

A Programmer's Tour of Microsoft Windows XP and Vista

Michael A. Covington
Artificial Intelligence Center
The University of Georgia

December 2006; revised May 2009

Introduction

This hastily written document is an introduction to Windows XP and Vista from the programmer's point of view. It is intended for people who are new to programming and programmers who are new to Windows.

NOTE: Windows Server 2003 and 2008 are closely related to Windows XP. Windows 7 is closely related to Windows Vista. Everything said here applies to them; it should be easy for you to check the details.

What is a computer?

A computer consists of:

- A **central processing unit**, which is a complex digital circuit that constantly fetches instruction codes from memory and does whatever the codes specify.
- Random-access memory (**RAM**), where data is stored while being processed.
- Read-only memory (**ROM**), a small amount of permanently recorded memory that the computer uses to start up – important because RAM goes blank when the computer is turned off, but ROM does not.
- One or more **disk drives** for storing data permanently. As you will see, the functioning of Windows revolves around the disk drive.

- Various **input-output devices** such as a keyboard, a video display, a printer, a network adapter, and so forth.

What is a PC?

By “PC” we mean any computer descended from the IBM Personal Computer (1982). It has been a very long descent! The Intel Pentium is completely compatible with software written for the earliest IBM PC, but today’s typical PCs have about 1000 times as much RAM and run about 1000 times as fast.

The original PC operating system was DOS (Microsoft Disk Operating System). In the mid-1980s, Microsoft began adding a mouse-and-windows user interface to DOS, and this led to **two Windows lineages**. Windows versions 1.0, 2.0, 3.0, 3.1, 95, 98, and ME basically consist of an ever-growing Windows system built on top of DOS. Windows NT, 2000, XP, 2003, and Vista are a separate operating system not built on DOS but able to emulate it.

There are two main kinds of Windows software, **Win32** and **.NET** (“dot net”), relying on two major subsystems of the Windows operating system.

Win32 has been the core of Windows since version 3.0 and takes advantage of the full memory capacity of the Pentium, freeing it from the limitations of DOS, which, after all, was designed for a much smaller computer. Win32 was programmed in C and C++ and relies on the programmer to make extensive use of *pointers* (numbers giving the memory location of data). Working with pointers is error-prone, and (like UNIX, which works the same way), Win32 leads to somewhat crash-prone and insecure software.

The **.NET Framework** is a new component of Windows that was provided as an update to Windows 2000, XP, and 2003 and is built into Vista. It provides a different way for programs to communicate with the operating system so that programmers do not have to use pointers, and so that programming errors are much easier to detect. (In .NET Framework, errors almost always produce a meaningful message rather than an uninterpretable crash.) Along with .NET Framework, Microsoft is promoting a set of new programming languages, among them C# (a derivative of Java).

What does Windows do for you?

Windows (of whatever version) is a large computer program that serves as an **operating system** for the PC – that is, it has the ability to load and run other programs, not just itself. To be precise, Windows can find a program on the disk

drive, copy it into memory, start the CPU running it, and regain control when the program finishes.

Windows also provides services to other programs that you are running. When Microsoft Word wants to put up an “open file” dialog box, for instance, it calls on Windows to do this for it. On a more mundane level, Windows keeps track of the low-level details of the disk drives, the video system, etc., so that programs can ask for files by name, write on the screen in standard fonts and colors, and so forth.

Multitasking

Windows is a **multitasking** operating system. That is, it can run many programs at once. It does this by **time-slicing** the CPU's attention. Each program runs for about a millisecond and then the CPU switches to the next one in rotation.

Most programs spend a lot of time waiting for input from the keyboard, data from a disk drive or network, or the availability of a printer or other output device. Thus, time-slicing can be very efficient. If a program is not ready to proceed when its turn comes up, the CPU moves immediately to the next program.

Multitasking doesn't just enable you to run more than one application program. It also enables Windows to perform routine tasks “in the background” while you work.

Multiuser

Windows is also a **multiuser** operating system. Whenever you use the computer, you **log in** as a specific “account.”

One account, called **Owner** in Windows XP Home and **Administrator** in most other versions, is privileged. It has the right to install software and make major changes in the machine configuration.

Other accounts do not have these privileges. It is better to do most of your work in a non-privileged account in order to limit the damage that could be done by a virus or an accidental mishap.

Each account has a separate desktop and “My Documents” folder. We'll get to the details of this soon.

People who learned to program in the 1980s keep forgetting that Windows is multiuser. Much academic software is unfortunately written so that it only works if you are logged in as Administrator or if you installed it under your own account. For example, we recently had this problem with *EndNote*.

With properly designed software, Administrator can install software for use by everyone, and can place items on the desktop and in the Start Menu to be seen by everyone – but not all software is properly designed. Doing things right doesn't take any extra effort, just some awareness of how computers work in the 21st Century.

Virtual memory

Windows is a **virtual memory** operating system. That is, it automatically uses disk space as a substitute for extra RAM, swapping information into and out of real memory as needed. Windows has a large, hidden “swap file” that enables it to function as if it had much more RAM than it actually does.

When you try to run a program with large memory requirements, such as a video editor, your computer won't stop in its tracks if you exceed the available RAM. Instead, the computer will slow down as swapping takes place.

How a computer starts up (“boots up”)

The process of turning a computer on is called **bootstrapping** (“booting”) because the computer has to “pull itself up by its own bootstraps.” Here's how it's done.

- (1) When the Pentium is powered on, it automatically starts fetching and executing instructions from a particular memory address. That address points to ROM BIOS (the read-only-memory basic-input-output-system), which is a small, permanently recorded program that tells the computer what to do next.
- (2) The program in the ROM BIOS tells the Pentium to read a particular sector of the main disk drive (the “boot sector”) and execute the instructions there.
- (3) In the boot sector of a properly formatted Windows hard disk is a program that carries out more of the startup process. It gives the Pentium the ability to read the rest of the disk, copy the appropriate part of Windows into RAM, and begin executing it.

