Places

Home

Tutorials

Windows

Windows FAQ

DirectX

DirectX and .NET

Fourier Analysis

Projects

m-math

Embedded T$_{E}$X

QuickRes NT

WebMessage

Information

Links

Personal

My CV

Optical Smartcards

Robot Ducks

MIDI Chord Recorder

Windows Programming FAQ

This FAQ is intended to answer many of the questions that newcomers to and intermediate users of the Windows API often encounter.

Contents

This FAQ

Can I mirror this FAQ/the tutorials/any other part of this site?

You may mirror the FAQ or the Windows programming tutorials, provided:

- You first send me an e-mail notifying me of your intention to do so.
- If I request that you remove such a mirror, you do so promptly ('promptly' being 'within seven days').
- You make all reasonable efforts to ensure that the mirror is kept up to date, such that changes are promptly propagated.
- If you are mirroring a tutorial, that tutorial is not a 'work in progress'.

Can I add a link to your site on mine?

Yes. I would appreciate a notification.

Can I add a link to my site on yours?

No. I don't have the time. Please don't send me links.

All Windows programs

What other FAQs are available?

- http://www.iseran.com/Win32/FAQ/faq.htm
- http://www.winprog.org/faq/

What message boards are available?

- http://www.cprogramming.com/
- http://www.programmersheaven.com/

What tutorials are available?

- Chemanuel has links to tutorials on his Web site: http://www.geocities.com/josempadron/eng/tutorials.htm.
- My Win32 API and DirectX tutorial: http://www.sunlightd.com/Windows/.

What books are available?

There are many, many books written on this subject. Make sure you buy a book directed at Win32, not Win16 (Windows 3.*x*), as many older books are.

The classic book is 'Programming Windows' by Charles Petzold (Microsoft Press), now in its 5th edition.

Petzold has a new book, 'Programming Windows with C#'. Should I buy this instead?

This is my opinion only.

Win32 is the current environment of choice. Everything you can do in Windows can be done with Win32, so I would recommend 'Programming Windows, 5th edition' at the moment, for beginning Windows programmers.

C# and .NET are well worth learning, but for the next couple of years, Win32 programming will be almost inescapable. I would move onto .NET only after having a solid grounding in Win32. In two or three years time, I may well revise that opinion - but not yet.

How do I create a Windows program?

You will need a Windows compiler. The most widely used is Microsoft Visual C++. There are a number of free compilers around, including Borland C++ 5.5, mingw32, lcc/Win32, and DJGPP with the RSX library.

These compiler systems will *only* create Windows applications, and so any 'DOS-like' applications are actually console applications (see below).

How do I port a DOS program to Windows?

There are numerous libraries available to help you port your program to Windows.

Most console applications should run fine; the following table lists equivalents of commonly-found functions. All these functions require windows.h.

Function	**Equivalent**
delay	Sleep
textcolor/textattr	SetConsoleTextAttribute (see below)
clrscr	See below
gotoxy	SetConsoleCursorPosition (see below)

If your application is written for the BGI (Borland Graphics Interface), Konstantin Knizhnik has written a BGI emulation layer for Windows called 'WinBGI'. This will allow you to run your BGI programs, mostly unmodified, in Windows.

If your application is written for the Allegro graphics library, there is a version prepared for Windows.

Why does my compiler not accept 'union REGS regs;'?

union REGS is a declaration used by old DOS compilers for their interrupt-calling functions. You cannot call interrupts in Windows, so any code using 'union REGS' should be rewritten using Windows services.

How do I run in Mode 13h?

Mode 13h is a BIOS video mode that describes 320x200 in 8-bit colour. Windows does not use the BIOS for video functions, nor does it allow direct access to the frame buffer in the same way. If you want to run full-screen in this video mode (or any other video mode) you should use DirectDraw. See above for tutorials.

How do I run another program?

Windows users are spoiled for choice in methods for running other applications. The methods include _spawn/_exec, system, WinExec, CreateProcess, ShellExecute and ShellExecuteEx.

