Windows Programming

en.wikibooks.org

December 29, 2013

Contents

1 Windows System Architecture

1.1 History

Windows was originally a 16-bit graphical layer for MS-DOS that was written by Microsoft. As it grew, it gained the ability to handle 32-bit programs and eventually became totally 32-bit when Windows NT and 2000 came out. After Windows 95, Microsoft began to remove dependencies on DOS and finally fully implemented the separation in Windows 2000. Windows has many advanced features as well as many platform specific problems. It possesses an Application Programming Interface that consists of thousands of mostly undocumented GUI functions as well as having varying degrees of MS-DOS compatibility. Additionally, with the advent of NT (New Technology), Windows relies completely on the NT kernel instead of its MS-DOS subsystem, the NT kernel is capable of emulating the necessary DOS functionality. In addition to the NT kernel, Microsoft has also introduced many API wrappers, such as the MFCs (Microsoft Foundation Classes), COM (Component Object Model), and .NET technologies.

The most popular languages for use on Windows include Visual Basic/VB6[1] and C/C++[2], although C++ is quickly being replaced by the .NET[3] platform, specifically C#[4] (C Sharp).

1.2 Windows Kernels

Windows 1.0, 2.0, and 3.11 are considered to be an older generation of Windows systems that were built to be a simple graphical layer over the MS-DOS operating system. Windows 95, Windows 98, and Windows ME were designed to bypass MS-DOS (although DOS was still present), and were all based on the same code structure known as the "9x Kernel". Windows NT 4.0, Windows 2000, Windows XP, Windows Vista, Windows 7, and Windows Server are all based on a collection of code known as the "NT Kernel".

1.3 System Architecture

The Windows NT Kernel is divided into several sections, here we will briefly discuss how the Windows operating system is put together. At the most basic level is the file NTOSKRNL.EXE, the kernel of the Windows operating system, and the most important

1 `http://en.wikibooks.org/wiki/Programming%3AVisual%20Basic%20Classic`
2 `http://en.wikibooks.org/wiki/Programming%3Ac`
3 `http://en.wikipedia.org/wiki/.NET_Framework`
4 `http://en.wikibooks.org/wiki/C%20Sharp%20Programming`

file on your computer. If you are interested in seeing this for yourself, you can find it in the C:\Windows\System32 folder (this can also be found using the following path %systemroot%\system32) on your own Windows NT machines.

NTOSKRNL.EXE provides some of the basic functionality of Windows, but one file alone cannot make the whole system work. NTOSKRNL relies heavily on a **Dynamic Link Library** (DLL) known as HAL.DLL. HAL stands for "Hardware Abstraction Layer", and is the portion of code that allows low-level mechanisms such as interrupts and BIOS communication to be handled independently.

If we consider Windows architecture as a layered architecture, with NTOSKRNL.EXE and HAL.DLL on the bottom layer, the next layer up contains two important files, NTDLL.DLL, and WIN32K.SYS. NTDLL contains a number of user-mode functions such as system call stubs and the run-time library (RTL) code, collectively known as the (largely undocumented) "Native API". Much of the run-time library code is shared between NTOSKRNL and NTDLL. WIN32K.SYS is a kernel-mode driver that implements windowing and graphics, allowing for user interfaces to be created.

The next layer up contains a number of libraries that will be of primary interest to us. This layer comprises what is called the **Win32 API**, and it contains (almost) all the functions that a user will need in order to program in Windows. The Win32 API is divided into 4 component parts, each one a .DLL:

kernel32.DLL

This contains most of the system-related Win32 API functions. Most of these functions are just wrappers around the lower-level NTDLL functions, but some functionality such as National Language Support (NLS) and console handling are not available in NTDLL.

advapi32.DLL

This contains other system-related functions such as registry and service handling.

gdi32.DLL

This contains a number of basic functions for drawing. These functions are all relatively simple, and allow the user to draw shapes (circles, rectangles, etc.) on the screen, to display and manipulate bitmaps, etc.

user32.DLL

This contains a number of functions that implement the familiar user-interface of Windows. Programs, message boxes, prompts, etc are all implemented using the User32 functions. User32 performs its tasks by calling system calls implemented by WIN32K.SYS.

In addition to the 4 primary libraries in the Win32 API, there are a number of other important libraries that a Windows programmer should become familiar with:

MSVCRT.DLL

MSVCRT.DLL is the dynamic link library that contains the implementations of the C standard library (**stdlib**) functions that C programmers should be familiar with. These are the functions defined in the common header files stdio.h, string.h, stdlib.h, etc.

WS2_32.DLL

This is the Winsock2 library, that contains the standard Berkeley socket API for communicating on the internet. We will talk about winsock programming later in this book.

1.4 Windows Does It

The Windows system, it might be surprising for some people to learn, is a very hands-on system. This is not a familiar concept for people who are just beginning C programming using the standard library. In a normal software project, there is typically a main function, and the main function in turn calls other functions that are defined in your project. In a Windows function, typically the programmer provides function pointers to the system, and Windows will make calls *into your program.* Also, in a Windows program, your code will sit idle when there is nothing to be done. Using the **message loop architecture**, Windows will send messages to your program when an event needs to be handled, and the program responds to the messages. If the program doesn't respond, the message is ignored.

For each program, Windows sets up a message queue structure to handle the message transmission process. Windows will maintain a listing of all the objects and system resources in use by a program, and will assign each one a **handle**. These handles are useless by themselves, but they can be passed to the system to reference particular objects and resources.

2 User Mode vs Kernel Mode

In **Windows** (and most modern operating systems), there is a distinction between code that is running in "user mode", and code that is running in "kernel mode". This chapter is going to point out some of the differences. Firstly, Intel CPUs have modes of operation called *rings* which specify the type of instructions and memory available to the running code. There are four rings:

- *Ring 0* (also known as *kernel mode*) has full access to every resource. It is the mode in which the Windows kernel runs.
- Rings 1 and 2 can be customized with levels of access but are generally unused unless there are virtual machines running.
- *Ring 3* (also known as *user mode*) has restricted access to resources.

The reason for this is because if all programs ran in kernel mode, they would be able to overwrite each others' memory and possibly bring down the entire system when they crashed.

2.1 Virtual Memory

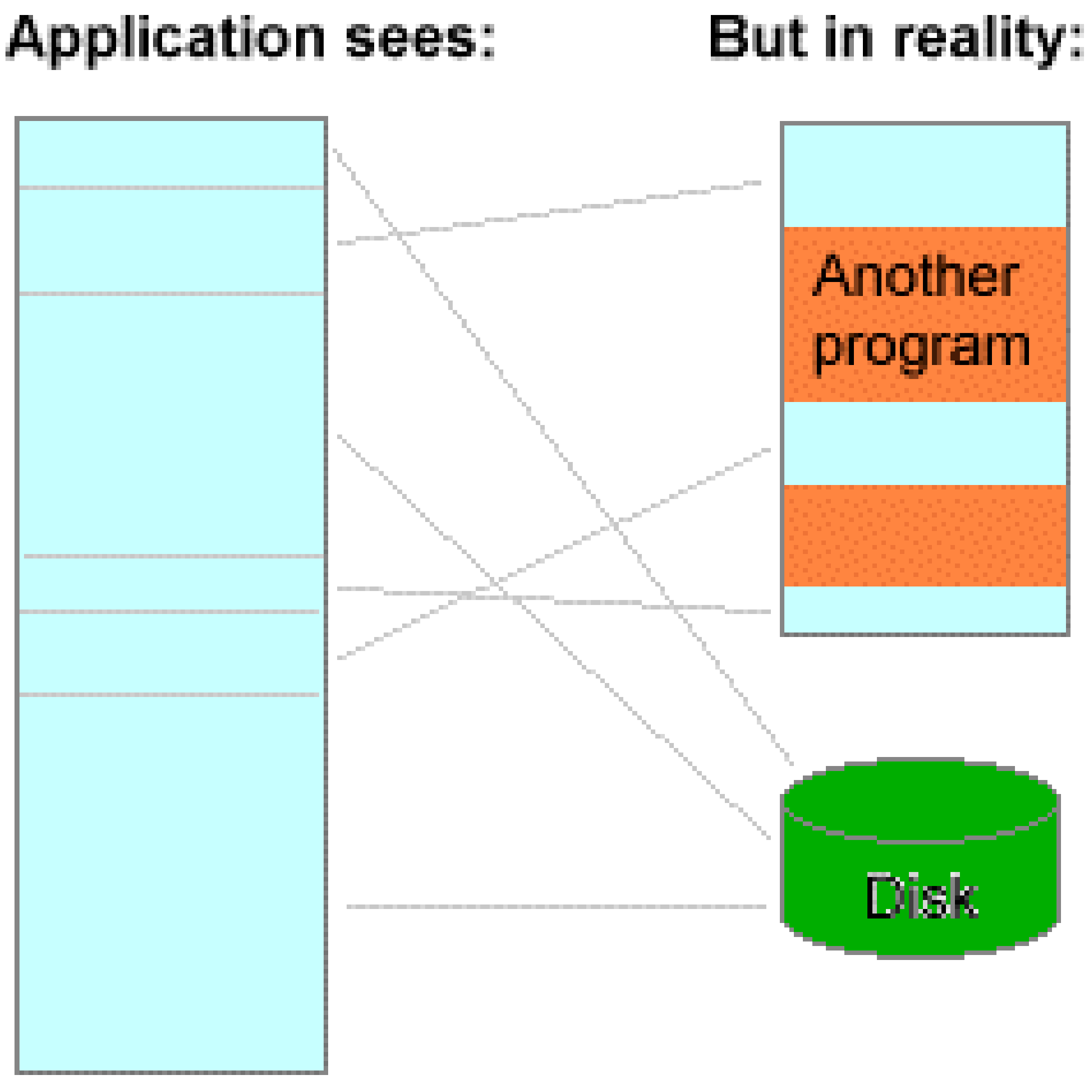


Figure 1 The program thinks it has a large range of contiguous addresses; but in reality the parts it is currently using are scattered around RAM, and the inactive parts are saved in a disk file.

When a program is started (e.g. a web browser or a word processor), it runs in its own *process*. A process contains its own "virtual" memory space and resources. Its memory is "virtual" because the process thinks memory is at address `0x12345678` may actually be at address `0x65f7a678` in physical memory. Similarly, two different processes may have different data stored at (to them) `0x00401000`. This is implemented by dividing memory into chunks called *pages*; on x86 systems one page is 4 kilobytes in size. Each page can have its own set of attributes, such as read-only/read-write. The CPU has a transparent mechanism for translating virtual addresses to physical addresses through a *page table* which the operating system sets up.

Virtual memory is useful for many reasons:

1. The process cannot access other process' memory,
2. Each page can have different protection settings (read-only or read-write, kernel-mode-only), and
3. Inactive memory regions of the process can be "paged out" (stored) to the *pagefile* and be retrieved by the operating system when needed. This is also done when the system is low on physical memory.

2.2 User Mode

Every process started by Windows (with the exception of the *System* "process") runs in user mode. In this mode, programs cannot modify paging directly and so have no way of accessing other programs' memory except through API functions. Programs in user mode also cannot interfere with interrupts and context switching.

2.3 Kernel Mode, Interrupts, and System Calls

When Windows is first loaded, the Windows kernel is started. It runs in kernel mode and sets up paging and virtual memory. It then creates some system processes and allows them to run in user mode. How does the CPU ever switch back to kernel mode then? This is not done automatically by the CPU. The CPU is often interrupted by certain events (timers, keyboard, hard disk I/O), and these are called *interrupts*. The kernel must first set up *interrupt handlers* to deal with these events. Then, whenever interrupts occur, the CPU stops executing the currently running program, immediately switches to kernel mode, and executes the interrupt handler for that event. The handler saves the state of the CPU, performs some processing relevant to that event, and restores the state of the CPU (possibly switching back to user mode) so the CPU can resume execution of the program.

When a program wants to call a Windows API function[1], it triggers an interrupt[2] which causes the CPU to switch to kernel mode and begin executing the desired API function. When the API function has finished processing, it switches back to user mode and resumes execution of the program. This is because API functions like `ReadProcessMemory` cannot work in user mode; the program can't access other programs' memory. In kernel mode, however, the API function can read any memory region without restriction.

1. Actually, Windows API functions eventually call a different API: the *Native API*. This is the API used by the Windows NT family of kernels. *This* is when the CPU switches to kernel-mode.

2. Modern CPUs have special, faster instructions for system calls, such as `sysenter` and `sysexit` on x86. These instructions cause the CPU to switch to ring 0, and then begin executing a handler set up by the operating system.

2.4 Context Switching

So, a program runs and calls API functions. How do other programs get a chance to run, then? Most of the time, programs simply allow the operating system to switch to another program because they are waiting for something (human input, hard disk). These programs are known as *unrunnable* programs, and since they make calls to the kernel to wait for something, the kernel knows to perform *context switching* to allow another program to run. This is done by:

1. Saving the current program's state (including registers),
2. Figuring out which program to run next,
3. and restoring a different program's state.

If a program (*thread* or *process* to be more accurate) runs for more than a certain period of time (the *thread quantum* or a processes *time slice*), the operating system will context switch to another program. This idea is called *preemption.* Preemption is accomplished by setting a timed interrupt in the processor that will invoke context switching. The time slice that is used may be different for each process.

2.5 Next Chapter

- C and Win32 API[1]

1 Chapter 3 on page 11

3 C and Win32 API

3.1 C and Windows

Many of the low-level functions in Windows were created using the C programming language. C code tends to be relatively small and fast compared to VB code or even C++ code, and has a shorter development time compared to raw assembly code. All of the DLLs in the Win32 API, and most of the kernel-level structures are implemented in C code.

3.2 Visual Basic and Windows

Visual Basic is essentially a "Windows-only" programming language - although some ports either exist or are in development for other systems, VB is not nearly as popular on other operating systems as it is on Windows. The VB runtime libraries tap into the Win32 API and the MFC libraries, to implement its functionality. The new version of VB, "VB.NET", uses the .NET platform, and therefore will not be covered in this wikibook. Old versions of VB, or "VB Classic" utilize the same Win32 API and MFC interfaces that C and C++ use to communicate with Windows. However, not all of the functions are identical, because VB uses different data types, and stores data differently than C or C++. As we discuss the Win32 API and MFC, we will attempt to show how the same functionality is implemented in VB.

Visual Basic uses the STDCALL calling convention for implementing function calls. Most of the Win32 API uses the STDCALL calling convention as well, so much of the hard work for interfacing the two is done by the compiler. C and C++, on the other hand, do not use STDCALL by default, and if you want to manually link a VB object code file with a C object file, you will need to explicitly make the C interface a STDCALL one.

3.3 COM and Windows

Microsoft implemented a technology known as the **Component Object Model** for Windows. COM essentially takes the object-oriented programming paradigm to the next level, by standardizing the class interface, and allowing classes to be written in different languages (C++, VB, etc.) and interfaced together seamlessly. COM programs (or "COM clients") can be written in most languages that allow object-orientation.

3.4 Other Languages

Many other programming languages have been implemented on Windows systems, and many of them have some sort of method for interfacing the Win32 API, or the MFC libraries. These interfaces are known collectively as *wrappers*, because they wrap the functionality of the Win32 API in another programming language. Common wrappers are provided for Perl, Ada, Python, PHP, AutoIT, etc.

3.5 Next Chapter

- <Windows.h>[1]

1 Chapter 4 on page 13

4 <Windows.h>

4.1 windows.h

The primary C header file for accessing the Win32 API is the `<windows.h>` header file. To make a Win32 executable, the first step is to include this header file in your source code. The windows.h header file should be included *before* any other library include, even the C standard library files such as stdio.h or stdlib.h. This is because the windows.h file includes macros and other components that may modify, extend, or replace things in these libraries. This is especially true when dealing with UNICODE, because windows.h will cause all the string functions to use UNICODE instead. Also, because many of the standard C library functions are already included in the Windows kernel, many of these functions will be available to the programmer without needing to load the standard libraries. For example, the function `sprintf` is included in windows.h automatically.

4.2 Child Header Files

There are a number of header files that are automatically included with windows.h. Many of these files cannot simply be included by themselves, because of dependencies. The windows.h header file is actually a relatively small file, that provides conditional inclusion for a number of other header files, definitions of a few important macros, etc.

4.3 Additional Header Files

This section will talk about some other interesting header files and libraries that can be included with your project, but which are not included by default with windows.h.

4.4 windows.h Macros

This section will briefly detail some of the changes that the user can make in the windows.h structure at compile time using macros.

4.4.1 WIN32_LEAN_AND_MEAN

The WIN32_LEAN_AND_MEAN macro causes several of the child headers to not be included in the build. This can help to speed up the compilation process.

4.4.2 UNICODE and _UNICODE

These macros, which we will discuss later, can generally be used interchangeably, but are frequently both defined together. These cause the program to be compiled with UTF-16 encoded strings instead of ASCII encoded strings. UTF-16 is one internationalized encoding based on the UNICODE standard and allows for more characters than the American ASCII encoding, although UTF-8 is often preferred in modern programming.

4.4.3 WINVER and _WIN32_WINNT

You must set these to a number greater or equal to 0x500 or you will not get some of the useful functions that are in Windows 2000 (and thus in any modern Windows) but were not in Windows 98.

4.5 Next Chapter

- Handles and Data Types[1]

1 Chapter 5 on page 15

5 Handles and Data Types

One of the first things that is going to strike many first-time programmers of the Win32 API is that there are tons and tons of old data types to deal with. Sometimes, just keeping all the correct data types in order can be more difficult than writing a nice program. This page will talk a little bit about some of the data types that a programmer will come in contact with.

5.1 Hungarian Notation

First, let's make a quick note about the naming convention used for some data types, and some variables. The Win32 API uses the so-called "Hungarian Notation" for naming variables. Hungarian Notation requires that a variable be prefixed with an abbreviation of its data type, so that when you are reading the code, you know exactly what type of variable it is. The reason this practice is done in the Win32 API is because there are many different data types, making it difficult to keep them all straight. Also, there are a number of different data types that are essentially defined the same way, and therefore some compilers will not pick up on errors when they are used incorrectly. As we discuss each data type, we will also note the common prefixes for that data type.

Putting the letter "P" in front of a data type, or "p" in front of a variable usually indicates that the variable is a pointer. The letters "LP" or the prefix "lp" stands for "Long Pointer", which is exactly the same as a regular pointer on 32 bit machines. LP data objects are simply legacy objects that were carried over from Windows 3.1 or earlier, when pointers and long pointers needed to be differentiated. On modern 32-bit systems, these prefixes can be used interchangeably.

5.2 LPVOID

LPVOID data types are defined as being a "pointer to a void object". This may seem strange to some people, but the ANSI-C standard allows for generic pointers to be defined as "void*" types. This means that LPVOID pointers can be used to point to different types of objects, without creating a compiler error. However, the burden is on the programmer to keep track of what type of object is being pointed to.

Also, some Win32 API functions may have arguments labeled as "LPVOID lpReserved". These reserved data members should never be used in your program, because they either depend on functionality that hasn't yet been implemented by Microsoft, or else they are only used in certain applications. If you see a function with an "LPVOID lpReserved" argument, you must always pass a NULL value for that parameter - some functions will fail if you do not do so.

LPVOID objects frequently do not have prefixes, although it is relatively common to prefix an LPVOID variable with the letter "p", as it is a pointer.

5.3 DWORD, WORD, BYTE

These data types are defined to be a specific length, regardless of the target platform. There is a certain amount of additional complexity in the header files to achieve this, but the result is code that is very well standardized, and very portable to different hardware platforms and different compilers.

DWORDs (Double WORDs), the most commonly occurring of these data types, are defined always to be unsigned 32-bit quantities. On any machine, be it 16, 32, or 64 bits, a DWORD is always 32 bits long. Because of this strict definition, DWORDS are very common and popular on 32-bit machines, but are less common on 16-bit and 64-bit machines.

WORDs (Single WORDs) are defined strictly as unsigned 16-bit values, regardless of what machine you are programming on. BYTEs are defined strictly as being unsigned 8-bit values. QWORDs (Quad WORDs), although rare, are defined as being unsigned 64-bit quantities. Putting a "P" in front of any of these identifiers indicates that the variable is a pointer. putting two "P"s in front indicates it's a pointer to a pointer. These variables may be unprefixed, or they may use any of the prefixes common with DWORDs. Because of the differences in compilers, the definition of these data types may be different, but typically these definitions are used:

```
#include <stdint.h>
```

```
typedef uint8_t BYTE;
typedef uint16_t WORD;
typedef uint32_t DWORD;
typedef uint64_t QWORD;
```

Notice that these definitions are not the same in all compilers. It is a known issue that the GNU GCC compiler uses the *long* and *short* specifiers differently from the Microsoft C Compiler. For this reason, the windows header files typically will use conditional declarations for these data types, depending on the compiler being used. In this way, code can be more portable.

As usual, we can define pointers to these types as:

```
#include <stdint.h>
```

```
typedef uint8_t * PBYTE;
typedef uint16_t * PWORD;
typedef uint32_t * PDWORD;
typedef uint64_t * PQWORD;
typedef uint8_t ** PPBYTE;
typedef uint16_t ** PPWORD;
typedef uint32_t ** PPDWORD;
typedef uint64_t ** PPQWORD;
```

DWORD variables are typically prefixed with "dw". Likewise, we have the following prefixes:

Data Type	**Prefix**
BYTE	"b"
WORD	"w"
DWORD	"dw"
QWORD	"qw"

5.4 LONG, INT, SHORT, CHAR

These types are not defined to a specific length. It is left to the host machine to determine exactly how many bits each of these types has.

Types

```
typedef long LONG;
typedef unsigned long ULONG;
typedef int INT;
typedef unsigned int UINT;
typedef short SHORT;
typedef unsigned short USHORT;
typedef char CHAR;
typedef unsigned char UCHAR;
```

LONG notation

LONG variables are typically prefixed with an "l" (lower-case L).

UINT notation

UINT variables are typically prefixed with an "i" or a "ui" to indicate that it is an integer, and that it is unsigned.

CHAR, UCHAR notation

These variables are usually prefixed with a "c" or a "uc" respectively.

If the size of the variable doesn't matter, you can use some of these integer types. However, if you want to exactly specify the size of a variable, so that it has a certain number of bits, use the BYTE, WORD, DWORD, or QWORD identifiers, because their lengths are platform-independent and never change.

5.5 STR, LPSTR

STR data types are string data types, with storage already allocated. This data type is less common than the LPSTR. STR data types are used when the string is supposed to be treated as an immediate array, and not as a simple character pointer. The variable name prefix for a STR data type is "sz" because it's a zero-terminated string (ends with a null character).

Most programmers will not define a variable as a STR, opting instead to define it as a character array, because defining it as an array allows the size of the array to be set explicitly. Also, creating a large string on the stack can cause greatly undesirable stack-overflow problems.

LPSTR stands for "Long Pointer to a STR", and is essentially defined as such:

```
#define STR * LPSTR;
```

LPSTR can be used exactly like other string objects, except that LPSTR is explicitly defined as being ASCII, not unicode, and this definition will hold on all platforms. LPSTR variables will usually be prefixed with the letters "lpsz" to denote a "Long Pointer to a String that is Zero-terminated". The "sz" part of the prefix is important, because some strings in the Windows world (especially when talking about the DDK[1]) are not zero-terminated. LPSTR data types, and variables prefixed with the "lpsz" prefix can all be used seamlessly with the standard library <string.h> functions.

5.6 TCHAR

TCHAR data types, as will be explained in the section on Unicode, are generic character data types. TCHAR can hold either standard 1-byte ASCII characters, or wide 2-byte Unicode characters. Because this data type is defined by a macro and is not set in stone, only character data should be used with this type. TCHAR is defined in a manner similar to the following (although it may be different for different compilers):

```
#ifdef UNICODE
#define TCHAR WORD
#else
#define TCHAR BYTE
#endif
```

1 http://en.wikipedia.org/wiki/Windows_Driver_Kit

5.7 TSTR, LPTSTR

Strings of TCHARs are typically referred to as TSTR data types. More commonly, they are defined as LPTSTR types as such:

```
#define TCHAR * LPTSTR
```

These strings can be either UNICODE or ASCII, depending on the status of the UNICODE macro. LPTSTR data types are long pointers to generic strings, and may contain either ASCII strings or Unicode strings, depending on the environment being used. LPTSTR data types are also prefixed with the letters "lpsz".

5.8 HANDLE

HANDLE data types are some of the most important data objects in Win32 programming, and also some of the hardest for new programmers to understand. Inside the kernel, Windows maintains a table of all the different objects that the kernel is responsible for. Windows, buttons, icons, mouse pointers, menus, and so on, all get an entry in the table, and each entry is assigned a unique identifier known as a HANDLE. If you want to pick a particular entry out of that table, you need to give Windows the HANDLE value, and Windows will return the corresponding table entry.

HANDLEs are defined as being unsigned 32-bit quantities in <windows.h>, but HANDLEs should never be used like integers. They are unique identifiers, and if you edit them, or use them in arithmetic, then they can never be used to get the table entry that they correspond to. In other words, HANDLEs should be stored, but they should never be changed by the programmer.

HANDLEs are generally prefixed with an "h". Below are a few special handles that are worth discussing: A handle is actually a pointer to a pointer to a memory location.Handles are unsigned integers that Windows uses internally to keep track of objects in memory. Windows moves objects like memory blocks in memory to make room, if the object is moved in memory, the handles table is updated.

5.8.1 HWND

HWND data types are "Handles to a Window", and are used to keep track of the various objects that appear on the screen. To communicate with a particular window, you need to have a copy of the window's handle. HWND variables are usually prefixed with the letters "hwnd", just so the programmer knows they are important.

Canonically, main windows are defined as:

```
HWND hwnd;
```

Child windows are defined as:

```
HWND hwndChild1, hwndChild2...
```

and Dialog Box handles are defined as:

```
HWND hDlg;
```

Although you are free to name these variables whatever you want in your own program, readability and compatibility suffer when an idiosyncratic naming scheme is chosen - or worse, no scheme at all.

5.8.2 HINSTANCE

HINSTANCE variables are handles to a program instance. Each program gets a single instance variable, and this is important so that the kernel can communicate with the program. If you want to create a new window, for instance, you need to pass your program's HINSTANCE variable to the kernel, so that the kernel knows where the new window belongs to. If you want to communicate with another program, it is frequently very useful to have a copy of that program's instance handle. HINSTANCE variables are usually prefixed with an "h", and furthermore, since there is frequently only one HINSTANCE variable in a program, it is canonical to declare that variable as such:

```
HINSTANCE hInstance;
```

It is usually a benefit to make this HINSTANCE variable a global value, so that all your functions can access it when needed.

5.8.3 HMENU

If your program has a drop-down menu available (as most visual Windows programs do), that menu will have an HMENU handle associated with it. To display the menu, or to alter its contents, you need to have access to this HMENU handle. HMENU handles are frequently prefixed with simply an "h".

5.9 WPARAM, LPARAM

In the earlier days of Microsoft Windows, parameters were passed to a window in one of two formats: WORD-length (16-bit) parameters, and LONG-length (32-bit) parameters. These parameter types were defined as being WPARAM (16-bit) and LPARAM (32-bit). However, in modern 32-bit systems, WPARAM and LPARAM are both 32 bits long. The names however have not changed, for legacy reasons.

WPARAM and LPARAM variables are generic function parameters, and are frequently type-cast to other data types including pointers and DWORDs.

5.10 Next Chapter

- Unicode[2]

2 Chapter 6 on page 23

6 Unicode

For a reference of Unicode standard, see Unicode[1].

6.1 Introduction to Unicode

w:Unicode[2] **Unicode** is an industry standard whose goal is to provide the means by which text of all forms and languages can be encoded for use by computers. Originally, text-characters were represented in computers using byte-wide data: each printable character (and many non-printing, or "control" characters) were implemented using a single byte each, which allowed for 256 characters total. However, globalization has created a need for computers to be able to accommodate many different alphabets from around the world.

The old codes were known as ASCII or EBCDIC, but it was apparent that neither of these codes were capable of handling all the different characters and alphabets from around the world. The solution to this problem created Unicode. Windows NT implements many of its core functions with a "wide" 16-bit characters set, close to Unicode standard, although it provides a series of functions that are compatible with the standard ASCII characters as well.

UNICODE characters are frequently called "Wide Characters", "Generic Characters", or "T Characters". This book may use any of these terms interchangeably.

6.1.1 Variable-Width Characters

Before Unicode, there was an internationalization attempt that introduced character strings with variable-width characters. Some characters, such as the standard ASCII characters would be 1 byte long. Other characters, such as extended character sets, were two bytes long. These types of character formats fell out of favor with the advent of UNICODE because they are harder to write and much harder to read. Windows does still maintain some functionality to deal with variable-width strings, but we won't discuss those here.

Unfortunately all advantages of using wide characters were lost because the number of characters needed quickly exceeded the 65,536 possible 16-bit values. Windows actually uses what is called UTF-16 to store characters, where a large number of characters actually take //two// words, these are called "surrogate pairs". This development is after much of the Windows API documentation was written and much of it is now obsolete. You

1 `http://en.wikibooks.org/wiki/Unicode`
2 `http://en.wikipedia.org/wiki/Unicode`

should never treat string data as an "array of characters", instead always treat it as a null-terminated block. For instance always send the entire string to a function to draw it on the screen, do not attempt to draw each character. Any code that puts a square bracket after a LPSTR is wrong.

At the same time, variable-width character-based strings made a big comeback in the multi-platform standard called UTF-8, which is pretty much the same idea as UTF-16 except with 8-bit units. Its primary advantage is that there is no need for two APIs. The 'A' and 'W' APIs would have been the same if this were used, and since both are variable-sized, it has no disadvantage. Although most Windows programmers are unfamiliar with it, you may see increased references to using the non-UNICODE API.

6.2 Windows Implementation

The Win32 API classifies all of its functions that require text input into two categories. Some of the functions have an "A" suffix (for ASCII), and some have a "W" suffix (for Wide characters, or Unicode). These functions are differentiated using the macro "UNICODE":

```
#ifdef UNICODE
#define MessageBox MessageBoxW
#else
#define MessageBox MessageBoxA
#endif
```

Because of this differentiation, when you receive a compiler error, you will get an error on "MessageBoxW" instead of simply "MessageBox". In these cases, the compiler is not broken. It is simply trying to follow a complex set of macros.

6.3 Unicode Environment

All Windows functions that require character strings are defined in this manner. If you want to use unicode in your program, you need to explicitly define the UNICODE macro before you include the windows.h file:

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

Also, some functions in other libraries require you to define the macro _UNICODE. The standard library functions can be provided in unicode by including the <tchar.h> file as well. So, to use unicode in your project, you need to make the following declarations in your project:

```
#define UNICODE
#define _UNICODE
```

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

Some header files include a mechanism like the following, so that when one of the two UNICODE macros is defined, the other is automatically defined as well:

```
#ifdef UNICODE
  #ifndef _UNICODE
    #define _UNICODE
  #endif
#endif
```

```
#ifdef _UNICODE
  #ifndef UNICODE
    #define UNICODE
  #endif
#endif
```

If you are writing a library that utilizes UNICODE, it might be worthwhile for you to include this mechanism in your header files as well, so that other programmers don't need to worry about including both macros.

6.4 TEXT macro

In C, to make a string of wide characters, you need to prefix the string with the letter "L". Here is an example:

```
char *asciimessage = "This is an ASCII string.";
w_char *unicodemessage = L"This is a Wide Unicode string.";
```

The data type "TCHAR" is defined as being a char type if unicode is not defined, and is defined as being a wide type if UNICODE is defined (in tchar.h). To make strings portable between unicode and non-unicode, we can use the TEXT() macro to automatically define a string as being unicode or not:

```
TCHAR *automessage = TEXT("This message can be either ASCII or UNICODE!");
```

Using TCHAR data types, and the TEXT macro are important steps in making your code portable between different environments.

Also, the TEXT macro can be written as:

```
TEXT("This is a generic string");
_T("This is also a generic string");
T("This is also a generic string");
```

All three of these statements are equivalent.

The TEXT macro is typically defined like this:

```
#ifdef UNICODE
#define TEXT(t) L##t
#define _T(t) L##t
#define T(t) L##t
#else
#define TEXT(t) t
#define _T(t) t
#define T(t) t
#endif
```

6.5 Unicode Reference

- see Unicode[3]

6.5.1 Control Characters

Unicode characters 0 to 31 (U+0000 to U+001F) are part of the C0 Controls and Basic Latin block. They are all control characters. These characters correspond to the first 32 characters of the ASCII set.