How a disk drive works

A disk drive consists of several rotating platters coated with the same kind of magnetic material as audio tapes.

Information is recorded on the disk drive by magnetizing patterns on its surface, much the way a tape recorder records sound. Any region of the disk can be demagnetized and re-magnetized with different data at any time. Thus, until it wears out (which takes many years), a disk drive is completely reusable.

Every disk is divided into **sectors** which are numbered. Each sector holds a block of information. (Actually, Windows uses the sectors in groups called **clusters**, rather than singly, but I'll keep things simple here. For gory details, look up *FAT32* in Wikipedia.)

Besides sectors filled with data, the disk drive has to have two special regions:

- The **allocation table** indicates which sectors are in use and how they are chained together (more about this shortly).
- The **directory** (more precisely, the **root directory**) lists the names of all the files (with a big exception to be noted later) and indicates what sector each file starts in.

When you delete (erase) a file, its name is removed from the directory, and its sectors are marked as available in the allocation table, but *the data are not erased from the disk* until the space is actually needed for something else. That is why “unerase” utilities work.

Windows also has a second safeguard against accidental deletion. When you delete a file, normally it is moved into a special directory called the “recycle bin” from which it can be completely retrieved.

Chaining sectors together

The sectors occupied by a file need not be adjacent (contiguous). A file can start in sector 30, continue in sector 15, then in sector 45, then in sector 12, and finally end in sector 99. The allocation table keeps track of this.

Naturally, the computer will run faster if large, frequently-used files occupy contiguous sectors. You can run **defragmentation** programs to rearrange the contents of the disk so that this is so.

Multiple directories (folders)

One of the great inventions of computer science is the concept of **multiple directories**, introduced with the UNIX operating system in 1978 and adopted by DOS 2.0 in 1983.

In graphical user interfaces, directories are called **folders**. A directory and a folder are absolutely the same thing.

The basic idea is that *the files on a disk aren't all listed in a single list*. In addition to the “root directory” (which you can think of as the root of a tree structure), there can be *any number of additional directories* which are actually special files containing directory information about more files.

Thus, you can organize your disk contents into a hierarchy of folders within folders. Crucially, *each folder is not a section of the disk drive*; it's just a place where files are listed. The files can be physically located anywhere on the disk.

Paths

In Windows, every disk drive has a drive letter, such as C:, and every directory has a name. The **path** to a file consists of its drive letter, one or more directory names, and the filename, like this:

C:\temp\myfile.txt

That means “file *myfile.txt* in directory *temp* on drive C.” Or:

C:\Program Files\Covington Innovations\TIP\Library\Eval1.tip-v

That is: “file *Eval1.tip-v* in folder *Library* which is in folder *TIP* which is in folder *Covington Innovations* which is in *Program Files* which is in the root directory of drive C.”

Word to the wise:

Paths are joined by \ (backslash), not / (forward slash).

Traditionally, on an early PC, A: and B: were two diskette drives and C: was the first hard disk.

File types and their significance

An important fact about computers is that **data means nothing apart from the programs that process it**. To the computer, a file is nothing more than a series of bits (ones and zeroes) in groups of eight (called bytes).

To ask what these bits *mean* is rather like asking what the barrel of ink means at the print shop. Computers are not reflectively conscious and do not “understand” anything.

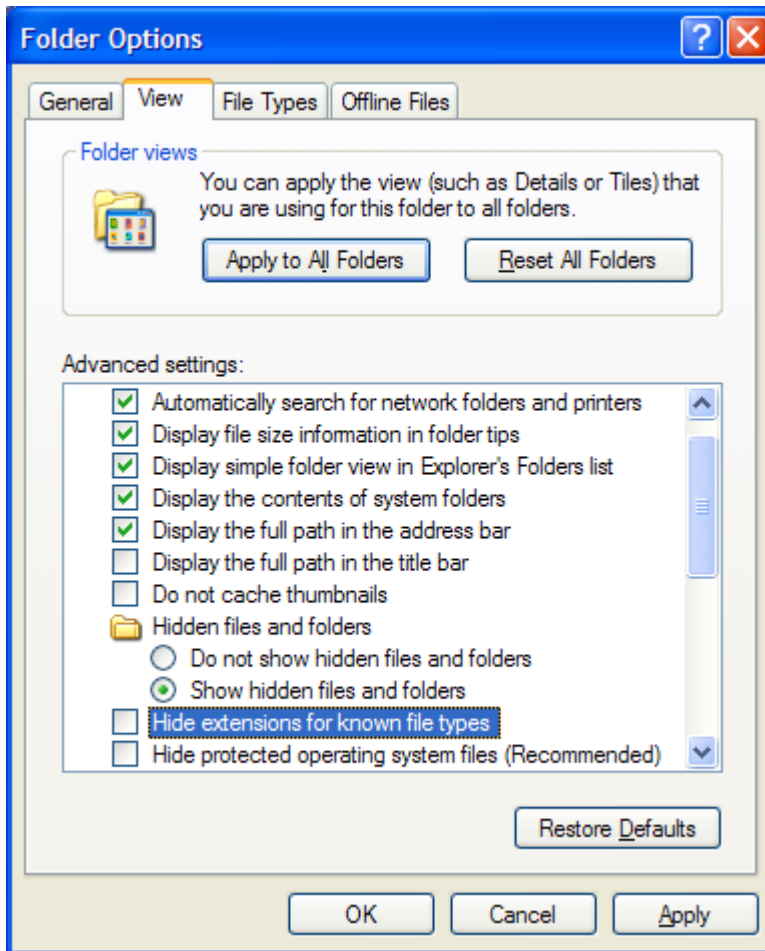
There are some standard codes, such as ASCII for simple text files, but in general, *any programmer can choose to represent information any way he wants* and write programs that use this representation. That is why rival word processors or graphics programs do not open each other’s files (or if they do, they’re unreliable, because they’re relying on programmer B guessing how programmer A did things).