_spawn/_exec

The _spawn and _exec families of functions are found in Windows compiler libraries for portability from UNIX. There is a variety of functions, including the ability to pass either a list or array of parameters, to search in the PATH or otherwise, and whether or not to pass an *environment* (a set of environment variables). Using _spawn, you can also specify whether you want the new process to close your program (_P_OVERLAY), to wait until the new process is finished (_P_WAIT) or for the two to run concurrently (_P_NOWAIT).

_spawn and _exec are useful for instances in which you have simple requirements for running the program, don't want the overhead of the Windows header file, or are interested primarily in portability.

system

system is an ANSI standard function. It operates by loading the command processor (COMMAND.COM in Windows 9x, CMD.EXE in Windows NT/2000) and passing it the command line you specify.

system is rarely, if ever, useful. The overhead of loading the command processor is large, and it should be ignored for this reason.

WinExec

WinExec is a function provided for compatibility with 16-bit Windows, and for that reason should not be used.

CreateProcess

All processes in Windows are eventually created by this function. It provides a bewildering array of parameters which cover every eventuality, but most of these can be set to default values. CreateProcess is the fastest way of loading a process, and is the preferred method.

ShellExecute/ShellExecuteEx

These two functions perform a slightly different task. While they will load and run a program (passing it on to CreateProcess) they also have the capability of loading the application associated with a document file. This is useful, for example, in starting a Web browser, when ShellExecute need only be passed the URL.

What is Unicode, and should I use it?

The Unicode consortium was formed to solve the problem of character encoding. The standard ANSI character set only covers the major Western European languages - ASCII only covers English. In their own words:

Unicode provides a unique number for every character,
no matter what the platform,
no matter what the program,
no matter what the language.

A noble statement... but useless without application support. Fortunately for non-Western people across the world (billions of them!) Unicode support is widespread. For our purposes, it's most important to notice that it's the native character set of Windows NT.

In Windows NT, you can choose either ANSI (8-bit) or Unicode (16-bit) characters. However, all Windows APIs use Unicode internally. Therefore, when you call any Windows API function (or a C library function that calls the Windows API) it has to translate the input strings from ANSI to Unicode, and back on return. This is slower than just using Unicode natively - so it's important that Windows NT applications should run on Unicode.

What's the catch?

The catch is that Windows 95 and 98 do not support Unicode APIs. This is due to their 16-bit Windows heritage. So if you call the Unicode APIs directly, you will lose the ability to run on Windows 9x.

The way around this is to make a program that can be compiled for either Unicode or ANSI without loss of functionality. You could use a lot of conditional code; fortunately, much of it is done for you. You can make your applications Unicode-compliant by:

- using Unicode in your data structures;
- using the _t forms of C library functions; and
- using the _T macro for inline strings.

You can compile for Unicode by defining the symbols UNICODE *and* _UNICODE in your compiler definitions, and for ANSI by not doing so.

Windows console mode programs

What are 'Windows console mode' programs?

All 32-bit Windows applications are able to access the same resources. They are all able to open windows, message boxes and dialogs, and use all Win32 API functions.

Win32 allows Windows applications to have a 'console', a text-based window similar to that provided to DOS applications. This console can be created using the AllocConsole function. Each application may have only one console open at any one time.

By setting a flag in the executable file, Win32 will create a console at startup for the application. This is the *only* difference, from Win32's point of view, between console and GUI applications. This flag is set by the Microsoft linker by specifying '/subsystem:console' rather than '/subsystem:windows'. The program is still a Win32 application, and still has access to all Win32 facilities. It is *not* a DOS application - DOS is not involved anywhere.

For compatibility's sake, Win32 C/C++ compilers use different start-up functions for console and GUI applications. Console applications are expected to have a standard 'main' function, while GUI applications are expected to have a 'WinMain' function that is very similar to that used by 16-bit Windows. However, even if an application uses a 'main' function and runs in a console window, it is still a Win32 application - not a DOS application.

Where can I find more information about Windows console mode?

The MSDN (Microsoft Developer Network) Library contains the definitive reference to programming Windows console mode applications.

http://msdn.microsoft.com/library/default.asp?URL=/library/psdk/winbase/conchar_4p6c.htm

How do I clear the screen?