Code point	**Decimal equivalent**	**Name**
U+0000	0	null character
U+0001	1	start of header
U+0002	2	start of text
U+0003	3	end of text
U+0004	4	end of transmission
U+0005	5	inquiry
U+0006	6	acknowledgment
U+0007	7	bell
U+0008	8	backspace
U+0009	9	horizontal tab
U+000A	10	line feed
U+000B	11	vertical tab
U+000C	12	form feed
U+000D	13	carriage return
U+000E	14	shift out
U+000F	15	shift in
U+0010	16	data link escape
U+0011	17	device control 1
U+0012	18	device control 2
U+0013	19	device control 3
U+0014	20	device control 4

3 http://en.wikibooks.org/wiki/Unicode

Code point	Decimal equivalent	Name
U+0015	21	negative acknowledgment
U+0016	22	synchronous idle
U+0017	23	end of transmission block
U+0018	24	cancel
U+0019	25	end of medium
U+001A	26	substitute
U+001B	27	escape
U+001C	28	file separator
U+001D	29	group separator
U+001E	30	record separator
U+001F	31	unit separator

6.6 Next Chapter

- Dynamic Link Libraries[4] (DLL)

4 Chapter 7 on page 29

7 Dynamic Link Libraries

7.1 Dynamic link libraries

Dynamic Link Libraries (DLLs) were introduced with the first releases of the Microsoft Windows operating system, and today are a fundamental structural component of the OS. They are found not only on the OS core, but as part of many of the frameworks created by Microsoft like the MFC, ATL, .NET, etc, even C and the C++ runtime libraries are distributed as DLLs.

DLLs allow certain code fragments to be compiled into a single library, and to be linked to by multiple programs. This means that only one copy of the library needs to exist, and multiple programs can share the functions and the data between them. The Windows system makes itself accessible, in several of its user-space features, by providing DLLs that programmers can use.

The difference between a DLL and a static library is that when you compile your programs, the DLL is not compiled into your executable, but instead remains a separate module. This feature helps to keep executable size low, and also allows for a DLL to be loaded into memory only when it is needed. As a self contained entity a DLL also permit kick and target updates to the system and to applications. By simply replacing a DLLs with a newer version with fixes or improvements, it is easy to extend the alteration to multiple dependent programs instantly.

The exact method of building a DLL file is dependent on the compiler you are using. However, the way in which DLLs are programmed is universal. We will talk about how to program DLL files in this chapter.

7.1.1 DLL hell

The common problem referred generically as "DLL hell[1]" has always been a bane to Windows programmers and it really doesn't seem to have a solution in the horizon. The problem was stated in the 90s and there was when the term was coined. The issue is on the permissibility of the OS to let incorrect DLLs version to be loaded upon the request from an application, that would invariably lead to a crash. Today an application will simply refuse to run.

1 `http://en.wikipedia.org/wiki/DLL%20hell`

7.2 __declspec

The **__declspec** keyword is a strange new keyword that is not part of the ANSI C standard, but that most compilers will understand anyway. **__declspec** allows a variety of non-standard options to be specified, that will affect the way a program runs. Specifically, there are two **__declspec** identifiers that we want to discuss:

- __declspec(dllexport)
- __declspec(dllimport)

When writing a DLL, we need to use the **dllexport** keyword to denote functions that are going to be available to other programs. Functions without this keyword will only be available for use from inside the library itself. Here is an example:

```
__declspec(dllexport) int MyFunc1(int foo)
```

The **__declspec** identifier for a function needs to be specified both in the function prototype and the function declaration, when building a DLL.

To "import" a DLL function into a regular program, the program must link to the DLL, and the program must prototype the function to be imported, using the **dllimport** keyword, as such:

```
__declspec(dllimport) int MyFunc1(int foo);
```

Now the program can use the function as normal, even though the function exists in an external library. The compiler works with Windows to handle all the details for you.

Many people find it useful to define a single header file for their DLL, instead of maintaining one header file for building a DLL, and one header file for importing a DLL. Here is a macro that is common in DLL creation:

```
#ifdef BUILDING_DLL
#define DLL_FUNCTION __declspec(dllexport)
#else
#define DLL_FUNCTION __declspec(dllimport)
#endif
```

Now, to build the DLL, we need to define the **BUILDING_DLL** macro, and when we are importing the DLL, we don't need to use that macro. Functions then can be prototyped as such:

```
DLL_FUNCTION int MyFunc1(void);
DLL_FUNCTION int MyFunc2(void);
.......
```

(just a note: Microsoft did not intend for this __declspec syntax to be used. Instead the intention was that the public api of a DLL would be declared in an "exports" file. However the above syntax, despite requiring the macro to switch it back and forth, was much more convenient and is pretty much used by all software today).

7.3 DllMain

When Windows links a DLL to a program, Windows calls the library's **DllMain** function. This means that every DLL needs to have a DllMain function. The DllMain function needs to be defined as such:

```
BOOL APIENTRY DllMain (HINSTANCE hInstance, DWORD reason, LPVOID reserved)
```

The keywords "BOOL", "APIENTRY", "HINSTANCE", etc., are all defined in <windows.h>. So, you must include that file even if you don't use any Win32 API functions in your library.

APIENTRY is just a keyword that Windows uses internally. So, you don't need to worry about it. The variable "hInstance" is the HINSTANCE handle for the library, and you can keep this and use it, or you can trash it. **reason** will be one of four different values:

DLL_PROCESS_ATTACH

a new program has just linked to the library for the first time.

DLL_PROCESS_DETACH

a program has unlinked the library.

DLL_THREAD_ATTACH

a thread from a program has linked to the library.

DLL_THREAD_DETACH

a thread from a program has just unlinked the library.

The DllMain function doesn't need to do anything special for these cases, although some libraries will find it useful to allocate storage for each new thread or process that is being used with the library.

The DllMain function must return TRUE if the library loaded successfully, or FALSE if the library had an error and could not load. If you return FALSE, the program will pop up a warning message and crash.

Here is a general template for a DllMain function:

```
BOOL APIENTRY DllMain (HINSTANCE hInst, DWORD reason, LPVOID lpReserved)
{
   switch (reason)
   {
     case DLL_PROCESS_ATTACH:
       break;
     case DLL_PROCESS_DETACH:
       break;
```

```
      case DLL_THREAD_ATTACH:
        break;
      case DLL_THREAD_DETACH:
        break;
    }
    return TRUE;
}
```

However, if you aren't interested in any of the reasons, you can remove the entire switch statement from your program and return TRUE.

7.4 Linking a DLL

DLL libraries can be linked to an executable in two ways: Statically and Dynamically.

7.4.1 Static Linking to a DLL

When static linking to a DLL, the linker will do all the work, and it will be transparent to the programmer that the functions are located in an external library. That is, it will be transparent if the library writer has properly used the _DECLSPEC modifier in the library's header file.

When compiling a DLL, the compiler will generate two files: the DLL library file, and a static-linking stub .LIB file. The .LIB file acts like a mini static library, that tells the linker to statically link the associated DLL file. When using a DLL in a project, you can either provide the linker with the .LIB stub file, or some linkers allow you to specify the DLL directly (and the linker will then try to find the .LIB file, or may even attempt to create the .LIB file automatically).

7.4.2 Loading DLLs Dynamically

The real power behind DLL files is that they can be loaded into your program dynamically at execution time. This means that while your program is running, it can search for and load in new components, without needing to be recompiled. This is an essential mechanism for programs that allow plugins and extensions to be loaded at execution time. To Dynamically load the DLL file, you can call the **LoadLibrary** function to get a handle to that library, and then pass that handle to one of several other functions to retrieve data from the DLL. The prototype for LoadLibrary is:

```
HMODULE WINAPI LoadLibrary(LPCTSTR lpFileName);
```

HMODULE is a HANDLE to a program module. *lpFileName* is the file name of the DLL you want to load. Keep in mind that when loading a module, the system will check in your PATH first. If you want the system to check in other specified directories first, use the **SetDllDirectory** function first.

Once a DLL is loaded, and you have a handle to the module, you can do various things:

- Use **GetProcAddress** to return a function pointer to a function in that library.
- Use **LoadResource** to retrieve a resource from the DLL.

Once you are finished with a DLL file, and you want to remove it from memory, you can call the **FreeLibrary** function with the DLL's module handle.

7.5 Next chapter

- Programming Windows With OSS Tools[2]

2 Chapter 8 on page 35

8 Programming Windows With OSS Tools

8.1 Getting an Open Source Compiler

With a fast internet connection, gcc --- a free compiler that can compile C, C++, and Fortran, as well as other languages and works on Windows --- can be obtained through Cygwin[1]. When installing Cygwin, select to install the "devel/gcc" package and optionally also "graphics/opengl", depending on what type of program you're planning to write. Another Windows port of the GCC compiler collection is MinGW (Minimalist GNU for Windows)[2], an approximately 10MB download 1/10th the size of Cygwin. You could also download DJGPP[3], but it is designed more for 32-bit console programs. An alternative to obtaining the gcc compiler via porting is to compile from command line within a virtualized[4] version of Linux, which not only comes with gcc, but whose kernel "... is written in the version of the C programming language supported by GCC..."

8.1.1 Command Line GCC Illustration

To compile, open a command-line interface and use the commands:

```
c:\cygwin\bin\gcc -s -Os -mno-cygwin -o <outputfilename> <inputfilename>
```

for command-line programs and

```
c:\cygwin\bin\gcc -s -Os -mwindows -mno-cygwin -o <outputfilename>
<inputfilename> -lopengl32 -lwinmm
```

for GUI programs.

Description of command-line flags:

- -s: makes the output program smaller by stripping out information that is useful for debugging crashed applications
- -Os: optimizes for size (-O3 for execution speed)
- -mno-cygwin: makes the program work without any external nonstandard .dll-files

1 http://www.cygwin.com/
2 http://www.mingw.org/
3 http://www.delorie.com/djgpp
4 http://en.wikipedia.org/wiki/Operating_system-level_virtualization

- -mwindows: makes it a GUI application
- -lopengl32: optionally lets you use OpenGL for graphics
- lwinmm: optionally adds some multimedia support such as sound, joysticks, and high resolution timers

One must be in the bin directory of Cygwin in order to compile with Cygwin, whereas in Linux gcc is already aliased to work in all directories.

8.1.2 Graphical Integrated Development Environments

For a more user-friendly environment, like Visual Studio, an excellent open source IDE for Windows is Bloodshed's Dev-C++[5]. It has a well-organised Graphical User Interface, and comes with everything necessary for setting up a Windows C/C++ development environment, installing and using the Mingw port of GCC by default (though it can be configured to work in combination any other GCC-based compiler such as Cygwin). Its source (written in Delphi) is available from the Bloodshed Website, as are the pre-compiled binaries. Dev-C++ has not been updated in quite some time, though, and alternatives like Code::Blocks[6] or CodeLite[7] have since emerged. Netbeans[8] is an IDE developed by Sun Microsystems as the official IDE for Java however Netbeans now supports multiple languages and is very extensible via its plugin system. Another open source IDE is Eclipse[9], which was initially developed by IBM. One more IDE, created specifically for C++, is Ultimate++[10].

8.2 GCC, DJGPP, MinGW and others

These are three well-known FOSS development toolsets for compiling applications, although they function in a way slightly different to Microsoft and Borland's Windows programming tools.

8.2.1 GCC

GCC stands for the *GNU Compiler Collection.* Although it provides support for compiling sources in several programming languages, it is most commonly used for C and C++ code. Because it is a compiler and not an IDE, approaches such as DJGPP wrap some more arcane details away from the user, easing the creation of programs, as well as providing other features, described below.

5 `http://www.bloodshed.net/devcpp.html`
6 `http://www.codeblocks.org`
7 `http://codelite.org/`
8 `http://www.netbeans.org`
9 `http://www.eclipse.org/`
10 `http://www.ultimatepp.org/`

8.2.2 DJGPP

DJGPP is an Integrated Development Environment (IDE) developed by DJ Delorie since 1989. It is a DOS application with 32-bit awareness and allows the creation of software with DPMI support. As a combined port of GCC and other GNU utilities, it is commonly used to write, compile and debug console applications. It can make use of libraries such as Allegro, in order to create applications with graphics, Other add-ons extend its functionality. It has support for cross-platform development, meaning that you may create software without needing a physical instance of a target machine type or operating system in order to build the executable that will be shipped to that configuration.

8.2.3 MinGW

MinGW stands for the *Minimalist GNU for Windows* and includes the needed header files for using the Win32 API; it supports creation of native Windows executables. As of late December 2006, Wikipedia defines the goal of MinGW as to "provide only a free compiler and toolchain, prioritizing performance". This contrasts the Cygwin objective of creating a POSIX compliant development environment. Like DJGPP, it can be used for cross-platform development. Because MinGW is available for other systems outside of Windows (such as Linux), even developers lacking a copy of Windows can create Win32 API applications. Furthermore, PE executables (.exe files) can be run using the WINE program (which translates Windows API calls to the corresponding UNIX system functions) to get a taste of what the application will look like - without actually having to install Windows.

8.2.4 Dev-C++

Dev-C++ is a free graphical IDE that comes packaged with MinGW and can also use the Cygwin compiler.

8.2.5 Other tools

GCC, DJGPP and MinGW do not have support for graphic-oriented resource and GUI editing commonly needed for Windows programming.

However open source libraries such as QT, GTK+ and wxWidgets provide user interface frontends as well as simplified models wrapping around the complexities of the Windows API. The minimal trade-off is that these may require special licensing and the bundling of runtime DLLs.

8.3 Cygwin

Cygwin is an environment made for Windows that emulates Unix behavior. It can be obtained from cygwin.com via a small setup.exe package that when run, asks for a selection of features that are then fetched. A fast internet connection is suggested, since Cygwin comprises many non-trivial shells, programs, and even GUIs.

The setup program allows the user to download the following parts, among others:

- Bash shell
- Development tools, with debugger, several languages supported by gcc compilers, as well as AWK, SED, YACC and Bison support
- Network tools, such as ssh, lynx web browser, nmap, and others
- Programmer text editors, such as GNU Emacs, nano and vi
- X, which is a very difficult environment to fetch for free on Windows otherwise

The software, after selected, is downloaded and installed to a folder at the root of the user's hard drive. It does not repartition the drive, require installation of any Unix system nor use a disk image. Files are simply installed in a Unix hierarchy within the cygwin folder.

To run Cygwin, the user will have a Start Menu program or Desktop shortcut called **Cygwin Bash Shell**. Upon opening it, a DOS-like window opens and displays a classic bash prompt. It is color coded, unlike its DOS underpinnings, and can interact with DOS by being aware of path settings and other system variables, so be aware of the fact that your Windows applications may be located from your system path variable and conflict with a folder's current filename because of DOS's implicit EXE extension expansion.

8.3.1 Directory information

As an added detail, if you need access to different drives in your computer, instead of using /**mnt**/, you have to cd into /**cygdrive**/ to see your drives listed. Seeing your C: drive's root, then, is just a matter of typing cd /**cygdrive**/**c** at the command line.

8.3.2 Emacs editor information

When using Emacs, the normal key combination to exit the program has no effect, by default, though adept users will probably find an Emacs keybinding trick to overcome this issue. Thus, when you want to exit the editor from command line mode, you may end up getting tired, and suspend the program to do a later kill of the process at the command line. Instead of pressing C-x C-c, to leave emacs, you must press F10 to get the menu. Once there, press f and then e. This summons the File Menu and later its Exit option, returning to the command line and prompting for the appropriate saving when some changed files need to be committed to disk.

8.3.3 Compiling without Cygwin library bindings

Compiling with g++, as stated in a previous section, needs to include the appropriate flags. gcc will normally bind the executable code to the cygwin1.dll file, unless your gcc flags explicitly include "-mno-cygwin" as a switch. To solve this, you can add this and all other switches to your .bashrc file, by using a known Unix option:

Open the .bashrc file at your Cygwin home directory in any editor. Look for the lines starting with "alias". There, you will see some shell commands tweaked to have specific options. Under the line containing

```
alias l='ls -CF'
```

you may add:

```
alias g++='g++ -mno-cygwin' #do not depend on cygwin1.dll on target system
```

In case you need to add more fine tuning tags, you may either add them to that same line after the -mno-cygwin switch, or just re-alias the g++ command so that the shell stacks up the current meaning of g++ to further tags. For example, you may add

```
alias g++='g++ -Wno-deprecated' #silence deprecation talk.
```

To keep the g++ compiler from reminding you about the standard practice of including <string> instead of the now deprecated <string.h> header.

8.3.4 No root nor su switching

Cygwin was created with a few differences in mind. It does not have a different root account, though it may claim files are owned by the **Administrator** user. Though this may be beyond the scope of the book, it is worth a line or two to say that you won't easily be able to experiment with programs that run under different user rights, because it does not seem that Cygwin tries to enforce account security. You may read any file in the OS that Windows allows Cygwin to access.

8.3.5 Running X

8.3.6 Other details

8.4 GNU Tools

8.5 Next Chapter

- Resource Scripts[11]

11 Chapter 9 on page 41

9 Resource Scripts

The Windows platform SDK includes a special utility called **rc.exe**, the **Resource Script Compiler**. Resource scripts are a powerful part of Windows programming that any good programmer should know how to utilize effectively.

For more information about specific resources, see the Resource Script Reference[1].

9.1 What is a Resource Script

The resource compiler compiles a special type of file known as a **Resource Script**. Resource scripts contain GUI data, and, when compiled, can be linked into a program. The program then can access the data contained in the resource script. Also interesting to note is that the Windows operating system accesses a program's resources for various purposes. For instance, right-click on a program and click "Properties". If available, click on the "Version" tab. This tab will contain a number of different text strings that describe the program. These text strings are all included from a resource script.

Also, people are all familiar with the fact that usually each program has a distinct icon. If you want your program executable to use a special icon you must include it in a resource script.

There are many different types of resources that can be stored in a program via a resource script. Resource scripts are not the only way to store this information in a program, but they are much easier to write than hard C or VB code to store and access them.

9.2 Types of resources

Here is a list of common resources:

- Drop-down Menus
- Popup Menus
- Text Strings
- Keyboard Accelerators (keypress combinations, such as [Ctrl]+[C] to copy text)
- Icons
- Bitmap Images
- Dialog Boxes
- Version information
- Mouse Cursors

1 Chapter 41 on page 183

9.3 Making a Resource Script

The syntax for writing resource scripts is similar to C. For instance, the resource script compiler uses the standard C preprocessor. This is important because you must include the header file <afxres.h> to make a resource script. Also, most resource scripts contain macro values, so programmers will frequently store the related macro definitions in a header file "resource.h". This way, the same macros can be included and used in the main program files.

We will talk more about resources in the following chapters; there's also an appendix where all common resource types are listed including information on how to use them. This appendix is located at Windows Programming/Resource Script Reference[2].

9.4 Using a Resource

Once a resource is stored in your executable there are various methods to access it. These methods differ depending on what type of resource you are trying to access. For instance if you want to access a string resource, you would need to use the **LoadString** function; correspondingly the **LoadIcon** function is needed to access an icon.

To access a resource you must have the instance handle of the executable file that contains the resource. This means, that if you obtain an instance handle to another executable (or DLL) file, you can access the resources remotely! Occasionally, programmers will create DLL files that contain nothing but resources for use in other programs.

9.5 MAKEINTRESOURCE

The **MAKEINTRESOURCE** keyword that we will occasionally see when dealing with resources is an important keyword, and a program will crash (or not even compile) if it is not used correctly so taking a minute to understand it is well worth the effort.

Resources are all stored with a name which can be a string or a numerical identifier. If it is numerical, the number must be no larger than an unsigned 16-bit integer (65535, max). Resources are all called by name, that is the system is expecting an unicode string with the name of the resource. However if we use a numerical identifier we need to inform the system that we are doing so, so that it doesn't get confused and try to treat your integer as a string. For that we pass the MAKEINTRESOURCE macro to make. The macro takes a single argument - the numerical identifier - and it returns a string pointer suitable for use in the system. We will demonstrate this later.

2 Chapter 41 on page 183

9.6 Next Chapter

- Message Loop Architecture[3]

3 Chapter 10 on page 45

10 Message Loop Architecture

10.1 WinMain

Programming Windows in C can be a little bit different from what most people are used to. First off, there isn't a **main()** function, but instead there is a **_tWinMain()** function to start your program off. _tWinMain() is defined as a macro in tchar.h as such:

```
#ifdef _UNICODE
#define _tWinMain wWinMain
#else
#define _tWinMain WinMain
#endif
```

This means that Windows functions can be written easily in Unicode or ASCII. Besides the difference in function name, _tWinMain also has different parameters than the standard main function:

```
int WinMain(HINSTANCE hThisInstance,
            HINSTANCE hPrevInstance,
            LPSTR lpszArgument,  // LPWSTR for wWinMain
            int iCmdShow);
```

HINSTANCE objects are references to a program instance. hThisInstance is a reference to the current program, and hPrevInstance is a reference to any previously running versions of this same program. However, in Windows NT, the hPrevInstance data object is always NULL. This means that we can't use the hPrevInstance value to determine if there are other copies of the same program running on your system. There are different ways of checking for other copies of a program running, and we will discuss those methods later.

lpszArgument essentially contains all the information that the argc and argv variables used to show, except that the command-line arguments are not broken up into a vector. This means that if you want to sort individual arguments from the command line, you need to either break them up yourself, or call one of the available functions to break it up for you. We will discuss these functions later.

The last argument, "int iCmdShow" is a data item of type int (integer) that determines whether or not a graphical window should be displayed immediately, or if the program should run minimized.

WinMain has a number of different jobs:

1. Register the window classes to be used by the program
2. Create any Windows used by the program

3. Run the message loop

We will explain each of those tasks in more detail.

10.1.1 Register Window Classes

Every little graphical detail you see on the screen is known as a "window". Each program with a box around it, each button, each text box are all called windows. In fact, all of these various objects all get created in the same manner. This must mean that there are a large number of options and customizations available in order to get things as different as textboxes and scroll bars from the same method. Each window has an associated "Window Class" that needs to be registered with the system, to specify the different properties of the window. This is done by following 2 steps:

1. Fill in the fields of the WNDCLASS data object
2. Pass the WNDCLASS object to the RegisterClass function.

Also, there is an "extended" version of this procedure, that can be followed with similar results:

1. Fill in the fields of the WNDCLASSEX object
2. Pass the WNDCLASSEX object to the RegisterClassEx function.

Either of these methods can be used to register the class, but the -Ex version has a few more options.

Once the class is registered, you can discard the WNDCLASS structure, because you don't need it anymore.

10.2 Create Windows

Creating Windows can be done with the CreateWindow or the CreateWindowEx functions. Both perform the same general task, but again, the -Ex version has more options. You pass some specifics to the CreateWindow function, such as the window size, the window location (the X and Y coordinates), the window title, etc. CreateWindow will return a HWND data object, that is a handle to the newly created window. Next, most programs will pass this handle to the **ShowWindow** function, to make the window appear on the screen. The way CreateWindow() creates a window is as follows:

```
hwnd=CreateWindowEx(
     WS_EX_CLIENTEDGE,
     g_szClassName,
     "The title of my window",
     WS_OVERLAPPEDWINDOW,
     CW_USERDEFAULT,CW_USERDAFAULT,240,120,
     NULL,NULL,hTnstance,NULL);
```

The first parameter WS_EX_CLIENTEDGE is the extended window style. Next we have the class name g_szClassName, this tells the system what kind of window to create. Since we want to create a window from the class we just registered, we use the name of that class. After that we specify our window name or title which is the text that will be displayed in the caption, or title bar on our window. The parameter we have as WS_OVERLAPPEDWINDOW is the Window Style parameter. There are quite a few of these and you should look them up and experiment to find out what they do. The next four parameters (CW_USERDEFAULT,CW_USERDAFAULT,240,120) are the X and Y co-ordinates for the top left corner of your window, and the width and height of the window. The X and Y co-ordinates are se to CW_USERDAFAULT to let the window choose where on the screen to put the window. Next, (NULL,NULL,hTnstance,NULL) we have the parent window handle , the menu handle, the application instance handle, and a pointer handle to window creation data. In windows, the window on the screen is arranged in a hierarchy of the parent and child windows. When one sees a button on a window, the button is the Child and it is contained within the window that is it's Parent. In this example, the parent handle is NULL because we have no parent, this is our main or top level window. The menu is NULL for now since we don't have one yet. The instance handle is set to the value that is passed in as the first parameter of the WinMain(). The creation data that can be used to send additional data to the window that is being created is NULL.

10.3 Message Loop

Once the window is created, the window will interact with the rest of the system by way of **messages**. The system sends messages to the window, and the window sends messages back. Most programs in fact don't do anything but read messages and respond to them!

Messages come in the form of an MSG data type. This data object is passed to the GetMessage() function, which reads a message from the message queue, or waits for a new message from the system. Next, the message is sent to the TranslateMessage function, which takes care of some simple tasks such as translating to Unicode or not. Finally, the message is sent to the window for processing using the DispatchMessage function.

Here is an example:

```
MSG msg;
BOOL bRet;
while( (bRet = GetMessage( &msg, NULL, 0, 0 )) != 0)
{
   if (bRet == -1)
   {
       // handle the error and possibly exit
   }
   else
   {
       TranslateMessage(&msg);
       DispatchMessage(&msg);
   }
}

return msg.wParam;
```

That last line will be explained later.

10.4 The Window Procedure

A window procedure may be named anything you want, but the general prototype for one is as follows:

```
LRESULT CALLBACK WinProc (HWND hwnd,
                          UINT msg,
                          WPARAM wParam,
                          LPARAM lParam);
```

An LRESULT data type is a generic 32-bit data object, that can be type-casted to contain any 32-bit value (including a pointer). The hwnd parameter is a handle to the window itself. The msg data value contains the current message from the operating system, and the WPARAM and LPARAM values contain the arguments for that message. For instance, if a button is pressed on the keyboard, the msg field will contain the message WM_KEYDOWN, and the WPARAM field will contain the actual letter pressed ('A' for instance), and the LPARAM field will contain information on whether or not the CTRL, ALT, or SHIFT buttons are down, and whether the type-matic repeat function has been triggered. Several macros have been defined that are very useful in separating out the WPARAM and LPARAM into different sized chunks:

LOWORD(x)

returns the low 16-bits of the 32-bit argument

HIWORD(x)

returns the high 16-bits of the 32-bit argument

LOBYTE(x)

returns the low 8-bits of the 16-bit argument

HIBYTE(x)

returns the high 8-bits of the 16-bit argument

For instance, to access the 2nd byte in the wParam field, we will use the macros as follows:

```
HIBYTE(LOWORD(wParam));
```

Since the window procedure only has two available parameters, these parameters are often packed with data. These macros are very useful in separating that information out into different fields.

Here is an example of a general Window Procedure, that we will explain:

```
LRESULT CALLBACK MyWinProc (HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
   switch (msg)
   {
       case WM_DESTROY:
           PostQuitMessage(0);
           break;
       default:
           return DefWindowProc (hwnd, msg, wParam, lParam);
   }
   return 0;
}
```

Most window procedures only contain a simple loop that searches for particular messages from the system, and then acts on them. In this example, we are only looking for the WM_DESTROY message, which is the message that the kernel sends to the window when the window needs to close. In response to the WM_DESTROY message, the window calls the PostQuitMessage function, which puts a WM_QUIT message (which is defined as 0) into the message queue. When the message loop (described above) gets the WM_QUIT message, it breaks from the loop, and returns the value from the PostQuitMessage function.

Any message that is not handled by the loop should be (must be) passed to the DefWindowProc function. DefWindowProc will perform some default actions based on the messages received, but it won't do anything interesting. If you want your program to do something, you will need to handle these messages yourself.

10.5 Messages

There are a few other messages that we will talk about later on:

WM_CREATE

Your window receives this message only once, when it is first created. Use this message to perform tasks that need to be handled in the beginning, such as initializing variables, allocating memory, or creating child windows (buttons and textboxes).

WM_PAINT

This message indicates that it is time for the program to redraw itself. Use the graphical functions to redraw whatever is supposed to be on the window. If you don't draw anything, then the window will either be a boring white (or grey) background, or if the background was not erased, will keep whatever image is already shown on it (which looks unstable.)

WM_COMMAND

This is a general message that indicates that the user has done something on your window. Either the user has clicked a button, or the user has selected a menu item, or the user has pressed a special "Accelerator" key sequence. The WPARAM and LPARAM fields will contain some descriptions on what happened, so you can find a way to react to this. If you do not process the WM_COMMAND messages, the user will not be able to click any buttons, or select any menu items, and that will be very frustrating indeed.

WM_CLOSE

The user has decided to close the window, so the kernel sends the WM_CLOSE message. This is the final chance to preserve the window as necessary - if you don't want it closed completely, you should handle the WM_CLOSE message and ensure that it does not destroy the window. If the WM_CLOSE message is passed to the DefWindowProc, then the window will next receive the WM_DESTROY message.

WM_DESTROY

The WM_DESTROY indicates that a given window is removed from the screen and will be unloaded from memory. Normally, your program can post the WM_QUIT message to exit the program by calling PostQuitMessage().

These are some of the most basic messages, and we will discuss other messages as the need arises.

10.6 Next Chapter

- Interfacing[1]

1 Chapter 11 on page 51

11 Interfacing

11.1 The Keyboard

When a key is pressed on the keyboard, the signal travels into the computer, where the kernel gets it. These signals, or "keycodes" as they are called, are raw data that needs to be translated into ASCII characters. The kernel performs this conversion, as well as obtaining other information about the keystroke. The necessary information is encoded into a message, and is sent to the currently active window. Simultaneously, the message is sent to the currently active **caret**.

11.1.1 Cursor Vs Caret

Inside Windows, there is a large terminology problem. So many different things all need to get their own names, so we can program with them, and keep everything straight. A perfect example of this is the difference between the **cursor** and the **caret**. The cursor is the graphical image that represents the mouse. It can either be an arrow for pointing, a hand, an hourglass, or an I-shaped text selector. The **caret**, on the other hand, is the blinking object that is used to enter text. When you type, the letter appears at the caret, and the caret moves forward by 1 space. It is important to keep these terms straight, because if you confuse them inside your program, you could have a lot of debugging to do.

11.1.2 Keypress Messages

There are a few keypress messages that the program can choose to handle. It is important to note that not all of these messages need to be handled in a program, and in fact it is usually a good idea to not handle many of them.

WM_KEYDOWN

The WM_KEYDOWN message indicates that a key has been pressed, or that it has been pressed and held down. If the key is held down, the keyboard goes into "Type-Matic" mode and generates keypress events repeatedly at a certain frequency. The kernel will generate a WM_KEYDOWN message for each of these, with the LPARAM message component containing the number of messages sent by the Kernel in succession, so that your program can choose to ignore some of the messages and not miss any of the information. In addition to the Key count, the LPARAM will contain the following information:

{| class="wikitable"

```
|-
! Bits of LPARAM !! Purpose
|-
| 0-15 || Key Count
|-
| 16-23 || Scan Code
|-
| 29 || Context Code
|-
| 30 || Previous State
|-
| 31 || Key Transition
|}
```

The scan code is the raw binary signal from the keyboard, which may not correspond to the ASCII value of the character. Notice that all buttons on a keyboard generate a scan code, including action buttons (Shift, ALT, CTRL). Unless you are trying to interact with the keyboard in a special way, you want to ignore the scan code. The Context Code determines if the ALT key is pressed at the same time. If the ALT key is pressed at the same time, the Context code is 1. The Previous State is the state of the button before the message was generated. The Key Transition determines whether the key is being pressed, or if it is being released. Most of the fields in the WM_KEYDOWN message can be safely ignored by most programs, unless you are trying to use Type-Matic functionality, or are trying to interface on a low level with the keyboard.

WM_KEYUP

This message is sent when a key that was being pressed has been released. Every key press will generate at least two messages: a Key down (when the button is pressed) and a Key Up (when the button is released). In general for most text-processing applications, the Key Up message can be ignored. The WPARAM is the value of the virtual character code, and the LPARAM is the same as for the WM_KEYDOWN message.

11.1.3 Accelerators

Windows users will no doubt be familiar with some of the common key combinations that are used with large windows programs. CTRL+C copies an object to the clipboard. CTRL+P prints the current document. CTRL+S saves the current document. There are dozens more, and it seems like each program has its own specific key combinations.

These key combinations are known in the Windows world as "Accelerators". A program that uses accelerators will define an **accelerator table**. This table will contain all the different key combinations, and the command identifier that they each map to. When an accelerator is pressed on the keyboard, the program doesn't receive the keypress messages, it instead receives a WM_COMMAND message, with the command identifier in the WPARAM field.

To translate the accelerator keypress combinations into WM_COMMAND messages, the function TranslateAccelerator needs to be used in conjunction with the message loop, as such:

```
while(GetMessage(&msg, NULL, 0, 0))
{
   if(!TranslateAccelerator(&msg))
   {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
   }
}
```

The TranslateAccelerator function will automatically dispatch the WM_COMMAND message to the appropriate window.

11.1.4 Using the Caret

Each program may only have 1 active caret, and worse than that, the entire system may only have 1 active caret on the screen at a time. When using a caret, the programmer needs to take special care to destroy the caret when it is not in use, and to recreate the caret when needed. This can be accomplished relatively easily by creating the caret on the WM_SETFOCUS message (when the window is made active), and by destroying the caret on the WM_KILLFOCUS and WM_DESTROY messages.

11.2 The Mouse

The mouse has more messages associated, because the mouse is capable of more unique tasks than the keyboard is. For instance, the mouse has at least 2 buttons (frequently 3 or more), it often has a trackball, and it can be hovered over objects on the screen. Each of these functions of the mouse can be handled via messages, so we are going to need several messages

11.2.1 Mouse Messages

There are a number of mouse messages that may be handled by the program.