Filename extensions

Normally, Windows filenames end with an *extension*, which is a series of letters after a period. For example, *myfile.txt* has *txt* as its extension.

Contrary to widespread belief, the extension is not limited to three letters. (It was under DOS.)

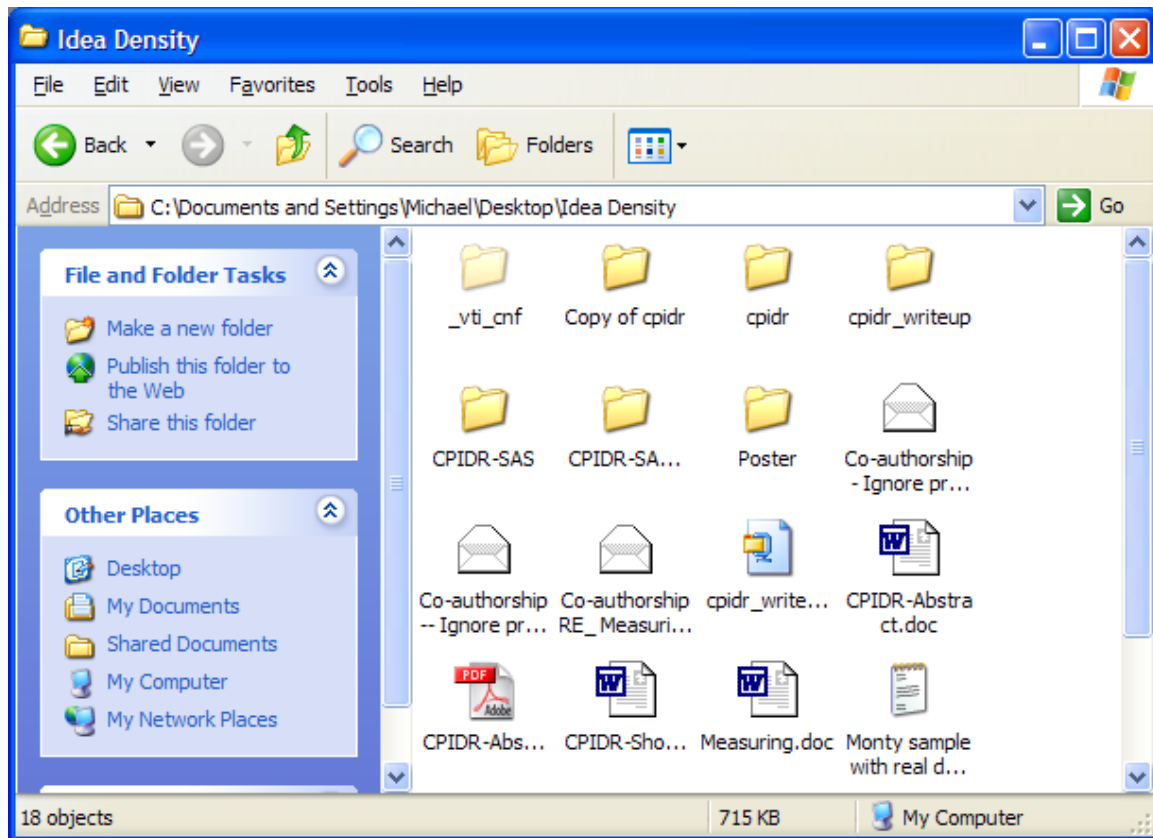
To see extensions, you must tell Windows to make them visible. Open any folder and go to Tools, Folder Options, View, and uncheck “Hide extensions...”:



In their somewhat limited wisdom, the people at Microsoft decided that ordinary Windows users shouldn't have to look at extensions. We are not ordinary Windows users.

Icons

Instead of looking at extensions, ordinary people distinguish files by **icons** (images). Here's an example:



Here you see several folders, several e-mail messages, a ZIP file, some Word documents, a PDF document, and a text document.

How does Windows know which icons to display? Simple. *Windows looks at the extensions.* Windows has a database called the Registry that contains, among many other things, a table specifying what icon to display for each type of file, as identified by extension.

There are three exceptions. Program files (.exe files) contain their own icons within them. Icon files (.ico files) display as themselves (of course). The icon for a whole disk (such as a CD or a removable disk drive) is specified in its *autorun.inf* file.

