmente nosotros no utilizaremos este mensaje para dibujar nuestro mundo 3D ya que llamando a nuestra función de render desde WinMain las cosas irán más rápidas.

Una vez visto esto pasamos a la función más importante de una aplicación Windows, es decir, la función **WinMain**. En nuestro ejemplo tiene un aspecto tal como...

```

//-----
// Función: WinMain
// Propósito: Es la función principal de toda aplicación de Windows
// Se encarga de crear la ventana, mostrarla y es la receptora de los
// mensajes que le son enviados a la aplicación
//-----
INT WINAPI WinMain(HINSTANCE hInstance, // Instancia a nuestra aplicación
                  HINSTANCE hPrevInstance, // Existen instancias previas de
nuestra aplicación?
                  LPSTR lpCmdLine, // Línea de comandos
                  int nShowCmd) // Cómo se despliega nuestra ventana?
{
    MSG msg; // handle a un mensaje
    WNDCLASSEX wc; // clase de ventana

    // Rellenamos la estructura de clase de ventana

    wc.cbSize = sizeof(WNDCLASSEX);
    wc.style = CS_CLASSDC;
    wc.lpfnWndProc = MsgProc;
    wc.cbClsExtra = 0L;
    wc.cbWndExtra = 0L;
    wc.hInstance = GetModuleHandle(NULL);
    wc.hIcon = NULL;
    wc.hCursor = NULL;
    wc.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
    wc.lpszMenuName = NULL;
    wc.lpszClassName = APP_NAME;
    wc.hIconSm = NULL;

    // Registramos la clase de ventana

    if(!RegisterClassEx(&wc))
    {
        MessageBox(NULL, "No se pudo registrar la clase de ventana", "Error",
MB_OK|MB_ICONEXCLAMATION);
        return 0;
    }

    // Creamos la ventana

    g_hWnd = CreateWindow(APP_NAME, // nombre de la clase
                        APP_NAME, // título de la ventana
                        WS_OVERLAPPEDWINDOW, // tipo de ventana
                        CW_USEDEFAULT, // posición X de la ventana
                        CW_USEDEFAULT, // posición Y de la ventana

```



```

WND_WIDTH, // ancho de la ventana
WND_HEIGHT, // alto de la ventana
GetDesktopWindow(), // ventana padre
NULL, // menú
wc.hInstance, // instancia de la aplicación
NULL); // no se suele usar

// Mostrar nuestra ventana

ShowWindow(g_hWnd, nShowCmd);
UpdateWindow(g_hWnd);

// Bucle de mensajes típico de Windows
while(GetMessage(&msg, NULL, 0, 0))
{
    // si se pulsa ESC salimos
    if(GET_KEYDOWN(VK_ESCAPE))
        PostQuitMessage(0);

    TranslateMessage(&msg);

    DispatchMessage(&msg);
}

// Salir de la aplicación

UnregisterClass(APP_NAME, wc.hInstance);

return (msg.wParam);
}

```

En primer lugar nos encontramos con una variable de tipo **MSG** que es un handle a un mensaje. Esta variable recibirá el primer mensaje que se encuentra en la cola de mensajes de nuestra aplicación y que Windows controla. Después tenemos una variable **wc** (window class) que es una estructura del tipo **WNDCLASSEX** la cual rellenaremos con las características que queramos para nuestra ventana. Así, en las siguientes líneas definimos estas características tales como tamaño de la estructura, estilo de la ventana, función que utilizaremos para manejar los mensajes, la instancia a nuestra aplicación, el color de fondo de nuestra ventana y el nombre de nuestra clase ventana...los demás parámetros se han inicializado a 0L ó a NULL ya que no nos interesan. Lo siguiente será registrar la clase con **RegisterClassEx**.

El siguiente paso es crear la ventana propiamente dicha. Para ello utilizamos la función **CreateWindow** que como podéis ver está bien comentada. Decir en este apartado que el tipo de ventana es un tipo estándar con el icono de sistema, el icono de minimizar, maximizar, cerrar y los bordes para modificar su tamaño. Esto cambiará en el momento en que demos entrada a nuestra primera aplicación a pantalla completa... pero eso se explica en la sección de DirectX. Continuando con las siguientes líneas, tenemos que mostramos nuestra ventana con el par de funciones **ShowWindow** y **UpdateWindow**. Seguidamente viene el bucle de mensajes que recibe/envía mensajes de/para Windows. Dentro de este bucle hemos puesto nuestra función **GET_KEYDOWN** con el parámetro **VK_ESCAPE** que es una macro definida en el archivo de cabecera **WinUser.h** del API de Win32. Esto lo que hace es que si se pulsa la tecla ESC se sale del programa...¡así de simple!. Pues bien, en el periodo en el que nuestra aplicación esté activa, se irá recorriendo este bucle permanentemente hasta que se pulse ESC. Más adelante modificaremos este bucle para dar cabida a nuestra función de render y otras más... ya que esta es la forma más óptima para especificar nuestras funciones que necesitan aprovechar al máximo la

potencia de nuestro ordenador y no tener que esperar a que Windows nos dé permiso para poder, por ejemplo, renderizar nuestra escena. Como íbamos diciendo, este bucle se recorre permanentemente hasta que el usuario pulsa la tecla ESC, momento en el que se sale del bucle y también de la aplicación no sin antes liberar el espacio ocupado en memoria por nuestra clase de ventana con la función

UnregisterClass.

