

Forum:[Hack Forums](#)

Thread: [Complexity: Introduction to the Win32 API](#)

Inso - June 10, 2013, 11:25 am

Introduction to the Win32 API

Hello and welcome to my tutorial on the Windows Application Programming Interface for C++. So what is it? The Windows API is a way for programmers to create graphical user interfaces (GUI) on the Windows operating system (OS). It allows the programmer to have a simpler way to do this rather than have to handle compatibility with different hardware and various other aspects of development between machines.

So why use the WinAPI? Aren't there easier packages to do this?

This is true, various frameworks such as MFC and .NET exist for rapid development of windows forms. Most of these frameworks work as wrappers for the Win32 API, where all of the code used would be wrapped within premade classes that cuts down on the amount of code that must be wrote.

The Win32 API should still be learned though as it can be helpful in the event of an error with one of these frameworks. With no understanding of how these frameworks really work, it will be difficult or more time consuming to fix any problems.

As this is my first tutorial on the Win32 API, I'm only going to show the Windows version of the popular Hello World program and explain it step by step. Because a bit more work would go into using the WinAPI, the Hello World program will be much bigger than expected.

This is a simple "Hello World" program using regular C++:

PHP Code

```
<?php
#include <iostream>
using namespace std;

int main() {
    cout << "Hello, World!";
    return 0;
}
```

This is a nice short program. In contrast, the Win32 requires much more code just for a simple window. Charles Petzold (who has wrote many books on programming with the Windows API) has been quoted as saying "The original hello world program in the Windows 1.0 SDK was a bit of a scandal. HELLO.C was about 150 lines long, and the HELLO.RC resource script had another 20 or so more lines. (...) Veteran C programmers often curled up in horror or laughter when encountering the Windows hello-world program".

Okay, so now onto our program. The first thing to do before anything else is done, is include the Windows header file, as this is the file where we get all the Windows functions, data structures, etc. This is done using something called a "preprocessor directive". This is a command that the compiler reads and responds to, in this case it bundles the windows header file along with your source into the program. Preprocessor directives are preceded by a hash (#) symbol. Unlike other statements in C++, it is not ended by a semicolon (;).

PHP Code

```
<?php
#include <Windows.h>
```

The next thing to do is create our main function. Console based programs in C++ will use an actual main() function, but when programming with Win32 you use either WinMain() or wWinMain() as Windows.h will provide its own version of the main function. The only difference between the two is wWinMain uses unicode and thus wide characters, whereas WinMain used ASCII character codes.

In this tutorial I will use the wWinMain() function. It has the following prototype:

PHP Code

```
<?php
int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE prevInstance, LPWSTR cmdLine,
int cmdShow);
```

hInstance is the handle to the windows instance. prevInstance is also handle to a windows instance, but it should always be 0 as it is a parameter only used on 16 bit Windows. cmdLine is the arguments given to the program as a unicode string if the program was run from the command prompt. cmdShow is a flag that tells Windows if the program should start maximised, minimised or normal.

Here is our empty wWinMain function:

PHP Code

```
<?php
#include <Windows.h>
```

```
int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE prevInstance, LPWSTR cmdLine,
int cmdShow) {
    return static_cast<int>(msg.wParam);
}
```

Because our program won't actually be using prevInstance or cmdLine, two additional functions can be called inside wWinMain which stops the compiler throwing errors at you for not using arguments.

PHP Code

```
<?php
#include <Windows.h>
```

```
int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE prevInstance, LPWSTR cmdLine,
int cmdShow) {
    UNREFERENCED_PARAMETER(prevInstance);
    UNREFERENCED_PARAMETER(cmdLine);
    return static_cast<int>(msg.wParam);
}
```

```
}
```

Okay, now the next step is to create a windows class. This is done using WNDCLASSEX, a type definition of the data structure tagWNDCLASSEX. The MSDN documentation defines it as follows:

PHP Code

```
<?php
typedef struct tagWNDCLASSEX {
    UINT cbSize; //Size in bytes of the structure
    UINT style; //Class style flags
    WNDPROC lpfnWndProc; //A pointer to the window procedure
    int cbClsExtra; //Number of extra bytes to allocate the window-class structure
    int cbWndExtra; //The number of extra bytes to allocate the window instance
    HINSTANCE hInstance; //A handle to the window procedure for the class
    HICON hIcon; //A handle to the class icon
    HCURSOR hCursor; //A handle to the class cursor
    HBRUSH hbrBackground; //A handle to the class background brush
    LPCTSTR lpszMenuName; //Pointer to string that specifies the resource name of the class menu
    LPCTSTR lpszClassName; //A pointer to a string specifying the class name
    HICON hIconSm; //A handle to a small icon for the window class
} WNDCLASSEX, *PWNDCLASSEX;
```

To create your new window class, put the following code in your wWinMain function, after the UNREFERENCED_PARAMETER() function and before the return statement.