WM_LBUTTONDBLCLK

The user double-clicked the left mouse button.

WM_LBUTTONDOWN

The user pressed the left mouse button.

WM_LBUTTONUP

The user released the left mouse button.

WM_MBUTTONDOWN

The user pressed the middle mouse button.

WM_MBUTTONUP

The user released the middle mouse button.

WM_MOUSEMOVE

The user moved the mouse cursor within the client area of the window.

WM_MOUSEWHEEL

The user rotated or pressed the mouse wheel.

WM_RBUTTONDOWN

The user pressed the right mouse button.

WM_RBUTTONUP

The user released the right mouse button.

In addition, the LPARAM field will contain information about the cursor location, in X-Y coordinates, and the WPARAM field will contain information about the state of the shift and CTRL keys.

The system will handle the graphical mouse movements, so you don't need to worry that your program is going to lock up the mouse. However, the program is capable of changing the mouse cursor, and sending mouse messages to other windows, if needed.

11.3 The Timer

Timers are used to space the flow of a program via pauses, so that a group of actions can be allowed to process before another group interacts with the result. In a strict sense, the Windows Timer is not a user input device, although the timer can send input messages to the window, so it is generally covered in the same manner as the mouse and the keyboard. Specifically, Charles Petzold's famous book, "Programming Windows", treated the timer as an input device.

A common use of a timer is to notify the program of the end of a pause so that it can erase an image previously painted on the screen, such as in screensavers that display various images from one folder. The native timer function, however, is not considered accurate for games or time-critical responses. The DirectX API is preferred for games.

Each time the specified time interval assigned to the Timer elapses, the system sends a WM_TIMER message to the window associated to the Timer.

The new Timer starts timing the interval as soon as it is created by the **SetTimer** function. When you create a Timer you retrieve a unique identifier that can be used by the **KillTimer** function to destroy the Timer. This identifier is also present in the first parameter of WM_TIMER message.

Let's see the functions syntax.

```
UINT_PTR SetTimer(
    HWND hWnd,              //Handle of the window associated  to the timer
    UINT nIDEvent,          //an identifier for the timer
    UINT uElapse,           //the time-out value, in ms
    TIMERPROC lpTimerFunc   //the address of the time procedure (see below)
);

BOOL KillTimer(
    HWND hWnd,              // Handle of the window associated to the timer
    UINT_PTR uIDEvent       // the identifier of the timer to destroy
);
```

If you want to reset an existing timer you have to set the first argument to NULL, and the second to an existing timer ID.

You can process WM_TIMER message in two different ways:

- by processing the WM_TIMER message in the window procedure of the window passed as first argument.

- by defining a TimerProc callback function (fourth argument) that process the message instead of a window procedure.

Let's see the first one.

```
#define IDT_TIMER1 1001
#define IDT_TIMER2 1002

...

SetTimer(hwnd,                  //handle of the window associated to the timer
         IDT_TIMER1,            //timer identifier
         5000,                  // 5 seconds timeout
         (TIMERPROC)NULL);      //no timer procedure, process WM_TIMER in the
 window procedure

SetTimer(hwnd, IDT_TIMER2, 10000, (TIMERPROC)NULL);

...

LRESULT CALLBACK WinProc (HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam){

    ...
   case WM_TIMER:
    switch(wParam)
    {
     case IDT_TIMER1:
       //process the 5 seconds timer
     break;

     case IDT_TIMER2:
       //process the 10 seconds timer
     break;
    }
   ...


/* to destroy the timers */
KillTimer(hwnd, IDT_TIMER1);
KillTimer(hwnd, IDT_TIMER2);
```

Now using a **TimerProc**.

```
VOID CALLBACK TimerProc(
    HWND hwnd,    // handle of window for timer messages
    UINT uMsg,    // WM_TIMER message
    UINT idEvent,    // timer identifier
    DWORD dwTime     // current system time
   );
```

It is called by the system to process an associated Timer's WM_TIMER message. Let's see some code.

```
#define IDT_TIMER1 1001
...

/* The Timer Procedure */
VOID CALLBACK TimerProc(HWND hwnd,
                    UINT uMsg,
                    UINT idEvent,
                    DWORD dwTime)
   {
          MessageBox(NULL, "One second is passed, the timer procedure is called,
 killing the timer", "Timer Procedure", MB_OK);

          KillTimer(hwnd, idEvent);
   }

...

/* Creating the timer */
SetTimer(hwnd, IDT_TIMER1, 1000, (TIMERPROC)TimerProc);

...
```

11.3.1 Timer Messages

The timer only comes with a single message, WM_TIMER, and the WPARAM field will contain the timer ID number.

11.3.2 Timer Modes

11.4 Next Chapter

- Window Creation[1]

1 Chapter 12 on page 57

12 Window Creation

On the Windows operating system, most user-interfacable objects are known as "windows". Each window is associated with a particular class, and once the class is registered with the system, windows of that class can be created.

12.1 WNDCLASS

To register a windows class, you need to fill out the data fields in a WNDCLASS structure, and you need to pass this structure to the system. First, however, you need to provide your class with a name, so that Windows (the system) can identify it. It is customary to define the window class name as a global variable:

```
LPTSTR szClassName = TEXT("My Class");
```

You can name it anything you want to name it, this is just an example.

After you have the class name, you can start filling out the WNDCLASS structure. WNDCLASS is defined as such:

```
typedef struct {
   UINT style;
   WNDPROC lpfnWndProc;
   int cbClsExtra;
   int cbWndExtra;
   HINSTANCE hInstance;
   HICON hIcon;
   HCURSOR hCursor;
   HBRUSH hbrBackground;
   LPCTSTR lpszMenuName;
   LPCTSTR lpszClassName;
} WNDCLASS, *PWNDCLASS;
```

For more information on this structure, see this Microsoft Developer's Network article[1].

Notice the last data field is a pointer to a string named "lpszClassName"? This is where you point to your class name that you've just defined. The field named "hInstance" is where you supply the instance handle for your program. We will break the rest of the fields up into a few different categories.

1 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/windowclasses/windowclassreference/windowclassstructures/wndclass.asp

12.1.1 The HANDLEs

There are a number of different data types in the WNDCLASS structure that begin with the letter "h". As we remember from our discussion of Hungarian notation, if a variable starts with an "h", the variable itself holds a HANDLE object.

HICON hIcon

This is a handle to the icon that your program will use, as located in the top left, and in the taskbar. We will discuss icons more later. However, in our example below, we will use a default value for this item.

HCURSOR hCursor

This is a handle to the standard mouse pointer that your window will use. In our example, we will use a default value for this also.

HBRUSH hbrBackground

This is a handle to a brush (a brush is essentially a color) for the background of your window. Here is a list of the default colors supplied by Windows (these colors will change depending on what 'theme' is active on your computer):

```
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
```

Because of a software issue, a value of 1 must be added to any of these values to make them a valid brush.

Another value that is worth mentioning in here is the "lpszMenuName" variable. lpszMenuName points to a string that holds the name of the program menu bar. If your program does not have a menu, you may set this to NULL.

12.1.2 Extra Fields

There are 2 "extra" data members in the WNDCLASS structure that allow the programmer to specify how much additional space (in bytes) to allocate to the class (cbClsExtra) and to allocate to each specific window instance (cbWndExtra). In case you are wondering, the prefix "cb" stands for "count of bytes".

```
int cbClsExtra;
int cbWndExtra;
```

If you don't know how to use these members, or if you don't want to use them, you may leave both of these as 0. We will discuss these members in more detail later.

12.1.3 Window Fields

There are 2 fields in the WNDCLASS that deal specifically with how the window will operate. The first is the "style" field, which is essentially a set of bitflags that will determine some actions that the system can take on the class. These flags can be bit-wise OR'd (using the | operator) to combine more then one into the style field. The MSDN WNDCLASS documentation[2] has more information.

The next (and arguably most important) member of the WNDCLASS is the lpfnWndProc member. This member points to a WNDPROC function that will control the window, and will handle all of the window's messages.

12.1.4 Registering the WNDCLASS

After the fields of the WNDCLASS structure have been initialized, you need to register your class with the system. This can be done by passing a pointer to the WNDCLASS structure to the RegisterClass function. If the RegisterClass function returns a zero value, the registration has failed, and your system has failed to register a new window class.

12.2 Creating Windows

Windows are generally created using the "CreateWindow" function, although there are a few other functions that are useful as well. Once a WNDCLASS has been registered, you can tell the system to make a window from that class by passing the class name (remember that global string we defined?) to the CreateWindow function.

```
HWND CreateWindow(
   LPCTSTR lpClassName,
   LPCTSTR lpWindowName,
   DWORD dwStyle,
   int x,
   int y,
   int nWidth,
   int nHeight,
   HWND hWndParent,
   HMENU hMenu,
   HINSTANCE hInstance,
```

2 http://msdn2.microsoft.com/en-us/library/ms633576.aspx

```
    LPVOID lpParam
  );
```

(See this MSDN article[3] for more information.)

The first parameter, "lpClassName" is the string associated with our window class. The "lpWindowName" parameter is the title that will be displayed in the titlebar of our window (if the window has a titlebar).

"dwStyle" is a field that contains a number of bit-wise OR'd flags, that will control window creation.

12.2.1 Window Dimensions

The "x" and "y" parameters specify the coordinates of the upper-left corner of your window, on the screen. If x and y are both zero, the window will appear in the upper-left corner of your screen. "nWidth" and "nHeight" specify the width and height of your window, in pixels, respectively.

12.2.2 The HANDLEs

There are 3 HANDLE values that need to be passed to CreateWindow: hWndParent, hMenu, and hInstance. hwndParent is a handle to the parent window. If your window doesn't have a parent, or if you don't want your windows to be related to each other, you can set this to NULL. hMenu is a handle to a menu, and hInstance is a handle to your programs instance value.

12.2.3 Passing Values to the New Window

To pass a value to the new window, you may pass a generic, LPVOID pointer (a 32-bit value) in the lpParam value of CreateWindow. Generally, it is a better idea to pass parameters via this method than to make all your variables global. If you have more than 1 parameter to pass to the new window, you should put all of your values into a struct, and pass a pointer to that struct to the window. We will discuss this in more detail later.

12.3 An Example

Finally, we are going to display a simple example of this process. This program will display a simple window on the screen, but the window won't do anything. This program is a bare-bones program, and it encompasses most of the framework necessary to make any Windows program do anything. Beyond this, it is easy to add more functionality to a program.

3 http://msdn2.microsoft.com/en-us/library/ms632679.aspx

```
#include <windows.h>

LPSTR szClassName = "MyClass";
HINSTANCE hInstance;
LRESULT CALLBACK MyWndProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInstance, LPSTR szCmdLine,
int iCmdShow)
{
   WNDCLASS wnd;
   MSG msg;
   HWND hwnd;

   hInstance = hInst;

   wnd.style = CS_HREDRAW | CS_VREDRAW; //we will explain this later
   wnd.lpfnWndProc = MyWndProc;
   wnd.cbClsExtra = 0;
   wnd.cbWndExtra = 0;
   wnd.hInstance = hInstance;
   wnd.hIcon = LoadIcon(NULL, IDI_APPLICATION); //default icon
   wnd.hCursor = LoadCursor(NULL, IDC_ARROW);   //default arrow mouse cursor
   wnd.hbrBackground = (HBRUSH)(COLOR_BACKGROUND+1);
   wnd.lpszMenuName = NULL;                     //no menu
   wnd.lpszClassName = szClassName;

   if(!RegisterClass(&wnd))                     //register the WNDCLASS
   {
       MessageBox(NULL, "This Program Requires Windows NT",
                        "Error", MB_OK);
       return 0;
   }

   hwnd = CreateWindow(szClassName,
                       "Window Title",
                       WS_OVERLAPPEDWINDOW, //basic window style
                       CW_USEDEFAULT,
                       CW_USEDEFAULT,       //set starting point to default
value
                       CW_USEDEFAULT,
                       CW_USEDEFAULT,       //set all the dimensions to default
value
                       NULL,                //no parent window
                       NULL,                //no menu
                       hInstance,
                       NULL);               //no parameters to pass
   ShowWindow(hwnd, iCmdShow);              //display the window on the screen
   UpdateWindow(hwnd);             //make sure the window is updated correctly

   while(GetMessage(&msg, NULL, 0, 0))      //message loop
   {
       TranslateMessage(&msg);
       DispatchMessage(&msg);
   }
   return msg.wParam;
}

LRESULT CALLBACK MyWndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
   switch(msg)
   {
       case WM_DESTROY:
           PostQuitMessage(0);
           return 0;
   }
```

```
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

12.4 -EX members

The Win32 API gains more functionality with each generation, although Microsoft faithfully maintains the API to be almost completely backwards-compatible with older versions of windows. To add more functionally, therefore, Microsoft needed to add new functions and new structures, to make use of new features. An extended version of the WNDCLASS structure is known as the "WNDCLASSEX" structure, which has more fields, and allows for more options. To register a WNDCLASSEX structure, you must use the RegisterClassEx function instead.

Also, there is a version of the CreateWindow function with extended functionality: CreateWindowEx. To learn more about these extensions, you can do a search on MSDN[4].

12.5 Dialog Boxes

Dialog Boxes are special types of windows that get created and managed differently from other windows. To create a dialog box, we will use the **CreateDialog**, **DialogBox**, or **DialogBoxParam** functions. We will discuss these all later. It is possible to create a dialog box by defining a WNDCLASS and calling **CreateWindow**, but Windows already has all the definitions stored internally, and provides a number of easy tools to work with. For the full discussion, see: Dialog Boxes[5].

12.6 Default Window Classes

There are a number of window classes that are already defined and stored in the Windows system. These classes include things like buttons and edit boxes, that would take far too much work to define manually. Here is a list of some of the pre-made window types:

BUTTON

A BUTTON window can encompass everything from a push button to a check box and a radio button. The "title" of a button window is the text that is displayed on a button.

SCROLLBAR

SCROLLBAR windows are slider controls that are frequently used on the edge of a larger window to control scrolling. SCROLLBAR types can also be used as slider controls.

MDICLIENT

4 `http://search.msdn.microsoft.com/search/`
5 Chapter 15 on page 85

This client type enables **Multiple Document Interface** (MDI) applications. We will discuss MDI applications in a later chapter.

STATIC

STATIC windows are simple text displays. STATIC windows rarely accept user input. However, a STATIC window can be modified to look like a hyperlink, if necessary.

LISTBOX, COMBOBOX

LISTBOX windows are drop-down list boxes, that can be populated with a number of different choices that the user can select. A COMBOBOX window is like a LISTBOX, but it can contain complex items.

EDIT, RichEdit

EDIT windows allow text input with a cursor. Basic EDIT windows also allow for copy+paste operations, although you need to supply the code to handle those options yourself. RichEdit controls allow for text editing and formatting. Consider an EDIT control being in Notepad.exe, and a RichEdit control being in WordPad.exe.

12.7 Menus

There are a number of different menus that can be included in a window or a dialog box. One of the most common (and most important) is the drop-down menu bar that is displayed across the top of a window of a dialog box. Also, many programs offer menus that appear when the mouse is right-clicked on the window. The bar across the top of the window is known as the "Menu Bar", and we will discuss that first. For some information about creating a menu in a resource script, see The Resource Script Reference Page[6], in the appendix to this book.

12.7.1 Menu Bar: From Resource Script

The easiest and most straight-forward method to create a menu is in a resource script. Let's say that we want to make a menu with some common headings in it: "File", "Edit", "View", and "Help". These are common menu items that most programs have, and that most users are familiar with.

> When creating menus, it is polite to create menus with common names, and in a commonly-accepted order, so that computer users know where to find things.

We create an item in our resource script to define these menu items. We will denote our resource through a numerical identifier, "IDM_MY_MENU":

6 Chapter 41 on page 183

```
IDM_MY_MENU MENU DISCARDABLE
BEGIN
   POPUP "File"
   POPUP "Edit"
   POPUP "View"
   POPUP "Help"
 END
```

The keyword **POPUP** denotes a menu that opens when you click on it. However, let's say that we don't want the "Help" menu item to pop up, but instead we want to click on the word "Help", and immediately open the help window. We can change it as such:

```
IDM_MY_MENU MENU DISCARDABLE
BEGIN
   POPUP "File"
   POPUP "Edit"
   POPUP "View"
   MENUITEM "Help"
 END
```

The **MENUITEM** designator shows that when we click on "Help", another menu won't open, and a command will be sent to the program.

Now, we don't want to have empty menus, so we will fill in some common commands in the "File" and "Edit" menus, using the same MENUITEM keyword as we used above:

```
IDM_MY_MENU MENU DISCARDABLE
BEGIN
   POPUP "File"
   BEGIN
       MENUITEM "Open"
       MENUITEM "Save"
       MENUITEM "Close"
   END
   POPUP "Edit"
   BEGIN
       MENUITEM "Cut"
       MENUITEM "Copy"
       MENUITEM "Paste"
   END
   POPUP "View"
   MENUITEM "Help"
 END
```

Now, in the "View" category, we want to have yet another popup menu, that says "Toolbars". When we put the mouse on the "Toolbars" command, a submenu will open to the right, with all our selections on it:

```
IDM_MY_MENU MENU DISCARDABLE
BEGIN
   POPUP "File"
   BEGIN
       MENUITEM "Open"
       MENUITEM "Save"
       MENUITEM "Close"
   END
```

```
    POPUP "Edit"
    BEGIN
        MENUITEM "Cut"
        MENUITEM "Copy"
        MENUITEM "Paste"
    END
    POPUP "View"
    BEGIN
        POPUP "Toolbars"
        BEGIN
            MENUITEM "Standard"
            MENUITEM "Custom"
        END
    END
    MENUITEM "Help"
  END
```

This is reasonably easy, to start with, except that now we need to provide a method for interfacing our menu with our program. To do this, we must assign every MENUITEM with a command identifier, that we can define in a headerfile. It is customary to name these command resources with an "IDC_" prefix, followed by a short text saying what it is. For instance, for the "File > Open" command, we will use an id called "IDC_FILE_OPEN". We will define all these ID tags in a resource header script later. Here is our menu with all the ID's in place:

```
 IDM_MY_MENU MENU DISCARDABLE
 BEGIN
    POPUP "File"
    BEGIN
        MENUITEM "Open", IDC_FILE_OPEN
        MENUITEM "Save", IDC_FILE_SAVE
        MENUITEM "Close", IDC_FILE_CLOSE
    END
    POPUP "Edit"
    BEGIN
        MENUITEM "Cut", IDC_EDIT_CUT
        MENUITEM "Copy", IDC_EDIT_COPY
        MENUITEM "Paste", IDC_EDIT_PASTE
    END
    POPUP "View"
    BEGIN
        POPUP "Toolbars"
        BEGIN
            MENUITEM "Standard", IDC_VIEW_STANDARD
            MENUITEM "Custom", IDC_VIEW_CUSTOM
        END
    END
    MENUITEM "Help", IDC_HELP
  END
```

When we click on one of these entries in our window, the message loop will receive a WM_COMMAND message, with the identifier in the WPARAM parameter.

We will define all our identifiers in a header file to be numerical values in an arbitrary range that does not overlap with the command identifiers of our other input sources (accelerator tables, push-buttons, etc):

```
//resource.h
#define IDC_FILE_OPEN       200
#define IDC_FILE_SAVE       201
#define IDC_FILE_CLOSE      202
#define IDC_EDIT_COPY       203
#define IDC_EDIT_CUT        204
#define IDC_EDIT_PASTE      205
#define IDC_VIEW_STANDARD   206
#define IDC_VIEW_CUSTOM     207
#define IDC_HELP            208
```

And we will then include this resource header both into our main program code file, and our resource script. When we want to load a menu into our program, we need to create a handle to a menu, or an **HMENU**. HMENU data items are identical in size and shape to other handle types, except they are used specifically for pointing to menus.

When we start our program, usually in the WinMain function, we will obtain a handle to this menu using an HMENU data item, with the **LoadMenu** function:

```
HMENU hmenu;
hmenu = LoadMenu(hInst, MAKEINTRESOURCE(IDM_MY_MENU));
```

We will discuss how to use this handle to make the menu appear in another section, below.

12.7.2 Menu Bar: From API Calls

12.7.3 Menu Bar: Loading a Menu

To associate a menu with a window class, we need to include the name of the menu into the WNDCLASS structure. Remember the WNDCLASS structure:

```
typedef struct {
   UINT style;
   WNDPROC lpfnWndProc;
   int cbClsExtra;
   int cbWndExtra;
   HINSTANCE hInstance;
   HICON hIcon;
   HCURSOR hCursor;
   HBRUSH hbrBackground;
   LPCTSTR lpszMenuName;
   LPCTSTR lpszClassName;
} WNDCLASS, *PWNDCLASS;
```

It has a data field called "lpszMenuName". This is where we will include the ID of our menu:

```
WNDCLASS wnd;
wnd.lpszMenuName = MAKEINTRESOURCE(IDM_MY_MENU);
```

Remember, we need to use the MAKEINTRESOURCE keyword to convert the numerical identifier (IDM_MY_MENU) into an appropriate string pointer.

Next, after we have associated the menu with the window class, we need to obtain our handle to the menu:

```
HMENU hmenu;
hmenu = LoadMenu(hInst, MAKEINTRESOURCE(IDM_MY_MENU));
```

And once we have the HMENU handle to the menu, we can supply it to our **CreateWindow** function, so that the menu is created when the window is created:

```
HWND CreateWindow(
   LPCTSTR lpClassName,
   LPCTSTR lpWindowName,
   DWORD dwStyle,
   int x,
   int y,
   int nWidth,
   int nHeight,
   HWND hWndParent,
   HMENU hmenu,
   HINSTANCE hInstance,
   LPVOID lpParam
);
```

We pass our HMENU handle to the hMenu parameter of the CreateWindow function call. Here is a simple example:

```
HWND hwnd;
hwnd = CreateWindow(szClassName, "Menu Test Window!",
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, hmenu,
            hInstance, 0);
```

As a quick refresher, notice that we are using default values for all the position and size attributes. We are defining the new window to be a WS_OVERLAPPEDWINDOW, which is a common, ordinary window type. Also the title bar of the window will say "Menu Test Window!". We also need to pass in the HINSTANCE parameter as well, which is the second-to-last parameter.

12.7.4 Right-Click Menus

12.8 Next Chapter

- User Interface Controls[7]

7 Chapter 13 on page 69

13 User Interface Controls

Some predefined window classes are intended for use as user interface controls. They're commonly known as "standard Windows controls" and "common controls".

Usages of these UI controls should be documented in task-oriented categories, not on an API-oriented basis.

13.1 Standard Windows Controls

Standard Windows controls are implemented in USER32.DLL (former USER.EXE), and later (Windows XP+) subclassified by COMCTL32.DLL for Luna look&feel.

These are always present. No initialization is necessary. Available with Windows 2, ListBox and ComboBox with Windows 3.

13.1.1 Static Control

Static controls are “controls” with *no* user interface.

A static control is a child window that displays either text, bitmap image, or icon. Static controls cannot be selected and do not receive focus from the keyboard. They can; however, receive mouse input with use of *SS_NOTIFY*. This will allow the control to detect click and double-click with the *WM_NOTIFY* event of its parent.

Static Text (Label)

A label should be placed *before* the control it refers. When done so, pressing Alt+Hotkey will move the focus to the next possible control automatically with no line of code.

Labels can also be used for showing the appropriate physical unit right to a Single-Line Edit Control. Because such text never acts as a label, the logical order (Z order) is not important. The style bit SS_NOPREFIX can be used to avoid problems with Ampersand (&) character.

Example Code

Create Label

The label Static Control should be a child of another window, either a main window or a child window. To create it, all you need to do is call CreateWindow() with the STATIC class and the parameters of your choice. Below is the basic code to create it.

```
/*Create a Static Label control*/

hwndLabel = CreateWindow(
                        TEXT("STATIC"),                    /*The name of the
 static control's class*/
                        TEXT("Label 1"),                   /*Label's Text*/
                        WS_CHILD | WS_VISIBLE | SS_LEFT,  /*Styles (continued)*/
                        0,                                 /*X co-ordinates*/
                        0,                                 /*Y co-ordinates*/
                        50,                                /*Width*/
                        25,                                /*Height*/
                        hwnd,                              /*Parent HWND*/
                        (HMENU) ID_MYSTATIC,               /*The Label's ID*/
                        hInstance,                         /*The HINSTANCE of
 your program*/
                        NULL);                             /*Parameters for main
 window*/
```

Using the Label

Setting Label's Text

To set the text of your Label (static control), use the SendMessage() function with the WM_SETTEXT message.

```
/*Setting the Label's text
/*You may need to cast the text as (LPARAM)*/

SendMessage(    hwndLabel ,     /*HWND*/        /*Label*/
                WM_SETTEXT,      /*UINT*/        /*Message*/
                NULL,           /*WPARAM*/      /*Unused*/
       (LPARAM) TEXT("Hello"));  /*LPARAM*/      /*Text*/
```

Image (Bitmap, Icon, Metafile)

With other style bits, bitmaps or icons can be shown – with no line of code when the source image is placed in the same resource.

Metafile support was added with Windows 95.

Rectangles (Divider lines when tall)

Static controls can be used to draw grouping rectangles or divider lines.

13.1.2 Button

Buttons are controls with a *simple* user interface.

Push Button

Everyone should be familiar with the windows push button. It is simply a raised square with text inside it, and when you click usually something happens.

Example Code

As with all windows controls the push button is a child of your window be it the main window or another child. So to implement it into your program you merely need to call the CreateWindow() function. Below is a line of code to create a button.

Create Button

```
// create button and store the handle


HWND hwndButton = CreateWindow (TEXT("button"),                      // The
 class name required is button
                               TEXT("Push Button"),                  // the
 caption of the button
                               WS_CHILD |WS_VISIBLE | BS_PUSHBUTTON,  // the
 styles
                               0,0,                                  // the left
 and top co-ordinates
                               100,300,                              // width
 and height
                               hwnd,                                 // parent
 window handle
                               (HMENU)ID_MYBUTTON,                   // the ID
 of your button
                               hInstance,                            // the
 instance of your application
                               NULL) ;                               // extra
 bits you dont really need
```

this will create the button for you but it will NOT do anything on a click. In order to do that you need to go into your Windows Procedure function and handle the WM_COMMAND event. In the WM_COMMAND event the low word in the wparam is the ID of the child that has caused the event. So to ensure that you have received a message from your button ensure that the ID's are the same.

Test button ID

```
// compare button ID to message ID
if(ID_MYBUTTON == LOWORD(wparam))
{
   /* it's your button so do some work */
}
```

So now you know that your button has been pressed you need to find out what has happened to it so we process the notification code that is stored in the High Word of the wparam. The notification code you need to watch for is BN_CLICKED.

Test notification

```
// compare Notification to message Notification
if(BN_CLICKED == HIWORD(wparam))
{
   /* the button has been clicked do some stuff */
}
```

One last thing you may need to know is that lparam contains the handle to the button pressed.

Ownerdrawn Button

This type can implement anything. In many cases, such buttons are disabled (WS_DISABLED, i.e. *no* user input) and used to draw anything else into a dialog.

Split button

Introduced with Windows Vista, this button has an additional combo box drop-down field, typically to change the behaviour of this push button before pressing.

Checkbox

Checkboxes can be solely or collected to a group. In a group, a multiple-choice selection can be done. Furthermore, a Three-State checkbox is possible. This state should be used for "Don't know", or "Differently checked choices below" as in Windows Setup Control Panel application.

For a good implementation of checkboxes, following rules should be followed at compile time, typically using a Resource Editor:

- All checkboxes should havethe BS_AUTOCHECKBOX style bit set, otherwise, program logic must handle clicks
- A solely checkbox or the first checkbox of a group should have WS_GROUP style bit set
- Other checkboxes should *not* have the WS_GROUP style bit
- Checkbox groups must be contiguously ordered
- Placing WS_TABSTOP is not mantadory. Typically, every checkbox get this style bit set.

If done so, the Arrow keys will move the focus as expected, and the space bar will toggle its state without one line of code.

A check box group should not be used for more than 7 options. If more options are available, or the count is unknown at compile time, a Checked List Box should be used.

Radio Button

Radio buttons are always in a group of at least 2, typically 3..7 choices. They release each-other, and only one can be selected (checked) an one time. Although possible, program logic should avoid that no or more than one radio button is checked. Windows (more precisely, the function IsDialogMessage() wich is automatically processed when calling MessageBox()) can ensure this behaviour with no line of code automatically when:

- All buttons have BS_AUTORADIOBUTTON style bit set
- The first Radio Button of a group has WS_GROUP style

- The other Radio Buttons has *no* WS_GROUP style
- The Radio Buttons are contiguous ordered in dialog template resp. CreateWindowEx() calls

Radio buttons should not have the WS_TABSTOP style bit set. Windows automatically sets the focus on TAB key to the selected radio button.

Radio buttons should not be used for more than 7 options. (This is out of the "3..7 choices rule" for good GUI design.) If more options are available, or the count is unknown at compile time, a combo box should be used.

Example Code
Thats all you need to implement a button in a Win32 Application using the API

Radio Button

```
HWND hRadio =CreateWindow(TEXT("button"), TEXT("Red"),
                WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON,
                20, 155, 100, 30, hDlg, (HMENU)ID_RED, GetModuleHandle(NULL),
 NULL);
```

Group Box

This is *not* a real button and has no interaction at all. It's class name is "BUTTON" too. This element is typically used to visually group radio buttons. It does not create a child element as its .NET counterpart.

Group boxes should be placed *before* its visual content. When done so, pressing Alt+Hotkey will move the focus to the next possible control automatically, with no line of code.

13.1.3 Scroll Bar

Whereas Scroll Bars can be attached to *any* window or control (and is *not* a window with a handle itself), a Scroll Bar *Control* is a true window with a handle. These can be created horizontally (SB_HORZ) or vertically (SB_VERT).

In the days before the Common Controls Progress Bar and Track Bar, someone had “misused” this control to set the volume for a speaker, or to show the progress of a lengthy process. Nowadays, single scroll bars are never used for such purposes.

However, scroll bars are necessary for sharing a Status Bar with a Horizontal Scroll bar, as known from Acrobat Reader or some office software. Because standard scroll bars are always as long as its window, not shorter.

13.1.4 Edit Control

The Edit control is the standard base object for text editing and display (it is commonly called a TextBox).

The Edit control has a quite large number of styles.

Single-Line Edit

Single-line edit controls are typically used for entering short descriptions, file names, parameters, and numbers. Although the style WS_VSCROLL is supported and shows a small vertical scroll bar like an up-down control, nothing else happens. Typically, ES_AUTOHSCROLL is used to move the content horizontally when text does not fit into the given space. The font can be changed programatically, but intermixing fonts and/or colors is not possible. For such purposes, the RichEdit control exists.

Multi-Line Edit

Multi-line edits look like Notepad.exe. Indeed, Notepad is just an Edit control with a frame window that supports the menu, load/save, resizing etc. All other functionality, even the context menu, the Unicode and Right-To-Left support, is already built into this control.

```
// create a text box and store the handle


HWND hwndText = CreateWindow(
                              TEXT("edit"),                              // The
 class name required is edit
                              TEXT(""),                                 //
 Default text.
                              WS_VISIBLE | WS_CHILD | WS_BORDER | WS_HSCROLL |
 WS_VSCROLL| ES_MULTILINE | ES_AUTOHSCROLL, // the styles
                              0,0,                                      // the
 left and top co-ordinates
                              100,300,                                  //
 width and height
                              hwnd,                                     //
 parent window handle
                              (HMENU)ID_MYTEXT,                         // the
 ID of your editbox
                              hInstance,                                // the
 instance of your application
                              NULL
                           );                                   // extra bits
 you dont really need
```

Here are the styles used in the example:

WS_BORDER	A thin border around the text area.
WS_HSCROLL	Display the horizontal scroll bar.
WS_VSCROLL	Display the vertical scroll bar.
ES_MULTILINE	This is a multiline text box.
ES_AUTOHSCROLL	No word wrapping.

```
    // Set the text.
    SendMessage(hwndText, WM_SETTEXT, 0, (LPARAM)"Hello");

    // Get the text.
    LRESULT iTextSize = SendMessage(hwndText, EM_GETLIMITTEXT, 0, 0);
    char *szText = new char[iTextSize];
    SendMessage(hwndText, WM_GETTEXT, iTextSize, (LPARAM)szText);
```

13.1.5 List Box

A list box can have tabulators so it can show somehow tabulated entries. This control is not so commonly used, except in spreadsheet-alike applications. It's main purpose is to deliver the Drop-Down Combo Boxes the pop-up List Box.

13.1.6 Combo Box

This control has three appearances.

Simple Combo Box (non-editable)

This type is very rarely used and not worth an explanation.

Non-editable Drop-Down Combo Box

Use this when the user has to choose out of some options.

On creation, use a large height. It sets the maximum height of the dropped-down list box. If less options are available, Windows will shrink its length automatically, but never extends. And scrolling here is annoying. (Note that Windows 3.x doesn't shink.) The height of the remaining control is 13 dialog units. (You need the number 13 when you combine it with similar Edit controls.)

