

Chapter 1

What Is a Registry and Why?

SOME USERS OF WINDOWS know exactly what the registry is a system designed to cause users and administrators to lose their hair. I know this is true because I can no longer feel the wind ruffling through my hair. Oh, I feel the wind; I just don't feel the hair.

The registry is a simple, hierarchical database of information that Windows operating systems (and some applications) use to define the configuration of the system. Originally, in the early, simple days of Windows (16-bit Windows versions especially), the same information that is now stored in the registry was stored in text files. Though these text files were simple, their organization made access to the information they contained too slow to keep up with increasingly speedy technology.

Many applications use the registry the same way, though some applications are now moving to separate storage locations for their data—a technique that allows the applications to easily back up and restore their configuration data.

In this chapter, I'll discuss the history of the registry and define it in greater detail. We'll cover the following topics:

- ◆ The development of the registry
- ◆ How the registry is organized
- ◆ How the registry is used
- ◆ Distinctions in terminology

The Registry: Past and Present

The development of the registry, like Windows, has been evolutionary. The registry was preceded by a pair of flat-text files, called `win.ini` and `system.ini`. While the performance with these files left something to be desired, they formed the basis for today's registry.

In fact, these two files live on today in Windows XP, though they are virtually unchanged from Windows NT version 4. The first registry to appear in Windows was created to solve a number of problems: poor performance (retrieving information from the original flat-text `.ini` files was cumbersome), size limitations (the `.ini` files could be only so large), and maintenance problems (the `.ini` files were organizationally impaired!).

Today, the Windows XP system `.ini` files contain only a few entries used by a few applications. (Most are legacy 16-bit applications, though a few new programs are also placing some items in the `win.ini` file, too!)

These system `.ini` files are of no importance to us, and we may safely ignore them. For Windows XP, it's the registry that is most important to the system, because it contains the heart and soul of Windows XP. Without the registry, Windows XP would be nothing more than a collection of programs, unable to perform even the basic tasks that we expect from an operating system. Every bit of configuration information that Windows XP has is crammed into the registry. Information about the system's hardware, preferences, security, and users—everything that can be set is set in there.

However, times are a-changing. Microsoft now realizes that if every application stores application-specific information in the system registry, then the system registry can grow to an enormous size. That isn't quite what Microsoft had in mind when they created the registry structure. Microsoft's policy now states that applications may (and should) use standalone `.ini` files as needed.

Some advantages to using application-specific `.ini` files include these:

- ◆ Individual applications sometimes need to be restored from backup. With an application-specific `.ini` file, it is not necessary to back up and restore the entire registry to reinstall any single application. (This eliminates the attendant problem of restoring one part of the registry only to lose another part during the restoration!)
- ◆ The system registry has a practical limited size. Granted, the size is large, but some applications have lately been adding substantial content to the registry without regard to the fact (sad as it is) that the registry is a *shared* resource that everyone, including the system, must use! Once the registry gets too large, some registry operations may take an excessive amount of time.

NOTE Microsoft limits the size of any object that is stored in a registry data key to 1MB. This limit is basically only meaningful for `REG_BINARY` objects, because strings and such are unlikely to become this large. If you must store more than 1MB in a registry object, then store the information in a file and store a pointer to the file in the registry. Without this limitation, the registry could easily grow to be the largest file on your system.

FOR WINDOWS BEFORE WINDOWS XP

Windows 2000 and earlier versions set restrictions on registry size. If you approach your registry limit, you'll get a message stating that you are low on registry quota. This indicates that the registry has grown too large for the current size allocation. Unless you change it, the registry size is set to 25 percent of the paged pool size; for most computers, the paged pool size is approximately equal to the amount of installed RAM, up to a maximum of 192MB. The registry can be set to 80 percent of the paged pool size (80 percent of 192MB is just under 154MB, though good sense says to round down to 150MB).