PHP Code

```
<?php
WNDCLASSEX wndClass = {0};

wndClass.cbSize = sizeof(WNDCLASSEX);
wndClass.style = CS_HREDRAW | CS_VREDRAW;
wndClass.lpfnWndProc = WndProc;
wndClass.hInstance = hInstance;
wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
wndClass.lpszMenuName = NULL;
wndClass.lpszClassName = "MyTutorialWindow";
```

In this snippet of code, we have created an instance of WNDCLASSEX and called it wndClass, then we initiate all of its values to 0 (0 is equivalent to null in the Windows API, somewhere in Windows.h there is a line reading #define NULL 0).

The size of the structure is set to the size of the WNDCLASSEX structure. Next it is given several style flags, in this case CS_HREDRAW and CS_VREDRAW. Flags are always separated by the bitwise OR symbol. CS_HREDRAW and CS_VREDRAW both redraw the entire window if a change is made to the horizontal or vertical client area respectively. A list of style flags can be found [here](#).

Next lpfnWndProc was set to WndProc, a function that we will be creating later. The hInstance is set to the application instance containing the window procedure for the class. hCursor is set using the LoadCursor function, which takes an HINSTANCE (optional) and LPCTSTR argument which is the name for the cursor to be used. To get a list of all cursor types available, use [this](#) link. this function will return a variable of type HCURSOR, which is the handle to the cursor. If the function fails it returns a NULL (0) value.

hbrBackground is the handle to the background brush for this class. It may be a colour value or a handle to a brush. If it is a colour value, it must use one of the following colour values:

COLOR_ACTIVEBORDER

COLOR_ACTIVECAPTION

COLOR_APPWORKSPACE

COLOR_BACKGROUND

COLOR_BTNFACE

COLOR_BTNSHADOW

COLOR_BTNTEXT

COLOR_CAPTIONTEXT

COLOR_GRAYTEXT

COLOR_HIGHLIGHT

COLOR_HIGHLIGHTTEXT

COLOR_INACTIVEBORDER

COLOR_INACTIVECAPTION

COLOR_MENU

COLOR_MENUTEXT

COLOR_SCROLLBAR

COLOR_WINDOW

COLOR_WINDOWFRAME

COLOR_WINDOWTEXT

Each of these values is equivalent to an integer. The reason we must use +1 is because the first value is equivalent to 0, so we add 1 so windows doesn't think we are supplying a NULL brush. We have set the menu name to null as we don't have one and we have called our class "MyTutorialWindow". Remember this name, as it will be used again later.

Next we must register this class. Add the following snippet of code after before the return statement.

PHP Code

```
<?php
```

```
if(!RegisterClassEx(&wndClass)) {
    return -1;
}
```

This if conditional check uses the RegisterClassEx function as it's condition. If RegisterClassEx successfully registers the class it will return true. If it fails it will return false. This statement uses the NOT operator (!), so it checks if RegisterClassEx returns false. If it does, the wWinMain function returns -1 and will terminate.

The next step is to create our window and the handle to it. Add the following code before the return statement.

```
PHP Code
<?php
HWND hwnd = CreateWindow(
    "MyTutorialWindow",
    "Hello, World!",
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT,
    CW_USEDEFAULT,
    500,
    300,
    NULL,
    NULL,
    hInstance,
    NULL
);
```

We have created a handle to a window called hwnd, which has been initialised using the CreateWindow() function. This takes 11 parameters. Although this is a function call, you may have noticed I have separated the arguments on different lines. C++ allows the use of whitespace like this, so it should be taken advantage of to improve readability of your code. In the MSDN document, CreateWindow is defined as follows:

```
PHP Code
<?php
HWND WINAPI CreateWindow(
    LPCTSTR lpClassName,
    LPCTSTR lpWindowName,
    DWORD dwStyle,
    int x,
    int y,
    int nWidth,
    int nHeight,
    HWND hWndParent,
    HMENU hMenu,
    HINSTANCE hInstance,
    LPVOID lpParam
);
```

It has a return value of HWND (Window Handle). It takes the argument of lpClassName (the class name we created earlier), lpWindowName (the text that will appear in the title label), the style flags (a full list can be found [here](#)), the initial x and y coordinates (in this case we used the default,

CW_USEDEFAULT), the dimensions of the window, the handle of the parent window (ours has no parent so it is NULL), the handle to any menu associated with the window, an instance to the module associated with the window, and a pointer to a value passed through CREATESTRUCT (this isn't necessary for our window so it has also been set to NULL).

After creating a window it is good practice to check if CreateWindow() failed. To do this, add the following snippet.

PHP Code

```
<?php
if(!hwnd) {
    return -1;
}
```

If this function is successful it will return an HWND for hwnd, but if it fails hwnd will be NULL which is equivalent to 0. This conditional check sees if hwnd is NULL, if it is the wWinMain function returns -1 and the program terminates. To display our window we now called the ShowWindow() function. This takes 2 arguments, the window handle of the window we want to display, and what state the window should open in.