Changing a filename extension

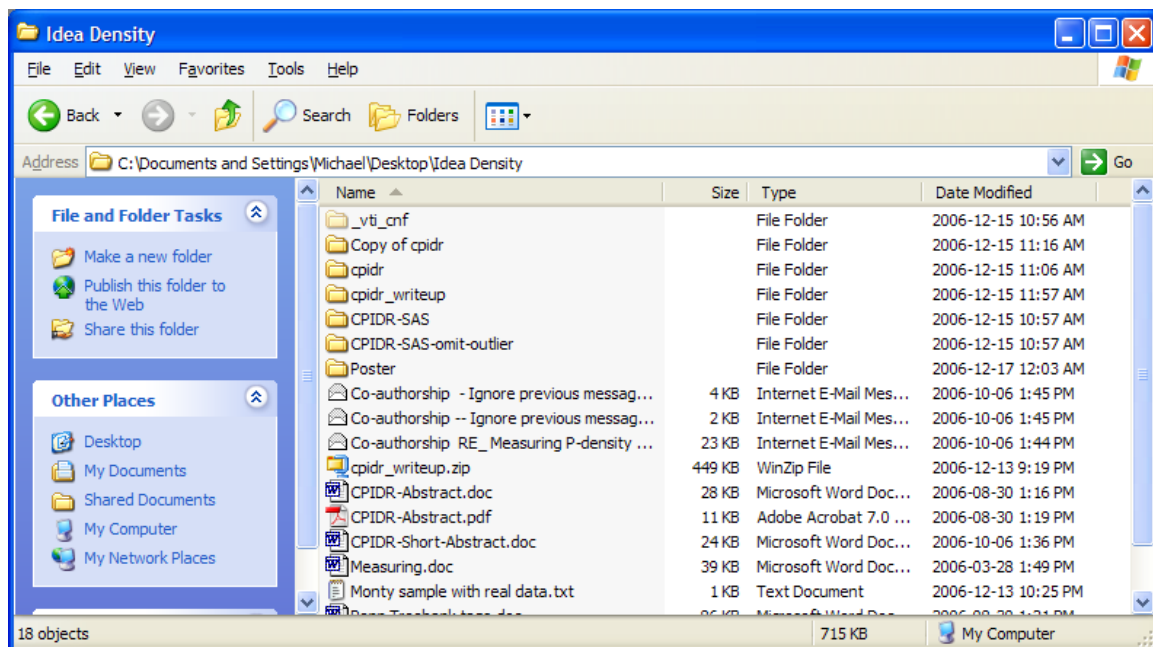
You are of course at liberty to rename a file in such a way as to change its extension.

When you do, this does not change the type of the file. It merely labels it falsely. If you rename *myfile.txt* as *myfile.bmp* you do not thereby turn it into a graphics file. You turn it into something that Windows will *try* to handle as a graphics file – but these attempts will be futile.

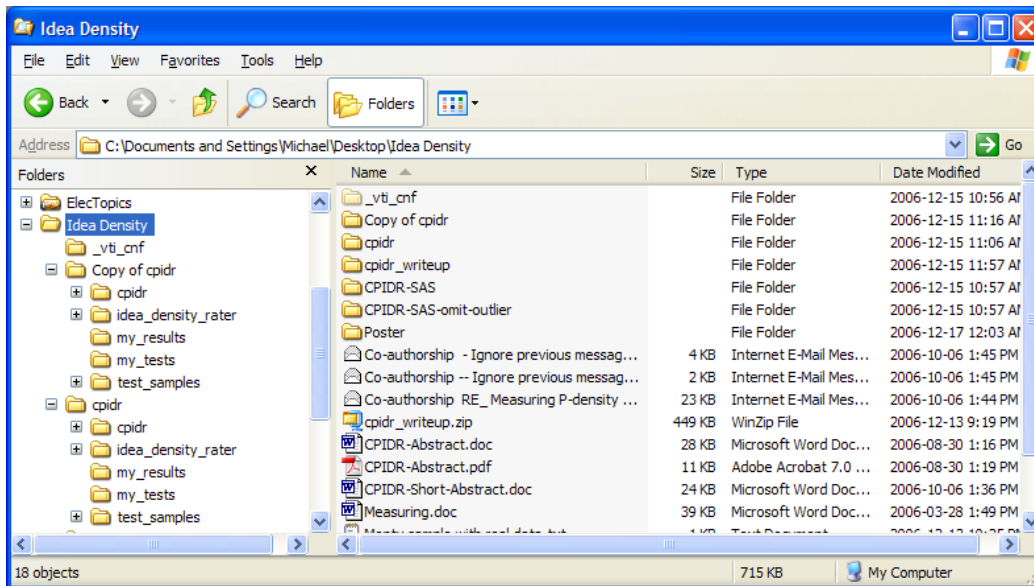
What you see on your screen

Folders

You've just seen what a folder looks like when opened. As an experienced Windows user, you know already that you can go to "View" and choose a different view of it, such as Details:



Or you can right-click on the folder and choose Explore instead of Open. Then you get a handy hierarchical view of it that (at least conceptually) goes all the way back to Windows 1.0:



In Vista, “Explore” is what normally happens when you click on a folder.

What happens when you click on a file

If you click on an executable program (an .exe file), Windows will run the program.

If you click on any other type of file, what happens depends on what information Windows has stored about how to “open” that type of file. Typically, text files open with a text editor such as Notepad, graphics files open with the appropriate program, and so on.

This information is stored in the Registry, which we’ll get to. You can customize it by right-clicking on the file and choosing “Open with...” Beware of doing this erroneously.

Shortcuts

A special kind of file that is very important in Windows is called a **shortcut**. This is a file that immediately redirects Windows to another file elsewhere.

By using shortcuts, you can put the same file or folder in several places without making multiple copies of it. For instance, an executable program can reside in

C:\Program Files (which is where programs are normally kept), and there can be copies of it on the desktop and in the Start Menu.

Internally, shortcuts have extension *.lnk*, but you do not see the extension even when Windows has been set to display extensions.

The desktop

The desktop is a folder

Fundamentally, your desktop is a folder, probably located at:

C:\Documents and Settings\username\Desktop (Windows XP)
C:\Users\username\Desktop (Windows Vista)

and the things you see on the desktop are the folders and files that it contains.

(In the paths above, *username* stands for whatever your username might be. Other locations are possible; your Windows system may be configured differently. By opening a folder on the desktop and then going up step by step, you can find out exactly where it is.)

The desktop also contains shared items

On your desktop you will also see any items that are located in

C:\Documents and Settings\All Users\Desktop (Windows XP)
C:\Users\Public\Desktop (Windows Vista)

This provides a convenient way to place things on everybody's desktop, especially shortcuts to installed software.