Bueno pues con esto tenemos una fabulosa ventana en pantalla. En las secciones de DirectX y Direct3D se explica como aprovecharla para pintar gráficos. Dentro de poco veremos aquí como manejar los recursos típicos de Windows (diálogos, botones, etc.).

Capítulo 2 - Reordenando todo

En este paso hemos organizado el código de inicialización de la ventana en una función aparte llamada **IniVentana**. Por lo que el contenido de esta función queda así:

```
HRESULT IniVentana(HINSTANCE hInstance)
{
    WNDCLASSEX wc; // clase de ventana

    // Rellenamos la estructura de clase de ventana
    wc.cbSize = sizeof(WNDCLASSEX);
    wc.style = CS_CLASSDC;
    wc.lpfnWndProc = MsgProc;
    wc.cbClsExtra = 0L;
    wc.cbWndExtra = 0L;
    wc.hInstance = GetModuleHandle(NULL);
    wc.hIcon = NULL;
    wc.hCursor = NULL;
    wc.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
    wc.lpszMenuName = NULL;
    wc.lpszClassName = APP_NAME;
    wc.hIconSm = NULL;

    // Registramos la clase de ventana

    if(!RegisterClassEx(&wc))
    {
        MessageBox(NULL, "No se pudo registrar la clase de ventana", "Error",
MB_OK|MB_ICONEXCLAMATION);
        return E_FAIL;
    }

    // Creamos la ventana

    g_hWnd = CreateWindow(APP_NAME, // nombre de la clase
                          APP_NAME, // título de la ventana
                          WS_OVERLAPPEDWINDOW, // tipo de ventana
                          CW_USEDEFAULT, // posición X de la ventana
                          CW_USEDEFAULT, // posición Y de la ventana
                          WND_WIDTH, // ancho de la ventana
                          WND_HEIGHT, // alto de la ventana
                          GetDesktopWindow(), // ventana padre
                          NULL, // menú
                          wc.hInstance, // instancia de la aplicación
                          NULL); // no se suele usar

    if (!g_hWnd)
    {
        MessageBox(NULL, "No se creó el objeto g_hWnd", "Error",
MB_OK|MB_ICONEXCLAMATION);
        return E_FAIL;
    }

    return S_OK;
}
```

Para mejorar el sistema en que nuestra **WinMain** manejará los mensajes de Windows y nuestra función principal del bucle del juego (nosotros la vamos a llamar **Render**) hemos introducido varios cambios. **WinMain** después de aplicarle esos cambios y la llamada a la nueva función **IniVentana** queda así:

```
//-----
// Función: WinMain
```



```
// Propósito: Es la función principal de toda aplicación de Windows
// Se encarga de crear la ventana, mostrarla y es la receptora de los
// mensajes que le son enviados a la aplicación
//-----
INT WINAPI WinMain(HINSTANCE hInstance, // Instancia a nuestra aplicación
                  HINSTANCE hPrevInstance, // Existen instancias previas de
nuestra aplicación?
                  LPSTR lpCmdLine, // Línea de comandos
                  int nShowCmd) // Cómo se despliega nuestra ventana?
{
    if(!SUCCEEDED(IniVentana(hInstance)))
        return 0;

    // Mostrar nuestra ventana

    ShowWindow(g_hWnd, nShowCmd);
    UpdateWindow(g_hWnd);

    // Bucle de mensajes típico de Windows

    MSG msg; // handle a un mensaje
    ZeroMemory(&msg, sizeof(msg));

    while(msg.message != WM_QUIT)
    {
        // si se pulsa ESC salimos
        if(GET_KEYDOWN(VK_ESCAPE))
            PostQuitMessage(0);

        if(PeekMessage(&msg, NULL, 0U, 0U, PM_REMOVE))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
        else
        {
            //Nuestra función de render aquí
        }
    }

    // Salir de la aplicación

    UnregisterClass(APP_NAME, wc.hInstance);

    return (0);
}
```

Esta es la manera más robusta/rápida de tratar los mensajes y tener a la vez nuestra función de Render ejecutándose constantemente mientras no suceda ningún evento en nuestra ventana.