PHP Code

```
<?php
ShowWindow(hwnd, cmdShow);
```

cmdShow will be given by the wWinMain function, it is one of the arguments we gave when writing it. After the first time the window is called and you decide to call ShowWindow again to change the state, the following values can be used instead of cmdShow:

SW_HIDE: 0, hides the window and activates another window

SW_SHOWNORMAL: 1, show the window in its normal state

SW_SHOWMINIMIZED: 2, activates the window and minimises it

SW_SHOWMAXIMIZED: 3, maximises the window

SW_SHOWNOACTIVATE: 4, similar to SW_SHOWNORMAL but the window is not activated

SW_SHOW: 5, activates the window and shows it in its current size and position

SW_MINIMIZE: 6, similar to SW_SHOWMINIMIZED but the next window in the Z order is activated

SW_SHOWMINNOACTIVE: 7, similar to SW_SJOWMINIMIZED but the window is not activated

SW_SHOWNA: 8, similar to SW_SHOW but the window is not activated

SW_RESTORE: 9, activates the window and displays it in its original size and position

SW_SHOWDEFAULT: 10, sets the window to a default size and position

SW_FORCEMINIMIZE: 11, minimises the window even if the thread that owns the window is not responding

To see these calls in action, you could change your ShowWindow call to the following:

PHP Code

```
<?php
ShowWindow(hwnd, cmdShow);
Sleep(2000);
ShowWindow(hwnd, SW_SHOWMAXIMIZED);
```

This will wait for 2 seconds, then maximise the window. Don't try this until the entire window is made though, as there are still some parts to add before the window will even work. While following the rest of this tutorial, keep only the original ShowWindow() call, you can try out the second one and the Sleep() afterwards.

Next I'm going to introduce the MSG structure. It is defined in the MSDN documentation as follows:

PHP Code

```
<?php
typedef struct tagMSG {
    HWND  hwnd;
    UINT  message;
    WPARAM wParam;
    LPARAM lParam;
    DWORD time;
    POINT pt;
} MSG, *PMSG, *LPMSG;
```

hwnd is the handle to the window that has a windows procedure that receives messages (you will learn more about this later), message is the message identifier, wParam and lParam contains additional information about this message, time is the time the message was sent, and pt is the position of the cursor at the time of the message being sent.

Now it's time to populate our MSG structure, add the following:

PHP Code

```
<?php
MSG msg = { 0 };

while(msg.message != WM_QUIT) {
    if(PeekMessage(&msg, 0, 0, 0, PM_REMOVE)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}
```

We first create an instance of the data structure, called msg and we set all of its member variables to 0. We have created a while loop that continues as long as msg.message is not WM_QUIT. WM_QUIT is the message used to quit a program. For each iteration of the loop we check to see if PeekMessage returns true. If it returns NULL there are no messages in the "message queue" (the queue where message are kept until they can be processed). PeekMessage() takes arguments of the address of our MSG structure, the handle to a window where messages should be received, the first and last in the range of messages to be examined, and a flag specifying how messages are handled. In this case it is PM_REMOVE (Removed from queue after being processed by

PeekMessage). Other flags that could be used (may be read about in more detail [here](#)) instead are:
PM_NOREMOVE: Not removed from queue

PM_REMOVE: Removed from queue

PM_NOYIELD: Prevents threads being released that are waiting for the caller to go idle

All messages are automatically processed, but if you only want one or more types of messages to be processed you can also use the following flags:

PM_QS_INPUT: Mouse and keyboard messages

PM_QS_PAINT: Paint messages

PM_QS_POSTMESSAGE: All posted messages

PM_QS_SENDMESSAGE: All sent messages

These flags (like the window style flags) are separated by the bitwise OR operator (|).

TranslateMessage() will translate messages into character messages which are put into the message queue. DispatchMessage() will take messages from the queue and sends it to the window procedure, which is the next function we are going to create. Just to recap, your current wWinMain function should be as follows:

PHP Code

```
<?php
int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE prevInstance, LPWSTR cmdLine,
int cmdShow) {
    UNREFERENCED_PARAMETER(prevInstance);
    UNREFERENCED_PARAMETER(cmdLine);

    WNDCLASSEX wndClass = { 0 };

    wndClass.cbSize = sizeof(WNDCLASSEX);
    wndClass.style = CS_HREDRAW | CS_VREDRAW;
    wndClass.lpfnWndProc = WndProc;
    wndClass.hInstance = hInstance;
    wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
    wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
    wndClass.lpszMenuName = NULL;
    wndClass.lpszClassName = "MyTutorialWindow";

    if(!RegisterClassEx(&wndClass)) {
        return -1;
    }

    HWND hwnd = CreateWindow(
        "MyTutorialWindow",
        "Hello, World!",
```

```

WS_OVERLAPPEDWINDOW,
CW_USEDEFAULT,
CW_USEDEFAULT,
500,
300,
NULL,
NULL,
hInstance,
NULL
);

if(!hwnd) {
    return -1;
}

ShowWindow(hwnd, cmdShow);

MSG msg = { 0 };
while(msg.message != WM_QUIT) {
    if(PeekMessage(&msg, 0, 0, 0, PM_REMOVE)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}

return static_cast<int>(msg.wParam);
}

```

Now onto our windows procedure function. First, we need to write the prototype for our function, before the wWinMain function definition and after the #include directive write the following:

PHP Code

```

<?php
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam,
LPARAM lParam);

```

What this does is create the prototype for our WndProc (Windows Procedure). It takes 4 arguments; the handle to the window, the message identifier being sent, the first message parameter, and the second message parameter.

