

Forum:[Hack Forums](#)

Thread: [What is .NET? What is MFC? What is Win API? Q & A's \(Part-6\)](#)

?si?qqo? - December 27, 2008, 8:15 am

Part One: <https://hackforums.net/showthread.php?tid=39070>

Part Two: <https://hackforums.net/showthread.php?tid=40287>

Part Three: <https://hackforums.net/showthread.php?tid=41526>

Part Four: <https://hackforums.net/showthread.php?tid=41528>

Part Five: <https://hackforums.net/showthread.php?tid=41530>

This thread is talking about .NET - MFC - Windows API in Questions & Answers form...I hope you like it ?

Q: What is .NET Framework? Explain in details please.

A: The .NET framework is the layer which is installed on an Operating system (like windows) which holds everything you need to run a .NET application. It contains the libraries which provide functionality to, for example, open files, read xml, and so on. This gives you the opportunity to write .NET applications, compile them and run them with only the .NET framework installed. Much like java does when you install the jvm (java virtual machine).

This is where visual studio comes in. VS.NET needs the .NET framework. VS.NET won't function without it. Whenever you install VS, the install procedure will also install the .NET framework. VS.NET makes it much easier to create applications on the .NET framework by generating source code when you i.e. build a windows or web form. You could say that VS.NET is a very fancy notepad for creating .NET applications in any way, shape or form.

Executables from IDE other than .NET do not require the end-user to have the .NET Framework...

==*==

Q: Do All Windows Come With .NET Framework?

A: It depends on the package of the operating system you are using. Most windows(xp, vista) has none builtin .net enabled. Applications developed on dot net provides more security during execution since the level of permission is set in user's machine configuration files. Other advantages are rapid application development, lesser code, JITted code is optimized for the environment, code safety, once built you can run your application in any platform that has dotnet framework installed (as of now windows and some flavours of linux).

You can ship dotnet framework redistributable along with your application so that your clients can install them before running your application, similar to how java applications bundle jvm and vb applications bundle vbruntime. I guess C runtime comes along with OS that is why you dont need to install them seperately. When you package your setup, there is a way to include the runtime.

By more security I meant user's machines are secured and also since the application is run by dotnet framework, its execution is validated.

==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*

Q: Do application .exe which coded under different IDE (other than .NET framework) work successfully without in need of .NET Framework?

A: An application that made by .NET needs .NET - it doesn't matter what IDE it is built with, whether notepad, visual studio, etc. If it is built without using .NET components, then it won't need .NET.

If you want to use your application with new technology experience you should use vb.net or c_sharp.net or any .net windows form application, but be aware must have a framework installed during installation on running the program in other OS.

==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==

Q: What is a native code?

A: Native code is machine code executed directly by the CPU. An example of native code, is the pure standard

C++ that does not contain any non-standard headers. Native code can be done in all types of programming environments.

==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==

Q: Are applications built on .NET is considered native code, explain?

A: No because .NET Framework has its own ready-made class libraries, etc.

.NET has same concept of JVM (Java Virtual Machine), so application written on .NET is not compiled directly to the machine. All .NET source code (of any language) is compiled to IL (Intermediate Language) during development. The IL is then converted to machine code at the point where the software is installed, or (more commonly) IL is then converted to machine code at run-time by a Just-In-Time (JIT) compiler. IL is similar Bytecode in Java.

Instead of compiling down to a bytecode-like structure, .NET languages are compiled into an Intermediate Language (in fact, it's called just that--IL). IL is a human readable language that looks sorta like an object-oriented version of assembler (sort of a high level machine language)...

In other words, whenever you compile your code in .NET, regardless the language you choose, it is converted to Intermediate Language (IL) which is also known as Microsoft Intermediate Language or Common Intermediate Language. What this means is that instead of code being compiled in actual machine code (code that the CPU would run), it is instead compiled into a high-level generic language. This means that whatever language you write your code in, when you compile it with .NET it will become IL. Since all languages eventually get compiled into the intermediate language, the runtime (Common Language Runtime - CLR) only has to worry about understanding and working with the intermediate language.

When you compile your code and it is translated to the intermediate language it is then simply stored in an assembly. When that assembly is used the CLR picks up that code and compiles it on-the-fly for the specific machine that is running the code. CLR only compiles the individual method that is being invoked. This kind of on-the-fly compilation, referred to as Just-in-Time Compilation, only happens once per method call. If the next time the method is called, no compilation occurs because the CLR has already compiled that code.

Some explanation has been taken from

<http://www.ondotnet.com/pub/a/dotnet/2005/09/06/what-is-dotnet.html#heading1>

==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==

Q: What Is The Main Difference Between .Net Paltform And .Net Framework.?