The Start Menu

Like the desktop, the Start Menu (accessed from the Start button) is a combination of two folders, one for the individual user and one for all users. In Windows XP, these are, respectively:

C:\Documents and Settings\username\Start Menu
C:\Documents and Settings\All Users\Start Menu

In Vista the names are much longer:

C:\Users\username\AppData\Roaming\Microsoft\Windows\Start Menu
C:\ProgramData\Microsoft\Windows\Start Menu

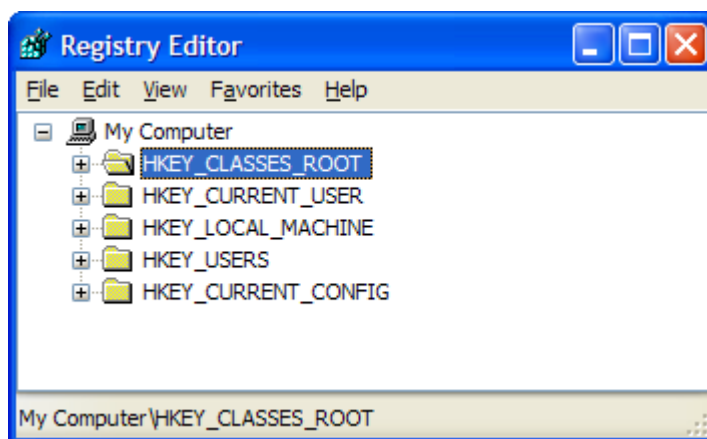
Compared to the desktop, much more of the Start Menu is in the All Users portion; also, almost everything on the Start Menu is a shortcut rather than an actual file or folder.

If you are logged in with sufficient privileges, you can right-click on Start and choose “Open” or “Open All Users” to access to the individual and the shared portions of the Start Menu as a folder.

The Registry

Windows maintains a huge database called the **registry** containing information about how Windows and your software are configured.

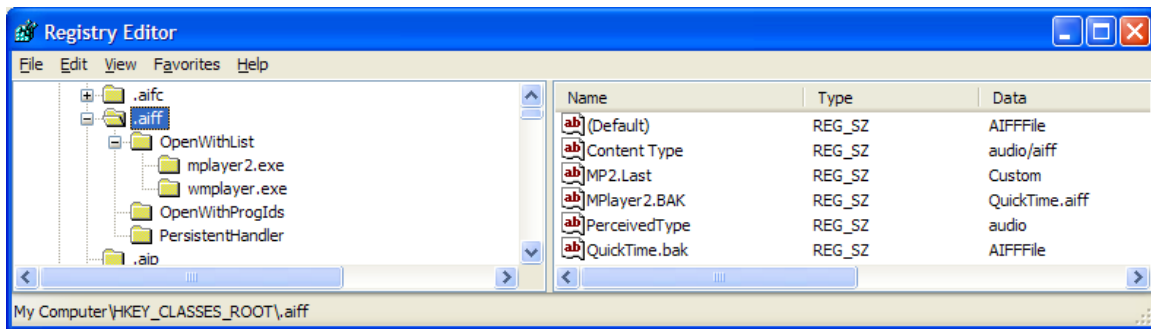
You can explore the Registry by going to Start, Run..., and typing **regedit**. You'll see something like this:



(Vista users, see p. 19 to see how to get to the “Run...” prompt.)

Please do not make any changes to the Registry unless you're 100% sure you know what you're doing! It's easy to ruin a Windows installation by making random changes. **When you make a change in Regedit, it takes effect immediately** – there is no “Save” button; what you type is saved right then.

The section called HKEY_CLASSES_ROOT is the easiest to understand. It specifies the meanings of filename extensions. Often these consist simply of references to another item in the same list, so you have to follow a chain of them to understand an entry, but here is a particularly simple one:



It says that files with extension *.aiff* are audio files that can be opened with *mplayer2.exe* or *wmplayer.exe* (two media players).

The content and workings of the Registry are *far* beyond the scope of this document.

How programming is done

Programming languages

If you wanted to, you could program the Pentium by giving it the actual binary codes that control it. Doing so would be extremely tedious. Instead, we always use a programming language.

Computers do not “understand” programming languages the way humans understand human languages. Rather, a programming language is a notation that enables us to describe computations precisely. Our descriptions are then translated into codes that control the computer.

Compilers vs. interpreters

Suppose you’ve written a program in a language such as C, C#, Java, or Basic. There are two basic ways to get the computer to run the program.

You can use a **compiler**, which is a computer program that translates your program into a file of actual machine code (an *.exe* file) which the computer can run directly.

Or you can use an **interpreter**, which is a computer program that reads your program, line by line, and does whatever it calls for.

Traditionally, C and C++ are compiled; Python and Prolog are interpreted. The advantage of compilation is that the end product runs faster (since it's in pure machine code) and can run on any computer that runs Windows (you don't have to give people the interpreter along with the program).

Java and C# introduce a third way of running programs called **just-in-time (JIT) compilation**. When you compile your program, it is translated, not into machine code, but into *intermediate code*, which resembles machine code but does not match it exactly. Then, at run time, it is translated into machine code.

This approach has two advantages. First, intermediate code is more concise than machine code, resulting in smaller files. Second, if you end up using a different kind of CPU in the future, only the JIT compiler ("JITter") will need to be changed, and this can be taken care of as part of the operating system.

JIT compilation is not time-consuming. In fact, compiling a concise intermediate-code file into machine code often takes less time than would be taken simply to read the larger machine-code file from the disk.

You will notice that the .exe files produced by a C# compiler are much shorter than those that come from a C or C++ compiler (assuming you're compiling a relatively large program). The reason is that the .exe files from the C# compiler rely on the .NET Framework to do JIT compilation from intermediate code to machine code when the user runs the program.

How software is installed

To "install" software on a computer means to make it usable, and that may mean any number of things.

If the software consists of a single .exe file, then this file can simply be placed on the machine in any convenient place, and the user can run it by clicking on it. I encourage you to program this way whenever possible.