Now below the wWinMain function definition, create our WndProc function:

PHP Code

```

<?php
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam,
LPARAM lParam) {
    return 0;
}

```

Now it's time to fill our WndProc function. First, we create a PAINTSTRUCT data structure that will hold information used to paint on the client window. After this we create a handle to the device context which can either be of type display, printer, memory, or information. The device context is used to support drawing or data retrieval on different types of devices.

PHP Code

```
<?php
PAINTSTRUCT paintStruct;
HDC hDC;
```

Now we have to create a switch which uses cases defined by the message identifiers:

PHP Code

```
<?php
switch(message) {
case WM_PAINT:
    break;

case WM_DESTROY:
    break;

default:
    break;
}
```

This switch takes the message identifier passed to the WndProc function and does different things depending on this message identifier. In our switch we have included the WM_PAINT identifier and the WM_DESTROY identifier, but we have also included a default case in the event of being passed a message identifier that doesn't have a specific set of instructions attributed to it.

When populating this switch, each case should be in a block enclosed by braces, this prevents some errors which are caused when braces are left out (I'm not sure why though). For example:

PHP Code

```
<?php
switch(variable) {
case VALUE:
{
    MyStatements...
}
break;
}
```

So first, in the WM_PAINT case, add the following:

PHP Code

```
<?php
{
hDC = BeginPaint(hwnd, &paintStruct);
EndPaint(hwnd, &paintStruct);
}
```

BeginPaint takes a handle to the window being repainted and the address of our paintStruct

structure. It uses these values and returns a device context handle but if it fails it will return NULL. EndPaint takes the handle to the window that has been repainted, and the address of paintStruct which should hold data passed to it from BeginPaint.

Now add the following to the WM_DESTROY case:

PHP Code

```
<?php
{
PostQuitMessage(0);
}
```

When the WM_DESTROY message identifier is sent, it will call PostQuitMessage. PostQuitMessage will take an argument that specifies the exit code (0 tends to mean everything executed correctly). The exit code is used as the wParam parameter of the WM_QUIT message. Because WM_QUIT is not associated with a window (it is a call for the entire application to quit) it does not need a case to be created in the window procedure. It is instead generated when PostQuitMessage is called.

Now add this to the default case:

PHP Code

```
<?php
return DefWindowProc(hwnd, message, wParam, lParam);
```

DefWindowProc takes 4 arguments, the handle to the window procedure that received the message, the message identifier, and two parameters for additional message information. It is used to handle any messages that haven't been handled by our switch cases in the WndProc function.

If you haven't caught on yet, the basic flow for a Windows program will be as follows:

Program starts and your WinMain/wWinMain function is called

Create your window class

Register this class

Create a window

Initiate the message loop

Check if the message queue contains anything

If it does, call the windows procedure while passing the handle to whatever window is using this callback procedure

Use the information in this message to decide what to do

Act on this message

Continue looping until WM_QUIT is posted as a message

Quit the application

And now your window is complete! This is a basic window that will say "Hello, World!" in the title bar. The entire source code can be seen in the spoiler below:

Spoiler ([Click to View](#))

PHP Code

```
<?php
/*
 * My C++ blank win32 application
 */

#include <Windows.h>

LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam,
LPARAM lParam);

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE prevInstance, LPWSTR cmdLine,
int cmdShow) {
    UNREFERENCED_PARAMETER(prevInstance);
    UNREFERENCED_PARAMETER(cmdLine);

    WNDCLASSEX wndClass = { 0 };

    wndClass.cbSize = sizeof(WNDCLASSEX);
    wndClass.style = CS_HREDRAW | CS_VREDRAW;
    wndClass.lpfnWndProc = WndProc;
    wndClass.hInstance = hInstance;
    wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
    wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
    wndClass.lpszMenuName = NULL;
    wndClass.lpszClassName = "MyTutorialWindow";

    if(!RegisterClassEx(&wndClass)) {
        return -1;
    }

    HWND hwnd = CreateWindow(
        "MyTutorialWindow",
        "Hello, World!",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        500,
        300,
        NULL,
        NULL,
        hInstance,
        NULL
    );

    if(!hwnd) {
```

```

    return -1;
}

ShowWindow(hwnd, cmdShow);

MSG msg = { 0 };
while(msg.message != WM_QUIT) {
    if(PeekMessage(&msg, 0, 0, 0, PM_REMOVE)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}

return static_cast<int>(msg.wParam);
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam,
LPARAM lParam) {
    PAINTSTRUCT paintStruct;
    HDC hDC;

    switch(message) {
    case WM_PAINT:
        {
            hDC = BeginPaint(hwnd, &paintStruct);
            EndPaint(hwnd, &paintStruct);
        }
        break;

    case WM_DESTROY:
        {
            PostQuitMessage(0);
        }
        break;

    default:
        return DefWindowProc(hwnd, message, wParam, lParam);
    }

    return 0;
}

```

Inso - June 10, 2013, 11:54 am

(Jun 10, 2013 11:51 AM)Shihanus Wrote: [CITE](#) Another user completely mistaking the idea of the WIN Api.