```
// create a combo box and store the handle


HWND hwndCombo = CreateWindow (TEXT("combobox"),                     // The
 class name required is combobox
                              TEXT(""),                              // not
 used, ignored.
                              WS_CHILD | WS_VISIBLE | CBS_DROPDOWNLIST, // the
 styles
                              0,0,                                   // the
 left and top co-ordinates
                              100,300,                               //
 width and height
                              hwnd,                                  //
 parent window handle
                              (HMENU)ID_MYCOMBO,                     // the
 ID of your combobox
                              hInstance,                             // the
 instance of your application
                              NULL) ;                                //
 extra bits you dont really need
```

The displayed height of the actual widget will be automatically changed depending on the font. The "unused" part of the height will be devoted to the size of the drop down menu.

For example, only 33 pixels of the 300 pixel height will be to the selected item. When the down button is clicked, the menu will be 267 pixels in height.

There are other combo box styles: CBS_SIMPLE (similiar to a list box) and CBS_DROPDOWN (similiar to CBS_DROPDOWNLIST but the selected field is editable).

```
// Add a list of strings to the combo box.

SendMessage(
             hwndCombo,                   // The handle of the combo box
             CB_ADDSTRING,                // Tells the combo box to append
 this string to its list
             0,                           // Not used, ignored.
             (LPARAM) "Item A"            // The string to add.
          );

SendMessage(hwndCombo, CB_ADDSTRING, 0, (LPARAM) "Item B");
SendMessage(hwndCombo, CB_ADDSTRING, 0, (LPARAM) "Item C");
SendMessage(hwndCombo, CB_ADDSTRING, 0, (LPARAM) "Item D");
SendMessage(hwndCombo, CB_ADDSTRING, 0, (LPARAM) "Item E");


// Select the default item to be "Item C".
SendMessage(
             hwndCombo,                   // The handle of the combo b,
             CB_SETCURSEL,                // Tells the combo box to select the
 specified index
             2,                           // The index of the item to select
 (starting at zero)
             0                            // Not used, ignored.
          );
```

Editable Drop-Down Combobox

Use it when the user can enter single-line text *or* can take some predefined strings. A typical example is choosing an URL or number with history function. Or selecting a port address, allowing to enter a yet-not-known port address.

13.2 Common Controls

Common controls were introduced roughly with Windows 3.11, and somehow constantly expanded both in functionality and in appearance. Note that, when using Windows XP Luna style or newer, the standard controls above gets automatically subclassified by comctl32.dll's code. A manifest resource controls this behaviour.

The version and functionality of Common Controls DLL heavily depends on version of Internet Explorer installed.

13.2.1 Header and Linking Requirements

Using Common Controls requires linking to comctl32.lib and including commctrl.h. Whereas some controls are there by default, some other need initialization with InitCommonControlsEx(). It depends on operating system, so it's wise to always call InitCommonControlsEx() with the right bits for the controls your application needs.

The function InitCommonControls() does nothing but ensures that the library is loaded when you don't link to any other Common Controls library function (like CreateToolBar()). Surely, the Windows 3.11 classes were registered at load time of the DLL.

Common controls are typically used like regular dialog elements. The dialog resource template, introduced roughly with Windows 2.0, reserves space for user class names if these are not STATIC, BUTTON, EDIT, LISTBOX, COMBOBOX, or SCROLLBAR, the standard controls above. Therefore, examples may show how to create a child window using CreateWindowEx(), but that's quite uncommon because including them into a dialog template is much easier.

13.2.2 Tool Bar

Introduced: Windows 3.11

13.2.3 Status Bar (Status Strip)

Note that the documentation of CalcEffectiveClientRect() is mostly erraneous: The first UINT/BOOL pair of the array is not used at all.

13.2.4 Date/Time Picker

Introduced: Windows 95

13.2.5 IP Address Input Field

13.2.6 Up/Down Control

Mostly it's attached to an edit window so it looks like one with a tiny vertical scroll bar. Indeed, the trick with the scroll bar was widely used in the days of Windows 3.x. But now it looks worse than this control.

Up/Down controls automate integer counting and limit watching. They can automatically fit to their “buddy” window, so no code for placement is necessary. Edit controls are automatically shrunken horizontally, as if a vertical scroll bar were added.

Possible buddy windows are single-line edits, progress bars, and trackbars.

Unluckily, Win32 Up/Down Controls won't work with decimals, in opposite to the .NET counterpart. So you have to code a bunch of boring lines to implement .NET behaviour.

13.2.7 Tab Control

This control is rarely used directly. The Property Sheet uses it heavily.

13.2.8 Tooltip

Balloon-style tooltips were introduced quite late, with Windows XP. Therefore, most programs still use standard black-on-yellow rectangular tooltips.

13.2.9 List View, Tree View

The Windows Explorer is a perfect example for both controls. The left pane usually shows a directory Tree, and the right pane a file List. Also, the desktop itself is (very similar to a) List View.

```
static const INITCOMMONCONTROLSEX icc={sizeof(icc),ICC_TREEVIEW_CLASSES};
InitCommonControlsEx(&icc);
```

13.2.10 Combo Box Ex

13.2.11 Checked List Box

13.2.12 Progress Bar

What it is

A standard bar graph that displays progress of an item. Shows a graphical representation of amount completed over amount total.

Example code

Create Progress Bar

The Progress Bar control should be a child of another window, either a main window or a child window. To create it, all you need to do is call CreateWindow() with the PROGRESS_CLASS class and the parameters of your choice. Below is the basic code to create it.

```
/*Create a Progress Bar*/

HWND hwndProgress = CreateWindow(
            PROGRESS_CLASS,        /*The name of the progress class*/
            NULL,             /*Caption Text*/
            WS_CHILD | WS_VISIBLE,    /*Styles*/
            0,            /*X co-ordinates*/
            0,            /*Y co-ordinates*/
            200,            /*Width*/
            30,            /*Height*/
            hwnd,             /*Parent HWND*/
            (HMENU) ID_MYPROGRESS,     /*The Progress Bar's ID*/
            hInstance,        /*The HINSTANCE of your program*/
            NULL);            /*Parameters for main window*/
```

Using the Progress Bar

Changing position

There are three ways to increment/decrement the progress bar. PBM_DELTAPOS steps by a given number. PBM_SETPOS sets a specific position. PBM_SETSTEP sets an increment/decrement number, and PBM_STEPIT steps with that number.

Delta Position

PBM_ DELTAPOS advances the progress bar with a number given as the wparam.

```
/*Advance progress bar by 25 units*/

SendMessage(    hwndProgress ,    /*HWND*/    /*Progress Bar*/
        PBM_DELTAPOS,     /*UINT*/    /*Message*/
        25,        /*WPARAM*/    /*Units*/
        NULL)        /*LPARAM*/    /*Unused*/
```

Set Position

PBM_ SETPOS advances the progress bar to the specified position in the WPARAM

```
/*Advances progress bar to specified position (50)*/

SendMessage(    hwndProgress ,    /*HWND*/    /*Progress Bar*/
        PBM_SETPOS,     /*UINT*/    /*Message*/
        50,        /*WPARAM*/    /*Units*/
        NULL)        /*LPARAM*/    /*Unused*/
```

Stepping Position

PBM_ SETSTEP specifies the amount of units to step. *PBM_ STEPIT* advances by the amount of units given with PBM_SETSTEP (default 10 units).

```
/*Advances progress bar by specified units*/

/*If progress bar is stepped when at the progress bar's maximum,
/* the progress bar will go back to its starting position*/

               /*Set the step*/
SendMessage(    hwndProgress ,    /*HWND*/    /*Progress Bar*/
        PBM_SETSTEP,     /*UINT*/    /*Message*/
        1,        /*WPARAM*/    /*Amount to step by*/
        NULL)        /*LPARAM*/    /*Unused*/

               /*Step*/
SendMessage(    hwndProgress ,    /*HWND*/    /*Progress Bar*/
        PBM_STEPIT,     /*UINT*/    /*Message*/
        NULL,        /*WPARAM*/    /*Unused*/
        NULL)        /*LPARAM*/    /*Unused*/
```

13.3 Next Chapter

- GDI and Drawing[1]

13.4 References

MSDN - Windows Controls[2]

1 Chapter 14 on page 81
2 `http://msdn2.microsoft.com/en-us/library/bb773173.aspx`

14 GDI and Drawing

This page will talk about graphics and drawing using the windows GDI libraries.

14.1 Device Contexts

What's in? And their defaults: <table border cellspacing=0 cellpadding=4> GDI objects (one per type)Windows versionSet functionGet function PenZero-width black pen (means 1 pixel width on any scaling)2SelectObject (SelectPen, SelectBrush, SelectFont)GetCurrentObject BrushWhite solid brush2 FontSystem default font2 PaletteSystem default palette (Notes below)3SelectPalette Color space?4SetColorSpace Destination bitmap (not for Metafile contexts)depends on which function creates the DC2SelectObject (SelectBrush) Clipping Regiondepends on which function creates the DC2GetClipRgn (GetClipBox) Numbers Mapping mode, scaling and offset1 unit = 1 pixel, origin = left/top2Get/Set MapMode, WindowOrgEx, WindowExtEx, ViewportOrgEx, ViewportExtEx Transformation matrix (rotation, shearing, scaling, offset)Equality4SetWorldTransform, ModifyTransformGetWorldTransform Text color, also used for black-and-white involving blt operationsblack2SetTextColorGetTextColor Background color, used for text / hatch brush / dotted line background and some blt operationswhite2SetBkColorGetBkColor Miter limit (clipping of sharp polygon corners)10.0F (ten times the line width)4SetMiterLimitGetMiterLimit Brush origin (for aligning a pattern brush)0/02SetBrushOrgExGetBrushOrgEx Enums and boolean switches Enable advanced GDI functionsNo4SetGraphicsModeGetGraphicsMode What's inside or outside a complex polygon (PolyFillMode)Alternate Rule2SetPolyFillModeGetPolyFillMode Text alignmenttop/left2SetTextAlignGetTextAlign Text / hatch brush / dotted line output with opaque background color or notwith background2SetBkModeGetBkMode

How bitmaps are stretched on blt operations?3SetStretchBltModeGetStretchBltMode How pen / brush and surface are combined (boolean operations)Copy pen2SetROP2GetROP2 Something else Path (Prepared lines)empty path4BeginPath, EndPath ...- Context stackempty2SaveDCRestoreDC

To be continued ...

14.2 Brush object

Windows uses brushes to paint colors and fill areas with predefined patterns. Brushes have a minimum size of 8X8 pixels and like pens, have three basic characteristic: size, pattern and color. With their 8X8 pixel minimum size,brushes are said to have a pattern, not a style as pens do. The pattern may be a solid color, hatched, diagonal or any other user definable combination, even a bitmap pattern.

```
case WM_PAINT:
{
    /* This will paint a red rectangle */
    HDC holdBrush;
    HDC hdc = BeginPaint(hwnd, &ps);
    HBRUSH hBrush = CreateSolidBrush(RGB(255,0,0));

    holdBrush = SelectObject(hdc, hBrush);
    Rectangle(hdc, 10, 10, 100, 100);

    /* Memory management has been omitted for brevity */
    EndPaint(hwnd, &ps);
    break;
}
```

What's behind the **brush** object?

On creation, see LOGBRUSH structure	**Modification**
Color (RGB color, can be dithered at realization)	SetDCBrushColor
Style (solid, hatched, or bitmap)	-
Bitmap (for bitmap style)	-
colspan=2 Hidden fields on realization	
Referencing device context (to track whether realization is still valid)	UnrealizeObject
Device-dependent, possibly dithered bitmap, typically 8x8 size	-

Note that a bitmap pattern brush behaves differently from a hatched brush, even when the bitmap looks like the hatch. Hatched brushes are transparent, whereas bitmap brushes are opaque. If a black-and-white bitmap is used, black is replaced by the Device Context's text color, and white is replaced by the background color. (This is exactly the BitBlt() behaviour.) SetBkMode() won't work. However, DC's ROP2 applies.

14.3 Pen object

Pens are used to create borders around shapes you have drawn.

Visible fields, see LOGPEN structure	**Windows version**	**Modification**
Width, in horizontal(!) logical units, 0 = 1 pixel	2	-
Style (solid or somehow dotted)	2	-
User dotting style (for user style selected)	4	-
Color	2	SetDCPenColor
Brush (Content: See above)	4	-
Geometric / Cosmetic flag	4	-
Hint how to draw line ends	4	-
Hint how to draw line joins	4	-

Visible fields, see LOGPEN structure	**Windows version**	**Modification**
Hidden fields		
Referencing device context (to track validity of other hidden fields)	2	UnrealizeObject?
Width in device pixels	2	-
Palette index of given color	2	-
Device-dependent bitmap for filling wide lines	2	-

Lines which result in widths larger than 1 device pixel (not logical units) are drawn as polygons with winding rule. Line colors are never dithered, except ExtCreatePen with a solid brush is used.

14.4 Font object

Fonts are for displaying text and symbols in various styles and sizes. Internal large differences exist for font management between Windows 2 (very basic), 3.1 (introducing TrueType), and 4+ (Unicode; world-transform rotation and mirroring; escapement can differ from rotation).

See LOGFONT structure for visible fields.

Hidden fields include

- Referencing device context
- Device-dependent bitmap font representation

Therefore, selecting a font into a Device Context (SelectObject) can be time-consuming, especially for large Asian fonts and large font sizes, to “paint” all the glyphs out of the TrueType template into the bitmap. Furthermore, as for all GDI objects, it's not a good idea to select one object from one to another context, because invalidating the hidden files can be a time-consuming process. Note that *any* GDI object cannot be selected into more that one DC.

14.5 Basic Drawing

In Windows, drawing is typically handled by the WM_PAINT message. The following is an example of how to draw a red square:

```
case WM_PAINT:
{
    PAINTSTRUCT ps;
    BeginPaint(hwnd, &ps);

    RECT rectangle = {50, 50, 250, 250};
    HBRUSH hbr = CreateSolidBrush(RGB(125, 0, 0));
    FillRect(ps.hdc, &rectangle, hbr);

    DeleteObject(hbr);
    EndPaint(hwnd, &ps);
}
break;
```

Firstly, we create the PAINTSTRUCT variable *ps*. This is a data structure containing information about the painting operation. The next line calls BeginPaint. This initializes *ps*, then fills it with relevant information. For this example, we only need the *hdc* member of *ps*. This is a handle to our window's Device Context. Next, we create a rectangle. This holds the coordinates we're going to paint this rectangle at. The coordinates are relative to the upper-left corner of the window's client area. We also have to create a brush, otherwise Windows won't know what color to paint the rectangle. Finally, we call FillRect, and pass the parameters *ps.hdc*, a pointer to *rectangle*, and *hbr*, our brush. This paints the rectangle directly to our window's device context, and from there it is painted on the screen. After every painting operation, it is necessary to clean up any GDI objects we use, in this case *hbr* and *ps*.

14.5.1 Advanced hint

Because Windows should respond to WM_PAINT and WM_PRINTCLIENT, a more general rule for writing a WM_PAINT handler is this:

```
case WM_PRINTCLIENT: OnPaint((HDC)wParam, NULL); break;
```

```
case WM_PAINT: {
  PAINTSTRUCT ps;
  BeginPaint(hwnd, &ps);
  OnPaint(ps.hdc, &ps.rcPaint);    // It's a good idea to manage the update
area. Other PAINTSTRUCT fields are of less usefulness.
  EndPaint(hwnd, &ps);
}break;
```

```
// Somewhere else
void OnPaint(HDC dc, RECT* rcUpdate) {
 …
}
```

14.6 Metafiles

14.7 Next Chapter

- Dialog Boxes[1]

1 Chapter 15 on page 85

15 Dialog Boxes

People are familiar with dialog boxes. They are the grey windows that pop up on Windows systems to display messages, and allow the user to set parameters. There are 3 types of dialog boxes: modeless, modal, and system modal.

Modal

Modal dialog boxes are generally used inside a program, to display messages, and to set program parameters. Modal dialog boxes come to the front of the screen, and you may not use the program while the modal dialog box is open. to continue using the program, the modal dialog box must be closed.

System Modal

System modal dialog boxes are like modal boxes, except that they supersede the entire desktop area. When a system modal dialog box is open, nothing else on the screen can be clicked or selected.

Modeless

Modeless dialog boxes are able to be deselected, and control can be taken away from a modeless dialog box and transferred to some other window. Modeless dialog boxes are frequently used as a fast and easy way to create a window, without having to register a window class. Modeless dialog boxes are common in the Windows control panel.

15.1 MessageBox

The most simple type of dialog box is the MessageBox function. The MessageBox function takes 4 parameters: a handle to a parent, a message, a title, and an option. If the parent handle is NULL, the message box is modeless. If you provide a handle for a parent window, the MessageBox can become Modal to the parent window.

MessageBox dialog boxes have a number of different options that can be specified: Button types, Icons, modality (modal/modeless), and text justification. These options are specified as bit flags, that can be used by bitwise ORing them together.

15.1.1 Buttons

Message boxes can have standard OK or Cancel buttons, or they can have a "Yes, No, Cancel" configuration, or a number of derivatives. Only one primary button scheme can be used per message box:

- MB_ABORTRETRYIGNORE: The message box contains three push buttons: Abort, Retry, and Ignore.
- MB_CANCELTRYCONTINUE: Same as MB_ABORTRETRYIGNORE, but preferred on Windows 2000/XP.
- MB_OK: The message box contains an "OK" button. This is the default.
- MB_OKCANCEL: The message box contains two push buttons: OK and Cancel.
- MB_RETRYCANCEL: The message box contains two push buttons: Retry and Cancel.
- MB_YESNO: The message box contains two push buttons: Yes and No.
- MB_YESNOCANCEL: The message box contains three push buttons: Yes, No, and Cancel.

To display an icon in the message box, specify one of the following values. In addition, a message box can add an additional "Help" button by specifying the "MB_HELP" flag. A "Default Button", a concept that we will see frequently in this chapter, is the button that is automatically selected when a dialog box is opened. Windows provides the ability to set the default button to any of the buttons on a message box, by using the MB_DEFBUTTON*x* macro. Here is an example:

```
MessageBox(NULL, "This is a Test", "Test", MB_OKCANCEL|MB_HELP|MB_DEFBUTTON2);
```

This will have a message box with an "OK", a "Cancel", and a "Help" button, and the "Cancel" button will be automatically selected.

15.1.2 Icons

A message box may have no icons, or it may have one. You shouldn't specify a message box to have multiple icons. The different icons, according to MSDN are:

- MB_ICONEXCLAMATION: An exclamation point icon appears in the message box.
- MB_ICONWARNING: An exclamation point icon appears in the message box.
- MB_ICONINFORMATION: An icon consisting of a lowercase letter i in a circle appears in the message box.
- MB_ICONASTERISK: An icon consisting of a lowercase letter i in a circle appears in the message box.
- MB_ICONQUESTION: A question mark icon appears in the message box.
  The question mark message icon is no longer recommended because it does not clearly represent a specific type of message and because the phrasing of a message as a question could apply to any message type. In addition, users can confuse the message symbol question mark with Help information. Therefore, do not use this question mark message symbol in your message boxes. The system continues to support its inclusion only for backward compatibility.
- MB_ICONSTOP: A stop sign icon appears in the message box.
- MB_ICONERROR: A stop sign icon appears in the message box.
- MB_ICONHAND: A stop sign icon appears in the message box.

15.1.3 Modality

Finally, the MessageBox can be defined as being Modal, Modeless, or System Modal, by using another identifier: MB_APPLMODAL, MB_SYSTEMMODAL, or MB_TASKMODAL. MB_APPLMODAL is the default value, and only works if a parent window handle was specified to the function. There are a number of other options available, check out MSDN[1] for more details.

15.2 Dialog Box Procedures

Dialog box procedures are slightly different from window procedures. Specifically, they return BOOL values, instead of LRESULT values. Also, dialog boxes do not have a default message processing function, because messages don't always need to be handled. Specifically, Windows manages dialog boxes, and Windows will handle the unused messages. If a dialog box processes a certain message, it should return TRUE. If the message is not processed, the function should return FALSE. Also, Dialog boxes do not get a WM_CREATE message, but instead get a WM_INITDIALOG message. Furthermore, when a dialog box has finished its business, it should call the **EndDialog** function.

Here is an example of a skeleton dialog box function:

```
BOOL CALLBACK MyDlgProc(HWND hDlg, UINT msg, WPARAM wParam, LPARAM lParam)
{
   switch(msg)
   {
      case WM_INITDIALOG:
          return TRUE;
      case WM_DESTROY:
          EndDialog(hDlg, 0);
          return TRUE;
   }
   return FALSE;
}
```

15.3 Creating Modal Dialog Boxes

Once a dialog box procedure has been defined, a dialog box can be created by calling either the **DialogBox** or **DialogBoxParam** function. These functions return an **NRESULT** value, that is the integer number passed to the EndDialog function in the dialog box procedure.

The DialogBox function will not return until the dialog box is closed. This means, essentially, that the program is frozen in time until we close the dialog box. The DialogBox function requires 2 handles: the module instance handle and the handle of the parent window. Also, the DialogBox function requires that a string be passed naming the resource

1 `http://msdn.microsoft.com/en-us/library/ms645505(v=vs.85).aspx`

where the dialog box is defined. The last argument to DialogBox is a pointer to the dialog box procedure function, that you have already defined.

To pass a parameter to a dialog box, the function DialogBoxParam can be used. DialogBoxParam has all the same parameters as the regular version, except it takes a fifth argument as a 32-bit pointer. This 32 bit value will be passed as the LPARAM element of the WM_INITDIALOG message.

15.3.1 Indirect Dialog Boxes

DialogBox and DialogBoxParam both require that the dialog box be defined in a resource. However, if you want to make the dialog box on the fly, you can use the **DialogBoxIndirect** or the **DialogBoxIndirectParam** functions. When defining a dialog box indirectly, we need to fill out a DLGTEMPLATE structure, and pass a pointer to that structure to the function, in place of a resource identifier. The DLGTEMPLATE contains fields for determining some of the characteristics of the dialog box, such as the dimensions and screen location.

The DLGITEMTEMPLATE structure is used to define individual dialog box items. For more information on this subject, search MSDN[2].

15.4 Creating Modeless Dialog Boxes

Modeless dialog boxes are a breed of a different color, and are more like windows than dialog boxes. First, we need to modify the message loop, to ensure that dialog box messages are routed correctly:

```
while(GetMessage(&msg, NULL, 0, 0))
{
   if(!IsDialogMessage(hDlg, &msg))
   {
       TranslateMessage(&msg);
       DispatchMessage(&msg);
   }
}
```

Now, there are 2 ways we can define a message box in a resource script, with a class or without. We will discuss each in turn.

15.5 Without Class

We can define a dialog box in a resource script with the DIALOG keyword. The resource will have an ID associated with it (either a number or a string), and this ID can be passed directly to the CreateDialog function.

2 http://msdn.microsoft.com/en-us/library/ms644997(v=vs.85).aspx

15.6 With Class

If we want to define a modeless dialog box in terms of a window class, we can use a few additions to make the job easier. First, we create a WNDCLASS structure with the information about our dialog box. However, there is one difference, in that we must set the cbWndExtra field to the value DLGWINDOWEXTRA value:

```
wnd.cbWndExtra = DLGWINDOWEXTRA;
```

Then, we register the class like normal. Since we are registering our window classes like normal windows, it shouldn't come as a surprise that Modeless dialog boxes use a regular window procedure, and not a dialog box procedure. Now, Windows identifies classes by name, so we should remember the name of our class. Let's say we named our class "MyDlgClass". We could create a dialog box resource as such:

```
MYDLGCLASS DIALOG DISCARDABLE 100, 100, 200, 200
CAPTION "My Dialog Box"
CLASS "MyDlgClass"
FONT 8, "MS Sans Serif"
BEGIN
...
END
```

Notice the field that says "CLASS"? This is the same string that we used in our WNDCLASS structure to name the class. It is important that these two strings be identical, because Windows needs this string to link the WNDCLASS and the dialog box resource together. Notice also that we used the string "MYDLGCLASS" to identify the dialog resource. This isn't mandatory, but it does make things convenient later on.

Now, instead of calling CreateWindow, we will call the easier-to-use function **CreateDialog**. We do not use the DialogBox function, because CreateDialog returns immediately, and doesn't halt program execution.

Here is an example:

```
HWND hDlg;
hDlg = CreateDialog(hInst, "MyDlgClass", hwndParent, MyDlgProc);
```

Here, we are saying that "hInst" is the instance handle of the application, and "hwndParent" is the handle to the parent window of our dialog box. If the hwndParent parameter is NULL, the dialog box won't have a parent. When the modeless dialog box is finished, it calls "DestroyWindow", not "EndDialog", like a modal dialog box would.

15.7 Common Dialog Boxes

The **Common Dialogs** is a library of functions that automatically produce some of the most common dialog boxes in Windows. This is an effort to make some amount of continuity between different programs, so that each different program doesn't create its own proprietary "File Open" dialog, for instance.

Each Common Dialog generally has a single function that takes a pointer to a structure. This structure is defined specifically for each different control. The common controls can be added to a project by including the <commdlg.h> header file, and linking to the **comdlg32.dll** library.

Some of the common controls available through this library are the "Choose Font" dialog box, the "File open" and "File save" boxes, and the "Color Palette" dialog box.

MSDN[3]

15.7.1 ChooseColor

The **ChooseColor** function brings up the color palette window, and returns a 32-bit color value to your program.

```
BOOL ChooseColor(LPCHOOSECOLOR lpcc);
```

ChooseColor takes a single argument, in the form of a pointer to a CHOOSECOLOR structure. This structure is initialized with various values, and when the function returns, the CHOOSECOLOR structure contains the color value code.

15.7.2 GetOpenFileName and GetSaveFileName

These two functions bring up the familiar file open and file save dialog boxes that are found in nearly every Windows application.

```
BOOL GetOpenFileName(LPOPENFILENAME lpofn);
BOOL GetSaveFileName(LPOPENFILENAME lpofn);
```

Both of these functions take a pointer to an OPENFILENAME structure. This structure controls such things as the file extensions that may be loaded, and the starting path to look in. When the function returns, the structure will contain the name of the file selected. Once your program has this information, you can use the File I/O API to access the file.

15.7.3 ChooseFont

The ChooseFont function brings up a familiar dialog box that allows the user to select a font and various font attributes such as size, underline/bold/italics, color, etc. This function takes a pointer to a CHOOSEFONT structure.

3 `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/userinput/commondialogboxlibrary.asp`

```
BOOL ChooseFont(LPCHOOSEFONT lpcf);
```

15.8 Dialog Box Resources

Dialog boxes can be specified in a resource script, to handle the tricky task of creating all the various child windows (buttons and editboxes, etc.) that a dialog box may contain. This process is described in detail in the Resource Script Reference[4] in the appendix. Here, we will discuss some of the basics of using a resource script to define a dialog box.

15.8.1 The DIALOG keyword

A dialog box resource is specified with the **DIALOG** (must be all caps) keyword. The DIALOG keyword is preceeded by the resource identifier, and is followed by a series of dimension values:

ID_DLGBOX DIALOG X, Y, CX, CY

X and Y are the location coordinates of the upper-left corner of the dialog box, in relation to the upper-left corner of the screen. Remember, all the coordinates start at (0,0) in the upper-left hand corner. The next set of numbers, CX and CY, are the dimensions of the dialog box. These dimensions do not include the title bar (if any), so setting your Y value to 0 will make a dialog box that is only a title bar.

```
    +---------------> X

    ["DialogBox"   [_][O][x]]
+   |                       |
|   |                       |
|   |                       |
|   |                       |
|   |                       |
|   |                       |
v   |                       |
Y   |                       |
    |                       |
    |                       |
    +-----------------------+
```

After the DIALOG declaration, there are a number of other fields that can be filled in, to provide information about your dialog box:

```
ID_DLGBOX DIALOG 100, 100, 200, 150
STYLE WS_OVERLAPPED | WS_CAPTION | WS_VISIBLE
CAPTION "Title Bar Text"
FONT 8, "MS Sans Serif"
```

4 Chapter 41 on page 183

The STYLE declaration contains all the window styles, bitwise OR'd, that you would have used in the WNDCLASS structure, or in the style field of the CreateWindow function. All the same values are available. The CAPTION is the title of the dialog box. The FONT is the point-size and the TrueType font to be used on all the surfaces of the dialog box. Any font and size can be specified, although if the font is too big, your dialog box will be very annoying.

Now, once we have our dialog box sized and shaped the way we want it, we can start to fill it with control buttons and edit boxes, and all sorts of other goodies. First, we use the BEGIN and END tags:

```
ID_DLGBOX DIALOG 100, 100, 200, 150
STYLE WS_OVERLAPPED | WS_CAPTION | WS_VISIBLE
CAPTION "Title Bar Text"
FONT 8, "MS Sans Serif"
BEGIN
  ...
END
```

Next, we can start to fill in the dialog box with buttons, checkboxes, or whatever we want, using the following format:

```
ID_DLGBOX DIALOG 100, 100, 200, 150
STYLE WS_OVERLAPPED | WS_CAPTION | WS_VISIBLE
CAPTION "Title Bar Text"
FONT 8, "MS Sans Serif"
BEGIN
  PUSHBUTTON "OK",  IDOK,      10, 10, 50,  15, WS_TABSTOP
  CHECKBOX "Box 1", IDC_CB1,   10, 30, 50,  15, WS_TABSTOP
  EDITTEXT          IDC_EDIT1, 10, 50, 100, 100
END
```

After the declaration, you may optionally include one or more style flags, to specify how you want a particular control to appear. The WS_TABSTOP identifier specifies which controls can be selected when you press the TAB key on the keyboard. When you press the TAB key, control switches among the dialog box controls in the same order that they are specified in the resource script (top to bottom).

15.9 Next Chapter

- Input/Output[5]

5 Chapter 16 on page 93

16 Input-Output

Many of the previous chapters have attempted to shed some light on the Windows graphical interface, but this chapter is going to start a detour into the inner-workings of the Windows operating system foundations. In this chapter, we are going to talk about Input and Output routines. This includes (but is not limited to) File I/O, Console I/O, and even device I/O.

16.1 File API

Files, like everything else in a windows platform, are managed by handles. When you want to read a file or write to one, you must first open a handle to that file. Once the handle is open, you may use the handle in read/write operations. In fact, this is the same with all I/O, including console I/O and device I/O: you must open a handle for reading/writing, and you must use the handle to perform your operations.

16.1.1 CreateFile

We will start with a function that we will see frequently in this chapter: **CreateFile**. CreateFile is the generic function used to open I/O handles in your system. Even though the name doesn't indicated it, CreateFile is used to open Console Handles and Device Handles as well. As the MSDN documentation says:

```
The CreateFile function creates or opens a file, file stream, directory,
physical disk, volume, console buffer, tape drive, communications resource,
mailslot, or named pipe. The function returns a handle that can be used to
access an object.
```

Now, this is a powerful function, and with the power comes a certain amount of difficulty in using the function. Needless to say, CreateFile is a little more involved than the standard C STDLIB **fopen**.