Clearing the console screen is not a standard function provided by Windows. The Microsoft Knowledge Base contains an article that describes how to write a function that clears the screen:

```
#include <windows.h>
void clrscr()
{
    COORD coordScreen = { 0, 0 };
    DWORD cCharsWritten;
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    DWORD dwConSize;
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);

    GetConsoleScreenBufferInfo(hConsole, &csbi);
    dwConSize = csbi.dwSize.X * csbi.dwSize.Y;
    FillConsoleOutputCharacter(hConsole, TEXT(' '), dwConSize,
        coordScreen, &cCharsWritten);
    GetConsoleScreenBufferInfo(hConsole, &csbi);
    FillConsoleOutputAttribute(hConsole, csbi.wAttributes, dwConSize,
        coordScreen, &cCharsWritten);
    SetConsoleCursorPosition(hConsole, coordScreen);
}
```

This function first obtains a handle to the console window. It retrieves its size, then fills the console window with spaces. Finally, it sets the cursor position to the top left hand corner of the window.

How do I move the cursor?

Cursor movement is accomplished by using the SetConsoleCursorPosition function. It is easy to write a function which performs the same as the old DOS function gotoxy:

```
#include <windows.h>
void gotoxy(int x, int y)
{
    COORD coord;
    coord.X = x; coord.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}
```

How do I change the colour of text?

Use the SetConsoleTextAttribute function. This takes a handle to the console (use GetStdHandle(STD_OUTPUT_HANDLE) to obtain it) and a colour value. You can obtain different colours through different combinations of the constants FOREGROUND_BLUE, FOREGROUND_GREEN, FOREGROUND_RED, FOREGROUND_INTENSITY, BACKGROUND_BLUE, BACKGROUND_GREEN, BACKGROUND_RED, and BACKGROUND_INTENSITY.

Colour	Value
Black	0
Red	RED
Green	GREEN
Blue	BLUE
Cyan	GREEN | BLUE
Magenta	RED | BLUE
Yellow	RED | GREEN
Dark Grey	RED | GREEN | BLUE
Light Grey	INTENSITY
Light Red	INTENSITY | RED
Light Green	INTENSITY | GREEN
Light Blue	INTENSITY | BLUE
Light Cyan	INTENSITY | GREEN | BLUE
Light Magenta	INTENSITY | RED | BLUE
Light Yellow	INTENSITY | RED | GREEN
	INTENSITY | RED | GREEN | BLUE

For example, to use red text on a black background:

```
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
   FOREGROUND_RED);
```

Light blue text on a dark grey background:

```
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
   FOREGROUND_BLUE | FOREGROUND_INTENSITY |
   BACKGROUND_RED | BACKGROUND_GREEN | BACKGROUND_BLUE);
```

Can I change the size of the console window?

Yes. Use the SetConsoleWindowInfo function.

Can I make a console window start up full screen programmatically?

No. There is currently no documented way to switch between windowed and full-screen mode in console applications.

Can I draw lines, circles, points, graphics in a console window?

No. There is no documented way to draw in a console window. You have a few options:

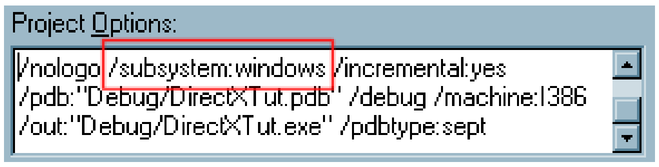
- Write a Windows GUI application (see above for tutorials).
- You can use the WinBGI library (see above) to port DOS graphics code that uses the BGI (Borland Graphics Interface).
- You can use the Win32 port of the Allegro library (see above) to port DOS graphics applications developed with DJGPP and Allegro.

Why am I getting 'unresolved external '_WinMain@16' when I compile my console application?

You are attempting to compile a console application as a GUI application.

- On Visual C++, you can create a console application project by selecting 'Win32 Console Application' from the New Project list. Alternatively, you can change an existing project to a console application by changing your project options.

  Change your linker settings (Project - Settings - Link tab - Project Options) to /subsystem:console (it should have had /subsystem:windows):



```
/nologo /subsystem:windows /incremental:yes
/pdb:"Debug/DirectXTut.pdb" /debug /machine:I386
/out:"Debug/DirectXTut.exe" /pdbtype:sept
```

Also, add the define _CONSOLE in the Preprocessor section of the C/C++ tab.