If the software consists of a large number of files, then the programmer should create a **deployment project** (an installer, a *setup.exe* or *.msi* file) to do the following:

- Put the software and related files in an appropriate folder under C:\Program Files
- Create shortcuts to the software from the Start Menu and/or desktop

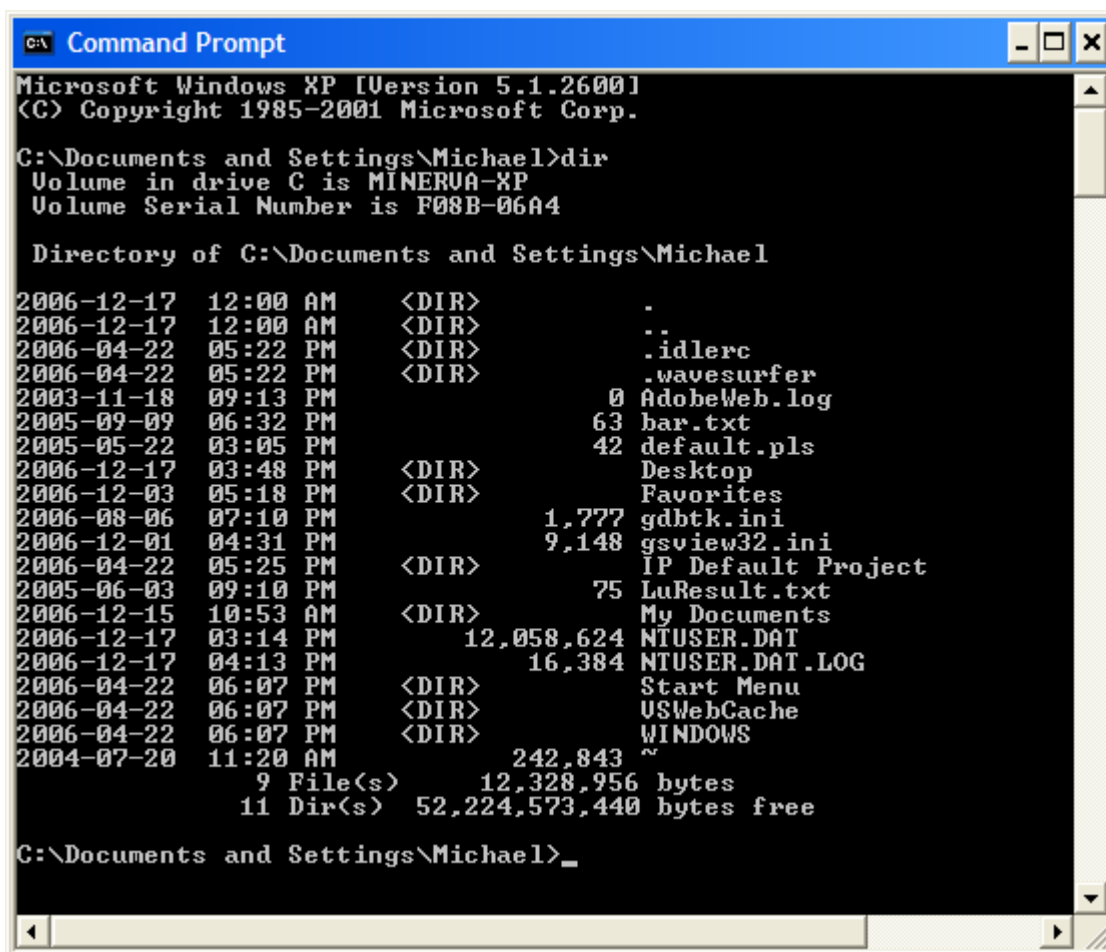
- Store information in the Registry so the software can be uninstalled from Control Panel

In Visual Studio, this process is largely automated.

It is very awkward if installing a software package makes changes to the computer which are not easily undone.

The command prompt (console)

You can type commands to Windows at the command prompt (console), like this:



```

C:\ Command Prompt
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Michael>dir
Volume in drive C is MINERVA-XP
Volume Serial Number is F08B-06A4

Directory of C:\Documents and Settings\Michael

2006-12-17  12:00 AM    <DIR>          .
2006-12-17  12:00 AM    <DIR>          ..
2006-04-22  05:22 PM    <DIR>          .idlerc
2006-04-22  05:22 PM    <DIR>          .wavesurfer
2003-11-18  09:13 PM             0 AdobeWeb.log
2005-09-09  06:32 PM             63 bar.txt
2005-05-22  03:05 PM             42 default.pls
2006-12-17  03:48 PM    <DIR>          Desktop
2006-12-03  05:18 PM    <DIR>          Favorites
2006-08-06  07:10 PM             1,777 gdbtk.ini
2006-12-01  04:31 PM             9,148 gsview32.ini
2006-04-22  05:25 PM    <DIR>          IP Default Project
2005-06-03  09:10 PM             75 LuResult.txt
2006-12-15  10:53 AM    <DIR>          My Documents
2006-12-17  03:14 PM      12,058,624 NTUSER.DAT
2006-12-17  04:13 PM      16,384 NTUSER.DAT.LOG
2006-04-22  06:07 PM    <DIR>          Start Menu
2006-04-22  06:07 PM    <DIR>          USWebCache
2006-04-22  06:07 PM    <DIR>          WINDOWS
2004-07-20  11:20 AM      242,843 ~
           9 File(s)      12,328,956 bytes
          11 Dir(s)    52,224,573,440 bytes free

C:\Documents and Settings\Michael>_

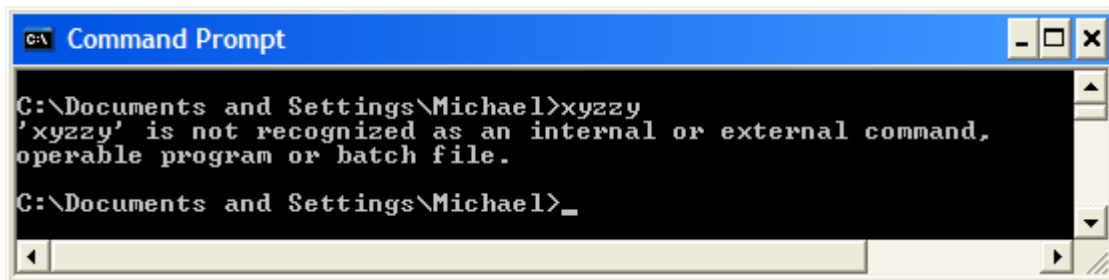
```

Here I've asked for the contents of a directory.

