History of the Win32 API

Interestingly, Win32 wasn't slated to be the original programming interface to what was then called Windows NT. Because the Windows NT project started as a replacement for OS/2 version 2, the primary programming interface was the 32-bit OS/2 Presentation Manager API. A year into the project, however, Microsoft Windows 3.0 hit the market and took off. As a result, Microsoft changed direction and made Windows NT the future replacement for the Windows family of products as opposed to the replacement for OS/2. It was at this juncture that the need to specify the Windows API arose—before this, in Windows 3.0, the API existed only as a 16-bit interface.

Although the Windows API would introduce many new functions that hadn't been available on Windows 3.1, Microsoft decided to make the new API compatible with the 16-bit Windows API function names, semantics, and use of data types whenever possible to ease the burden of porting existing 16-bit Windows applications to Windows NT. This explains why many function names and interfaces might seem inconsistent: –this was required to ensure that the then new Windows API was compatible with the old 16-bit Windows API.

Services, Functions, and Routines

Several terms in the Windows user and programming documentation have different meanings in different contexts. For example, the word *service* can refer to a callable routine in the operating system, a device driver, or a server process. The following list describes what certain terms mean in this book:

- **Windows API functions** Documented, callable subroutines in the Windows API. Examples include *CreateProcess*, *CreateFile*, and *GetMessage*.
- **Native system services (or system calls)** The undocumented, underlying services in the operating system that are callable from user mode. For example, *NtCreateUserProcess* is the internal system service the Windows *CreateProcess* function calls to create a new process. For a definition of *system calls*, see the section "System Service Dispatching" in Chapter 3, "System Mechanisms."
- **Kernel support functions (or routines)** Subroutines inside the Windows operating system that can be called only from kernel mode (defined later in this chapter). For example, *ExAllocatePoolWithTag* is the routine that device drivers call to allocate memory from the Windows system heaps (called *pools*).
- **Windows services** Processes started by the Windows service control manager. For example, the Task Scheduler service runs in a user-mode process that supports the *at* command (which