A: .NET Framework is a set of library/ddls provided by Microsoft which can be used to develop programs.
.NET Platform is something like C#.NET, vb.NET, C++.NET, where as .NET Framework is a defined set of dlls which can be used by any .NET Compatible program (C#.NET, vb.NET, C++.NET, etc).

In short, the .NET Framework is a DLL used to implement the .NET platform. Otherwise, the platform e.g. application built in C#.NET wouldn't execute if .NET Framework is not installed in the machine because .NET framework got set of libraries/dlls that make the C#.NET executable application work.

Got it now? Hopefully yes!

==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==

Q: What is Win32 API?

A: Win32 is the native user mode programming interface for 32-bit (and supported by 64-bit) Windows versions. In other words, it is a set of code (API) for Windows OS that lets you create GUI Application for Windows OS. All these GUI programs you have in your Windows OS have been created by Win32 API.

==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==

Q: Isn't .NET & Other IDE's such as Borland VCL are the one that create GUI applications?! Any idea?

A: Well, you seem like you got lots of confusions....
All these IDE's which lets your create GUI's, are actually a tool that makes it easy for you to build GUI's. These tools are making Win32 API automatically for you behind the scenes, so you don't go through the hassle and write the Win32 API by yourself...You can do it! Many tutorials out there teaches you how to code Win32 API manually. Because Win32 API requires lots of coding just to make a regular simple GUI, people (developers - companies - whatever) have developed new ways (IDE's) to create GUI in easy ways, .NET & Borland VCL are good examples. However, at the end it all goes back to the core which is Win32 API....

==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==

Q: What is Win32 API, i mean what is "API"?

A: An application programming interface (API) is a set of functions, procedures, methods, classes or protocols that an operating system provides, to support requests made by computer programs.

Every OS has its own set of API...Linux has its own API, Windows has its own API, and so on...

==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==

Q: Since we make GUI's on Windows by coding win32 API. Win32 API has been constructed and

written and which language then?!

A: All of the DLLs in the Win32 API, and most of the kernel-level structures are implemented (written) in C code. In the end, it doesn't really matter how these API's are implemented. All that matters is how you can make use of the functionality, and that's exactly what an API describes : how to make use of the functionality provided by it.

==*

Q: is it a must to use Win32 API to create applications?

A: not really! you do not need Win32 API if you want to make applications for consoles. But you see! when you code in console, you won't have any hassles on how to make a window, text boxes, buttons, etc. Therefore, coding in win32 API will make you code alot of stuff for the GUI.

==*

Q: Does Win32 API Has Its Own Logic? Does It Differ Alot from Console Programming?!

A: Technically speaking: It's not a lot different. It groups functions that serve similar purposes. The basic difference probably is that it targets different aspects such as e.g. the Windows GUI, in GUI things are addressed differently - such as files, handling sockets, etc.

Handling sockets is modeled closely to the standard socket programming scheme. You will find some special things that are covered there also, but in general. So not really a huge different, its just the gui thing & how to handle things in the window...

==*

Q: To code a network application, you need to use Winsock API! Now whats this? Whats the different between Win32 API (Windows API) & Winsock API? Explain

A: Windows API is simply a general interface to several system provided functions. There is API for GUI's, API for sockets, API for dealing with file systems & devices, etc.

The Win32 API (or more correctly the Windows API) is the superset of all standard API's provided by the Windows operating system. Note that the Winsock API is a (small) part of the Windows API. In other words, the Windows API is a collection of all the core API's provided by the Windows operating system. The Winsock API is a (small) part of that.

Any application that wants to make use of functionality provided by the Windows OS should make use of the Windows API.

The Winsock library can be used in any kind of Windows application, whether it's a console application, GUI, etc. In other words, if you want to make GUI network application you will beed to use Winsock API with the GUI API, simply winsock library and gui library. There's nothing in the Winsock library that would interfere with GUI library. To fully understand it, you need to read and do some examples.