Why is it that you idiots think just because it's the WIN Api, you're going to make GUIs?

That's not the WIN Api is for!

It's allows for the user to interact with the OS, simply that. I've been using the WIN Api for a couple of years now, and never created a GUI. Does that mean I'm doing something wrong?

I'm using it for forms creation in this thread and tutorial, if I was wanting to create a tutorial about

networking or using the OS shell then I would have mentioned those reasons. As of now, I haven't looked at those uses of it so I'm not going to mention them. I'm sharing what I (think) I know so far. I like to try learn something and then post a tutorial and get feedback. I can then see if I've learned it correctly, and if I get constructive feedback about mistakes, I can fix it and I've learned something new.

pr3fix - June 10, 2013, 11:58 am

You should have linked the MSDN documentation, so that people will know where to look at when they need explanation on certain function or parameter to pass to a function. Eventually, they will remember these and it'll be much easier.

Shihanus - June 10, 2013, 11:58 am

(Jun 10, 2013 11:54 AM)Inso Wrote: [\[CITE\]](#) I'm using it for forms creation in this thread and tutorial, if I was wanting to create a tutorial about networking or using the OS shell then I would have mentioned those reasons. As of now, I haven't looked at those uses of it so I'm not going to mention them. I'm sharing what I (think) I know so far. I like to try learn something and then post a tutorial and get feedback. I can then see if I've learned it correctly, and if I get constructive feedback about mistakes, I can fix it and I've learned something new.

I didn't even necessarily mean networking or OS Shells.

If you wanted to give a tutorial on the WIN API. You could have started with something even more basic.

```
Code #define WIN32_LEAN_AND_MEAN
#include <windows.h>
int main()
{
    MessageBoxA(GetDestkopWindow(),"Hello world!","MessageBox!",MB_OK);
    return 0;
}There you have it, Win32 API with Hello World.
```

Inso - June 10, 2013, 12:01 pm

(Jun 10, 2013 11:58 AM)Shihanus Wrote: [\[CITE\]](#) I didn't even necessarily mean networking or OS Shells.

If you wanted to give a tutorial on the WIN API. You could have started with something even more basic.

```
Code #define WIN32_LEAN_AND_MEAN
#include <windows.h>
int main()
{
    MessageBoxA(GetDestkopWindow(),"Hello world!","MessageBox!",MB_OK);
    return 0;
}There you have it, Win32 API with Hello World.
```

I did think of that but I thought it wouldn't be worth a tutorial on just the message box. I was trying to learn off several sites and one of them was the MSDN docs, two of them mentioned the message box at the start but it only seemed to take up a small space for the overall tutorial.

I myself don't know too much about it (the entire WinAPI, just starting it basically), currently I'm just learning about adding controls but I don't know enough about it either which is why I left it out too. I'm a bit annoyed about that, one site I was looking at had some great tutorials that I was learning from but the next part of the series is being wrote and isn't up yet. ./

(Jun 10, 2013 11:58 AM)pr3fix Wrote: [\[CITE\]](#) You should have linked the MSDN documentation, so that people will know where to look at when they need explanation on certain function or parameter to pass to a function. Eventually, they will remember these and it'll be much easier.

I'll see if I can edit it in now, the thread was too big so it had to be done as two posts and once they merge it won't let you edit because of the character limit.

If not, the MSDN is referenced for a few of the functions and structures in there, so hopefully they'll be able to navigate from those links.

EDIT: Nope, sorry can't.

Quote:There was an error performing the update.

The message is too long. Please enter a message shorter than 18000 characters (currently 20733).

Krofnica - June 10, 2013, 1:32 pm

I stopped reading at

Code Console based programs in C++ will use an actual main() function, but when programming with Win32 you use either WinMain() or wWinMain() as Windows.h will provide its own version of the main function.

Inso - June 10, 2013, 1:41 pm

(Jun 10, 2013 01:32 PM)Krofnica Wrote: [\[CITE\]](#) I stopped reading at

Code Console based programs in C++ will use an actual main() function, but when programming with Win32 you use either WinMain() or wWinMain() as Windows.h will provide its own version of the main function.

I don't see what's wrong with that? It might not be as simple as I've wrote it, from what I've read it's the CRT that provides an implementation of main() but I thought it would just be simpler to write it as I did.