```
HANDLE CreateFile(
 LPCTSTR lpFileName,
 DWORD dwDesiredAccess,
 DWORD dwShareMode,
 LPSECURITY_ATTRIBUTES lpSecurityAttributes,
 DWORD dwCreationDisposition,
 DWORD dwFlagsAndAttributes,
 HANDLE hTemplateFile);
```

As can be guessed, the "lpFileName" parameter is the name of the file to be opened. "dwDesiredAccess" specifies the desired access permissions for the file handle. In the most basic sense, for a file, this parameter can specify a read operation, a write operation, or an execute operation. However, don't be fooled, there are many many different options that can be used here, for different applications. The most common operations are GENERIC_READ, GENERIC_WRITE, and GENERIC_EXECUTE. These can be bitwise-OR'd to have read+write access, if needed.

File handles can be optionally shared or locked. A shared file can be simultaneously opened and accessed by other processes. If a file is not shared, then other programs attempting to access the file will fail. The "dwShareMode" specifies whether or not the file can be accessed by other applications. Setting dwShareMode to zero means that the file access cannot be shared, and other applications attempting to access the file, while the file handle is open, will fail. Other common values are FILE_SHARE_READ and FILE_SHARE_WRITE which allow other programs to open read handles and write handles, respectfully.

The lpSecurityAttributes is a pointer to a SECURITY_ATTRIBUTES structure. This structure can help to secure the file against unwanted accesses. We will discuss security attributes in a later chapter. For now, you can always set this field to NULL.

The dwCreationDisposition member would be better named "dwCreateMode" or something similar. This bit flag allows you to determine how the file is to be opened, according to different flag values:

CREATE_ALWAYS

Always creates a new file. If the file exists already, it will be deleted, and overwritten. If the file does not exist, it is created.

CREATE_NEW

If the file exists, the function fails. Otherwise, creates a new file.

OPEN_ALWAYS

Opens the file, without erasing the contents, if the file exists. Creates a new file if the file does not exist.

OPEN_EXISTING

Opens the file, without erasing the contents, only if the file exists already. If the file does not exist, the function fails.

TRUNCATE_EXISTING

Opens the file, only if the file exists. When the file is opened, all the contents are deleted, and the file is set to 0 bytes long. If the file does not exist, the function fails. When opening with TRUNCATE_EXISTING, you must specify a GENERIC_WRITE flag as the access mode, or the function will fail.

The dwFileAttributes member specifies a series of flags for controlling File I/O. If the CreateFile function is being used to create something that isn't a File handle, this parameter is not used and may be set to 0. For accessing a normal file, the flag FILE_ATTRIBUTE_NORMAL should be used. However, there are also options for FILE_ATTRIBUTE_HIDDEN, FILE_ATTRIBUTE_READONLY, FILE_ATTRIBUTE_ARCHIVE, etc.

Finally, the hTemplateFile member can be specified if you want the new file handle to mimic the properties of an existing file handle. This can be set to NULL if not used.

16.1.2 ReadFile WriteFile

Once a file handle is opened, ideally we would like to interact with the specified file. We can do this most directly by using the ReadFile and WriteFile functions. Both of them take similar parameters:

```
BOOL ReadFile(
 HANDLE hFile,
 LPVOID lpBuffer,
 DWORD nNumberOfBytesToRead,
 LPDWORD lpNumberOfBytesRead,
 LPOVERLAPPED lpOverlapped);
```

```
BOOL WriteFile(
 HANDLE hFile,
 LPCVOID lpBuffer,
 DWORD nNumberOfBytesToWrite,
 LPDWORD lpNumberOfBytesWritten,
 LPOVERLAPPED lpOverlapped);
```

In both, the hFile parameter is the handle to the file that we obtained with CreateFile. The lpOverlapped parameter is used only for a special I/O mode known as "Overlapped I/O Mode", which we will discuss later. For simple I/O, the lpOverlapped parameter can be set to NULL.

In ReadFile, the lpBuffer is a pointer to a generic buffer to receive the data. This data may not be character data, so we don't call it a LPSTR type. "nNumberofBytesToRead" is the number of bytes that should be read, and "lpNumberOfBytesRead" is the actual number of bytes that were read. If lpNumberOfBytesRead is zero, the file has no more data in it.

In WriteFile, the lpBuffer parameter points to the data that should be written into the file. Again, it isn't specifcally character data. nNumberOfBytesToWrite is the maximum number of bytes to write, and the lpNumberOfBytesWritten returns the number of bytes that were actually written to the file.

16.1.3 CloseHandle

When you are done with a file handle, you should close it with the CloseHandle function. CloseHandle only takes one parameter, the file handle you wish to close. If you do not close your handle, Windows will automatically close the handle when the program closes. However, it is a more expensive operation for Windows to do it for you, and can waste time on your system. It is a good idea to always explicitly close all your handles before you exit your program.

Failure to close a handle is known as "handle leaks", and are a common form of memory leakage that can cause your program, and your entire system, to lose resources and operate more slowly. The handle itself occupies only 32-bits of information, but internally the kernal maintains a large amount of data and storage for every handle. Failure to close a handle means that the kernal must maintain all the associated information about the handle. It also costs the kernel additional time and resources to check through all the old unused handles when it is looking for information about a current handle.

16.1.4 Memory-Mapped Files

Memory-Mapped files provides a mechanism to read and write to a file using regular pointers and array constructs. Instead of reading from the file using ReadFile, you can read from the file using a memory pointer. The system does this by reading in the file to a memory page, and then writing changes to that page onto the physical disk. There is a certain amount of additional overhead to read the file into memory at first, and to write it back after the mapping is completed. However, if there are many accesses to the file, it can be much more convenient in the long run.

16.1.5 Overlapped I/O

"Overlapped" I/O is the term Microsoft uses to describe asynchronous I/O. When you want to do I/O, either to a file or to an external device, you have two options:

Synchronous (non-overlapped)

You request the I/O from the system, and wait till the I/O has completed. The program will stop running until the I/O has completed.

Asynchronous (overlapped)

You send a request to the system, and the system completes that request in parallel with your program. Your program can continue to do processing work, and the system will automatically send notification when your request has been completed.

Synchronous I/O is much easier to use, and is much more straight forward. In synchronous I/O, things happen sequentially, and when the I/O function has returned, you know that the transaction is complete. However I/O is typically much slower then any other operation in your program, and waiting on a slow file read, or a slow communications port can waste lots of valuable time. In addition, if your program is waiting for a slow I/O request, the graphical interface will appear to hang and be non-responsive, which can annoy the user.

Programmers can avoid these delays by using dedicated threads or a thread pool to execute synchronous I/O operations. But threads have significant overhead, and creating too many of them exhausts system resources. Asynchronous I/O avoids this overhead, and is thus the preferrable API for high-performance high-load server applications.

Asynchronous I/O is more complicated to use: It requires the use of the OVERLAPPED structure, and the creation of a handler function that will be called automatically by the system when the I/O is complete. However, the benefits are obvious in the efficiency of the method. Your program can request multiple transactions without having to wait for any of them to complete, and it can also perform other tasks while the system is performing the required task. This means that the programs will appear more responsive to the user, and that you can spend more time on data processing, and less time waiting for data.

16.2 Console API

16.2.1 Allocating a Console

A console can be allocated by calling the **AllocConsole** function. Normally we need not do so if we are creating a "console process" (which contains the **main** function) because they are already attached to a console. However we can create a console for "GUI process" (which entry point is **WinMain**) and perform I/O operation on the newly created console. It should be noted that each process can only be associated with one console. If the process has already attached to a console, calling AllocConsole will return FALSE.

After calling AllocConsole, the Windows Command Prompt window will appear.

A console can be freed by calling **FreeConsole**.

16.2.2 Getting a Console Handle

Upon the creation of console, the standard output, standard input and standard error handles (we call them the "standard devices") will be initialized. These handles are essential for any console I/O operations. They can be obtained by calling **GetStdHandle**, which accepts a parameter specifying the handle of the standard device to be obtained. The parameter can be any of the following:

STD_OUTPUT_HANDLE

Specifies the standard output device, which is used for outputting data to the console.

STD_INPUT_HANDLE

Specifies the standard input device, which is used for reading input from the console.

STD_ERROR_HANDLE

Specifies the standard error device, which is mainly used for outputting error.

If the function succeeded, the return value is the handle to the standard device specified. If failed, it will return INVALID_HANDLE_VALUE.

16.2.3 High Level I/O

The <**stdio.h**> or <**iostream**> (C++ only) header files contain the functions typically used for high level console I/O. The high level I/O are typically "buffered". Such functions including **printf**, **scanf**, **fgets** etc. If we wish to do unbuffered I/O, we can use the **fread** or **fwrite** functions and pass **stdin**, **stdout** or **stderr** to the parameter specifying the standard input, standard output and standard error devices respectively. It is generally not advisable to combine the use of both high level and low level I/O however.

These functions are designed to be portable and act as an abstraction to the low level system I/O functions.

16.2.4 Low Level I/O

The low level console I/O can be done by using several API functions such as **WriteConsole**, **ReadConsole**, **ReadConsoleInput** etc.

```
BOOL WriteConsole(
 HANDLE hConsoleOutput,
 const VOID *lpBuffer,
 DWORD dwNumberOfCharsToWrite,
 LPDWORD lpNumberOfCharsWritten,
 LPVOID lpReserved
);
```

```
BOOL ReadConsole(
 HANDLE hConsoleInput,
 LPVOID lpBuffer,
 DWORD dwNumberOfCharsToRead,
 LPDWORD lpNumberOfCharsWritten,
 LPVOID pInputControl
);
```

Please note that the "Chars" referred is actually the number of **TCHAR**, which can be 2-bytes wide when **UNICODE** is defined. It is NOT the number of bytes.

The ReadConsoleInput can be used to read keystrokes, which can't be done with C or C++ standard library. There are many more functions which provide powerful I/O functions.

16.2.5 Colors and Features

There are many exciting API functions that provide additional controls over the console. One of the more commonly used function is the **SetConsoleTitle** which is used to set the console title text. We can also alter the position of the cursor by using the **SetConsoleCursorPosition** function.

We can output text with different foreground and background colors by **SetConsoleTextAttribute**. We can also change the size of the screen buffer by **SetConsoleScreenBufferSize**.

For an extensive documentation of all the Console API one can consult MSDN.

16.3 Device IO API

Interaction between a program and a device driver can be complicated. However, there are a few standard device drivers that may be used to access standard ports and hardware. In most instances, interacting with a port or a piece of hardware is as easy as opening a handle to that device, and then reading or writing to it like a file. In most instances, these ports and devices can be opened using the CreateFile function, by calling the name of the device instead of the name of a file.

16.3.1 Getting a Device Handle

16.3.2 Device IO Functions

16.3.3 Warnings about Device IO

16.4 Completion Ports

16.5 Next Chapter

- File Management[1]

1 Chapter 17 on page 101

17 File Management

This chapter will discuss some of the details of file and directory management. Some topics discussed will include moving and deleting files and directories, enumerating files, locking files, encrypting files, and accessing the Recycle Bin.

17.1 File Attributes

File attributes can be used to write-protect, hide, and un-hide files, or as behind-the-scenes file maintenance. This information can be accessible using the **GetFileAttributes** function. The attributes are called Read Only, Hidden, Archive, and System, and are described below:

Read Only: A file that is marked Read Only cannot be altered. It can be read, but it cannot be changed or deleted.

Hidden: By default, hidden files do not appear in a directory listing.

Archive: The Archive attribute can be used to selectively back up or copy files; it is most helpful in DOS.

System: System files are files flagged for use by the operating system and are not usually displayed in a directory listing.

17.2 Copying, Moving and Renaming Files

To copy files or directory there are two functions: **CopyFile** and the extended **CopyFileEx**.

```
BOOL CopyFile(
    LPCTSTR lpExistingFileName, // ptr to the name of the file/directory to copy
    LPCTSTR lpNewFileName,    // ptr to filename to copy to
    BOOL bFailIfExists       // flag for operation if file exists
   );
```

lpExistingFileName: is a pointer to a string that contains the file/directory to copy

lpNewFileName: is a pointer to a string with the new file/directory path

bFailIfExists: if this parameter is TRUE and the new file already exists, the function fail; if it is FALSE it will overwrite the existing file/directory.

If the function fail, the return value is zero, otherwise nonzero.

CopyFileEx is a bit more complicated. You can also specify a callback routine that is called each time a portion of the file has been copied. You can also cancel the copy operation, and if you want restart it later.

```
BOOL CopyFileEx(
    LPCWSTR lpExistingFileName,        // pointer to name of an existing file
    LPCWSTR lpNewFileName,        //pointer to filename to copy to
    LPPROGRESS_ROUTINE lpProgressRoutine,    // pointer to the callback function
    LPVOID lpData,              //data to be passed to the callback function
    LPBOOL pbCancel,              //flag that can be used to cancel the
 operation
    DWORD dwCopyFlags              //flags that specify how the file is copied
   );
```

lpProgressRoutine: You can set a callback function, so that it is called every time a portion of the file has been copied (it must be in the PROGRESS_ROUTINE form, see below).

lpData: Data to pass to the callback function (can be NULL).

pbCancel: if this flag is set to TRUE during the copy operation, the operation is cancelled.

dwCopyFlags: Specifies how the file is copied. Can be a combination of the following values

COPY_FILE_FAIL_IF_EXISTS The copy operation fail if lpNewFileName already exists

COPY_FILE_RESTARTABLE The copy progress is tracked in the file. You can restart the copy process later using the same values for lpExistingFileName and lpNewFileName

The definition of the copy progress routine:

```
DWORD WINAPI CopyProgressRoutine(

    LARGE_INTEGER TotalFileSize,            // total file size of the file, in
 bytes
    LARGE_INTEGER TotalBytesTransferred,   // total number of bytes transferred
 since operation started
    LARGE_INTEGER StreamSize,              // size of current stream, in bytes
    LARGE_INTEGER StreamBytesTransferred,    // total number of bytes
 transferred for this stream
    DWORD dwStreamNumber,    // the current stream
    DWORD dwCallbackReason,    // reason for callback
    HANDLE hSourceFile,    // handle to the source file
    HANDLE hDestinationFile,    // handle to the destination file
    LPVOID lpData    // data passed by CopyFileEx
   );
```

You can use the first four parameters to display an indicator showing the percentage completed of the process.

dwCallbackReason: can have the following values

CALLBACK_CHUNK_FINISHED Another part of the data was copied.

CALLBACK_STREAM_SWITCH A stream was created and it is about to be copied. This is the reason when the Copy Routine Callback is called for the first time.

This function must return one of the following values:

PROGRESS_CONTINUE Continue with the copy

PROGRESS_CANCEL Cancel the copy operation and delete the target file

PROGRESS_STOP Stop the operation (it can be restarted later)

PROGRESS_QUIET Continue the copy operation, but don't call the Copy Progress Routine Again

---

Now, let's see how to move and rename files. The function we need are **MoveFile** and **MoveFileEx**. These function are used to both move or rename a file. When you call one of these function the system simply copy the file to the new location (with a different name if you also rename it) and deletes the old file.

```
BOOL MoveFileEx(
    LPCTSTR lpExistingFileName,    // address of name of the existing file
    LPCTSTR lpNewFileName,    // address of new name for the file
    DWORD dwFlags            // flag to determine how to move file
   );
```

17.3 Deleting Files

The API function used to delete files is **DeleteFile** . The file to delete must be closed, or the function will fail. If the function fails, the return value is 0, if it succeeds it's nonzero.

```
BOOL DeleteFile(
        LPCTSTR lpFileName     // pointer to name of file to delete
   );
```

To delete a directory you can use the **RemoveDirectory** function, but first you have to delete all it's files and subdirectoryes (it must be **empty**).

```
BOOL RemoveDirectory(
         LPCTSTR lpPathName     // address of directory to remove
   );
```

17.4 Directories

17.4.1 Directory changes

ReadDirectoryChangesW can be used to watch a directory (or volume). It can pass notifications when some file action is done in the target directory.

17.5 File Enumeration

17.6 Locking Files

17.7 Encrypting Files

17.8 Compressing Files

msdn[1]

17.9 Next Chapter

- Memory Subsystem[2]

1 `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/fileio/fs/file_management_functions.asp`

2 Chapter 18 on page 105

18 Memory Subsystem

C programmers will undoubtedly be familiar with the stdlib memory allocation functions, malloc, realloc, calloc, etc. These functions are based off a number of other functions in the Win32 API that deal with memory segments.

18.1 Windows Memory

When talking about the memory subsystem, there are 4 distinct types of memory that Windows manages, and each type of memory has a number of different functions to allocate and free that memory.

Virtual Memory

The Virtual Memory subsystem allocates and manages memory by pages. This means that memory can only be allocated in blocks of 4 Kbytes (or larger, depending on OS version) at a time. This is a fantastically large amount of memory for most applications, and using the virtual memory functions in most programs is overkill. However, some programs do need use of entire pages for storage, so the Virtual Memory subsystem can be used for that.

Heap Memory

The heap is an area of memory that usually takes up a whole page, or a fraction of a page. Each process is allocated a heap immediately by windows, called the process heap. The stdlib functions such as malloc will allocate memory from this memory region. The heap can be divided up into small memory segments, for use with variables and dynamic storage.

Global Memory

Windows maintains at least 1 page of memory for use as general-purpose global memory. This memory can be read or written by any process running on the computer, using the global memory functions. Data in the global memory space can be shared among various programs, and items sent to the windows clipboard are generally stored in global memory (so it can be "pasted" into any program). Global memory is limited, so it should not be used without a specific need.

LocalMemory

Local memory has similarities to both global memory and heap memory. It is local to the process, but the memory is managed by the Windows memory manager, and not by the program. We will discuss this later.

18.2 Virtual Memory Subsystem

The virtual memory functions, as explained above, allocate memory in terms of pages. Pages are generally 4 Kbytes of memory, so most applications won't need to allocate an entire page (much less more then 1 page). The Virtual memory system is essentially the primitive function base that the other memory functions utilize to perform their tasks. For instance, the heap is comprised of 1 or more pages, and the heap functions will allocate pages using the virtual memory system when needed.

When virtual memory blocks are allocated, they are not actually being utilized, they are simply reserved by the system for future use. Other functions need to be used to segment the virtual memory pages into useful segments. Since virtual memory is allocated by pages, a number of special paging features can be used on virtual memory that can not be used on other types of memory. For instance, pages can be locked (to prevent read/write access), or they can be protected from any particular access mode (read, write, execute).

That said, there are a number of functions in the virtual memory subsystem that can be used:

VirtualAlloc

VirtualFree

Protectiveness

Virtuosity

Virtual Query

hi

18.3 Heap Memory

Each program is provided with a default process heap, but a process may optionally allocate any number of additional heaps, if more storage is needed. The heap functions will manage their virtual memory usage automatically, and therefore heaps can be set to grow if they are being filled up with data. If a heap is allowed to grow automatically, the heap functions will automatically allocate additional pages as needed. On the x86 architecture the heap grows in size towards higher memory addresses.

To use heap memory, a heap must first be allocated (or a handle must be obtained to the default heap). Once you have obtained a handle to a heap, you can pass that handle to the memory allocation functions, to allocate memory from that particular heap.

The stdlib memory functions (malloc, realloc, calloc, free) are all used very similarly to the heap functions, so programmers familiar with the stdlib functions may be able to figure out what many of the heap functions are doing, by examining their names:

HeapCreate

Allocates a heap, and returns a handle to that heap. All the other heap functions will use this handle to uniquely identify the particular heap that you are accessing. By maintaining multiple handles, your program may interface with multiple separate heaps.

HeapDestroy

This function closes a heap handle, and deallocates the heap memory so that other processes can use it.

GetProcessHeap

This function returns a handle to the default process heap. Each program gets a default heap when it is loaded, and in most applications this should be enough space to store data items.

HeapAlloc

Similar to the STDLIB "malloc" function, HeapAlloc allocates storage space on the heap, and returns a pointer to that space.

HeapReAlloc

Similar to the STDLIB "realloc" function, HeapReAlloc reallocates the storage space of a variable to be a different size.

HeapFree

Frees a memory object on the heap. Attempts to use the same pointer to access memory after it has been freed will cause an error.

18.4 Global Memory

Windows maintains a certain amount of global heap memory. This memory is limited compared to regular process heaps, and should not be accessed unless global memory is specially required.

When data has been written to the global memory, you don't get a pointer to that data, but instead you get a handle for that data. Once you give your data to the global memory manager, the system is in charge of it. Remember, a handle is not a pointer, and should never be used as one. The system will manage the memory in the global memory section, moving it between pages, and defragmenting it, et cetera.data doesn,t reside within a single segment

GlobalAlloc

GlobalFree

GlobalDiscard

GlobalLock

GlobalUnlock

GlobalFlags

GlobalSize

GlobalHandle

18.5 Local Memory

Local memory, in this sense, is not the kind of storage that programs utilize internally, on the stack and otherwise. Instead, Windows manages a special section of memory that it dubs to be "Local Memory", and it provides a number of functions to allocate and manage this special memory. Local memory is similar to global memory in the sense that data is written to the local location, and the system returns a handle to that data. The system will manage the data, just like in global memory. The local functions are named very similarly to the global memory functions. However, the global memory and local memory functions should never be mixed. For instance, a global memory handle should never be closed with the LocalFree function.

LocalAlloc

LocalFree

LocalDiscard

LocalFlags

LocalLock

LocalReAlloc

LocalUnlock

LocalHandle

18.6 Next Chapter

- Multitasking[1]

1 Chapter 19 on page 109

19 Multitasking

19.1 Processes and Threads

Current versions of Windows are multitasking operating systems. In this chapter, we will discuss some of the tools and API functions that are involved in multitasking, threading, and synchronization for use in Windows.

First, let's explain a little bit of terminology. A **process** is a single program, with a single entry point, and a single exit point. A **thread** is a part of a process. A process has at least 1 thread; but can have more than 1 thread. When created, processes and threads run automatically and are alloted time slices of execution time by the scheduler in a round-robin fashion. The operating system may activate and deactivate any thread or process at any time. For this reason, we will need to control access to program resources such as global memory and output devices.

If multiple processes are working together, the resulting group is known as a **job**. Jobs can also be managed by Windows.

19.1.1 Managing Processes

- CreateProcess, etc

Jobs

19.1.2 Threads

A few functions are used when dealing with threads, like `CreateThread`, `ResumeThread`, `SuspendThread`, and `TerminateThread`.

If the volatility of threads is disconcerting, Windows also provides an execution object known as a **fiber** that only runs when activated by the parent thread.

CreateThread

The `CreateThread` function accepts a few parameters:

- a pointer to the function to be executed within the thread.
- a pointer to a variable to be passed to the thread's function.

The CreateThread function creates a new thread for a process. The creating thread must specify the starting address of the code that the new thread is to execute. Typically, the starting address is the name of a function defined in the program code. This function takes a single parameter and returns a DWORD value. A process can have multiple threads simultaneously executing the same function.

The following example demonstrates how to create a new thread that executes the locally defined function, ThreadFunc.

```
DWORD WINAPI ThreadFunc( LPVOID lpParam )
{
   char szMsg[80];
   wsprintf( szMsg, "ThreadFunc: Parameter = %d\n", *lpParam );
   MessageBox( NULL, szMsg, "Thread created.", MB_OK );
   return 0;
}
VOID main( VOID )
{
   DWORD dwThreadId, dwThrdParam = 1;
   HANDLE hThread;
   hThread = CreateThread(
       NULL,                        // no security attributes
       0,                           // use default stack size
       ThreadFunc,                  // thread function
       &dwThrdParam,                // argument to thread function
       0,                           // use default creation flags
       &dwThreadId);                // returns the thread identifier

  // Check the return value for success.
  if (hThread == NULL)
     ErrorExit( "CreateThread failed." );
  CloseHandle( hThread );
}
```

For simplicity, this example passes a pointer to a DWORD value as an argument to the thread function. This could be a pointer to any type of data or structure, or it could be omitted altogether by passing a NULL pointer and deleting the references to the parameter in ThreadFunc. It is risky to pass the address of a local variable if the creating thread exits before the new thread, because the pointer becomes invalid. Instead, either pass a pointer to dynamically allocated memory or make the creating thread wait for the new thread to terminate. Data can also be passed from the creating thread to the new thread using global variables. With global variables, it is usually necessary to synchronize access by multiple threads.

Note:
User-Mode Scheduling (UMS) is a light-weight mechanism that applications can use to schedule their own threads. Available only on 64-bit versions of Windows 7 and Windows Server 2008 R2.

Passing Parameters

Thread Local Storage (TLS)

A single process can (usually) spawn 2000 threads. This is because the default stack size allocated by the linker is 1MB per thread. 1MB x 2000 is around 2GB which is the maximum a user-process can access. Following is a sample code which spawn many threads till a limit is reached:

```
// Threads.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
#include <process.h>
#include <conio.h>
#include <windows.h>
unsigned int __stdcall myThread (LPVOID params) {
        printf ("Inside thread");
        Sleep (INFINITE);
        return 0;
}
int _tmain(int argc, _TCHAR* argv[])
{
        int i = 0;
        unsigned int dwThreadId = 0;
        while (true) {
                HANDLE h = (HANDLE) _beginthreadex (NULL, 0, myThread, NULL, 0,
&dwThreadId);
                if (h == NULL) break;
                ++i;
        }
        printf ("\n\nI spawned %d threads", i);
        Sleep (10000);
        return 0;
}
```

Priority

Synchronization

Events

Mutexes

Critical Sections

Spin Locks

Debugging

The naming of a thread, a special feature, that only works with a few windows debuggers, is extremely useful for debugging threads, especially when debugging programs with a lot of threads. It consists in generating a special runtime exception (0x406D1388) which allows the program to pass the name for the thread to the debugger.

```
//! NameThread - Names a threads for the debugger.
//!
//! Usage: NameThread( -1, "MyThreadIsNowNamed" );
//!
void NameThread( unsigned long a_ThreadID, char* a_ThreadName )
{

  typedef struct tagTHREADNAME_INFO
  {
    unsigned long m_Type;// must be 0x1000
    char*         m_Name;// pointer to name (in user addr space)
    unsigned long m_ThreadID;// thread ID (-1=caller thread)
    unsigned long m_Flags;// reserved, must be zero
  } THREADNAME_INFO;

  THREADNAME_INFO info;
  info.m_Type = 0x1000;
  info.m_Name = a_ThreadName;
  info.m_ThreadID = a_ThreadID;
  info.m_Flags = 0;

  __try
  {
    RaiseException( 0x406D1388, 0, sizeof(info)/sizeof(unsigned long),
                    (unsigned long*) &info );
  }
  __except( EXCEPTION_CONTINUE_EXECUTION )
  {
  }

}
```

19.1.3 Fiber

19.2 Next Chapter

- Interprocess Communication[1]

1 Chapter 20 on page 113

20 Interprocess Communication

When we have multiple threads and fibers working together in a single process, or when we have multiple processes working together in a job, we often need to allow processes and threads to communicate with each other. This is done using a series of different tools provided by Windows.

20.1 Pipes

20.1.1 Unnamed Pipes

20.1.2 Named Pipes

20.2 Mailslots

20.3 Sockets

20.4 Next Chapter

- MDI Programs[1]

1 Chapter 21 on page 115

21 MDI Programs

Multiple Document Interface (MDI) applications are a very common and popular type of application. MDI applications allow a single program window to contain multiple open workspaces simultaneously. Creating them isn't particularly tricky, and this chapter will attempt to explain the process in detail.

21.1 Steps to Making an MDI

These are the steps to making an MDI application. We will explain each of them.

1. Register the Frame and Child window classes.
2. Modify the message loop.
3. Create the frame window.
4. Create the MDI Client
5. Create MDI child windows with the client.

21.2 Register the Window Classes

Like any other windows application, we need to register the window classes. The main window (the frame, or the "parent" window) needs to be created. Frame windows are just like any other window, but they generally have a background color of COLOR_APPWORKSPACE. Also, the child windows in an MDI application are not allowed to have their own menus, so the Frame window must manage the application menu. Remember, that different child windows usually require different menu choices (or different menus all together), so the Frame window needs to maintain a current record of the active child window, and set the menu accordingly.

The child window class (all of them) needs to be created with WS_CHILD class. An MDI application may have any number of different types of child windows (a famous example is the use of spreadsheets and charts in Microsoft Excel), and they all need to be registered before use.

21.3 Modify the Message Loop

MDI applications use an almost-normal message loop, but if you want to use accelerators in your program, you need to add a new step:

```
while(GetMessage(&msg, hwndFrame, 0, 0))
{
   if(!TranslateMDISysAccel(hwndClient, &msg) &&
      !TranslateAccelerator(hwndFrame, hAccel, &msg))
   {
       TranslateMessage(&msg);
       DispatchMessage(&msg);
   }
}
```

This way the accelerators get routed to the correct destination among all the different child windows. If you are not using accelerators, you don't need to worry about this step.

21.4 The Frame Window

Frame Windows get created like any other windows. Remember, however, that frame windows are just a backdrop for the action that is happening in the child windows, so you shouldn't get to crazy or fancy with your background artwork.

21.5 The Client Window

You need to create a window of type "MDICLIENT". MDICLIENT windows are defined internally in Windows, so you don't need to worry about what it is or what it does. When you create an MDI client window, you first need to fill out the fields in the CLIENTCREATESTRUCT structure. You pass a pointer to this structure as the LPARAM data field in the CreateWindow function.

21.6 MDI Child Windows

Creating MDI child windows is a little bit different from creating normal windows. To create an MDI child window, you must fill out the MDICREATESTRUCT data structure. The MDICREATESTRUCT is very similar to the WNDCLASS structure, except it is more limited. After creating this structure, you must send it as a message to the MDICLIENT window:

```
hwndChild = (HWND)SendMessage(hwndClient, WM_MDICREATE, 0,
(LPARAM)mdicreatestruct);
```

The message function will return a handle to the newly created child window.

21.7 The "Window" Menu

Many MDI applications will offer a "Window" popup menu on the menu bar, to manage the MDI child windows. This menu frequently has options to "Tile" or "Cascade" the child windows, and it frequently maintains a listing of the currently available child windows in the application. It turns out that the MDI Client window will manage this menu for you, if you just pass it the correct handle to the menu.

21.8 Next Chapter

- Registry API[1]

1 Chapter 22 on page 119

22 Registry API

22.1 The Registry

The **registry** is a system-defined hierarchical central database in x86, x64 and Mobile versions of Microsoft Windows System. The Windows Registry API provides methods for an application to save and query data stored in the registry. Although the registry was present even in 3.x Windows, it was used to store much less data.

The registry stores configuration settings for software, information about operating system configuration, current hardware profile, drivers' settings etc. It was also designed to replace the old-fashioned way of storing application data in .ini files. On 16-bits Windows systems applications used to store data in configuration files with .ini extensions.

Values are grouped and stored in **keys**. The registry stores keys in tree format, with keys containing values and **sub-keys**.

Applications should only modify keys created by itself, and changing or deleting other application's keys and values is not recommended. Also, an application shouldn't modify keys containing important system settings. We should keep these two tips in mind when editing registry (of course there are situations when we want to change some system settings but this is very rare).

22.2 Opening a Key

Keys must be opened before they can be used. To do this, we use either of

RegOpenKeyEx()

To open any key.

RegCreateKeyEx()

To create a key or open it if it exists already.

22.3 Values

The value of a key can be obtained by calling RegQueryValueEx().

The RegSetValueEx() function is used to associate a value with a key.

22.4 Key Hierarchy

22.5 Next Chapter

- Security API[1]

1 Chapter 23 on page 121

23 Security API

This page will talk about the Windows security API.

23.1 Security Overview

API

23.2 User Permissions

23.3 Editing ACLs

23.4 Security Reminders

23.5 Next Chapter

- Winsock[1]

1 Chapter 24 on page 123

24 Winsock

Winsock is the name of the library in Windows that handles the Berkely Socket API. Technically, this library is not part of the Win32 API, although there are some windows-specific issues that need to be considered when programming a Winsock application.

24.1 Making a Winsock Project

You can add Winsock to your programming project by including the <winsock2.h> header file. This header file is for the 32-bit version of the library. For the 16-bit version, include the file <winsock.h>.

winsock.dll is the 16-bit version of the library, and ws2_32.dll is the 32-bit version. You must instruct your linker to link to the appropriate version of the library.

24.2 Initializing Winsock

Before calling any of the Winsock routines, the library must first be initialized by calling the **WSAStartup** function. This function requires a pointer to the WSADATA structure. You do not need to initialize this structure, because the call to WSAStartup will fill in all the fields of the structure. You may, optionally, read the values from this structure, and use the results in your program. This is not necessary, however. WSAStartup also requires that you specify the version of winsock that you wish to use. The most current version of winsock is version 1.1, although the newest version of the library is version 2.0. To specify this parameter, pass the major and minor versions to the MAKEWORD macro. Here is an example:

```
WSADATA wd;
WSAStartup(MAKEWORD(2, 0), &wd);
```

Here is the definition of the WSADATA structure. From this data structure, you can determine some important system metrics, including the version of your library, the maximum number of simultaneous sockets available, etc.