- On Borland C++ 5.5 (the command-line compiler), remove the command line options '-W' and '-laa'.

Windows GUI programs

Why do Windows GUI programs have so much code in them?

Don't forget that Windows GUI programs actually do a *lot*. Typically, you will have movable, sizable windows with full mouse support, icons, cursors, portability between wildly different systems...

My tutorial illustrates that if you want to do simple programs, you need not have hundreds of lines of code - but those hundreds of lines of code are still useful, since they offer you so much flexibility.

Should I learn MFC, and when?

MFC (Microsoft Foundation Classes) is a class library developed by Microsoft for Visual C++, but also available with Borland C++. It is useful for creating standard Windows GUI applications of four types:

- Simple dialog-based applications
- Complex OLE controls
- Single-window applications, especially based on files or database records
- Multiple document applications

MFC offers a jump start on these applications with its AppWizard. It also offers many facilities that are very useful in an application: templated list, array and map classes; a flexible and fast string class; support for ActiveX controls; and a large library of third-party code that is very easy to integrate into your application (CodeGuru: http://www.codeguru.com/).

It is usually best to become familiar with the Windows API before beginning MFC. Reading Petzold will give you a firm foundation on which you will be able to pick up MFC easily.

Where can I learn MFC?

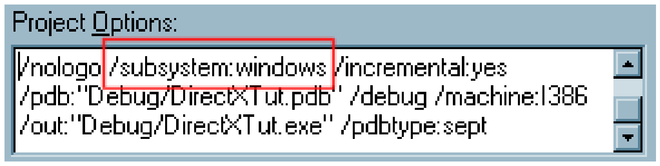
The best place to learn MFC is the Scribble tutorial, which comes with Visual C++. The tutorial (along with several others which demonstrate other aspects of MFC) are in the MSDN Library at http://msdn.microsoft.com/library/default.asp?URL=/library/devprods/vs6/visualc/vctutor/tutorhm.htm.

Why am I getting 'unresolved external _main' when I compile my GUI application?

You are attempting to compile a GUI application as a console application.

- On Visual C++, you can create a GUI application project by selecting 'Win32 Application' from the New Project list. Alternatively, you can change an existing project to a GUI application by changing your project options.

Change your linker settings (Project - Settings - Link tab - Project Options) to /subsystem:windows (it should have had /subsystem:console):



```
/nologo /subsystem:windows /incremental:yes
/pdb:"Debug/DirectXTut.pdb" /debug /machine:I386
/out:"Debug/DirectXTut.exe" /pdbtype:sept
```

Also, remove the define _CONSOLE in the Preprocessor section of the C/C++ tab.

- In Borland C++ 5.5 (the command line compiler), you will have to specify the command line options '-W -laa'. Make sure you get the case correct (capital W).

How do I 'skin' a program?

Programs that appear to have drastically changed the layout and appearance of their windows (e.g. WinAmp, Windows Media Player 7) are usually simply windows with no borders or captions. These programs then draw their entire interface in the client area of the window.

In order to inform Windows which parts of the 'skinned' windows represent the various components of the window (borders, captions and so on), the windows handle the WM_NCHITTEST message. This message is sent whenever Windows needs to know what window component is at which point. If the window decides the point lies over the caption bar, it returns HTCAPTION; if over the bottom resizing border, it returns HTBOTTOM, and so on.

Transparent areas present some difficulty, since versions of Windows prior to Windows 2000 required the clumsy method of *window regions* to support transparency. The programs create a GDI *region* from combinations of rectangles, polygons, rounded rectangles and ellipses. Windows 2000 introduced the concept of *layered* windows, which allow the use of transparency and alpha-blending in a normal window, with much improved performance. To read more about layered windows, see this article, or search for "layered windows" in the MSDN Library.

Where do I get a resource editor?

I get dozens of e-mails asking about resource editors. The best remains the one in Visual Studio, which is very reasonably priced. A free resource editor is available as part of LCC-Win32.