Rough example:

```
#include <windows.h>    //Windows.h is for the Win32 API in C++
```

```
#include <winsock2.h>    //Winsock2.h is our winsock library
.
.
.

if(bind(sServer, (LPSOCKADDR)&sinServer, sizeof(sinServer)) == SOCKET_ERROR){
/* failed at starting server /
  MessageBox(NULL, (LPCTSTR)"Could not bind the server!", (LPCTSTR)"Server::Error",
  MB_OK|MB_ICONERROR);
  WSACleanup();
  return 0;
}
.
.
.
```

windows.h is a Windows-specific header file which contains declarations for all the function in the Windows API. It defines a very large number of Windows specific functions that can be used in C, C++, etc. There are a number of child header files that are automatically included with windows.h. Some child headers of windows.h are included automatically, others are not.

windows.h may include any of the following header files automatically:

- except.h Exception handling
- stdarg.h variable-argument functions (standard C header)
- windef.h various macros and types
- winnt.h various macros and types (for Windows NT)
- basetsd.h various types
- guiddef.h the GUID type
- ctype.h character classification (standard C header)
- string.h strings and buffers (standard C header)
- winbase.h kernel32.dll: kernel services
- winerror.h Windows error codes
- wingdi.h GDI (Graphics Device Interface)
- winuser.h user32.dll: user services
- winnls.h NLS (Native Language Support)
- wincon.h console services
- winver.h version information
- winreg.h Windows registry
- winnetwk.h WNet (Windows Networking)
- winsvc.h Windows services and the SCM (Service Control Manager)
- imm.h IME (Input Method Editor)

Extra includes:

- cderr.h CommDlgExtendedError function error codes
- commdlg.h Common Dialog Boxes
- dde.h DDE (Dynamic Data Exchange)
- ddeml.h DDE Management Library
- dlgs.h various constants for Common Dialog Boxes
- lzexpand.h LZ (Lempel-Ziv) compression/decompression
- mmsystem.h Windows Multimedia
- nb30.h NetBIOS
- rpc.h RPC (Remote procedure call)
- shellapi.h Windows Shell API

wincrypt.h - Cryptographic API
winperf.h - Performance monitoring
winresrc.h - used in resources
winsock.h - Winsock (Windows Sockets), version 1.1
winsock2.h - Winsock (Windows Sockets), version 2
winspool.h - Print Spooler
ole2.h - OLE (Object Linking and Embedding)
objbase.h - COM (Component Object Model)
oleauto.h - OLE Automation
oleclid.h - various GUID definitions

The functionality provided by the Windows API can be grouped into eight categories:

Base Services: provide access to windows system, things like file systems, devices, processes and threads, and error handling.

Advanced Services: Included are things like the Windows registry, shutdown/restart the system (or abort), start/stop/create a Windows service, etc.

Graphics Device Interface: Provide the functionality for outputting graphical content to monitors, printers and other output devices.

User Interface: Provides the functionality to create GUI stuff, such as windows, buttons, receive mouse and keyboard input, etc.

Common Dialog Box Library: Provides applications the standard dialog boxes for opening and saving files, choosing color and font, etc. grouped under the User Interface category.

Common Control Library: Include things like status bars, progress bars, toolbars and tabs. It is grouped under the User Interface category.

Windows Shell: allows applications to access the functionality provided by the operating system shell, as well as change and enhance it. grouped under the User Interface category.

Network Services: Gives networking capabilities of the operating system. Its sub-components include NetBIOS, Winsock, NetDDE, RPC and many others.

Extra Info: The Windows API mostly concerns itself with the interaction between the operating system and an application. For communication between the different Windows applications among themselves, you need to look at Component Object Model (COM).

Visit Wikipedia website to learn more.

==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==

Q: Is Win32 API Used Only for Creating Application that has GUI Facilities?

A: No. Windows API is a set of programming interfaces. It not only includes GUI interfaces, it provides access to: file system, network, threads, graphics, etc. It is the ONLY way to talk to Windows...You can use the API to make any number of programs, including those without a forms interface (background process). Win32 API simply refers to windows specific APIs. You are free to use the APIs in console or windows applications.

==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==

Q: "Many background processes that you never actually see are also built using the Win32 API." Isn't console programming is the one that is usually used to make background processes?

A: It can be a console or it can be a windows application which does not have a visible window. If you do not CreateWindow()/ShowWindow(SW_HIDE), then this will make a background process that you never actually see.

Win32 API could start with WinMain() in case of windows forms or main() in case of console application. In other words, to use WinMain() you need to include the header <windows.h>, but main() does not need <windows.h>....

In some specific cases you may need to call windows API, for example if you want to access the system information, this does not need a GUI application but you have to call a WIN32 API to get the information. For example, when you code for console & in a sudden you decided to access system information, at that point you will have to include <windows.h> so you get some functionality that allows you to do that regardless whether you are using console or GUI.

start by main() for application on consoles. WinMain() for making application on windows forms.

==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==

Q: "Every application that runs in Windows, uses WinAPI in one way or another" is this true?

A: Not really. Windows API is required only to talk to windows OS specific functions. As i said you call <windows.h> when you need to. For example, the basic "hello world" program does not call any windows API. It depends on your need to call a windows api or not.