Capítulo 3 - Cuadros de diálogo

En este paso vamos a añadir un pequeño cuadro de diálogo con un edit box, un combo box y tres botones (Prueba, Aceptar y Cancelar). Para empezar hemos creado una hoja de recursos nueva con un icono (**IDI_ICON1**), un menú (**IDR_MENU1**) y un diálogo (**IDD_DIALOG1**) que hemos editado para que tenga los elementos mencionados. En **WinMain** lo único que hemos cambiado ha sido la inicialización de una nueva variable global llamada **hInst** de tipo **HINSTANCE** con el valor de **hInstance** para que todas las funciones puedan acceder al valor de la instancia de nuestra aplicación:

```
//-----
// Función: WinMain
// Propósito: Es la función principal de toda aplicación de Windows
// Se encarga de crear la ventana, mostrarla y es la receptora de los
// mensajes que le son enviados a la aplicación
//-----
INT WINAPI WinMain(HINSTANCE hInstance, // Instancia a nuestra aplicación
                  HINSTANCE hPrevInstance, // Existen instancias previas de
nuestra aplicación?
                  LPSTR lpCmdLine, // Línea de comandos
                  int nShowCmd) // Cómo se despliega nuestra ventana?
{
    hInst=hInstance;
    .
    .
    .
    .
}
```

Y en **IniVentana** los cambios también han sido son pocos. En la estructura **wc** de tipo **WNDCLASSEX** hemos definido los nuevos recursos para que cargue el icono y el menú. El icono lo cargamos con la función **LoadIcon** que recibe en primer lugar la instancia de la aplicación y después el recurso que tiene que obtener, y el menú lo cargamos directamente a través de su recurso con la función **MAKEINTRESOURCE**:

```
WNDCLASSEX wc; // clase de ventana

// Rellenamos la estructura de clase de ventana
wc.cbSize = sizeof(WNDCLASSEX);
wc.style = CS_CLASSDC;
wc.lpfnWndProc = MsgProc;
wc.cbClsExtra = 0L;
wc.cbWndExtra = 0L;
wc.hInstance = GetModuleHandle(NULL);
wc.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_ICON1));
wc.hCursor = NULL;
wc.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
wc.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);
wc.lpszClassName = APP_NAME;
wc.hIconSm = NULL;
```

Ahora viene lo más interesante, los cambios en **MsgProc**:

```
//-----
// Función: MsgProc
// Propósito: Manejar los mensajes de la aplicación
//-----
LRESULT WINAPI MsgProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    switch(msg)
    {
        case WM_CREATE: //Inicio de la aplicacion
            MessageBox(hWnd, "La aplicacion ha sido creada.", "Alerta",
MB_ICONEXCLAMATION | MB_OK);
    }
}
```



```

        break;

    case WM_QUIT: //Fin de la aplicacion
        MessageBox(hWnd, "La aplicación se va a terminar.",
"Alerta", MB_ICONEXCLAMATION | MB_OK);
        break;

    case WM_COMMAND:
        switch(wParam)
        {
            case IDM_ARCHIVO_SALIR: //Menu --> Salir
                PostMessage(hWnd, WM_QUIT, 0, 0L);
                break;

            case IDM_ARCHIVO_DIALOGO: //Menu --> Dialogo
                //Creamos el nuevo diálogo
                DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG1), hWnd,
(DLGPROC) ProcedimientoDialogo);
                break;
        }
        break;

    case WM_LBUTTONDOWN: //Boton Izqdo
        MessageBox(hWnd, "Has hecho click con el boton izquierdo en el área
de cliente.", "Alerta", MB_ICONEXCLAMATION | MB_OK);
        break;

    case WM_RBUTTONDOWN: //Boton Dcho
        MessageBox(hWnd, "Has hecho click con el boton derecho en el área de
cliente.", "Alerta", MB_ICONEXCLAMATION | MB_OK);
        break;

    case WM_DESTROY:
        PostQuitMessage(0);
        return 0;

    case WM_PAINT:
        ValidateRect(hWnd, NULL);
        return 0;
    }

    return DefWindowProc(hWnd, msg, wParam, lParam);
}

```

Como se puede observar hemos añadido unos cuantos tratamientos a nuevos mensajes como son: **WM_CREATE** (cuando se crea la aplicación...), **WM_QUIT** (cuando se destruye...), **WM_LBUTTONDOWN** (cuando se pulsa el botón izquierdo...) y **WM_RBUTTONDOWN** (cuando se pulsa el botón derecho...). Estos cuatro mensajes nos dicen cuando suceden estos eventos y en caso de que sucedan nosotros ponemos el código que queremos que se ejecute. Bien, pues aún nos queda un último mensaje por analizar que es **WM_COMMAND**. Este mensaje nos indica que se ha efectuado un *click* sobre alguno de nuestros elementos del menú y **wParam** nos indica exactamente cual. Nosotros hemos puesto que si es el que está asociado con el botón de salir, termine la aplicación, y si es el botón de diálogo, que ejecute tal diálogo. A continuación podemos observar la llamada a la función que nos abre el diálogo:

```

DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG1), hWnd, (DLGPROC)
ProcedimientoDialogo);