Krofnica - June 10, 2013, 2:34 pm

(Jun 10, 2013 01:41 PM)Inso Wrote: [\[CITE\]](#) I don't see what's wrong with that? It might not be as simple as I've wrote it, from what I've read it's the CRT that provides an implementation of main() but I thought it would just be simpler to write it as I did.

CRT defines a function which does some initialization before user-defined main function begins with execution. That function is usually called maincrtstartup, and is entry point of all c(++) programs. It does not only call main with prototypes:

```
int main();  
int main(int, char**);
```

But it can also call:

```
int __stdcall WinMain(HINSTANCE, HINSTANCE, LPSTR, int);
```

windows.h does not define its own startup function :P Its all done by crt.

Inso - June 10, 2013, 2:40 pm

Oh right, guess I've learned something today. :P Thanks for the correction.

Deesr - July 8, 2013, 8:39 am

Thanks for this, I'm sure this will come in handy! Bookmarked.

hansp - July 14, 2013, 9:28 am

Thank you very much :) I learned a lot and improved my C++ Skills.

And because I am a beginner at C++ I ported your code to C#. Pretty useless, but a nice challenge

Spoiler ([Click to View](#))Code using System;

using System.Runtime.InteropServices;

//Credits to hansp, MSDN and pinvoke.net :P

//References to System.dll and System.Drawing.dll

namespace ConsoleApplication1

```
{

    class Program
    {
    static void Main()
    {
        if (createWindow() == 0) Console.WriteLine("Everything should have worked :)");
        else Console.WriteLine("Something failed :(");
        Console.ReadLine();
    }
    static int createWindow()
    {
        IntPtr hInstance = System.Diagnostics.Process.GetCurrentProcess().Handle;
        API.WNDCLASSEX wndClass = new API.WNDCLASSEX();
        wndClass.cbSize = (uint)Marshal.SizeOf(typeof(API.WNDCLASSEX));
        wndClass.style = API.CS.CS_HREDRAW | API.CS.CS_VREDRAW;
        wndClass.lpfnWndProc = WndProc;
        wndClass.hInstance = hInstance;
        wndClass.hCursor = API.LoadCursor(IntPtr.Zero,
32512/*API.IDC_STANDARD_CURSORS.IDC_ARROW*/);
        wndClass.hbrBackground = (IntPtr)6; // (HBRUSH)(COLOR_WINDOW(5) + 1) = 6
        wndClass.lpszMenuName = null;
        wndClass.lpszClassName = "MyTutorialWindow";
        API.RegisterClassEx(ref wndClass);
        IntPtr hwnd = API.CreateWindowEx(
        0,
        "MyTutorialWindow",
        "Hello, World!",
        API.WindowStyles.WS_OVERLAPPEDWINDOW,
        API.CW_USEDEFAULT,
        API.CW_USEDEFAULT,
        500,
        300,
        IntPtr.Zero,
        IntPtr.Zero,
        hInstance,
        IntPtr.Zero);

        if (hwnd == IntPtr.Zero)
        {
            return -1;
        }

        API.ShowWindow(hwnd, API.ShowWindowCommands.Show);
        API.UpdateWindow(hwnd);
    }
}
```

```

API.MSG msg = new API.MSG();
while (API.GetMessage(out msg, IntPtr.Zero, 0, 0) != 0)
{
    API.TranslateMessage(ref msg);
    API.DispatchMessage(ref msg);
}
return msg.WParam.ToInt32();
}
private static IntPtr WndProc(IntPtr hWnd, uint message, IntPtr wParam, IntPtr lParam)
{
    API.PAINTSTRUCT paintStruct;
    IntPtr hDC;
    switch (message)
    {
        case API.WM_PAINT:
        {
            hDC = API.BeginPaint(hWnd, out paintStruct);
            API.EndPaint(hWnd, ref paintStruct);
        }
        break;

        case API.WM_DESTROY:
        {
            API.PostQuitMessage(0);
        }
        break;

        default:
            return API.DefWindowProc(hWnd, message, wParam, lParam);
    }
    return IntPtr.Zero;
}
}
class API
{
    public const int CW_USEDEFAULT = unchecked((int)0x800);
    public const uint WM_DESTROY = 0x0002;
    public const uint WM_PAINT = 0x000F;

    [DllImport("user32.dll")]
    public static extern IntPtr LoadCursor(IntPtr hInstance, int lpCursorName);
    [DllImport("user32.dll")]
    [return: MarshalAs(UnmanagedType.U2)]
    public static extern short RegisterClassEx([In] ref WNDCLASSEX lpwcx);
    [DllImport("user32.dll")]
    [return: MarshalAs(UnmanagedType.Bool)]
    public static extern bool ShowWindow(IntPtr hWnd, ShowWindowCommands nCmdShow);
    [DllImport("user32.dll", SetLastError = true)]
    public static extern IntPtr CreateWindowEx(WindowStylesEx dwExStyle, string lpClassName,
    string lpWindowName,
    WindowStyles dwStyle, int x, int y, int nWidth, int nHeight, IntPtr hWndParent, IntPtr hMenu,
    IntPtr hInstance, IntPtr lParam);
    [DllImport("user32.dll")]