The most important thing to know about the command prompt is that **it is not DOS**. It looks like the screen of an early DOS-based PC, but you're actually talking to Windows, with all its capabilities.

The full use of the command prompt is beyond the scope of this document. However, note that:

- You can still create programs for the command prompt (“console applications”). Many beginning programming courses have the students do this, although I advocate getting away from this.
- You are always “in” a particular directory (the “current directory”), which you can change with the **cd** command (e.g., **cd c:\temp**). In the example above, the current directory is **C:\Documents and Settings\Michael**.
- When you type a command, Windows goes through a specific process to interpret it. Suppose for example you type **xyzzzy** (which is not a command as far as I know). The following things will happen:
 - Windows will check whether **xyzzzy** is built into the console system (like **dir**). It is not.
 - Windows will then look for a program named **xyzzzy** (such as *xyzzzy.exe* or *xyzzzy.bat*) in the current directory. There is none.
 - Windows will then do the same thing, in order, in all the directories that are listed in the search path (which we’ll define shortly).
 - Finally Windows will report that there is no such command:



```
C:\> Command Prompt
C:\Documents and Settings\Michael>xyzzzy
'xyzzzy' is not recognized as an internal or external command,
operable program or batch file.
C:\Documents and Settings\Michael>_
```

To get to a command prompt, go to Start, Programs, Accessories, or go to Run... and type **cmd**. (Vista users see p. 19 concerning “Run...”.)

If you want to see a DOS command prompt (and re-live the 1980s), go to Run... and type **command**. This is not the same as **cmd**.

Network testing from the command prompt

One of the most useful things you can do at the command prompt is test your network connection using traditional commands that have changed little since UNIX days. Here are some examples of commands you can type:

ipconfig /all

Display the status of all your network connections.

ipconfig /renew

Release and renew your DHCP number (i.e., “repair” your connection).

ping address

Send a test packet to the address and wait for a reply. Most addresses don't answer pings. If you have a Linksys router, try:

ping 192.168.1.1

and your router will reply.

nslookup address

Look up the IP number from the name, or vice versa. This tests whether you have a working nameserver, particularly if you look up an address you've never used before. For example, type

nslookup www.yale.edu

and you should get the IP number of a computer at Yale. This does not test whether you can actually communicate with the computer at Yale.

net view

net view \\machine

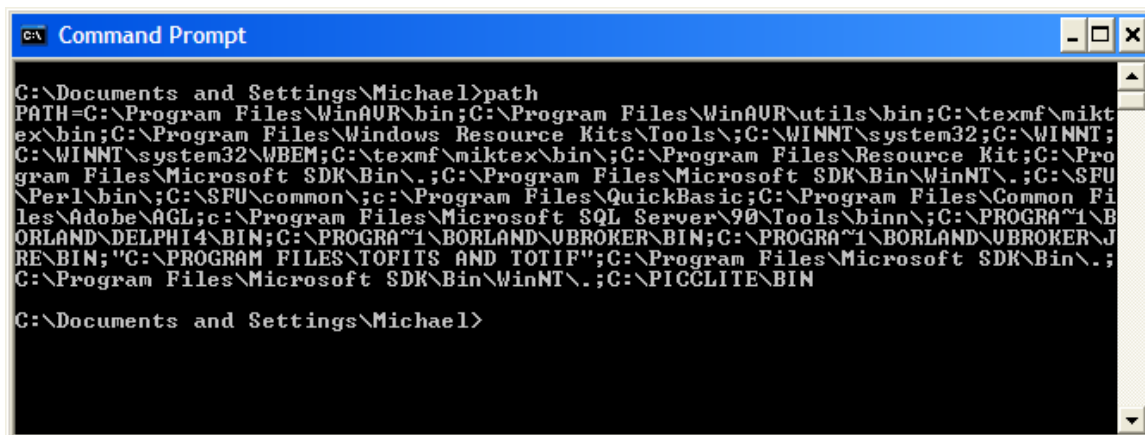
View your local Windows network (workgroup).

.BAT files

You can create a text file containing several command-prompt commands, give it a name ending in *.cmd* or *.bat*, and run it by double-clicking on it. These are not as powerful as UNIX shell scripts, but they are capable of doing useful work and you will encounter them from time to time. (Several powerful scripting languages are provided, including VBScript and, in Vista, PowerShell.)

The search path

The set of directories in which Windows looks for commands can be seen by typing **path** at a command prompt:



```

C:\Documents and Settings\Michael>path
PATH=C:\Program Files\WinAUR\bin;C:\Program Files\WinAUR\utils\bin;C:\texmf\mikt
ex\bin;C:\Program Files\Windows Resource Kits\Tools\;C:\WINNT\system32;C:\WINNT;
C:\WINNT\system32\WBEM;C:\texmf\miktex\bin\;C:\Program Files\Resource Kit;C:\Pro
gram Files\Microsoft SDK\Bin\.;C:\Program Files\Microsoft SDK\Bin\WinNT\.;C:\SFU
\Perl\bin\;C:\SFU\common;c:\Program Files\QuickBasic;C:\Program Files\Common Fi
les\Adobe\AGL;c:\Program Files\Microsoft SQL Server\90\Tools\bin\;C:\PROGRA~1\B
ORLAND\DELPHI4\BIN;C:\PROGRA~1\BORLAND\UBROKER\BIN;C:\PROGRA~1\BORLAND\UBROKER\J
RE\BIN;"C:\PROGRAM FILES\TOFITS AND TOTIF";C:\Program Files\Microsoft SDK\Bin\.;
C:\Program Files\Microsoft SDK\Bin\WinNT\.;C:\PICCLITE\BIN
C:\Documents and Settings\Michael>

```

Yours is almost certainly smaller than this one.