```
typedef struct WSAData {
 WORD wVersion;
 WORD wHighVersion;
 char szDescription[WSADESCRIPTION_LEN+1];
 char szSystemStatus[WSASYS_STATUS_LEN+1];
 unsigned short iMaxSockets;
```

```
    unsigned short iMaxUdpDg;
    char FAR* lpVendorInfo;
  } WSADATA, *LPWSADATA;
```

The definition of WSAStartup is as follows:

```
  int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);
```

This function returns zero on success, and will return non-zero on failure. These error codes can be handled, or the program can abort.

24.3 Exiting Winsock

After a program has completed, it must call WSACleanup, to unregister itself from the listing, and to free any resources in use by the library. WSACleanup takes no parameters, and returns zero on success. A non-zero return value signifies an error in the cleanup process.

24.4 Sockets as Handles

It has been said that unlike UNIX, Win32 does not allow sockets to be read/written using the file I/O functions. This is only partially true. Sockets may not be accessed using the standard-library functions such as fread, fwrite, fprintf, etc. However, if we cast our SOCKET structures to HANDLE structures, we can use the Win32 File I/O API to interface with the sockets. For instance, we can now use ReadFile and WriteFile to write to sockets, and any routines that we have written around these APIs can be used when writing to the network.

Under Win32, do not attempt to cast a SOCKET to a FILE type, and use the stdio.h file functions. This will result in some sort of error (most likely a bad error).

24.5 Advanced Win32 Sockets

Win32 has a full complement of socket functions, including bind, accept, socket, listen and recv. However, Win32 also provides a number of advanced function varieties that allow for advanced operation modes. For instance, using the advanced socket functions allow Overlapped I/O mode, asynchronous modes, events, etc. These functions can be explored on MSDN[1].

1 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/winsock/winsock_functions.asp

25 Microsoft Foundation Classes (MFC)

25.1 Microsoft Foundation Classes (MFC)

In essence, MFC is a SDK[1] interface, a library consisting in a set of classes that act as wrappers around portions of the Windows API[2], so that C++ programmers may program Windows using some concepts of the object-oriented programming (OOP) paradigm and the C++ language (the Win32 API is based on C, as seen in C and Win32 API Section[3] of the book). One should learn the Win32 API or at least have some ideas since some functions are absent from the MFC and would help you to better understand the SDK.

Some tools, such as Microsoft Visual Studio[4], are capable of automatically generating large amounts of MFC skeleton code for use in a project. Because of this, most MFC tutorials or reference materials will teach the subject using the automated Visual Studio tools, and leave out some of the gritty details. In this book where possible we try to be neutral.

MFC was first oriented mostly for enterprise-level programming projects, created in an age most code was done in C and Object Oriented Programming was only in the realm of Smalltalk.

Since the release of Visual Studio 6.0 and the MFC 6.0 little was known of the future support to the MFC since the company was favoring the .NET Framework. Version 7.0, 7.1 and 8.0 were mostly extensions to support the new OSs and to aid developers in migrating to the new framework. Since then information on the future of the MFC could be only extracted from Steve Teixeira, Microsoft Corporation, June 2005 paper - **MFC: Visual Studio 2005 and Beyond**, on the release of Visual Studio 2008 Service Pack 1, Microsoft seems to once again be actively supporting the MFC.

Many users today find it acceptable for a low complexity program to have a memory footprint of 30-80Mb (this is common in Java or .Net applications), low response times or "outside of your control" applications like the internet now provides. It is therefore debatable if the impact of use of MFC in small applications outweighs the benefits the libraries provides. Most of the software made specifically for Windows today uses MFC.

You should prefer the Win32 API SDK, or an alternative wrapper for it, if you do not intend to:

1. **Make use of a complex GUI, use the document/view architecture or complex controls.**
   This will increase the use of system resources (memory use, exe and install size).

1 `http://en.wikipedia.org/wiki/Software%20development%20kit`
2 `http://en.wikipedia.org/wiki/Windows%20API`
3 Chapter 3 on page 11
4 `http://en.wikipedia.org/wiki/Microsoft%20Visual%20Studio`

2. **Use other libraries that depend on the MFC.**
3. **Have a complex install for your application.**
   To distribute MFC-Based project you must build it with static MFC libraries or distribute your project with the needed MFC dlls. There is no guarantee that your customer has the required dlls for your program. Old MFC-Based programs must work with new MFC versions. They should but don't do it always. Your customer will not be very happy if after installing your project some of his old software products begin hanging.

25.1.1 MFC and C++

The MFC design principle is an attempt for simplification. The wrapper classes were designed so to simplify some tasks and automates others. Because of those facts, however, a certain amount of fine-tunable control was lost from the raw Win32 API or excessive automation was archived. The MFC has been recognized as having serious design flaws and inconsistencies and has not been actively maintained. The C++ language and best practices have evolved and today this constitutes a barrier for the utilization of the framework.

As MFC predates the STL standardization in to the C++ language, it implements its own versions of the STL containers, not as complete and even inconsistent, this simplistic solutions the MFC implementations tend to be faster, however you should prefers to use the STL when ever you can, it will make the code more C++ standard and permit easier portability in converting the code to multi-platform.

Multiple Inheritance

The MFC class library does not use and Multiple Inheritance and was not designed with full support for Multiple Inheritance.

Since most MFC classes derive from CObject using Multiple Inheritance will be cause problems of ambiguity (any reference to CObject member functions will have to be disambiguated). Static member functions, including `operator new` and `operator delete` must also be disambiguated.

The best option is to avoid the use of Multiple Inheritance with MFC but take a look at Using C++ Multiple Inheritance with MFC[5] (msdn.microsoft.com) for the needed information on how to bypass the limitations.

25.1.2 MFC Conventions

The MFC uses the Hungarian notation[6]. It uses prefixes, like "m_" to indicate a member variable or "p" to indicate a pointer, and the rest of the name is normally written out in CamelCase (the first letter of each word is capitalized).

CObject as the Root for most MFC Classes

5 http://web.archive.org/web/20090911094049/http://msdn.microsoft.com/en-us/library/62x3wzxy(VS.71).aspx

6 http://en.wikibooks.org/wiki/C%2B%2B%20Programming%2FProgramming%20Languages%2FC%2B%2B%2FCode%2FStyle%20Conventions%23Hungarian%20Notation

All the significant classes in MFC derive from the CObject class. The CObject does not have any member data, but does have some default functionality.

25.2 Using MFC

MFC requires header files that are separate from the standard <windows.h> header file. The core of the MFC system requires the inclusion of **<afxwin.h>**. Other header files that are of some use are **<afxext.h>** (for MFC extensions), and **<afxcmn.h>** (for the MFC common dialog boxes).

Simply changing the header files, unfortunately, is still not enough. The MFC DLL libraries must be linked to by the project, because the DLL files contain the class definitions that will be used throughout every program. To use MFC, you must link to the MFC libraries

25.2.1 stdafx.h

stdafx.h is the standard include for MFC projects - that is, if you create a new MFC project, a stdafx.h will automatically be created for you. It will include all the rest of the necessary MFC header files.

25.2.2 theApp

```
extern CYourAppClass theApp;
```

Use in the header file of your application class and then include it wherever you need to use **theApp**.

You could try the *AfxGetApp* function to get a pointer to **theApp**, an efficient method of accessing members of your application is to make a pointer to **theApp** a member variable of the class which needs it -- for example:

```
class CMyDialog : public CDialog
{

  // other class stuff here...

  // Attributes
  public:
    CMdiApp* m_pApp;
};
```

and make sure you initialize *m_pApp* in the constructor or else will be accessing a **NULL** pointer.

```
CMyDialog::CMyDialog(CWnd* pParent /*=NULL*/): CDialog(CMyDialog::IDD, pParent)
{
          //{{AFX_DATA_INIT(CMyDialog)
          //}} AFX_DATA_INIT
          // Outside the special-format comments above...
          m_pApp = (CMdiApp*)AfxGetApp( );
}
```

and voila! Now any time you need access to your application, you got it!

```
m_pApp->m_nMemberVar;
m_pApp->MemberFunction(nParam1, strParam2);
```

25.3 Getting Started

First and foremost, it must be mentioned, that MFC is not the brand of C++ programming that "looks like C". MFC makes heavy use of the object-oriented features of C++, which can seem "dense" or even unreadable to a new C++ programmer. It is highly recommended that the reader become very familiar with C++ concepts such as classes and hierarchies now, if they are not familiar concepts yet.

The root class for MFC is the **CObject** class. CObject itself does not support multiple inheritance, but derivative classes do. Each application begins in a class derived from **CWinApp**. Every program must have a CWinApp class, and each application may only have one. CWinApp contains a number of functions for initializing an application, and controlling the instance handle (similar to the HINSTANCE member of the WinMain function). Programs that want to display a window must utilize a derivative of the **CWnd** class.

25.4 Basic MFC Program

We will outline here a basic MFC program that will create a simple window, but won't handle any user input. From this basic outline, we will be able to tackle more difficult subjects.

```
#include <afxwin.h>  //basic MFC include

//class derived from CFrameWnd, which is derived from CWnd
class Basic_Window:public CFrameWnd
{
   public:
       Basic_Window()
       {
            Create(NULL, "Basic MFC Window");
            // In some cases you might want to use
            // Create(NULL, _T(":Basic MFC Window"));
       }
};

//class derived from CWinApp, which is the main instance of our application
class MyProgram:public CWinApp
{
      //a pointer to our window class object
      Basic_Window *bwnd;
   public:

      //this is essentially our "entry point"
      BOOL InitInstance()
      {
           bwnd = new Basic_Window();
           m_pMainWnd = bwnd;
           m_pMainWnd->ShowWindow(1);
```

```
            return 1;
        }
};

//the program class instance pointer
MyProgram theApp;
```

As we see here, we are immediately relying on class definitions and inheritance, so it is a good idea for readers who are not completely familiar with these topics to brush up on them before continuing.

25.5 Global Variables

MFC provides a number of global variables, that are instantiated and then utilized in the underlying MFC framework to compile your program. We can see this in our basic example code when we use the variable **m_pMainWnd**.

25.6 Cleanly closing an MFC application

The generic solution is `PostQuitMessage([exit code]);`, but take care to cleanup any lingering resources (closing documents, deallocating memory and resources, destroying any additional windows created etc), on the other hand using `AfxGetMainWnd()->PostMessage( WM_CLOSE );` can be a better method in some situations, since it triggers the correct shutdown sequence. This is especially important on MDI/SDI applications because it gives a chance for documents to prompt for save before exit or for the user to cancel the exit.

25.7 Wait mouse cursor

To display a busy/wait mouse cursor the MFC as added a simple helper class. Instantiate a CWaitCursor class inside a function and a wait cursor is then displayed for the duration of that function, auto destruction of the class will restore the cursor state.

```
CWaitCursor aWaitCursor;
```

25.8 Threads

As it should be expected the MFC also wraps the Win32 thread primitives in special classes and functions of its own. For generic information of threads and C++ see the C++ Programming Wikibook section on Multitasking[7]. The MFC devised threads in worker threads and GUI threads.

7 http://en.wikibooks.org/wiki/C%2B%2B%20Programming%2FThreading

25.8.1 Worker threads

Worker threads are especially useful to perform background tasks, or any asynchronous work that doesn't require user intervention, a print job, calculations, waiting for an event, etc. To create a worker thread, the simplest way is to implement a function that will perform the desisted work, and then create the thread with `AfxBeginThread()` that will use that specific function.

25.8.2 Communication

25.8.3 Exit strategy

Never use `TerminateThread()` unless you have no other choice to guarantee a thread termination. It is poor practice and dangerous.

A proper thread exit can be archived by a normal return from the thread (completion) or by signaling it to return prematurely. Since we are working on Windows and on a 32 bit bound CPUs (making 32 bits accesses atomic), it is considered safe (but not portable) to use a shared `bool` variable to indicate to a thread to exit, any other synchronization methods can be used if available.

Since the only issue regarding thread termination is on aborting a job or exiting the program without having threads running, when dealing with worker threads, in the class that created the thread on the destructor you should signal the thread to abort and then use `WaitForSingleObject()` to wait for the thread to terminate.

26 Classes Hierarchy

26.1 MFC Classes Hierarchy

CObject

The root/base class for most of the MFC libary. Some of the features it makes available to the programmer are serialization support, run-time class information, and object diagnostic output. Any derived class from CObject, can exploit these features.

Class CObject *in AFX.H*

Field Summary

```
static const AFX_CORE_DATA CRuntimeClass classCObject
```

Constructor Summary

```
protected  CObject()

private  CObject( const CObject& objectSrc )

 virtual ~CObject()
```

Method Summary

```
void PASCAL operator delete( void* p )

virtual CRuntimeClass* GetRuntimeClass() const

BOOL IsKindOf( const CRuntimeClass* pClass ) const

BOOL IsSerializable() const

void* PASCAL operator new( size_t, void* p )

void* PASCAL operator new( size_t nSize )

virtual void Serialize( CArchive& ar )
```

CCmdTarget

CWinThread

CWinApp

CWnd

CListCtrl

This class encapsulates the functionality of a list view control, which is a control that displays a collection of items, each consisting of an icon and a label.

Class **CListCtrl** *in AFXCMN.H*

CObject

```
   |
   +--CCmdTarget
         |
         +--CWnd
               |
               +--CListCtrl
```

class CListCtrl extends CWnd

Fields inherited from class CWnd

```
m_hWnd, wndTop, wndBottom, wndTopMost, wndNoTopMost, m_hWndOwner, m_nFlags,
m_pfnSuper, m_nMsgDragList, m_nModalResult, m_pDropTarget, m_pCtrlCont,
m_pCtrlSite, messageMap
```

Fields inherited from class CCmdTarget

```
messageMap, commandMap, dispatchMap, connectionMap, interfaceMap, eventsinkMap,
m_dwRef, m_pOuterUnknown, m_xInnerUnknown, m_xDispatch, m_bResultExpected,
m_xConnPtContainer
```

Fields inherited from class CObject

```
classCObject
```

CToolBar

This class encapsulates the functionality of a control bars that have a row of bitmapped buttons and/or separators. CToolBar objects are usually embedded members of frame-window objects derived from the class CFrameWnd or MDIFrameWnd.

CTreeCtrl

CException

CArray

CFile

CDC

CGdiObject

CFont

CHttpArgList

26.1.1 Subclassing

standard technique for customizing the behavior of a class

Window Subclassing

26.1.2 Window message route

The mechanism by which MFC routes messages is also called the Message Map system that was created to wrap the old C API and reduce the level of complexity for programmers.

```
// a Message Map example

BEGIN_MESSAGE_MAP( ClNOMainFrameWnd, CFrameWnd )
  ON_WM_SYSCOMMAND()
  ON_WM_CREATE()
  ON_WM_SIZE()
  ON_WM_CLOSE()
  ON_WM_MEASUREITEM()
  ON_WM_INITMENUPOPUP()
  ON_WM_MENUCHAR()
  ON_WM_DESTROY()

  ON_REGISTERED_MESSAGE( D_WM_REQUEST, OnMsgReqToShowWnd )

  ON_COMMAND( ID_APP_TRAYCLICK, OnTrayClick )
  ON_MESSAGE( C_WM_TIPACTION, OnTipOfTheDayAction )

END_MESSAGE_MAP()
```

Message map

The message map macros:

ON_COMMAND and ON_BN_CLICKED

the ON_COMMAND and ON_BN_CLICKED are the same, MFC command and control macro is preset to handle the Windows message WM_COMMAND and the notification routing mechanism uses the command ID to decide where to route to. Notifications with control notification code of zero (BN_CLICKED) are interpreted as commands.

ON_UPDATE_COMMAND_UI

the ON_UPDATE_COMMAND_UI macro.

26.2 Component Object Model (COM)

27 COM and ActiveX

People familiar with Windows have doubtless heard a number of different terms used, such as COM, DCOM, COM+, ActiveX, OLE, etc. What precisely are all these things, and how are they related?

27.1 Introduction to COM

Component Object Model (COM) is a binary-interface standard for software componentry introduced by Microsoft in 1993. It is used to enable interprocess communication and dynamic object creation in a large range of programming languages. The term COM is often used in the Microsoft software development industry as an umbrella term that encompasses the OLE, OLE Automation, ActiveX, COM+ and DCOM technologies. COM essentially lays a framework for classes to be created in any language, and to be instantiated into any other languages. A class may be written in C++, and called from VisualBasic, et cetera. Essentially, COM defines an interface that all languages must follow, if they want to participate in COM programs.

27.2 COM Libraries

COM classes, written in a compileable language (such as C++ or VisualBasic) may be compiled into DLL libraries. DLL libraries that contain COM classes are not compatible with DLL libraries written in C that contain regular C functions. However, DLL files may be compiled such that they contain both standard C functions, and COM classes.

Files that contain COM classes are refered to as "COM modules", or "COM Components". Just like with other dynamic linking applications, Windows will manage the COM interface between components automatically, so long as all the modules properly use the COM specifications.

27.3 UUID

The COM system relies on numerical identifiers known as **Universally Unique Identifiers** (UUID). A UUID is a 128 bit number that is essentially guaranteed to be unique COM identifier throughout the entire world. The UUID number is a hash value based on the MAC address of your primary network card (if your computer has one) and the real-time clock value of your processor. This means that the only way to generate 2 UUIDs that are identical would be to generate both at exactly the same time on exactly the same computer. Thankfully, this is impossible to do.

COM components are given a UUID number, so that they can be differentiated from all other COM components, and any given COM module can be identified by a single numeric identifier.

27.4 COM Complaints

28 DCOM and COM+

After the original versions of COM were introduced, there became a clear need to expand the functionality of the COM model, and to make some changes to the general framework. Later versions of COM were given various names, such as DCOM and COM+.

28.1 DCOM

DCOM stands for "Distributed COM", and is a protocol for linking COM components from across a local area network (LAN). DCOM allows COM components to be stored on external "COM servers", and used from other computers in the network.

28.2 MTS

28.3 COM+

COM plus = COM + MTS

28.4 .NET

When the COM design team tried to create a framework that was more easy to use then COM, more portable, and more universal, they created the .NET framework. Originally, .NET was supposed to be introduced as the next version of COM, but instead Microsoft made it into its own entity. COM development has essentially stopped, to focus more attention on the .NET platform.

Because .NET has its roots in COM, it shares many of the guiding principals of COM. First, many different OO languages can be compiled into a single intermediate language. This allows software written in C++, C#, VB.NET, and other .NET languages to be easily interfaced together to create mixed-language programs quickly. In addition, .NET has strong support for distributed computing (a la DCOM). The .NET platform is quickly being standardized, and is available on several different platforms. Therefore, .NET is not strictly a Windows-programming topic, and will likely not be covered much throughout the rest of this book.

29 Device Driver Introduction

29.1 Types of Drivers

Windows device drivers generally come in 2 flavors: **Virtual Device Drivers** (VXD) and **Windows Driver Model** (WDM). VXD style drivers are older, and are less compatible, while WDM drivers are supposed to be fully code-compatible all the way back to Windows 95.

29.2 Driver History

In the old days of DOS, the computer was free land where anything goes. To that end, developers wrote their own hardware drivers, conforming to no specific specification or interface, using real-mode assembly code. With the advent of Windows 3.0, the operating system began to take a more hands-on approach to application management, by creating and maintaining a variety of virtual machines, to execute different programs in different processor contexts. Drivers could no longer exist as non-conformist real-mode DOS drivers, but instead had to mitigate access between multiple programs, running more or less in parallel with each other. Windows 3.0 changed the "real devices" into managed resources known as "virtual devices", and replaced the real-mode drivers with new **virtual device drivers** (VDD).

The Windows NT product line existed as a separate entity from the "regular" windows brand. These two operating systems were completely different in almost every imaginable way, except perhaps that the shells looked similar. Windows NT was a fully-managed operating system, and unauthorized resource accesses were blocked by the NT kernel. This meant that in Windows NT, device drivers needed to interface with the computer through specific methods, while standard windows drivers (Windows 3.0, 3.1, 3.11, 95, 98, Me) could access hardware directly, without any sort of management. The drivers for both systems at this point, were generally written in assembly language, as well.

Realizing that the market was split between Windows and Windows NT, Microsoft saw a need to introduce a single driver model, so that device drivers could be portable between Windows and Windows NT. In addition, Microsoft knew that drivers had to be writable in a higher-level language, like C, in order to be code-compatible for different hardware systems. To meet these needs, Microsoft created the **Windows Driver Model** (WDM). WDM drivers are compiled using the DDK, they are written in C, and they follow exacting specifications that ensure they can be executed on any windows system. This book will attempt to focus on WDM drivers, but will include notes on writing DOS TSR drivers, and VDDs as well.

29.3 Driver Issues

Device Drivers operate in kernel mode so writing, testing, and debugging drivers can be a tricky task. Drivers should always be well tested before they are installed.

Since device drivers do not operate in user mode, the user mode libraries (kernel32.dll, user32.dll, wingdi.dll, msvcrt.dll) are not available to a device driver. Instead, a device driver must link directly to ntoskrnl.exe and hal.dll which provide Native API and executive services.

29.4 Writing a Driver

Device drivers are typically written in C, using the **Driver Development Kit** (DDK[1]). There are functional and object-oriented ways to program drivers, depending on the language chosen to write in. It is generally not possible to program a driver in Visual Basic or other high-level languages.

Because drivers operate in kernel mode, there are no restrictions on the actions that a driver may take. A driver may read and write to protected areas of memory, it may access I/O ports directly, and can generally do all sorts of very powerful things. This power makes drivers exceptionally capable of crashing an otherwise stable system.

The Windows platform DDK comes with header files, library files, and a command-line compiler that can be used to write device drivers in C or C++. There is no graphical interface to the DDK compiler.

29.5 Device Driver Stack

Windows implements device drivers in a highly-modular fashion, and it is important that we discuss some vocabulary before we continue the discussion of driver programming any further. The drivers necessary for any particular device are arranged in a driver stack, and are connected together internally by a singly-linked list, that starts at the bottom of the stack (the **root driver**), and terminates at the highest level driver. Each driver *must* contain at least 2 modules, a root driver, and a **function driver**. This combination, with some optional additions, constitute the whole of what people generally call a complete "device driver". Function Drivers will be the most common type of driver to be written, and will be of a primary focus in this wikibook.

Microsoft realized that certain classes of devices all behave similarly, and it would be a gigantic waste of time for every hardware manufacturer to have to write the entire driver code from scratch. To this end, Windows allows for a type of driver known as a **class driver**. Class drivers are themselves not complete function drivers, but class drivers can be dynamically linked to a regular function driver, and can simplify the development process quite a bit. It is possible to write your own class driver, but 3rd party programmers

1 http://en.wikibooks.org/wiki/Windows_Programming/Device_Driver_Introduction#External_Links

generally don't worry about it. In general, Microsoft will supply the class drivers, and driver developers will tap into those class drivers. This ensures that class drivers are fully microsoft tested and certified, and that they are very versatile.

Another classification of driver is the **filter driver**. There are two general types of filter driver, an **upper filter driver**, and a **lower filter driver**. Upper filter drivers exist in the stack above the function driver, and--as their name implies--they filter the incoming I/O requests. Lower filter drivers are placed in the stack between the function driver and the root driver. Filter drivers are generally implemented as bug fixes, or as quick hack extensions for preexisting drivers.

Here is a general diagram of a driver stack:

```
Upper filter driver
      |
      |
  Function Driver <-------> Class Driver
      |
      |
Lower Filter Driver
      |
      |
  Root driver
      |
      |
   Hardware
```

29.6 Buses and Physical Devices

For simplification, let us use the term "bus" to refer to any place on your computer where information can travel from one place to another. This is a very broad definition, and rightfully so: the term "bus" needs to account for everything from USB, Serial ports, PCI cards, Video outputs, etc. Each bus is controlled by its own root driver. There is a USB root driver, a PCI root driver, and so on.

Let's now consider a mythical construct known as the **root bus**, a structure that all other buses connect into. A root bus object doesn't actually physically exist in your computer, but it is handy to think about it. Plus, the root bus has its own driver. The root bus driver object is responsible for keeping track of the devices connected on any bus in your entire computer, and ensuring that the data gets to where it is all going.

29.7 PnP

Plug-n-Play (PnP) is a technology that allows for the hardware on the computer to be changed dynamically, and the PnP software will automatically detect changes, and allocate important system resources. PnP gets its own root driver, that communicates closely with the Root bus driver, to keep track of the devices in your system.

29.8 Device Namespace, and Named Devices

29.9 "Arbitrary Context"

Drivers execute in the context of whatever thread was running when windows accessed the driver. To this end, we say that drivers execute in an "arbitrary context". Therefore, it is not good practice for a driver programmer to make any assumptions about the state of the processor at the entry point to a driver. There are a few issues that arise with this, so we will discuss them here.

29.9.1 Floating Point Arithmetic

Drivers that want to use MMX or floating point arithmetic may find they are in for some undue difficulty. Because a driver may be entered in any context, at any time, the floating point unit may contain partial results and unhandled exceptions from the user mode program that was interrupted to call the driver. It is not enough to simply save the context and then to restore it, because any unhandled exceptions may become "unhandleable", and raise a system error or a bug check. There are only certain times when microsoft recommends using floating point arithmetic, and we will discuss them later.

29.10 External Links

- Understanding the Windows Driver Model[2] - An introduction to the basic concepts needed for WDM programming
- WDM I/O Concepts[3] - Understanding the I/O concepts needed for WDM programming
- Kernel-Mode Driver Framework[4] - the .ISO download includes the Driver Development Kit (DDK)

2 http://www.scribd.com/doc/61354342/Introduction-to-WDM
3 http://www.scribd.com/doc/61435580/WDM-Input-Output-Concepts
4 http://www.microsoft.com/whdc/driver/wdf/KMDF.mspx

30 The DDK

30.1 What it is

The Windows DDK comes on CD with a number of different options. Specifically, the DDK is capable of writing device drivers for a number of different platforms (Win9x, WinNT, WinServer), and is capable of targeting several different processors (IA-32, IA-64, and Itanium). Installing the necessary components for every platform on every processor can take up allot of harddisk space. In addition, the DDK comes with many examples of how to write everything from parallel port controllers to file system drivers, display drivers, and ethernet drivers. Installing all the examples can also take up a large amount of disk space.

See also: Obtaining the DDK[1]

30.2 Checked and Free Environments

The DDK comes with a number of different batch files, that will create a suitable driver programming environment. The different batch files will create an environment to program for a specific OS/Processor combination. Each different environment will set a number of global variables that the compiler and linker will read to make decisions about the code generation. Compiling a driver using the wrong environment will lead to a driver that may not be compatible with your system (and may destabilize your computer).

For each target platform, there are two environments available: **checked** and **free**. The Checked environment is essentially a debug environment, that will add additional debugging bloat to your compiled drivers, and will perform additional error checking and warnings. The Free environment, however, does not contain any debugging information, and should only be used to compile a fully-debugged device driver.

30.3 The Tools

The DDK comes with the compiler: **cl.exe**, the linker: **link.exe**, and an assembler: **ml.exe**. When writing device drivers, it is recommended that you only use these particular files, and that you do not use any other versions of these programs that you may have on your computer (either from the SDK or Visual Studio). The DDK will also come with a number of header files. Some of the header files you may be familiar with, but some of them will contain function prototypes for kernel-mode libraries, that most programmers are unfamiliar with.

1 Chapter 43 on page 199

30.4 Opaque Members

The DDK defines a number of different data structures for use in device drivers. However, because drivers operate at kernel mode, it is important to realize that many of the fields of these structures are for use only internally to the kernel, and your driver should not alter them. These data fields are known as "Opaque" members of the structure, and should be treated as if they were a "protected" or "private" member of a C++ class. Data fields that are not explicitly mentioned in this Wikibook should be considered suspect, and if there are any outstanding questions about the availability and use of a particular data field, consult MSDN or the DDK documentation.

30.5 Warning

Device drivers should always be thoroughly debugged before installing on a system. If a faulty device driver is installed in the system, it could compromise the kernel at boot, making the entire system unstable, if not completely unusable. Uninstalling a faulty driver can be difficult, especially if you need to perform a rescue boot operation on your computer. Device drivers should be written with care, and fully debugged before installing.

31 Driver Structure

Drivers, unlike user-mode programs, have a number of different entry points, and each entry point is specifically designed to handle a single type of interface. In this chapter, we will talk about the required interfaces, and we will also talk about adding additional interfaces.

31.1 DRIVER_OBJECT

Driver routines all receive a pointer to a DRIVER_OBJECT structure. The driver must fill in the appropriate fields of this structure, so that Windows can learn the whereabouts of the different access routines, and any other information about the driver.

31.2 I/O Request Packets

31.3 Device Object Structures

31.3.1 DeviceExtensions

31.4 DriverEntry and DriverUnload

31.5 StartIo and AddDevice

31.6 Service Routines

32 Driver API

It should come as no surprise that kernel mode drivers cannot use the same functions as user-mode applications. Device Drivers, for this reason, must rely on the available kernel mode functions. This page will talk about some of the available libraries and functions that can be utilized in a driver.

32.1 Function Prefixes

1. include <stdio.h>
2. include <iostream.h>
3. include <conio.h>

class

32.2 Native API

32.3 String Functions

32.3.1 Safe String Library

32.3.2 UNICODE_STRING

32.4 Kernel Mode Functions

33 Programming Shell Extensions

The Windows Shell consists primarily of explorer.exe, the graphical user interface that displays folders, icons, and the desktop. Explorer.exe is written primarily in C++, so to write extension modules is going to require OO programming. However, there are several functions, in <windows.h> that a C program can use to interact with the shell to perform some basic tasks. First, we will describe some of the basic areas of your shell.

33.1 The Shell

explorer.exe, the Windows shell program, has a number of different functions that can be used to cause your program to perform tasks like the shell would. We will run over a few of them here:

33.1.1 ShellExecute

The ShellExecute function takes a file and a pathname as arguments, and essentially performs whatever task the shell would perform if the file in question was double-clicked. For instance, calling a ShellExecute on "MyFile.txt" would open notepad, and would display MyFile.txt. Similarly, calling ShellExecute on a hyperlink will automatically open Internet Explorer (or your default browser) and will open the specified URL.

```
HINSTANCE ShellExecute(HWND hwnd,
                       LPCTSTR lpOperation,
                       LPCTSTR lpFile,
                       LPCTSTR lpParameters,
                       LPCTSTR lpDirectory,
                       INT nShowCmd);
```

hwnd

The handle to the parent window of the operation

lpOperation

The text name of the operation. These are the strings that appear in the menu when you right-click an icon. Some common values are "edit", "run", or "execute".

lpFile

the name of the target file

lpParameters

If lpFile specifies an executable file, the lpParameters field contains the commandline arguments to the executable, if any

lpDirectory

specifies what directory to perform the operation in

nShowCmd

33.2 The Desktop

33.3 The Taskbar

33.4 The System Tray

The system tray is the area in the lower-right hand side of the screen that contains the clock and a number of different icons. Icons can be added to the system tray by using a simple API call. The function call to be used is the **Shell_NotifyIcon** function, and we will explain it here.