```

```

public static extern bool UpdateWindow(IntPtr hWnd);
[DllImport("user32.dll")]
public static extern int GetMessage(out MSG lpMsg, IntPtr hWnd, uint wMsgFilterMin, uint
wMsgFilterMax);
[DllImport("user32.dll")]
public static extern int TranslateMessage([In] ref MSG lpMsg);
[DllImport("user32.dll")]
public static extern IntPtr DispatchMessage([In] ref MSG lpMsg);
[DllImport("user32.dll")]
public static extern void PostQuitMessage(int nExitCode);
[DllImport("user32.dll")]
public static extern IntPtr DefWindowProc(IntPtr hWnd, uint uMsg, IntPtr wParam, IntPtr lParam);
[DllImport("user32.dll")]
public static extern IntPtr BeginPaint(IntPtr hwnd, out API.PAINTSTRUCT lpPaint);
[DllImport("user32.dll")]
public static extern bool EndPaint(IntPtr hWnd, [In] ref API.PAINTSTRUCT lpPaint);
public delegate IntPtr WNDPROC(IntPtr hWnd, uint uMsg, IntPtr wParam, IntPtr lParam);

public enum WindowStyles : uint
{
    WS_BORDER = 0x800,
    WS_CAPTION = 0xc00,
    WS_CHILD = 0x400,
    WS_CLIPCHILDREN = 0x200,
    WS_CLIPSIBLINGS = 0x400,
    WS_DISABLED = 0x800,
    WS_DLGFRAME = 0x400,
    WS_GROUP = 0x20000,
    WS_HSCROLL = 0x100,
    WS_MAXIMIZE = 0x100,
    WS_MAXIMIZEBOX = 0x10000,
    WS_MINIMIZE = 0x200,
    WS_MINIMIZEBOX = 0x20000,
    WS_OVERLAPPED = 0x0,
    WS_OVERLAPPEDWINDOW = WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU |
WS_SIZEFRAME | WS_MINIMIZEBOX | WS_MAXIMIZEBOX,
    WS_POPUP = 0x800u,
    WS_POPUPWINDOW = WS_POPUP | WS_BORDER | WS_SYSMENU,
    WS_SIZEFRAME = 0x40000,
    WS_SYSMENU = 0x80000,
    WS_TABSTOP = 0x10000,
    WS_VISIBLE = 0x100,
    WS_VSCROLL = 0x200
}

public enum WindowStylesEx : uint
{
    WS_EX_ACCEPTFILES = 0x0010,
    WS_EX_APPWINDOW = 0x00040000,
    WS_EX_CLIENTEDGE = 0x00200,
    WS_EX_COMPOSITED = 0x0200,
    WS_EX_CONTEXTHELP = 0x00400,
    WS_EX_CONTROLPARENT = 0x00010000,
    WS_EX_DLGMODALFRAME = 0x001,

```

```

WS_EX_LAYERED = 0x00080000,
WS_EX_LAYOUTRTL = 0x00400,
WS_EX_LEFT = 0x00,
WS_EX_LEFTSCROLLBAR = 0x00004000,
WS_EX_LTRREADING = 0x00,
WS_EX_MDICHILD = 0x0040,
WS_EX_NOACTIVATE = 0x0800,
WS_EX_NOINHERITLAYOUT = 0x00100,
WS_EX_NOPARENTNOTIFY = 0x004,
WS_EX_OVERLAPPEDWINDOW = WS_EX_WINDOWEDGE | WS_EX_CLIENTEDGE,
WS_EX_PALETTEWINDOW = WS_EX_WINDOWEDGE | WS_EX_TOOLWINDOW |
WS_EX_TOPMOST,
WS_EX_RIGHT = 0x00001000,
WS_EX_RIGHTSCROLLBAR = 0x00,
WS_EXRTLREADING = 0x00002000,
WS_EX_STATICEDGE = 0x00020000,
WS_EX_TOOLWINDOW = 0x0080,
WS_EX_TOPMOST = 0x008,
WS_EX_TRANSPARENT = 0x0020,
WS_EX_WINDOWEDGE = 0x00100
}
public enum ShowWindowCommands : int
{
    Hide = 0,
    Normal = 1,
    ShowMinimized = 2,
    Maximize = 3,
    ShowMaximized = 3,
    ShowNoActivate = 4,
    Show = 5,
    Minimize = 6,
    ShowMinNoActive = 7,
    ShowNA = 8,
    Restore = 9,
    ShowDefault = 10,
    ForceMinimize = 11
}
public enum IDC_STANDARD_CURSORS : int
{
    IDC_ARROW = 32512,
    IDC_IBEAM = 32513,
    IDC_WAIT = 32514,
    IDC_CROSS = 32515,
    IDC_UPARROW = 32516,
    IDC_SIZE = 32640,
    IDC_ICON = 32641,
    IDC_SIZENWSE = 32642,
    IDC_SIZENESW = 32643,
    IDC_SIZEWE = 32644,
    IDC_SIZENS = 32645,
    IDC_SIZEALL = 32646,
    IDC_NO = 32648,
    IDC_HAND = 32649,