Earlier versions of Windows adjust the registry size based on the currently installed RAM. Several registry entries affect registry size, though most users will find that the defaults are acceptable for their use. To create a very large registry, ensure that the amount of RAM installed is sufficient and set the `RegistrySizeLimit` and `PagedPoolSize` entries.

Organization

The registry is organized into five major sections. These sections are called *hives*, which are analogous to root directories on your hard drive. Each hive, by definition, has its own storage location (a file) and log file. If necessary, a given hive can be restored without affecting the other hives in the registry.

Inside a hive you find both keys (and subkeys, analogous to directories and subdirectories on your hard disk) and values. The term *value* (or data value, as it is sometimes called) refers to the information, or data, assigned to a key, making the key analogous to a file on your hard drive as well.

A key or subkey may have zero, one, or more value entries, a default value, and from zero to many subkeys. Each value entry has a name, data type, and a value:

- ◆ The entry's name is stored as a Unicode character string.
- ◆ The entry's type is stored as an integer index. The type is returned to the querying application, which must then map this type to the type that the application knows.
- ◆ The entry's value is stored as necessary to allow efficient retrieval of the data when needed.

Both the Windows XP operating system and applications store data in the Windows XP registry. This is both good and bad. It is good because the registry makes an efficient, common storage location. Here's the bad part: as I mentioned earlier, as more and more applications and systems store information in the registry, it grows larger, and larger, and larger.

It is most unusual for the registry to get smaller—I'm unaware of any application that does a really complete job of cleaning up all of its own registry entries when the application is uninstalled. Many applications leave tons of stuff in the registry when they are uninstalled, and not many applications clean up unused entries as a routine process. The end result is that the registry will grow, like Jack's magic beanstalk, as time goes on.

NOTE *From time to time in this book I'll refer to hives, keys, subkeys, and values using the generic term object. When the term object is used, assume that the item could be any valid item in the registry!*

Hives and Their Aliases

There are five main, or top level, hives in the Windows XP registry, and accepted abbreviations for each:

- ◆ HKEY_CLASSES_ROOT, a.k.a. HKCR
- ◆ HKEY_CURRENT_USER, a.k.a. HKCU
- ◆ HKEY_LOCAL_MACHINE, a.k.a. HKLM
- ◆ HKEY_USERS, a.k.a. HKU
- ◆ HKEY_CURRENT_CONFIG, a.k.a. HKCC

NOTE *The Windows 98 and Windows Me (Millennium Edition) HKEY_DYN_DATA hive, which has no abbreviation, does not exist in Windows XP, though Microsoft had originally intended to include information about Plug and Play in this hive. So where is PnP data saved if the HKEY_DYN_DATA hive is gone? Windows XP supports PnP, and Microsoft decided to integrate PnP data with the main registry rather than use a separate hive.*

Each hive begins with HKEY_. HKEY is an abbreviation for “hive key,” though the significance of this is not terribly important in understanding the registry. The H also signifies that the name is a “handle” for a program to interface with the registry. These handles are defined in the file `winreg.h`, included with the Windows XP SDK (Software Development Kit).

The registry contains duplication—sort of. For example, you’ll notice that everything in HKEY_CURRENT_USER is also contained in the hive HKEY_USERS. But these aren’t two different sets of the same information; rather, they’re two names for the same set of information. Microsoft needed to make some parts of the registry appear to be in two places at one time. But they didn’t want to copy these sections, because that could have created problems with keeping each of the two sections updated. Instead, they created an alias, or another name, for some registry components. The alias points to the original component and is updated whenever the original is. These aliases are created solely by Windows. You, as a user, can’t create an alias in the registry no matter how hard you try!

The most common alias is the registry hive HKEY_CURRENT_USER. It is an alias to either the .DEFAULT user or the current user in HKEY_USERS. If you take a quick peek at HKEY_USERS, you will see several keys there: one is .DEFAULT, and the others are named with long strings of characters. These are SIDs (security identifiers), which Windows XP uses to identify users. One of these subkeys for the currently logged-on user consists of just the SID, while the other consists of the SID suffixed with _Classes. For example, on one Windows XP server, the administrator has the two subkeys HKEY_USERS\S-1-5-21-1004336348-842925246-1592369235-500 and HKEY_USERS\S-1-5-21-1004336348-842925246-1592369235-500_Classes. I’ll clear up what a SID is and how it is used in Chapter 17.