```
WINSHELLAPI BOOL WINAPI Shell_NotifyIcon(DWORD dwMessage, PNOTIFYICONDATA
pnid);
```

This function takes 2 arguments. The first argument is a message, and the second argument contains more information on the message. There are 3 messages possible:

```
NIM_ADD    Add a new icon to the system tray
NIM_DELETE Delete an icon from the system tray
NIM_MODIFY Modify an existing icon in the system tray
```

We can see that the second argument is a pointer to the NOTIFYICONDATA structure. This structure contains fields as such:

```
typedef struct _NOTIFYICONDATA {
 DWORD cbSize;
 HWND hWnd;
 UINT uID;
 UINT uFlags;
 UINT uCallbackMessage;
 HICON hIcon;
 WCHAR szTip[64];
} NOTIFYICONDATA, *PNOTIFYICONDATA;
```

cbSize

This should reflect the size of the NOTIFYICON structure. If we have a structure called "nid", we usually assign this member as follows:

```
nid.cbSize = sizeof(NOTIFYICONDATA);
```

hWnd

This field contains the handle of the parent window. When the notify icon is clicked, a corresponding message will be sent to this window.

uID

This is a numerical identifier for the icon in question. A program can have up to 12 different icons at once.

uCallbackMessage

This is the message that will be sent to your window. This message should be WM_USER or greater. The WPARAM field of the message will contain the uID of the icon in question.

uFlags

This member tells the shell which fields are valid fields. If a field does not contain any information, or if you don't want to use a particular field to set or modify a value, do not list them in the uFlags member. The possible values are NIF_ICON, NIF_MESSAGE, and NIF_TIP.

hIcon

a handle to the icon to be displayed in the system tray. must be a 16x16 icon.

szTip

A short string that contains a "tooltip" for the icon. When the mouse is hovered over the icon, the tooltip will be displayed. leave this blank, if you do not want to have a tool tip displayed.

33.5 The Recycle Bin

34 Extending IE

Similarly to explorer.exe, the Microsoft Internet Explorer, iexplore.exe, can also be extended by creating a **Browser Helper Object** (BHO). A BHO may be a single toolbar button, an entire toolbar, or even any number of other objects in the browser. BHOs are generally programmed using C++.

35 Programming Screen-savers

Screensavers are graphical programs that will exercise the computer's screen while the computer is not in use, to prevent damage to the screen. Screensavers can also be very aesthetic, interesting, and entertaining however.

35.1 Requirements

Screensaver programs are essentially normal executable programs, with a few small differences:

1. They have a .scr extension, instead of a .exe
2. They have a ScreenSaverProc, instead of a WindowProc.
3. They call DefScreenSaverProc, instead of DefWindowProc.

Also, screensavers must contain a configuration dialog box, that the shell can call when you click on the screensaver and select "properties". The last requirement is that a screensaver must have a string resource, at resource 1, with a description of the screensaver.

35.2 How they Work

Windows will send a number of different arguments to a screensaver, on the commandline, depending on what mode of operation the screensaver must take. Here are some of them:

- /a <hwnd>
- /s
- /c <hwnd>
- /p <hwnd>

The -a option tells the screensaver program to set the associated password, if the screensaver is selected to be password protected. The -s option tells the screensaver to run the graphics, and start saving the screen. The -c option tells the screensaver to display the configuration dialog, and the -p option tells the screensaver to run in preview mode.

The WinMain function should decode the command line, and obtain both the switch (a, s, p, c) and the handle, if specified. The Handle value will be in an ascii string, so that will need to be converted to an integer, and stored into an appropriate HWND variable. The handle provided is the handle to the parent window of the screensaver, which is usually the screensaver tab in the Windows Display dialog box. The screensaver doesn't have a parent window if it is in screensaving mode.

If WinMain has received a -s or a -p, the WinMain should call the ScreenSaverProc. If it has received the -c option, WinMain should call the Dialog box procedure function. If the screensaver receives the -a option, it should bring up a dialog to change the password.

35.3 Starting a Screensaver

36 Programming CMD

In Windows NT (2000, XP, Vista, 7, etc.) one is able to write batch files that are interpreted by the Command Prompt (cmd.exe). They can be used to automate file-system tasks such as backups or basic installations and can be used with other command-line utilities as well. The batch files can be considered to be a simple scripting language with logic and jumps. The advantages of using batch files are the ease of writing them, the ability to edit the files without compiling anything, their cross-compatibility across Windows NT Operating Systems and their inherent ability to manipulate file systems due to their basis on MS-DOS. Batch file scripts are not case-sensitive, although strings and data is. The file is controlled by a list of commands separated into lines which are run like normal commands would be at the Command Prompt, although some functionality is different. Batch files can be run from Windows Explorer but the console used to display them closes automatically at the end of the batch file, so a command at the end that prevents instantaneous exit is needed for any remaining output to be read before the console window closes. Although the batch files are able to manipulate their environment, such as color settings and environment variables, the changes are all temporary, as in a standard Command Prompt session. Color settings, however, are retained on later editions of Windows NT. In order to try to learn about batch files, it is useful to understand Command Prompt commands. See: Guide to Windows commands[1].

36.1 Batch File

The script is kept inside a batch file, with the extension .bat or .cmd. Although .bat is more recognisable, as it was used in the MS-DOS environment that preceded the Command Prompt, the Command Prompt's interpretations of batch files is very different to the manner of interpreting DOS batch files, and .cmd files are only interpreted by the Command Prompt, so using the .cmd extension prevents mis-use in older environments.

Execution starts at the top of the file and ends at the end. When the end of the file is reached, the file exits to the Command Prompt if it was invoked from there, or the console window closes if it was invoked from Windows Explorer or the START command.

36.2 ECHO Command

Typically, batch files start with the 'echo off' command, which stops the input and prompt from being displayed during execution, so only the command output is displayed. The '@'

1 `http://en.wikibooks.org/wiki/Guide%20to%20Windows%20commands`

symbol prevents a command from having input and its prompt displayed, so it is used on the 'echo off' command to prevent that first command from displaying the input and prompt:

```
@ECHO OFF
```

In order to print lines, the ECHO command is used again, but this time with a text parameter other than 'off':

```
ECHO.Hello World!
```

The period ('.') is used to prevent confusion with the attempt to output ON or OFF rather than turn input and prompt displaying on and off. The last line in a code should then be:

```
ECHO ON
```

'Echo on' will turn the input/prompt display back on, just in case the program exits to the Command Prompt where without the command 'echo on', there will be no visible prompt left for use.

36.3 Hello World Example

Using the code above, we can make a hello world program like so:

```
@ECHO OFF
ECHO.Hello World!
ECHO ON
```

36.4 Comments

In batch files there are two ways of writing comments. Firstly there is the form:

```
REM Comment here.
```

This form is included as it was in the MS-DOS batch file script. The other form is this:

```
::Comment here.
```

This form is generally favoured, for being faster to execute and write, and also for being easy to differentiate from normal commands. For this type of comment only two double-colons ('::') are needed and the comment ends at the end of the line. Batch files have no multi-line comment types.

You can also add a comment to the end of the command:

```
command &::Comment here.
```

36.5 Variables

We access variables with the SET command. SET lets us set the value for a variable, and delete a variable. The variables used are environment variables, which are set up to correspond to the system's environment variables, although they are not the actual environment variables of the system and changing them will not change system environment variables. For example:

```
SET name=John Smith
```

This command creates an environment variable called name, and sets its value to the string "John Smith". The first space is ignored as the value entered records from the first non-whitespace character encountered after the '=' sign.

'Set' also allows the storing of integers specifically, using the /A parameter:

```
SET /A number=38
```

This command creates an environment variable named number with the integer value 38, not the string value "38". This number can be involved in arithmetic and numerical logic, but without the /A parameter all that is stored are the characters '3' and '8', which can't be used as numbers.

Variables are accessed by surrounding the name with '%'s. This substitutes the value straight into a command, and does not point to the variable. For example:

```
@ECHO OFF

SET name=John Smith
ECHO %name%

ECHO ON
```

This file would return the text 'John Smith'. A string number can be converted to an integer number in a similar fashion:

```
@ECHO OFF

SET num=38
SET /A num=%num%

ECHO ON
```

This code creates a string called num, then overwrites it with an integer with the numerical value represented by the string contents of num, in this case 38. 'num' is now an integer number ready for use as a number.

36.6 Input

The set command can also be used for input:

```
SET /P var=Enter a value for var:
```

This command displays "Enter a value for var:" and when the user enters the data, var is given that value.

Be aware, if the user presses enter without entering anything then the value in var is unchanged, so for the sake of a prompt it is often best to give a default value, or clear the value for the variable first if it has been used before:

```
SET var=
SET /P var=Enter a value for var:
```

Below is an example:

```
@ECHO OFF
SET /P answer = Enter name of file to delete:
DEL /P %answer%
ECHO ON
```

This batch file gets the name of a file to delete and then uses the DEL command with the prompt parameter '/P' to ask the user if they're sure they want to delete the file.

36.7 Flow Control

36.7.1 Conditionals

IF

The IF command can be used to create program logic in batch files. The IF command allows three basic checks, on the ERRORLEVEL, the equality of two strings, and the existence of a file or folder. The first check on the ERRORLEVEL will check to see if it is greater than or equal to a certain number:

```
IF ERRORLEVEL 5 ECHO.The ERRORLEVEL is at least 5.
```

For this style the first parameter is always ERRORLEVEL, and the second is the value it checks against. In this case, if the ERRORLEVEL is at least 5 then the command at the end of the line is executed, outputting the message "The ERRORLEVEL is at least 5.". The second form is a check between two strings:

```
IF "%str1%"=="Hello." ECHO.The strings are equal.
```

Here the first parameter is two strings either side of the double '=', symbolising a check to see if they are equal. If the variable str1 is exactly equal to "Hello.", a check which is case-sensitive, then "The strings are equal." is outputted. In the case that you wish to make the check case-insensitive you would rewrite it as following:

```
IF /I "%str1%"=="Hello." ECHO.The strings are equal.
```

Now, for example, str1 could contain "HELLO." but the check would still result in the command being executed at the end as the check is now case-insensitive. The final basic IF type is the existence check, to see if a file or folder exists.

```
IF EXIST myfile.txt TYPE myfile.txt
```

Here if the file "myfile.txt" exists in the current folder then the command `TYPE myfile.txt` is executed which displays the contents of "myfile.txt" in the console window.

All of the preceding examples have an optional NOT parameter that can be written after the IF which will execute the command at the end of the line if the condition is *not* true. For example:

```
IF NOT EXIST myfile.txt ECHO.File missing.
```

Which will output "File missing." if the file "myfile.txt" is not existent in the current folder. There are a few other IF types with command extensions, which can be seen with the `IF /?` command at the command prompt.

ELSE

The ELSE operator can be used with a combination of brackets to provide multi-line logical statements that provide an alternative set of commands if the condition is not true.

```
IF condition (
   commands to be executed if the condition is true
) ELSE (
   commands to be executed if the condition is false
)
```

Unlike some languages, in batch files the scripting requires that the lines `IF condition (`, `) ELSE (` and `)` are written very specifically like that. It is possible, however, to re-write it to use single-line outcomes all on one line:

```
IF condition (command if true) ELSE command if false
```

Below is an example of the ELSE operator in use:

```
@ECHO OFF
::Prompt for input.
SET /P answer=Enter filename to delete:
IF EXIST %answer% (
 DEL /P %answer%
) ELSE (
 ECHO.ERROR: %answer% can not be found in this folder!
)
ECHO ON
```

This batch file will delete a file, unless it doesn't exist in which case it will tell you with the message "ERROR: %answer% can not be found in this folder!".

Unlike in most computer languages, multiple multi-line IF...ELSE style statements can't be nested in batch files.

36.7.2 Jumps

You can control program flow using the GOTO statement. Batch files don't have all elements for structured programming scripting, however some elements of structured programming such as functions can be simulated. The simplest way of controlling program flow, however, is the GOTO statement which jumps to a specified label.

```
GOTO labelnam
```

This code will direct program flow to the label labelnam, which is found at the first occurance of this line:

```
:labelnam
```

It is important to remember that labels only store 8 characters, so if a label is longer than 8 characters only the first 8 will be seen. This means the labels labelname1 and labelname2 can't be distinguished from each other as their only difference occurs past the first 8 characters. Although not strictly incorrect, it is better to avoid using label names longer than 8 characters to avoid these distinguishing problems easily.

Here is the example from earlier redesigned to loop until asked to stop:

```
@ECHO OFF

:prompt
::Clear the value of answer ready for use.
SET answer=
SET /P answer=Enter filename to delete (q to quit):

IF EXIST %answer% (
 DEL /P %answer%
 GOTO prompt
)
IF /I "%answer%"=="q" GOTO :EOF

::By this point an error must have occurred as all
::the correct entries have already been dealt with.
ECHO.ERROR: Incorrect entry!
GOTO prompt

ECHO ON
```

Take note of the command `GOTO :EOF`. This command will take the script to the end of the file and end the current batch script.

36.7.3 FOR Looping

Runs a specified command for each file in a set of files.

```
   FOR %variable IN (set) DO command [command-parameters]
```

```
 %variable  Specifies a single letter replaceable parameter.
 (set)      Specifies a set of one or more files.  Wildcards may be used.
 command    Specifies the command to carry out for each file.
 command-parameters
            Specifies parameters or switches for the specified command.
```

To use the FOR command in a batch program, specify %%variable instead of %variable. Variable names are case sensitive, so %i is different from %I.

Batch File Example:

```
 for %%F IN (*.txt) DO @echo %%F
```

This command will list all the files ending in .txt in the current directory.

If Command Extensions are enabled, the following additional forms of the FOR command are supported:

```
   FOR /D %variable IN (set) DO command [command-parameters]
```

If set contains wildcards, then specifies to match against directory names instead of file names.

```
    FOR /R [[drive:]path] %variable IN (set) DO command [command-parameters]
```

Walks the directory tree rooted at [drive:]path, executing the FOR statement in each directory of the tree. If no directory specification is specified after /R then the current directory is assumed. If set is just a single period (.) character then it will just enumerate the directory tree.

```
    FOR /L %variable IN (start,step,end) DO command [command-parameters]
```

The set is a sequence of numbers from start to end, by step amount. So (1,1,5) would generate the sequence 1 2 3 4 5 and (5,-1,1) would generate the sequence (5 4 3 2 1)

```
    FOR /F ["options"] %variable IN (file-set) DO command [command-parameters]
    FOR /F ["options"] %variable IN ("string") DO command [command-parameters]
    FOR /F ["options"] %variable IN ('command') DO command [command-parameters]
```

or, if usebackq option present:

```
    FOR /F ["options"] %variable IN (file-set) DO command [command-parameters]
    FOR /F ["options"] %variable IN ('string') DO command [command-parameters]
    FOR /F ["options"] %variable IN (`command`) DO command [command-parameters]
```

filenameset is one or more file names. Each file is opened, read and processed before going on to the next file in filenameset. Processing consists of reading in the file, breaking it up into individual lines of text and then parsing each line into zero or more tokens. The body of the for loop is then called with the variable value(s) set to the found token string(s). By default, /F passes the first blank separated token from each line of each file. Blank lines are skipped. You can override the default parsing behavior by specifying the optional "options" parameter. This is a quoted string which contains one or more keywords to specify different parsing options. The keywords are:

```
    eol=c           - specifies an end of line comment character
                      (just one)
    skip=n          - specifies the number of lines to skip at the
                      beginning of the file.
    delims=xxx      - specifies a delimiter set.  This replaces the
                      default delimiter set of space and tab.
    tokens=x,y,m-n  - specifies which tokens from each line are to
                      be passed to the for body for each iteration.
                      This will cause additional variable names to
                      be allocated.  The m-n form is a range,
                      specifying the mth through the nth tokens.  If
                      the last character in the tokens= string is an
                      asterisk, then an additional variable is
                      allocated and receives the remaining text on
                      the line after the last token parsed.
    usebackq        - specifies that the new semantics are in force,
                      where a back quoted string is executed as a
                      command and a single quoted string is a
                      literal string command and allows the use of
```

```
                     double quotes to quote file names in
                     filenameset.
```

Some examples might help:

```
FOR /F "eol=; tokens=2,3* delims=, " %i in (myfile.txt) do @echo %i %j %k
```

would parse each line in myfile.txt, ignoring lines that begin with a semicolon, passing the 2nd and 3rd token from each line to the for body, with tokens delimited by commas and/or spaces. Notice the for body statements reference %i to get the 2nd token, %j to get the 3rd token, and %k to get all remaining tokens after the 3rd. For file names that contain spaces, you need to quote the filenames with double quotes. In order to use double quotes in this manner, you also need to use the usebackq option, otherwise the double quotes will be interpreted as defining a literal string to parse.

%i is explicitly declared in the for statement and the %j and %k are implicitly declared via the tokens= option. You can specify up to 26 tokens via the tokens= line, provided it does not cause an attempt to declare a variable higher than the letter 'z' or 'Z'. Remember, FOR variables are single-letter, case sensitive, global, and you can't have more than 52 total active at any one time.

You can also use the FOR /F parsing logic on an immediate string, by making the filenameset between the parenthesis a quoted string, using single quote characters. It will be treated as a single line of input from a file and parsed.

Finally, you can use the FOR /F command to parse the output of a command. You do this by making the filenameset between the parenthesis a back quoted string. It will be treated as a command line, which is passed to a child CMD.EXE and the output is captured into memory and parsed as if it was a file. So the following example:

```
FOR /F "usebackq delims==" %i IN (`set`) DO @echo %i
```

would enumerate the environment variable names in the current environment.

In addition, substitution of FOR variable references has been enhanced. You can now use the following optional syntax:

```
%~I         - expands %I removing any surrounding quotes (")
%~fI        - expands %I to a fully qualified path name
%~dI        - expands %I to a drive letter only
%~pI        - expands %I to a path only
%~nI        - expands %I to a file name only
%~xI        - expands %I to a file extension only
%~sI        - expanded path contains short names only
%~aI        - expands %I to file attributes of file
%~tI        - expands %I to date/time of file
%~zI        - expands %I to size of file
%~$PATH:I   - searches the directories listed in the PATH
              environment variable and expands %I to the
              fully qualified name of the first one found.
              If the environment variable name is not
```

```
                    defined or the file is not found by the
                    search, then this modifier expands to the
                    empty string
```

The modifiers can be combined to get compound results:

```
    %~dpI       - expands %I to a drive letter and path only
    %~nxI       - expands %I to a file name and extension only
    %~fsI       - expands %I to a full path name with short names only
    %~dp$PATH:I - searches the directories listed in the PATH
                  environment variable for %I and expands to the
                  drive letter and path of the first one found.
    %~ftzaI     - expands %I to a DIR like output line
```

In the above examples %I and PATH can be replaced by other valid values. The %~ syntax is terminated by a valid FOR variable name. Picking upper case variable names like %I makes it more readable and avoids confusion with the modifiers, which are not case sensitive.

36.8 Pipes

this is mainly used to redirect the output of one program to another program

```
a | b
```

means execute "a" and what all output "a" gives to the console - give that as "b" s input

```
dir | find ".htm"
```

will give a list of file which contain ".htm" in their names

The output of a command as well as possible errors could be redirected also to files (Note the 2 in front of the >>):

```
ACommand >>TheOutputOfTheCommandLogFile.log
2>>TheErrorOutputOfTheCommandFile.log
```

36.9 Functions

Functions are denoted by prepending a colon to their name. Functions are defined after the main script at the end of the file. For the handling of parameters and return values use the following structure.

```
CALL :functionname %param1% %param2% ...
ECHO %result%
```

```
:functionname
SETLOCAL
commands using parameters %1, %2, ... and setting %retval%
...
ENDLOCAL & SET result=%retval%
```

Bat file with next content outputs "42" as result of execution

```
:: describes and calls function for multiplication with 2 arguments
@ECHO OFF
CALL :multiply 21 2
ECHO %result%

:multiply
SETLOCAL
set retval=0
set left=%1
set right=%2
:: use '/A' for arithmetic
set /A "retval=left*right"
ENDLOCAL & SET result=%retval%
```

36.10 Command-Line Interfacing

let's say we want to call a program "MyProgram" from the command prompt. we type the following into our prompt (the .exe file extension is unnecessary):

```
C:\>myprogram.exe
```

And this will run the myprogram executable. Now, let's say we want to pass a few arguments to this program:

```
C:\>myprogram arg1 arg2 arg3
```

Now, if we go into the standard main function, we will have our argc and argv values:

```
int main(int argc, char *argv[])
```

Where:

```
argc = 4
argv[0] = "myprogram" (the name of the program - deduct 1 from argc to get the
number of arguments)
argv[1] = "arg1"
argv[2] = "arg2"
argv[3] = "arg3"
```

This shouldn't come as a big surprise to people who have any familiarity with standard C programming. However, if we translate this to our WinMain function, we get a different value:

```
int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrev, LPSTR CmdLine, int
iCmdShow)
```

we will only have one value to accept the command line:

```
CmdLine = "myprogram arg1 arg2 arg3".
```

We can also use the function **GetCommandLine** to retrieve this string from any point in the application. If we want to parse this into a standard argv/argc pair, we can use the function **CommandLineToArgvW** to perform the conversion. It is important to note that CommandLineToArgvW only works on unicode strings.

When we return a value from our C program, that value gets passed to the CMD shell, and stored in a variable called "ERRORLEVEL". ERRORLEVEL is the only global variable that is not a string, and it can contain a number from 0 to 255. By convention, a value of zero means "success", while a value other then zero signifies an error.

Let's say we wanted to write a C program that returns the number of arguments passed to it. This might sound like a simple task in C, but it is difficult to accomplish in batch script:

```
int main(int argc, char *argv[])
{
   return (argc - 1);
}
```

And we will name this program "CountArgs.exe". Now, we can put this into a batch script, to pass it a number of arguments, and to print out the number passed:

```
countargs.exe %*
ECHO %ERRORLEVEL%
```

We can, in turn, call this script "count.bat", and run that from a command prompt:

```
C:\>count.bat arg1 arg2 arg3
```

and running this will return the answer: 3.

NOTE: Actually, this can be accomplished with a batch file without resorting to a C program, by simply using CMD delayed variable expansion via the "/V:ON" parameter:

```
/V:ON   Enable delayed environment variable expansion using ! as the
delimiter. For example, /V:ON would allow !var! to expand the variable
```

```
var at execution time.  The var syntax expands variables at input time,
which is quite a different thing when inside of a FOR loop.
```

Then use a simple batch file like the following to count parameters:

```
set COUNT=0
for %%x in (%*) do ( set /A COUNT=!COUNT!+1 )
echo %COUNT%
```

Or an easier way, without having to enable & use 'delayed environment expansion', would be to do the following:

```
set COUNT=0
for %%x in (%*) do set /A COUNT+=1
echo COUNT = %COUNT%
```

This returns the same answer as the C program example.

36.11 Console Control Handlers

37 Sample FTP script

A simple program that will ftp one file. **Do not name the file ftp.bat or it will recurse**.

```
set ftpUser=
set ftpPass=
set ftpSite=
set file=

@echo off

cls

: FTP the stuff
> script.ftp ECHO USER %ftpUser%
>> script.ftp ECHO %ftpPass%
>> script.ftp ECHO put %file%
>> script.ftp ECHO quit
FTP -v -n -s:script.ftp %ftpSite%
@type NUL >script.ftp


: Delete scripts
del script.ftp /Q

cls
```

Sample FTP[1]

1 http://en.wikibooks.org/wiki/Category%3AWindows%20Programming

38 Windows Script Host

The DOS programming model, with it's clumsy batch scripting capabilities was quickly wearing out, and Microsoft needed to add an alternative scripting environment to Windows. The result of this was the **Windows Script Host** (WSH). WSH comes in two flavors: a console mode version (csh.exe or cscript.exe) and a non-console version (wsh.exe or wscript.exe).

38.1 WSH Languages

WSH actually isn't a new language, but is instead an environment for programming in other **Active Scripting** languages. Active Scripting languages are programming languages that are implemented as COM components. Any Active Scripting language installed on your computer can be used with WSH. By default, WSH can be programmed using JScript and VBScript, but other Active Scripting languages can be installed and run in WSH. One of the more popular 3rd party Active Scripting languages is PerlScript, which is often installed as part of the ActivePerl distribution.

WSH scripts also include a certain amount of XML code, that helps to control the environment of the running script. The language being used to script must be specified in an XML tag.

WSH is an Object Oriented environment, and WSH provided a large number of objects that the Active Scripting languages may tap into.

Another Microsoft technology, **Active Server Pages** (ASP) is a similar program to WSH in that it is a language-neutral scripting environment that uses installed Active Scripting languages. However, ASP is used as a server-side scripting tool for generating webpages. WSH, in comparison is more well suited for use in scripting a local environment (although WSH can be used as a server-side website scripting engine as well, if you dare!).

38.2 Writing a WSH Script

A WSH script file is a plain-text file, that may use a generic ".wsh" file extension. Also, many other file extensions are associated with WSH, and may be more descriptive then the simple .wsh extension. For instance, a WSH script that is primarily VBScript may have the extension ".vbs", and a WSH script that is JScript may have the extension ".js".

Each WSH file must have an XML "JOB" object. Each script may have multiple Jobs, but each job must have an unique ID. For instance, here are some sample job tags:

```
<job id="Task1">
```

```
<job id="GetHarddriveInfo">
```

```
<job id="CreateLogFile">
```

Next, we must specify the scripting language that we will be using to implement a specific job:

" tag. At the end of a job, we close with a "</job>" tag.

38.3 "WScript" Object

WSH provides the **WScript** object, that has a number of methods and fields for use in a WSH script. The WScript object never needs to be instantiated, and it is always available in any script. As an example, the "Echo" method of the WScript object prints out a certain string. The csh.exe (Console Version) of WSH will print the string to the console, while the wsh.exe (Non-Console Version) will print the string to a message box. Here is a multi-lingual example:

```
<job id="TestEcho">

    </job>
```

Note, of course, that you can't use the PerlScript if you haven't installed it on your computer.

38.4 XML Tags

There are a number of other XML tags that can be used in a WSH script.

38.4.1 <description >

The <description> tag contains a certain amount of plain text that describes the current file. For instance:

```
<description>
  This script writes a status entry to the log file, and updates the website.
</description>
```

Now, this may seem like a difficult way to just write a comment, but the <description> tag also has another purpose. Let's say that we save the above code snippet into a script file, such as "testdesc.wsh". Now, if we run our script with the "/?" argument, we get the following result:

```
C:\>testdesc.wsh /?
This script writes a status entry to the log file, and updates the website.
```

When we send the argument "/?" to our script, all the text that is written in the <description> tag is printed to the console.

39 JScript

JScript, Microsoft's proprietary version of JavaScript, is a default language in Windows Script Host. JScript is very similar to JavaScript, and was originally created so that JavaScript programmers could put their skills to use scripting a local Windows environment.

39.1 Further Reading

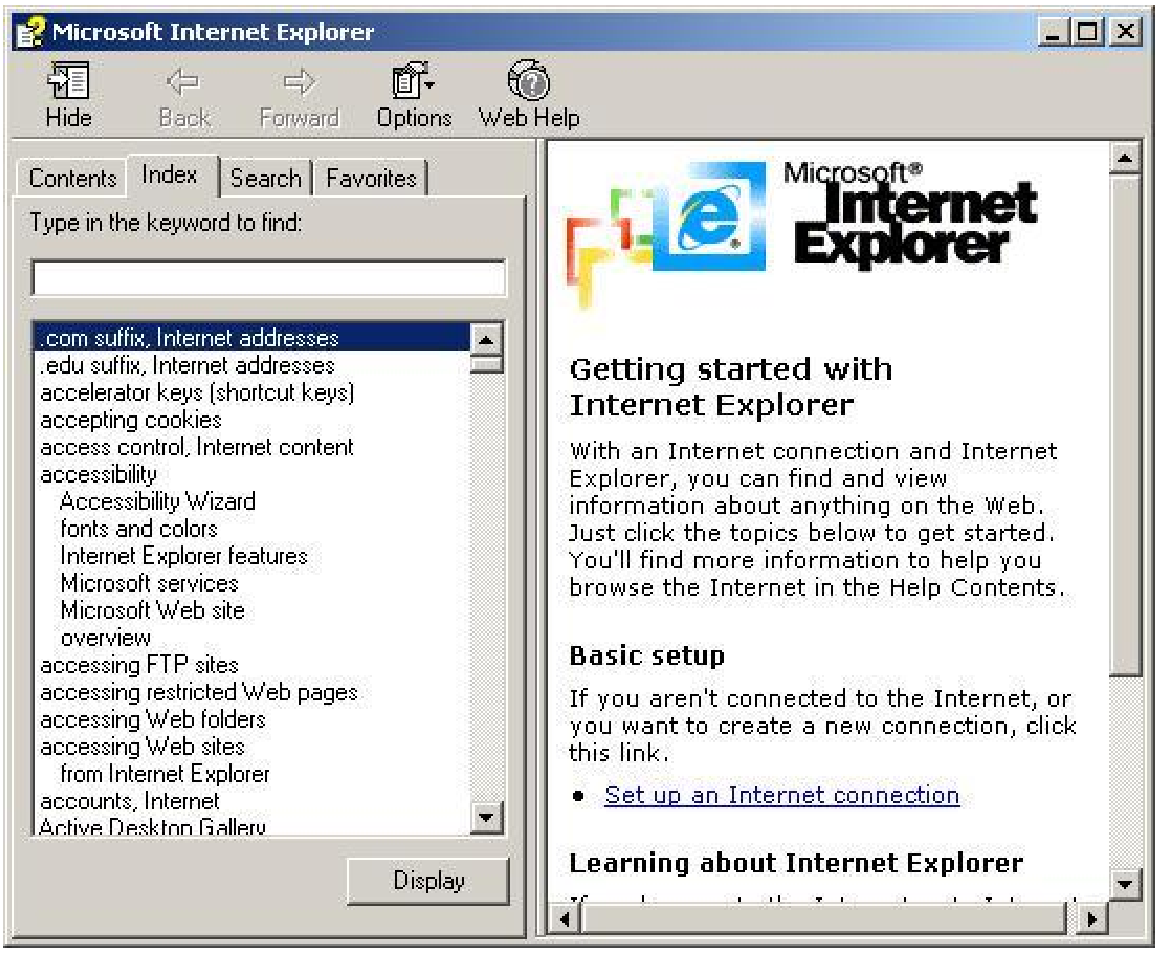
- JavaScript[1]

ja:JScript[2]

1 `http://en.wikibooks.org/wiki/JavaScript`
2 `http://ja.wikibooks.org/wiki/JScript`

40 Compiled HTML Help

In most programs, the menu will have a popup item called "Help". In the help menu, there is frequently an option for "Contents" or "Index" or "Topics". When you click on these options, it brings up a help box.


Microsoft Internet Explorer
Hide
Back
Forward
Options
Web Help
Contents
Index
Search
Favorites
Type in the keyword to find:
Display
Microsoft® Internet Explorer

.com suffix, Internet addresses
.edu suffix, Internet addresses
accelerator keys (shortcut keys)
accepting cookies
access control, Internet content
accessibility
Accessibility Wizard
fonts and colors
Internet Explorer features
Microsoft services
Microsoft Web site
overview
accessing FTP sites
accessing restricted Web pages
accessing Web folders
accessing Web sites
from Internet Explorer
accounts, Internet
Active Desktop Gallery

Getting started with Internet Explorer

With an Internet connection and Internet Explorer, you can find and view information about anything on the Web. Just click the topics below to get started. You'll find more information to help you browse the Internet in the Help Contents.

Basic setup

If you aren't connected to the Internet, or you want to create a new connection, click this link.

- Set up an Internet connection

Learning about Internet Explorer

Figure 2

These help boxes are known as "Compiled HTML Help Modules", or .chm files. These help files can be compiled from HTML source code using the Microsoft HTML Help Workshop, available as a free download from Microsoft.

40.1 Making HTML Help Modules

HTML help modules can be compiled from source files containing a mixture of HTML, DHTML, CSS and JavaScript. This is the same exact HTML that is used in writing web pages, so people with web experience will have a leg up creating HTML help modules. HTML, CSS and JavaScript are languages that are very well suited for graphical applications, but the functionality is far less then C or VB. The Help Workshop contains a number of different options, as to whether you want to include an index window, if you want to allow searching and indexing, if you want to have a Table of Contents, et cetera. A default page may be chosen to be displayed when the Help Module is first loaded. Pages in a help module may use hyper links to link to other pages in the module, other files on the computer, or even resources from the internet.

40.2 HTML Help API

The HTML Help Workshop comes with a header file ("htmlhelp.h") and a library file ("htmlhelp.lib") that must be utilized in your project to be able to call HTML help modules from your program. Windows.h must be included before the Htmlhelp.h file, or the compiler will spew out errors. Htmlhelp.h defines a single function, **HtmlHelp** that can be called from inside your program:

```
HWND WINAPI HtmlHelp(HWND hwndCaller, LPCTSTR pszFile, UINT uCommand, WORD_PTR
dwData);
```

This function returns a handle to the html help window, and your program can interact with this window at a relatively high level, if you want it to. The first parameter is the handle to the calling (parent) window. If you set this parameter, the help window will be a child of your window, and can communicate with it. If you set this to NULL, the help window will essentially be autonomous. The "pszFile" parameter is the filename of the target help module. The "uCommand" may contain a number of different commands, some of which are very complicated and involved. The simplest (and most common) is the **HH_DISPLAY_TOPIC** command, which displays help like normal, and allows the user to view and interact with the help module. The "dwData" may optionally contain additional data, depending on the command used.

If we want to open a help module, located at c:/help/myhelp.chm, we can use the following command:

```
hwndHelp = HtmlHelp(hwndParent,
                    "C:\help\myhelp.chm",
                    HH_DISPLAY_TOPIC,
                    NULL);
```

Now, let's say that we want to open a specific page inside our help module. Let's say that our module contains a page called "example.htm". We can open the help directly to that page, by changing the destination string a little bit:

```
hwndHelp = HtmlHelp(hwndParent,
                    "C:\help\myhelp.chm::example.htm",
                    HH_DISPLAY_TOPIC,
                    NULL);
```

And further more, we can force the page to be displayed in a particular window of our help viewer, if we have created more then one window for it. Let's say we have created a window called "SideWin" in our html help module. We can load our page (example.htm) into SideWin using the following syntax:

```
hwndHelp = HtmlHelp(hwndParent,
                    "C:\help\myhelp.chm||example.chm>SideWin",
                    HH_DISPLAY_TOPIC,
                    NULL);
```

For more advanced options, and a discussion of some of the other commands, you should check out the related material at MSDN.

40.3 Creating an HTML Help Module

The HTML help workshop has 5 buttons across the top, In order, from left to right, they are:

New

create a new HTML file for editing

Open

open an existing HTML file for editing

Compile

Compile the current project into a help module

View

view the current help project

Help

Opens a help viewer window

If you click on the "New" button (or alternately select "File>New" from the menu bar), you will be prompted to select what type of new thing you want to create. To start, you should create a new project.

Selecting to create a new project will open up a wizard, that will walk you through the steps of creating a new project.

40.4 HTML Help Files

There are 3 different file types that may be included in an HTML Help project: HTML source files (.HTM) Table of Contents files (.HHC) and Index Files (.HHK). Once you have created a new project, you may edit and add all of these types of files to your project. Each HTML help module may only have one .HHC and one .HHK file.

40.5 The Project Sidebar

When you start a new project, and complete the wizard, you will see a sidebar, on the left-hand-side of the help workshop window, that will allow you to edit the project options, the table of contents, and the index.

40.6 Decompiling

On Windows, a CHM file can be extracted to plain HTML with the command:

```
hh.exe -decompile extracted filename.chm <!-- Windows 7 64bit get error file could not be opened
-->
```

This will decompress all files embedded in *`filename.chm`* to folder *`extracted`*.

You can also use HTML Help Workshop to decompile CHM files. File -> Decompile...

7-Zip[1] will open .chm files as archives allowing access to all content.

On Windows, select the topmost Topic, right-click and select Print. On the popup dialog, select "Print this heading and all subtopics.", click OK. Before selecting a printer, look in %HOMEPATH%\Local Settings\Temp for a file named ~hh*.htm. This is the concatenated HTML. The image reference will refer to the .chm file at the time of the print.

On Linux systems which use apt[2] as a packaging tool, a CHM file is extracted to plain HTML with extract_chmLib (first command is for a Debian[3] based OS):

```
  $ sudo apt-get install libchm-bin
  $ extract_chmLib tero.chm tero/
```

1 http://en.wikibooks.org/wiki/7-Zip
2 http://en.wikibooks.org/wiki/Advanced%20Packaging%20Tool
3 http://en.wikibooks.org/wiki/Debian

41 Resource Script Reference

This appendix page will attempt to list the different types of resources, and will attempt to show how to use those resources.

41.1 General construction

Resource script files are human-readable text files in either ANSI or Unicode (more strictly: UTF-16 with *byte order mark* (BOM)) format. To intermix different languages in ANSI format, a quirks `#pragma` exist to switch the code page in between. Unicode, `#pragma` switch, and the `LANGUAGE` statement are only supported for Win32.

A typical small file may look like