==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==

Q: There is something called MFC (Microsoft Foundation Class Library) came after Win32 API! What is MFC?

A: MFC is a library that wraps portions of the Windows API in C++ classes, and thus allows a more object oriented way of interacting with the API. MFC made things easier. Wrapper generally refers to a type of packaging.

==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==

Q: What are the Differences between Win32 and .NET programming?

- A: - All .NET languages are Object-Oriented but Win32 isn't.
- .NET eventually calls Win32 functions! but Win32 functions call system services directly.
 - .NET provides a managed environment in which most of the resource management is automatic but, as using Win32, you should take care of resources (memory, handles, etc...)
 - A .NET application needs .NET framework on target machine, but a Win32 application only needs Windows itself (assuming you correctly handle version differences).

= * = * = * = * = * = * = * = * = * = * = * = * = * = * = * = * = * = *

A: You dont need the .net frame work to run application consists of pure Win32 API & MFC.

A: No. You cant create MFC applicaition using Dev C++. MFC is microsoft classes, which is available only with microsoft visual C++. Play around Dev-C++ and see what kind of libraries & facilities it has!

~~= * = * = * = * = * = * = * = * = * = * = * = * = * = * = * =~~

A: Managed code is computer program code that executes under the management of a virtual machine (such as .NET, or JVM), unlike unmanaged code, which is executed directly by the computer's CPU. The benefits of managed code include programmer convenience and enhanced security guarantees. In unmanaged code the memory allocation, type safety, security, etc needs to be taken care of by the developer. This makes unmanaged code prone to memory leaks like buffer overruns and pointer overrides and so forth.

Note: .NET is basically created for helping you making managed code. However, you can make code unmanaged codes on .NET too ... You need to search about it...

—*—

A: MDI - Multi document interface (allows you to open several windows)
SDI - Single document interface (single window to deal with)

—*—

A: Nothing to do with threading. (isn't this similar to threading? - looks like it but it isn't, make one doc busy and the app will not respond for the other docs.)

==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==

Q: Can C++.NET Let Hackers Write Malicious Code, Such as Trojans?

A: You can write a malicious program with .Net (any language), but it's not preferred. First off, typically you need to make sure the user's computer supports .Net; second the code access security provided by .Net makes it even more difficult to effectively penetrate or compromise a system; There have been a few viruses written in .Net, but they're all low risks.

Further details: http://en.wikipedia.org/wiki/Code_Access_Security

==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==

Q: Do you have any extra advices?

A: Regarding GUI's hm...It is better to get the basics on Win32 API (GUI Library) play around with it. Then move to MFC. Finally, move to .NET... Win32 API low level - MFC medium level - .NET high level...

To create strong milicious code, such as trojans, malware, worms, it is recommended to use Win32 API & MFC....
.NET is good for non-milicious codes. Win32 API & MFC for unamanaged code. .NET basically is for managed code.

So start with Win32 API -> MFC -> .NET

At the point, when you use MFC or .NET, you will have better understading on how things works behind the scenes since you already learned Win32 API. This can help you in many ways...

Note: i'm not supporting or encouraging you to code any milicious code! this is just for illustration purposes...

==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==

Q: Is there a way i can convert my MFC source code to .NET easily?

A: <http://www.dudelabs.com/>

==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==

Q: Is MFC Considered Managed or Unmanaged Code?

A: MFC is unmanaged code. It does not need .NET framework...

"Unmanaged" does not necessary mean "unsafe", as well as "managed" does not mean "good" or "safe" or whatever. These words (managed/unmanaged) were used for marketing purposes, just to make people think that the new tecnology (.Net) is better & safer then the old one (Win32API). Managed application still can eventually crash or behave oddly.

MFC is a library that wraps portions of the Windows API in C++ classes, so it is unmanaged since Win32 API is unmanaged & MFC just wraps Windows API to make it easier for you. MFC pretty safe, however - if you use it properly. You just need to have a good plan to make code works safely (safely means not causing any memory leaks by mistake, etc)! Its all about you. MFC & Win32 API

put you in full control, unlike .NET tries to handle things for you to reduce unsafety as much as possible.

I hope this general overview kinda helped you! Go and read further for more details, if you want to!...

==*==

Q. Is the code below considered Win API code?

```
#include <iostream.h>

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

A. This is C++ standard. It is not Win API code. C++ is higher level than WinApi. WinAPI is what microsoft has offered to developers so they can write applications for windows OS. All Windows programs must interact with the Windows API regardless of the language.

==*==

Check out the next tutorial! Feel free to ask...

Drop a comment

Wolskie - December 27, 2008, 5:14 pm
Nice easy to understand tutorial. Its brillent!