NOTE *The default user, used when no user is logged on, has only one subkey, named .DEFAULT. (How do you edit the registry when no one is logged on? Simply by using remote registry editing, with a different computer.)*

There are also other aliases in the registry. For example, the registry key HKEY_LOCAL_MACHINE\System\CurrentControlSet is an alias to one of the other control sets—ControlSet001, ControlSet002, or sometimes ControlSet003. Again, this is that same magic; only one registry object is there, it just has two names. Remember, in modifying a specific registry key or subkey; don’t be surprised when another registry key or subkey seems to magically change also!

Data Values

A value may contain one or, in some instances, more than one data item. The only type of multiple-item value entry that the registry editor can handle is REG_MULTI_SZ, which may contain zero, one, or more strings.

Data is stored in a number of different formats. Generally the system uses only a few simple formats, while applications, drivers, and so forth may use more complex types defined for a specific purpose. For example, REG_RESOURCE_LIST is a complex registry type used primarily by drivers. Though it would be inefficient, all registry data could be considered to be REG_BINARY data.

Data types for value entries include:

- ◆ REG_BINARY
- ◆ REG_COLOR_RGB
- ◆ REG_DWORD
- ◆ REG_DWORD_BIG_ENDIAN
- ◆ REG_DWORD_LITTLE_ENDIAN
- ◆ REG_EXPAND_SZ
- ◆ REG_FILE_NAME
- ◆ REG_FILE_TIME
- ◆ REG_FULL_RESOURCE_DESCRIPTOR
- ◆ REG_LINK
- ◆ REG_MULTI_SZ
- ◆ REG_NONE
- ◆ REG_QWORD
- ◆ REG_QWORD_LITTLE_ENDIAN
- ◆ REG_RESOURCE_LIST
- ◆ REG_RESOURCE_REQUIREMENTS_LIST
- ◆ REG_SZ
- ◆ REG_UNKNOWN

NOTE *REG_QWORD* was new to Windows 2000 and is a quad-word (64-bit) numeric entry; *REG_QWORD_LITTLE_ENDIAN* is the same as *REG_QWORD*.

Applications may access each of these data types. Additionally, some applications store data in formats that only they understand. Actually, a provision in the registry allows the storing application to assign a specific type to the registry data. Any application or component that doesn't understand the format would simply treat the data as a *REG_UNKNOWN* type and read the data as binary.

NOTE *Oops, did I say something special? Yes! Don't forget that applications can and do store data in the registry, and that data needn't be one of the established registry data types.*

How the Registry Is Used

How does Windows XP use the registry? When is the registry first opened and used?

WHAT IS WINDOWS XP?

Windows XP comes in a number of versions, including a Home version and a Professional version. Windows XP Home is configured for home users. Windows XP Professional, which is configured to work as a workstation client, is a somewhat more powerful configuration for business users. Throughout this book, I'll point out any differences in usage between the Home and Professional versions.

While not the focus of this book, Windows XP also comes in a number of server versions named Windows XP .NET. Microsoft has planned several server product offerings, including Windows XP .NET Server and Windows XP .NET Advanced Server. We don't expect that there will be major changes in .NET's use of the registry.

The registry is a tree-based hierarchical system that offers quick access to data stored in almost any format. Actually, the registry is a rather flexible database. Registry information comes from a number of sources:

- ◆ From installing Windows XP
- ◆ From booting Windows XP
- ◆ From applications, systems, and user interaction

Every component of Windows XP uses the registry, without exception. A set of APIs allows both Windows XP and other applications to access registry information easily and quickly.