```
#include <windows.h>
#define IDC_STATIC -1

100 ICON "ProgIcon.ico"

10 MENU
{        // or BEGIN
 POPUP "&File"
 {
  MENUITEM "E&xit",IDCANCEL
 }
}        // or END
```

Using *curly braces* or BEGIN/END depends on your preference. The old, space-wasting style is the BEGIN/END pair, coming from the Pascal heritage of MacOS and Win16 API calls. C programmers typically prefer curly braces. The Visual Studio Resource Editor always generates BEGIN/END pairs, besides lots of housekeeping stuff.

Beginning with some header inclusion and `#define` statements, every resource is included as either

```
id_of_resource  resource_type  [memory management flags]  "filename"
```

or

```
id_of_resource  resource_type  [memory management flags]
BEGIN
 subsequent data
END
```

An exception from this rule is

- The `LANGUAGE` statement, can be placed almost everywhere (Win32 only)
- The `DIALOG` and the `VERSIONINFO` resource, there are additional statements between heading line and BEGIN
- The `STRINGTABLE` resource, where no resource ID is before the keyword. Instead, every string is prefixed with an ID

`id_of_resource` and `resource_type` can be either a string or a number. No quotes are there! A number is the preferred method for identification. All predefined resource types are numbers. But watch out! If you use an unknown ID for the resource compiler (let's say MANIFEST for Visual Studio 6), you will get neither error nor warning, and the resource is built with string resource type. Windows XP, using `ExecProcess()`, will not find that intended manifest and your program will show up in old visual style.

Conditional compilation with #if / #ifdef / #endif is also supported.

Expressions for IDs are limited to very simple math, no boolean operators are permitted.

41.1.1 Under the hood

Resources are compiled into a three-level directory structure:

1. Resource type (MENU, DIALOG etc.)
2. Resource ID (the number that is ahead the Resource Type – for STRINGTABLE, the ID of a group of upto 16 strings)
3. Resource language (the currently active language, given by command-line option or LANGUAGE statement; Win32 only)

The content of following data depends on actual resource type. Mostly, it's binary.

Read access to the binary data of an arbitrary resource is done with

```
FindResource()        // get a handle
LoadResource()        // get the binary size
LockResource()        // get a pointer; Win32: This is a simple macro, Win16:
This is a function call.
…            // do something
UnlockResource()    // Win32: This is a do-nothing macro, Win16: This is a
function call.
FreeResource()        // release
```

Because bitmap, icon, cursor, dialog, string table, and menu resources are not officially documented and a bit hard to parse, programmers should use specialized resource load functions for these resource types instead. See description and examples for these types below.

41.1.2 Identifiers

Identifiers are generally named in a certain way, although the reader and all programmers are free to alter this naming scheme. It is simply a suggestion. Identifiers generally start with the prefix "ID", followed by a letter that denotes the type of identifier:

- IDS: A string resource
- IDM: A menu resource
- IDC: A command identifier
- IDD: A dialog box resource
- IDA: An Accelerator table resource
- IDI: An Icon or bitmap resource
- IDB: A Bitmap resource
- ID: A custom resource, or an uncommon resource type.

Sometimes, the command identifiers in a menu are given an "IDM_" prefix, to distinguish between commands from other sources.

There is no need to use symbolic identifiers. In some cases, identifiers complicate access to numerically adjanced controls in a dialog or menu. In any case, identifiers don't help non-English programmers to read a software source. Numbers never need translation. And both identifiers and numbers need explanation.

IDs are allowed in range 0..65535 and preferred in range 1..32767.

41.1.3 LANGUAGE

This keyword has different scope:

- Local (for one resource) if located below the resource line, like

```
21 MENU
LANGUAGE 7,1 // or, LANG_GERMAN, SUBLANG_GERMAN
{
 POPUP "&Datei"  // = "&File"
 …
```

This language applies to that menu

- Global (for all next resources) if located somewhere else

Language-neutral resources, like culture-free icons, VersionInfo, and Manifests, should be always set to LANGUAGE 0,0 (or, more verbously, LANG_NETUTRAL,SUBLANG_NEUTRAL).

Note to always check images against culture dependency! A typical mistake is the plug symbol for a mains-supplied notebook: It shows undoubtly an American plug, even in Europe. Obviously, images containing letters or text are culture-dependent.

41.1.4 Memory Management Flags

There are some Memory Management Flags from Win16 heritage, like MOVEABLE, FIXED, etc. See LocalAlloc() for some flags.

DISCARDABLE

Resources are loaded into memory when the program is run. However, if a resource is not in use, and if Windows does not need them immediately, resources can be optionally unloaded from memory until needed. To specify that it is okay to unload an unused resource from memory, you may list the **DISCARDABLE** keyword with the resource. DISCARDABLE resources allow more efficient memory usage, but can slow down your program if they need to be loaded from disk.

The DISCARDABLE keyword is ignored for 32-bit Windows, but remains for compatibility. `http://blogs.msdn.com/oldnewthing/archive/2004/02/02/66159.aspx` 32 bit resources are never loaded but mapped into memory.

41.2 Icons

Icons can be stored in a resource file using the `ICON` keyword. Here is a general example of using an icon in a resource script:

```
IDI_ICON<n> ICON [DISCARDABLE] "iconfile.ico"
```

Windows Explorer will display the binary executable with the first icon from the script. For instance, if we load two icons, as such:

```
IDI_ICON1 ICON DISCARDABLE "icon1.ico"
IDI_ICON2 ICON DISCARDABLE "icon2.ico"
```

And we define our macros as such in the corresponding `resource.h`:

```
#define IDI_ICON1 1
#define IDI_ICON2 2
```

The executable file will have *icon1.ico* as its icon.

To load an icon from an executable module, assuming we have an instance handle to the module (`hInst` in the following example), we can get a handle to the icon as such:

```
HICON hIcon;
hIcon = LoadIcon(hInst, MAKEINTRESOURCE(IDI_ICON1));
```

This will return a handle to the icon associated with the identifier "IDI_ICON1". Icon identifiers are generally prefixed with an "IDI_" which is short for "ID for an Icon".

The second parameter to the `LoadIcon()` function is a pointer to a string. String pointers are 32 bit values. However, if the most significant 16 bits are all zero, Windows will treat the value as a resource number, and not a string. To make the conversion between a string and a 16-bit integer, Microsoft provides the **MAKEINTRESOURCE** macro. Similarly, we could have used a string to define our Icon:

```
MYICON1 ICON DISCARDABLE "icon1.ico"
```

And we could load this string by name:

```
HICON hIcon;
hIcon = LoadIcon(hInst, "MYICON1");
```

String identifiers for resources are case insensitive.

`WNDCLASSEX` has handle values for two icons: a large icon and a small icon. The small icon is the icon used in the upper-left corner. Small icons are generally 16 pixels square. Larger icons are 32 pixels square. If no small icon handle is provided, the large icon will be shrunk down to fit.

If the `LoadIcon()` function is supplied with a NULL instance handle, Windows will supply a default icon for use.

Recently, the Win32 API provides the **LoadImage** function for loading icons, bitmaps, and mouse cursors from a single function. You can find more information about this function on MSDN.

Internally, Icons are stored under numeric resource type RT_ICON == 3, and are grouped under RT_GROUP_ICON == 14.

41.3 Bitmaps

Bitmaps can be loaded similarly to Icons in resource files:

```
(bitmap ID or name) BITMAP [DISCARDABLE] "bitmapfile.bmp"
```

Bitmaps can be accessed with the aptly named **LoadBitmap** function (again, new versions of the Win32 API prefer you use **LoadImage** to load a bitmap, icon, or cursor). LoadBitmap returns an HBITMAP handle type:

```
HBITMAP hBmp;
hBmp = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAP1));
```

Or, if we have named our bitmap resource:

```
hBmp = LoadBitmap(hInst, "MyBitmapRes");
```

Bitmaps are large resources, and if windows can't load the bitmap into memory (or if the ID or name value is invalid), the function will return a NULL value. Make sure you test this value before you use the handle.

Bitmaps must be unloaded from memory by passing the handle to the DeleteObject() function. You can find more information about this on MSDN

Bitmap identifiers generally use a "IDB_" prefix, to indicate that it is the ID of a bitmap.

Internally, Bitmaps are stored under numeric resource type RT_BITMAP == 2.

41.4 Mouse Cursors

Mouse cursors are specified similarly to icons and bitmaps, and are loaded with the **LoadCursor** function.

Internally, cursors are stored under numeric resource type RT_CURSOR == 1, and are grouped under RT_GROUP_CURSOR == 12.

As for any resource that implies binary data, the use of an external file with "filename" is recommended. However, most resource compilers allow to inline binary data to the resource file in this way:

```
42 ICON
{
 123,4567,0x89AB,0xCDEF
 '\x01','\x23',"ajx"
}
```

with this rule, coming from Win16 heritage:

- Numbers (decimal or hexadecimal) are stored contiguously as 16-bit little-endian quantities (unaligned)
- Characters are stored as 8-bit quantities

41.5 String Tables

A resource script can have many string tables, although this is unneccessary: the tables aren't differentiated (i.e. they get merged), and each string object, in any table, must have a unique identifier. Strings in a string table also may not use names, but instead must use numeric identifiers. After all, it doesn't make any sense to have to address a string with a string, does it?

Here is a general string table:

```
STRINGTABLE DISCARDABLE
BEGIN
   IDS_STRING1, "This is my first string"
   IDS_STRING2, "This is my second string"
   ...
END
```

It is important to note that in place of the BEGIN and END keywords, the programmer may also use the more C-like curly brackets, as such:

```
STRINGTABLE DISCARDABLE
{
   IDS_STRING1, "This is my first string"
   IDS_STRING2, "This is my second string"
   ...
}
```

Some people prefer one over the other, but they are all the same to the resource compiler.

Strings can be loaded using the **LoadString** function. LoadString is more involved then the LoadBitmap or LoadIcon functions:

```
int LoadString(HINSTANCE hInstance, UINT uID, LPTSTR lpBuffer, int nBufferMax);
```

The hInstance parameter, as we know, is the instance handle for the module that contains the string. The uID parameter contains the string number that we are trying to access. lpBuffer is the character array variable that will receive the string, and the nBufferMax number tells windows what the maximum number of characters that can be loaded is. This count is a security precaution, so make sure not to allow Windows to write character data beyond the end of the string. MSDN displays a large warning on the page for this function, and it is important that programmers heed this warning. msdn[1]

Windows will automatically zero-terminate the string, once it is written to the buffer. LoadString will return the number of characters that were actually written into the string, in case the number of characters is less then the maximum number allowed. If this return value is 0, the string resource does not exist, or could not be loaded.

Strings can have "\0" in the middle. As strings are saved as counted strings, LoadString returns the number of characters saved, including the zeroes in between. But most Resource Editors fail with such strings.

Internally, stringtables are stored under numeric resource type RT_STRING == 6, in groups of upto 16 adjanced IDs.

1 `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/resources/strings/stringreference/stringfunctions/loadstring.asp`

41.6 Accelerators

Keyboard accelerators are a common part of nearly every windows application, and therefore it is a good idea to simplify the job of creating accelerators by putting them in a resource script. Here is how to create an accelerator table:

```
(Accelerator Table ID or name) ACCELERATORS [DISCARDABLE]
BEGIN
   (key combination), (Command ID)
   ...
END
```

Key combinations are specified in terms of either a string literal character ("A" for instance) or a virtual key code value. Here are some examples:

```
IDA_ACCEL_TABLE ACCELERATORS DISCARDABLE
BEGIN
   "A", IDA_ACTION_A //Shift+A
END
```

Now, when the key combination "Shift+A" is pressed, your window procedure will receive a WM_COMMAND message with the value IDA_ACTION_A in the WPARAM field of the message.

If we want to use combinations of the "Alt" key, or the "Ctrl" key, we can use the ALT and CONTROL keywords, respectively:

```
IDA_ACCEL_TABLE ACCELERATORS DISCARDABLE
BEGIN
   "a", IDA_ACTION_A, ALT          //Alt+A
   "b", IDA_ACTION_B, CONTROL      //Ctrl+B
   "c", IDA_ACTION_C, ALT, CONTROL //Alt+Ctrl+A
END
```

Also, we can use the "ˆ" symbol to denote a CONTROL key code:

```
IDA_ACCEL_TABLE ACCELERATORS DISCARDABLE
BEGIN
   "ˆa", IDA_ACTION_A //Control+A
END
```

Similarly, if we want to be super hackers, would could use the ASCII code directly:

```
IDA_ACCEL_TABLE ACCELERATORS DISCARDABLE
BEGIN
   65, IDA_ACTION_A, ASCII //65 = "A", Shift+A
END
```

Or, we could refer to keys (including non-alphanumeric keys) with their Virtual Key Code identifiers, by using the VIRTKEY identifier:

```
IDA_ACCEL_TABLE ACCELERATORS DISCARDABLE
BEGIN
   VK_F12, IDA_ACTION_F12, VIRTKEY             //press the "F12 Key"
   VK_DELETE, IDA_ACTION_DEL, VIRTKEY, CONTROL //Ctrl+Delete
END
```

Now, If we make an accelerator correspond to a menu command, the menu command will light up when we press the accelerator. That is, the menu will light up unless we specify the "NOINVERT" keyword:

```
IDA_ACCEL_TABLE ACCELERATORS DISCARDABLE
BEGIN
   "A", IDA_ACTION_A, NOINVERT //Shift+A (non inverted menu selection)
END
```

To Load an accelerator table, we need to use the **LoadAccelerators** function, as such:

```
HACCEL hAccel;
hAccel = LoadAccelerators(hInst, MAKEINTRESOURCE(IDA_ACCEL_TABLE));
```

Again, we could have given our resource a string name, and used that string to load the table.

When using accelerators, we need to alter our message loop to intercept the keypress messages, and translate them into command messages according to our accelerator table rules. We use the **TranslateAccelerator** function, to intercept the keypress messages, and translate them into command messages, as such:

```
while ( (Result = GetMessage(&msg, NULL, 0, 0)) != 0)
{
    if (Result == -1)
    {
        // error handling
    }
    else
    {
        if (!TranslateAccelerator(hwnd, haccel, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
}
```

Also, if we are writing an MDI application, we need to intercept Accelerator messages from the child windows, we use the **TranslateMDISysAccel** function also:

```
while ( (Result = GetMessage(&msg, NULL, 0, 0)) != 0)
{
    if (Result == -1)
    {
        // error handling
    }
    else
    {
        if (  !TranslateMDISysAccel(hwndClient, &msg)
           && !TranslateAccelerator(hwndFrame, haccel, &msg) )
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
}
```

Where "hwndFrame" is the handle to the frame window, and "hwndClient" is the handle to the MDI client window.

Internally, Accelerators are stored under numeric resource type RT_ACCELERATOR == 9.

41.7 Menus

Menus can be defined in a resource script using the MENU keyword. There are 2 types of items that appear in a menu, the top level "POPUP" menu items, and the secondary "MENUITEM" items. These are defined in a menu as such:

```
(ID or name) MENU [DISCARDABLE]
BEGIN
   POPUP "File"
   POPUP "Edit"
   BEGIN
      MENUITEM "Copy", IDM_EDIT_COPY
      MENUITEM "Paste", IDM_EDIT_PASTE
   END
   ...
 END
```

We have included a few examples here, so that you can see the difference between a POPUP and a MENUITEM. When we have a menu with the ID_MENU identifier, we can load it into our program as such:

```
HMENU hmenu;
hmenu = LoadMenu(hInst, MAKEINTRESOURCE(ID_MENU));
```

Once we have this handle, we can pass it to the CreateWindow function, and apply it to our window.

When a menu item is selected, the host program receives a WM_COMMAND message, with the menu item identifier in the WPARAM parameter. If we have a basic window procedure switch-case statement, we can see this as follows:

```
case WM_COMMAND:
   switch(WPARAM)
```

```
    {
       case IDM_EDIT_COPY:
          //handle this action
          break;
       case IDM_EDIT_PASTE:
          //handle this action
          break;
    }
    break;
```

In a menu, if we want to associate a menu item with an accelerator, we can define it as such:

```
ID_MENU MENU DISCARDABLE
BEGIN
   POPUP "File"
   POPUP "Edit"
   BEGIN
      MENUITEM "&Copy", IDM_EDIT_COPY
      MENUITEM "&Paste", IDM_EDIT_PASTE
   END
   ...
 END
```

Notice how we put the ampersand (&) in front of the "C" in "Copy" and the "P" in "Paste". This means that those letters will be underlined, but more importantly, if an accelerator key combination is pressed, those items in the menu will be highlighted (unless the NOINVERT tag is specified in the accelerator table). If an ampersand is placed before a POPUP menu item, pressing ALT+ that letter will popup that menu. For instance, lets define our menu:

```
ID_MENU MENU DISCARDABLE
BEGIN
   POPUP "&File"
   POPUP "&Edit"
   BEGIN
      MENUITEM "Copy", IDM_EDIT_COPY
      MENUITEM "Paste", IDM_EDIT_PASTE
   END
   ...
 END
```

Now, if we press ALT+F, we will pop open the File menu, and if we press ALT+E it will open the Edit menu. That's pretty nice functionality for only a single extra character to type.

Internally, Menus are stored under numeric resource type RT_MENU == 4.

41.8 Version Information

A program can include certain information about its version, and its author in a resource script. This version information appears when you right-click the executable in Windows, and click "Properties". In the properties dialog box, this information appears on the

"Version" tab.

```
BLOCK "StringFileInfo"
  BEGIN
      BLOCK "040904E4"
      BEGIN
          VALUE "CompanyName",      "My Company.\0"
          VALUE "FileDescription",  "A Win32 program."
          VALUE "FileVersion",      "1.0.0.0\0"
          VALUE "ProductName",      "The product name.\0"
          VALUE "ProductVersion",   "1.0\0"
          VALUE "LegalCopyright",   "My Company.\0"
      END
  END
  BLOCK "VarFileInfo"
  BEGIN
      VALUE "Translation", 0x409, 1252
  END
```

Internally, VersionInfo is stored under numeric resource type RT_VERSION == 16. It was introduced with Windows 3.

41.9 Dialog Boxes

Dialog box resources follow a general pattern:

```
(Dialog ID or name) DIALOG [DISCARDABLE] x, y, width, height
TITLE "(dialog box title)"
[CLASS "(class name)"]
FONT "(font name)"
BEGIN
   ...
END
```

if a dialog box is not being associated with a class, the CLASS field does not need to be filled in. All strings listed as being in quotes *must be in quotes in the resource script* or there will be an error. Individual items in a dialog box are then specified between the BEGIN and END tags.

Internally, Dialogs are stored under numeric resource type RT_DIALOG == 5.

41.9.1 Generic Controls

CONTROL classname,windowname,id,left,top,width,height,windowflags

41.9.2 Specific Buttons

41.9.3 Edit Boxes

41.10 Manifests

Manifest resources contain UTF-8 encoded XML description of operating system and DLL dependency. Mostly, this resource dictates to use the Windows XP Version 6.0 comctl32.dll, to have Luna style for the standard and common controls.

Internally, Manifests are stored under numeric resource type RT_MANIFEST == 24. It was introduced with Windows 4.1.

41.11 User-type resources

User-type resources should use some greater resource type identifiers, or RT_RCDATA == 10.

42 Obtaining the SDK

42.1 The Windows Platform SDK

The Microsoft Windows **Software Development Kit** is a simple, free set of tools and libraries to create windows programs in C and C++. The SDK contains the standard windows header files, the link libraries, the compiler (cl.exe), and the linker (link.exe). The SDK is available as a free download from Microsoft.

Download the SDK[1]

42.2 Visual Studio

Another option is Microsoft Visual C++[2], a commercial IDE and is an option worth looking into for serious Windows programmers. MSVC++ is almost a de facto standard Windows programming IDE.

The best option for a beginner is Microsoft's .NET platform. .NET developers can use Microsoft Visual Studio.NET[3], Borland C#Builder (discussed earlier), the Visual Studio Express products[4] and SharpDevelop[5].

The Visual Studio Express products are Microsoft-provided stripped down versions of Visual Studio, designed to provide an entry-level product for Windows programmers. They are currently free and in beta, but they are expected to become a low-priced commercial product once they are released. As of this writing, Visual Studio Express supports Managed C++, C#, Visual Basic .NET, and J# for Windows programming. All Visual Studio Express products use the same compilers as the commercial Visual Studio .NET.

42.3 Borland Compiler

In addition, Borland has released several of its compilers for free (although still proprietary) as Borland Command Line compiler tools[6]. Borland requires users to log in in order to download these tools. They are quite small and fast, more so than gcc and DJGPP, and

1 `http://www.microsoft.com/downloads/en/details.aspx?FamilyId=E6E1C3DF-A74F-4207-8586-711EBE331CDC&displayl`
`en`
2 `http://msdn.microsoft.com/visualc/`
3 `http://msdn.microsoft.com/vstudio/`
4 `http://lab.msdn.microsoft.com/express/`
5 `http://www.icsharpcode.net/OpenSource/SD/`
6 `http://www.borland.com/products/downloads/download_cbuilder.html`

can build Windows applications with ease, especially for those accustomed with Borland tools.

Other notable downloads from Borland are C++BuilderX[7] and and C#Builder[8], both with IDEs. Both products impose restrictions on your software and must not be for commercial use.

42.4 Other Tools

SharpDevelop is a free and small IDE. It supports C# and partially supports Visual Basic.NET. However, you also need to obtain Microsoft's .NET SDK[9], which can be freely downloaded from Microsoft's site.

It's also possible to develop Windows programs in Forth[10] (see the Forth article at Wikipedia[11]) using WinForth, Win32Forth, SwiftForth, and RetroForth. Several of these can generate native Windows applications with much less bloat than C/C++.

7 http://www.borland.com/products/downloads/download_cbuilderx.html
8 http://www.borland.com/products/downloads/download_csharpbuilder.html
9 http://msdn.microsoft.com/netframework/downloads/updates/default.aspx
10 http://en.wikibooks.org/wiki/Forth
11 http://en.wikipedia.org/wiki/Forth%20

43 Obtaining the DDK

43.1 What is the DDK?

Microsoft offers a free **Driver Development Kit**, for people who are interested in writing device drivers, system services, and other kernel-mode code. The DDK comes with a number of different libraries, extensive documentation, link libraries, a special compiler and linker, and many examples for new driver programmers.

This book discusses the DDK in depth in throughout Section 4, but specifically in the chapter about the DDK[1].

43.2 Obtaining the DDK

The DDK is available free from microsoft, but not as a download. To obtain the DDK, you must order the software on CD, and this comes with a small shipping and handling fee. The DDK has a number of different options, and installing the DDK with every option will require a large amount of disk space.

The DDK used to be available as a download, but to conserve bandwidth, Microsoft has changed their policy to only offer the DDK as a CD. The SDK, and a number of other CDs are also available to order from Microsoft for free (plus shipping and handling), so this might be worthwhile to check out as well.

Order the DDK[2]

1 Chapter 30 on page 143
2 `http://www.microsoft.com/whdc/devtools/ddk/default.mspx`

44 Obtaining MASM

44.1 Obtaining MASM

Many people claim that Microsoft has discontinued the use of MASM, but this is not the case. MASM is still maintained by Microsoft for use in house, and is updated on a needs basis. This means that MASM is only updated when presented with a specific need, such as the migration to 64 bits, or the migration to Vista. MASM comes as part of the standard DDK distribution (on the CD), and can also be found for download on the Internet, or can be ordered separately by contacting Microsoft sales and service at 1-800-426-9400.

45 Obtaining the HTML Help Workshop

45.1 Introduction to HTML Help

The HTML Help Workshop is a tool for creating compiled HTML help applets, that are common on many platforms. The compiled Help modules may then be used as stand-alone information sources, or as help-modules from inside a program. HTML help modules are relatively easy to create, and only require a knowledge of basic HTML to get started.

HTML Help modules can be separated by topic, and the HTML Help Workshop will automatically compile a table of contents, and create an option to search, if specified.

45.2 Obtaining the HTML Help Workshop

The HTML Help Workshop can be obtained from Microsoft as a free download. The HTML Help SDK should contain all the header files, library files, and object files that are needed to link help files into existing applications.

Download HTML Help[1]

1 `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/htmlhelp/html/hwMicrosoftHTMLHelpDownloads.asp`

46 Windows Programming/Key Combinations

Here is a key combination guide for computers:

46.1 IBM compatible

46.1.1 DOS

Cntrl+Alt+Delete - performs a soft reset

Left arrow, right arrow - Move cursor backward or forward one character

Cntrl + left arrow, right arrow - Move cursor backward or forward one word

Home, End - Move cursor to start or end of line

Up, down arrow - Scroll up and down stored (command) buffer

Page up, down - Places oldest or most recent (command) in command line

Insert - Toggle between insert and over-type modes. (square cursor denotes over-type mode)

Esc - Delete current line

F1 - Repeat text typed in preceding line character by character

F3 - Repeat text typed in preceding line (complete)

F5 - Cycle through entire current (command) buffer

F7 - Display all entries in current (command) buffer with line numbers

Alt+F7 - Delete all entries in current (command) buffer

F8 - As F5, If preceded by entry of character string, will load latest buffer line with corresponding characters (from start of buffer entry line)

F9+Buffer line number - Displays relevant buffer line; normally F7 first to show lines with numbers

Cntrl+C - Close down most applications and return to prompt

Shift+Alt+Print Screen - High efficiencey mode. Black and greens.(MTK sometimes... im a wizurd...)

46.1.2 Windows

All versions

- **Tab** Switching between controls in active window
- **Alt+F4** Closes the active window
- **Win+M** Minimize most programs
- **Win+D** Shows the desktop
- **Cntrl+Tab** Switching between Tabs in active window
- **Cntrl+C** Copies selected text, images or files into the clipboard
- **Cntrl+X** Cuts selected text, images or files into the clipboard
- **Cntrl+V** Paste data from the clipboard
- **Cntrl+A** Select all
- **Cntrl+Z** Undo
- **Shift+Del** Delete file permanently without keep in Recycle Bin (used by Windows Explorer)

Windows Vista or Newer

- **Win+Tab** switching between programs with the Windows Flip 3D effect

Windows Versions Before Windows 95

- **Cntrl+Alt+Del** brought up a blue screen

Category:Windows Programming[1]

1 `http://en.wikibooks.org/wiki/Category%3AWindows%20Programming`

47 Lists of Available APIs and related libraries

- Telephony Application Programming Interface (TAPI)[1] provides computer telephony integration and enables PCs running Microsoft Windows to use telephone services.

1 http://en.wikipedia.org/wiki/Telephony%20Application%20Programming%20Interface

48 GNU Free Documentation License

48.1 0. PREAMBLE

48.2 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

48.3 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

48.4 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the

48.5 4. MODIFICATIONS

9. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
10. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
11. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
12. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
13. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified version.
14. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
15. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

48.6 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

48.7 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

48.8 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

49 How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME.

Permission is granted to copy, distribute and/or modify this document

under the terms of the GNU Free Documentation License, Version 1.3

or any later version published by the Free Software Foundation;

with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in the section entitled "GNU

Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the

Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

50 Contributors

Edits	User
50	Adrignola[1]
1	Avicennasis[2]
1	Aya[3]
2	Az1568[4]
1	Boyage[5]
16	Carl Turner[6]
1	Carlo.milanesi[7]
1	Curtaintoad[8]
4	Darklama[9]
4	Derbeth[10]
5	EvanCarroll[11]
1	Glaisher[12]
1	Guanaco[13]
1	Hahc21[14]
1	HethrirBot[15]
1	Imp Wit[16]
1	Intermediate-Hacker[17]
1	J36miles[18]
3	Jguk[19]
8	Jomegat[20]
5	Maveric149[21]

1 http://en.wikibooks.org/wiki/User:Adrignola
2 http://en.wikibooks.org/wiki/User:Avicennasis
3 http://en.wikibooks.org/wiki/User:Aya
4 http://en.wikibooks.org/wiki/User:Az1568
5 http://en.wikibooks.org/wiki/User:Boyage
6 http://en.wikibooks.org/wiki/User:Carl_Turner
7 http://en.wikibooks.org/wiki/User:Carlo.milanesi
8 http://en.wikibooks.org/wiki/User:Curtaintoad
9 http://en.wikibooks.org/wiki/User:Darklama
10 http://en.wikibooks.org/wiki/User:Derbeth
11 http://en.wikibooks.org/wiki/User:EvanCarroll
12 http://en.wikibooks.org/wiki/User:Glaisher
13 http://en.wikibooks.org/wiki/User:Guanaco
14 http://en.wikibooks.org/wiki/User:Hahc21
15 http://en.wikibooks.org/wiki/User:HethrirBot
16 http://en.wikibooks.org/wiki/User:Imp_Wit
17 http://en.wikibooks.org/wiki/User:Intermediate-Hacker
18 http://en.wikibooks.org/wiki/User:J36miles
19 http://en.wikibooks.org/wiki/User:Jguk
20 http://en.wikibooks.org/wiki/User:Jomegat
21 http://en.wikibooks.org/wiki/User:Maveric149

5	Mike.lifeguard[22]
1	Mjchael[23]
1	NithinBekal[24]
154	Panic2k4[25]
5	Pcu123456789[26]
6	QuiteUnusual[27]
1	Ramac[28]
8	Recent Runes[29]
5	Robert Horning[30]
1	SB Johnny[31]
1	Sigma 7[32]
1	Van der Hoorn[33]
2	Webaware[34]
215	Whiteknight[35]
2	Withinfocus[36]
8	Wj32[37]
1	Wknight8111[38]
2	Xerol[39]

22 `http://en.wikibooks.org/wiki/User:Mike.lifeguard`
23 `http://en.wikibooks.org/wiki/User:Mjchael`
24 `http://en.wikibooks.org/wiki/User:NithinBekal`
25 `http://en.wikibooks.org/wiki/User:Panic2k4`
26 `http://en.wikibooks.org/wiki/User:Pcu123456789`
27 `http://en.wikibooks.org/wiki/User:QuiteUnusual`
28 `http://en.wikibooks.org/wiki/User:Ramac`
29 `http://en.wikibooks.org/wiki/User:Recent_Runes`
30 `http://en.wikibooks.org/wiki/User:Robert_Horning`
31 `http://en.wikibooks.org/wiki/User:SB_Johnny`
32 `http://en.wikibooks.org/wiki/User:Sigma_7`
33 `http://en.wikibooks.org/wiki/User:Van_der_Hoorn`
34 `http://en.wikibooks.org/wiki/User:Webaware`
35 `http://en.wikibooks.org/wiki/User:Whiteknight`
36 `http://en.wikibooks.org/wiki/User:Withinfocus`
37 `http://en.wikibooks.org/wiki/User:Wj32`
38 `http://en.wikibooks.org/wiki/User:Wknight8111`
39 `http://en.wikibooks.org/wiki/User:Xerol`

List of Figures

- EPL: Eclipse Public License. `http://www.eclipse.org/org/documents/epl-v10.php`

Copies of the GPL, the LGPL as well as a GFDL are included in chapter Licenses[40]. Please note that images in the public domain do not require attribution. You may click on the image numbers in the following table to open the webpage of the images in your webbrower.

40 Chapter 51 on page 225

1	Original uploader was Philcha[41] at en.wikipedia[42]	
2	MonoBot, Webaware, Whiteknight	
3	penubag[43] Yellow concept by User:Vipersnake151[44], penubag[45] Yellow concept by User:Vipersnake151[46]	

41 `http:////en.wikipedia.org/wiki/User:Philcha`
42 `http://en.wikipedia.org`
43 `http:////commons.wikimedia.org/wiki/User:Penubag`
44 `http:////commons.wikimedia.org/wiki/User:Vipersnake151`
45 `http:///wiki/User:Penubag`
46 `http:///wiki/User:Vipersnake151`

51 Licenses

51.1 GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow. TERMS AND CONDITIONS 0. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion. 1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work. 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary. 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures. 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee. 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

* a) The work must carry prominent notices stating that you modified it, and giving a relevant date. * b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to “keep intact all notices”. * c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it. * d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate. 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

* a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange. * b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge. * c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b. * d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements. * e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying. 7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

* a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or * b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or * c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or * d) Limiting the use for publicity purposes of names of licensors or authors of the material; or * e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or * f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way. 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10. 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so. 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it. 11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's “contributor version”.

A contributor's “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law. 12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program. 13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such. 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version. 15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

51.2 GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference. 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License. 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies. 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document. 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

* A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. * B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. * C. State on the Title page the name of the publisher of the Modified Version, as the publisher. * D. Preserve all the copyright notices of the Document. * E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. * F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. * G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. * H. Include an unaltered copy of this License. * I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. * J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. * K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. * L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. * M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version. * N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section. * O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version. 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements". 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document. 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate. 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title. 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it. 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document. 11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (C) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with … Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

51.3 GNU Lesser General Public License