```

```

    IDC_APPSTARTING = 32650,
    IDC_HELP = 32651
}
[StructLayout(LayoutKind.Sequential)]

public struct MSG
{
    public IntPtr hwnd;
    public uint message;
    public IntPtr wParam;
    public IntPtr lParam;
    public uint time;
    public POINT pt;
}
[StructLayout(LayoutKind.Sequential)]

public struct WNDCLASSEX
{
    public uint cbSize;
    public CS style;
    public WNDPROC lpfnWndProc;
    public int cbClsExtra;
    public int cbWndExtra;
    public IntPtr hInstance;
    public IntPtr hIcon;
    public IntPtr hCursor;
    public IntPtr hbrBackground;
    public string lpszMenuName;
    public string lpszClassName;
    public IntPtr hIconSm;
}
[StructLayout(LayoutKind.Sequential)]
public struct POINT
{
    int x;
    int y;
}

[StructLayout(LayoutKind.Sequential)]
public struct PAINTSTRUCT
{
    public IntPtr hdc;
    public bool fErase;
    public RECT rcPaint;
    public bool fRestore;
    public bool fIncUpdate;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 32)]
    public byte[] rgbReserved;
}
[Flags]
public enum CS : uint
{
    CS_VREDRAW = 0x0001,

```

```

CS_HREDRAW = 0x0002,
CS_DBLCLKS = 0x0008,
CS_OWNDC = 0x0020,
CS_CLASSDC = 0x0040,
CS_PARENTDC = 0x0080,
CS_NOCLOSE = 0x0200,
CS_SAVEBITS = 0x0800,
CS_BYTEALIGNCLIENT = 0x1000,
CS_BYTEALIGNWINDOW = 0x2000,
CS_GLOBALCLASS = 0x4000,
CS_IME = 0x00010000,
CS_DROPSHADOW = 0x00020000
}

```

[StructLayout(LayoutKind.Sequential)]

```
public struct RECT
```

```
{
    public int Left, Top, Right, Bottom;
```

```
    public RECT(int left, int top, int right, int bottom)
```

```
{
    Left = left;
    Top = top;
    Right = right;
    Bottom = bottom;
}
```

```
    public RECT(System.Drawing.Rectangle r) : this(r.Left, r.Top, r.Right, r.Bottom) { }
```

```
    public int X
```

```
{
    get { return Left; }
    set { Right -= (Left - value); Left = value; }
}
```

```
    public int Y
```

```
{
    get { return Top; }
    set { Bottom -= (Top - value); Top = value; }
}
```

```
    public int Height
```

```
{
    get { return Bottom - Top; }
    set { Bottom = value + Top; }
}
```

```
    public int Width
```

```
{
    get { return Right - Left; }
    set { Right = value + Left; }
}
```

```
public System.Drawing.Point Location
{
    get { return new System.Drawing.Point(Left, Top); }
    set { X = value.X; Y = value.Y; }
}
```

```
public System.Drawing.Size Size
{
    get { return new System.Drawing.Size(Width, Height); }
    set { Width = value.Width; Height = value.Height; }
}
```

```
public static implicit operator System.Drawing.Rectangle(RECT r)
{
    return new System.Drawing.Rectangle(r.Left, r.Top, r.Width, r.Height);
}
```

```
public static implicit operator RECT(System.Drawing.Rectangle r)
{
    return new RECT(r);
}
```

```
public static bool operator ==(RECT r1, RECT r2)
{
    return r1.Equals(r2);
}
```

```
public static bool operator !=(RECT r1, RECT r2)
{
    return !r1.Equals(r2);
}
```

```
public bool Equals(RECT r)
{
    return r.Left == Left && r.Top == Top && r.Right == Right && r.Bottom == Bottom;
}
```

```
public override bool Equals(object obj)
{
    if (obj is RECT)
        return Equals((RECT)obj);
    else if (obj is System.Drawing.Rectangle)
        return Equals(new RECT((System.Drawing.Rectangle)obj));
    return false;
}
```

```
public override int GetHashCode()
{
    return ((System.Drawing.Rectangle)this).GetHashCode();
}
```

```
public override string ToString()
{

```

```
return string.Format(System.Globalization.CultureInfo.CurrentCulture,
"{{Left={0},Top={1},Right={2},Bottom={3}}}", Left, Top, Right, Bottom);
}
}
}
}
```

rekcahcave - August 3, 2013, 4:48 pm

Nice, thanks for the tutorial! I will reference it later!

Vexane - August 3, 2013, 9:34 pm

Book. Marked.

35 Characters

Still 35 Characters

beastmode16 - August 4, 2013, 11:09 am

Apparently, very informative tutorial.. got bookmarked it now! Thanks a lot for this man!

Makaveli The Don - August 11, 2013, 2:39 pm

Great guide. Was good to read. Thanks. ?