Chapter One

1. Microsoft Windows and Visual C++

Enough has already been written about the acceptance of Microsoft Windows and the benefits of the graphical user interface (GUI). This chapter summarizes the Windows programming model (Win32 in particular) and shows you how the Microsoft Visual C++ components work together to help you write applications for Windows. Along the way, you might learn some new things about Windows.

2. The Windows Programming Model

No matter which development tools you use, programming for Windows is different from old-style batch-oriented or transaction-oriented programming. To get started, you need to know some Windows fundamentals. As a frame of reference, we'll use the well-known MS-DOS programming model. Even if you don't currently program for plain MS-DOS, you're probably familiar with it.

2.1 Message Processing

When you write an MS-DOS-based application in C, the only absolute requirement is a function named main. The operating system calls main when the user runs the program, and from that point on, you can use any programming structure you want. If your program needs to get user keystrokes or otherwise use operating system services, it calls an appropriate function, such as getchar, or perhaps uses a character-based windowing library.

When the Windows operating system launches a program, it calls the program's WinMain function. Somewhere your application must have WinMain, which performs some specific tasks. Its most important task is creating the application's main window, which must have its own code to process messages that Windows sends it. An essential difference between a program written for MS-DOS and a program written for Windows is that an MS-DOS-based program calls the operating system to get user input, but a Windows-based program processes user input via messages from the operating system.

> Many development environments for Windows, including Microsoft Visual C++ version 6.0 with the Microsoft Foundation Class (MFC) Library version 6.0, simplify programming by hiding the WinMain function and structuring the message-handling process. When you use the MFC library, you need not write a WinMain function but it is essential that you understand the link between the operating system and your programs.

Most messages in Windows are strictly defined and apply to all programs. For example, a WM_CREATE message is sent when a window is being created, a WM_LBUTTONDOWN message is sent when the user presses the left mouse button, a WM_CHAR message is sent when the user types a character, and a WM_CLOSE message is sent when the user closes a window. All messages have two 32-bit parameters that convey information such as cursor coordinates, key code, and so forth. Windows sends WM_COMMAND messages to the appropriate window in response to user menu choices, dialog button clicks, and so on. Command message parameters vary depending on the window's menu layout. You can define your own messages, which your program can send to any window on the desktop. These user-defined messages actually make C++ look a little like Smalltalk.