```

A esta función le pasamos nuestro **hInst**, el identificador del diálogo que queremos usar, el **hWnd** y la rutina que va a servir como tratamiento de ese diálogo. Esta rutina la hemos definido así:

```

//-----

```



```
// Función: ProcedimientoDialogo
// Propósito: Manejar el nuevo dialogo de la aplicación
//-----
BOOL CALLBACK ProcedimientoDialogo(HWND hWndDlg,UINT Msg,WPARAM wParam,LPARAM
lParam)
{
    char Mensaje[256], Aux[256];

    //Creamos tres cadenas para el combo box
    char *Cadena[] = {"Nivel 1", "Nivel 2", "Nivel 3"};

    switch(Msg)
    {
        case WM_INITDIALOG:

            //Rellenamos el combo box con las tres cadenas que hemos definido
            SendDlgItemMessage(hWndDlg, IDC_COMBO1, CB_ADDSTRING, 0, (LPARAM)
Cadena[0]);
            SendDlgItemMessage(hWndDlg, IDC_COMBO1, CB_ADDSTRING, 0, (LPARAM)
Cadena[1]);
            SendDlgItemMessage(hWndDlg, IDC_COMBO1, CB_ADDSTRING, 0, (LPARAM)
Cadena[2]);

            //Ponemos dos frases inciales en el combo box y en el edit box
            SetDlgItemText(hWndDlg, IDC_EDIT1, "Pon tu nombre aquí");
            SetDlgItemText(hWndDlg, IDC_COMBO1, "Indica un nivel aquí");

            //Ponemos el foco en el edit box
            SetFocus(GetDlgItem(hWndDlg, IDC_EDIT1));

            return FALSE;

        case WM_CLOSE:
            EndDialog(hWndDlg, TRUE);
            return TRUE;

        case WM_COMMAND:
            switch(LOWORD(wParam))
            {
                case IDCANCEL:
                    EndDialog(hWndDlg, TRUE);
                    break;

                case IDOK:
                    //Tomamos el contenido del edit box y lo pasamos a 'Mensaje'
                    GetDlgItemText(hWndDlg, IDC_EDIT1, Mensaje, sizeof(Mensaje));
                    //Le añadimos parentesis
                    strcat(Mensaje, " (");
                    //Tomamos el contenido del combo box y lo pasamos a 'Aux'
                    GetDlgItemText(hWndDlg, IDC_COMBO1, Aux, sizeof(Aux));
                    //Se lo añadimos a 'Mensaje'
                    strcat(Mensaje, Aux);
                    //Le añadimos parentesis
                    strcat(Mensaje, ") (Boton Ok)");
                    //Lo mostramos mediante message box
                    MessageBox(hWndDlg, Mensaje, "Diálogo", MB_OK);
                    //Terminamos el dialogo
                    EndDialog(hWndDlg, TRUE);
                    break;

                case IDC_PRUEBA:
                    //Tomamos el contenido del edit box y lo pasamos a 'Mensaje'
                    GetDlgItemText(hWndDlg, IDC_EDIT1, Mensaje, sizeof(Mensaje));
```



```
        //Le añadimos parentesis
        strcat(Mensaje, " (");
        //Tomamos el contenido del combo box y lo pasamos a 'Aux'
        GetDlgItemText(hWndDlg, IDC_COMBO1, Aux, sizeof(Aux));
        //Se lo añadimos a 'Mensaje'
        strcat(Mensaje, Aux);
        //Le añadimos parentesis
        strcat(Mensaje, ") (Boton Prueba)");
        //Lo mostramos mediante message box
        MessageBox(hWndDlg, Mensaje, "Diálogo", MB_OK);
        break;
    }

    return TRUE;
}
return FALSE;
}
```

Esta rutina es exactamente un **MsgProc** exclusivo para este diálogo y se tratará de la misma manera. Como podemos observar tiene la misma estructura de tratamiento de mensajes. La función está bastante bien comentado y no tiene mucha complicación para entender. Sólo hay que puntualizar que **SetDlgItemText()** sirve para mandarle una cadena al recurso en cuestión y **GetDlgItemText()** para lo contrario, pero todas estas funciones de cada recurso (edit box, combo box, etc.) pueden verse cómodamente en la ayuda del SDK.