The Run... item on the Start Menu

As an alternative to using the command prompt, you can type a single command at the Run... item on the Start Menu.

Of course, if the command merely writes its output on the console, you won't see it because the console will close as soon as the program finishes.

In Windows Vista, the Run... item does not appear on the Start Menu unless you enable it. Right-click on the start button, choose Properites, Customize, and check Run Command.

Environment variables

Following a UNIX and DOS tradition, much of the configuration information about your computer is stored in a place much more accessible than the registry, specifically a set of **environment variables** that can be retrieved by name. You will probably use these in programs.

To see your environment variables, go to a command prompt and type **set**. Here's a rather copious example:

```

C:\Documents and Settings\Michael>set
ALLUSERSPROFILE=C:\Documents and Settings\All Users
APPDATA=C:\Documents and Settings\Michael\Application Data
BaseName=C:\Program Files\Microsoft SDK\Include\BROffice.Mak
Bkoffice=C:\Program Files\Microsoft SDK\
CLASSPATH=C:\PROGRAM~1\Borland\ubroker\lib\ubcpp.jar
CommonProgramFiles=C:\Program Files\Common Files
COMPUTERNAME=MINERUA
ComSpec=C:\WINNT\system32\cmd.exe
DISPLAY=localhost:0.0
EDITOR=vi
FP_NO_HOST_CHECK=NO
HOMEDRIVE=C:
HOMEPATH=\Documents and Settings\Michael
INCLUDE=C:\Program Files\Microsoft SDK\Include\
INETSDK=C:\Program Files\Microsoft SDK\
INTERIX_COMPILERDIR=/dev/fs/C/Program Files/Microsoft Visual Studio .NET/Uc7
INTERIX_ROOT=/dev/fs/C/SFU/
INTERIX_ROOT_WIN=C:\SFU\
LD_LIBRARY_PATH=/usr/lib:/usr/X11R6/lib
LIB=C:\Program Files\Microsoft SDK\Lib\
LOGONSERVER=\MINERUA
MSSdk=C:\Program Files\Microsoft SDK\
Mstools=C:\Program Files\Microsoft SDK\
NUMBER_OF_PROCESSORS=1
OPENNT_ROOT=/dev/fs/C/SFU/
OS=Windows_NT
Path=C:\Program Files\WinAUR\bin;C:\Program Files\WinAUR\utils\bin;C:\texmf\mikt
ex\bin;C:\Program Files\Windows Resource Kits\Tools\;C:\WINNT\system32;C:\WINNT;
C:\WINNT\system32\WBEM;C:\texmf\miktex\bin;C:\Program Files\Resource Kit;C:\Pro
gram Files\Microsoft SDK\Bin\;C:\Program Files\Microsoft SDK\Bin\WinNT\;C:\SFU
\Perl\bin;C:\SFU\common\c;\Program Files\QuickBasic;C:\Program Files\Common Fi
les\Adobe\AGL;c:\Program Files\Microsoft SQL Server\90\Tools\bin\;C:\PROGRAM~1\B
ORLAND\DELPHI4\BIN;C:\PROGRAM~1\BORLAND\UBROKER\BIN;C:\PROGRAM~1\BORLAND\UBROKER\J
RE\BIN;C:\PROGRAM FILES\TORITS AND TOTIP";C:\Program Files\Microsoft SDK\Bin\.;
C:\Program Files\Microsoft SDK\Bin\WinNT\.;C:\PICCLITE\BIN
PATHEXT=.COM;.EXE;.BAT;.CMD;.UBS;.UBE;.JS;.JSE;.WSF;.WSH
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_IDENTIFIER=x86 Family 15 Model 2 Stepping 7, GenuineIntel
PROCESSOR_LEVEL=15
PROCESSOR_REVISION=0207
ProgramFiles=C:\Program Files
PROMPT=$P$G
SESSIONNAME=Console
SFUDIR=C:\SFU\
SFUDIR_INTERIX=/dev/fs/C/SFU/
SystemDrive=C:
SystemRoot=C:\WINNT
TEMP=C:\DOCUME~2\Michael\LOCALS~1\Temp
TMP=C:\DOCUME~2\Michael\LOCALS~1\Temp
USERDOMAIN=MINERUA
USERNAME=Michael
USERPROFILE=C:\Documents and Settings\Michael
US71COMINTOOLS=C:\Program Files\Microsoft Visual Studio .NET 2003\Common7\Tools\
US80COMINTOOLS=C:\Program Files\Microsoft Visual Studio 8\Common7\Tools\
windir=C:\WINNT
XAPPLRESDIR=/usr/X11R6/lib/X11/app-defaults
XCMSDB=/usr/X11R6/lib/X11/xcms.txt
XKEYSYMDB=/usr/X11R6/lib/X11/XKeysymDB
XNLSPATH=/usr/X11R6/lib/X11/locale

C:\Documents and Settings\Michael>_

```

Most of these *should* puzzle you, but notice such things as **windir** (where Windows is installed), **ALLUSERSPROFILE**, **COMPUTERNAME**, **HOMEPATH**, **USERNAME**, and **USERPROFILE**. They provide a quick way for a program to find out where it's running and how Windows is set up.