Windows XP starts to use the registry at the very beginning stages of system bootup. The Windows XP boot process is based on which file format is installed, though the important parts are identical in either case. The unimportant parts are the loading of the specific drivers to read the NTFS file system.

NOTE *Throughout this book, I'm referring to Windows XP installed on an Intel x86 platform. There are differences in the boot process on RISC-based systems (such as the Digital Alpha system), though these differences are not terribly significant, considering how the registry is used. However, it seems that non-Intel systems are becoming very unusual, and they probably will receive little or no support from Microsoft in the future.*

The Windows XP boot process consists of the following steps:

1. The system is powered up, the video is initialized, and the hardware self-tests are performed. The BIOS performs these tests, which are called POSTs (power-on self-tests). Usually, the memory test is the most visible one; its progress is shown on most computer screens.
2. After running POST, the system initializes each adapter. If the adapter has its own built-in BIOS, the adapter's BIOS is called to perform its own initialization. For IDE adapters (most computers have either two or four IDE adapters), each connected drive (there may be up to two drives for each IDE adapter, allowing for a total maximum of eight IDE type drives) is queried for its specifications and access method.

Some adapters, such as Adaptec's SCSI adapters, display messages and allow the user to interact. Some adapters that don't have a BIOS aren't initialized until Windows XP loads their drivers much later in the boot-up process.

3. After all the adapters that have a BIOS have been initialized, the system boot loader reads in the sector located at the very beginning of the first bootable disk drive and passes commands to this code. This sector is called the *boot sector*, or the MBR (Master Boot Record), and it is written by the operating system when the operating system is installed.
4. The code in the MBR then loads the NTLDR file. (This file has no extension, though it is an executable file.) Once loaded, the MBR passes control to the code in NTLDR.
5. NTLDR then switches into 32-bit mode. (Remember, an Intel x86 processor always boots into 16-bit real mode.) It then loads a special copy of the necessary file system I/O files and reads in the file `boot.ini`.
6. The file `boot.ini` has information about each operating system that can be loaded. Remember, Windows XP supports multiboot configurations. It is trivial to create a Windows XP installation that can boot Windows NT, Windows XP, and Windows 95 or Windows 98. The boot loader can even boot two different copies of Windows XP with either the same or different version numbers. NTLDR then processes `boot.ini`, displaying boot information that allows the user to select which operating system will be loaded. At this point, let's assume that Windows XP will be loaded.
7. When you select Windows XP to be loaded, NTLDR loads the file `ntdetect.com`. This program then collects information about the currently installed hardware and saves this information for the registry. Most of this information is stored in the `HKEY_LOCAL_MACHINE` hive.
8. Once NTDETECT has detected the hardware, control is passed back to NTLDR, and the boot process continues. At this point, the registry has been substantially updated with the current hardware configuration, which is stored in `HKEY_LOCAL_MACHINE\Hardware`.
9. The prompt to select the configuration is then presented. This prompt, "Press spacebar now to invoke Hardware Profile/Last Known Good menu," allows you to force Windows XP to use a specific configuration as stored in the registry hive `HKEY_LOCAL_MACHINE`.
10. Following the detection of NTDETECT, NTLDR loads and initializes the Windows NT kernel, loads the services, and then starts Windows.
11. When the kernel is loaded, the HAL is also loaded. (The HAL—Hardware Abstraction Layer—is used to manage hardware services.) Next, the registry system subkey `HKEY_LOCAL_MACHINE\System` is loaded into memory. Windows XP scans the registry for all drivers with a start value of zero. This includes those drivers that should be loaded and initialized at boot time.
12. You can see the beginning of the next stage, kernel initialization. The screen switches to a blue background, and you see a message about the Windows XP build number and the number of system processors. Again, the system scans the registry and finds all drivers that must be started at the kernel initialization stage.
13. From this point, Windows XP starts various components and systems. Each component and system reads the registry and performs various tasks and functions. In the final stage, the program that manages the user logon, WinLogon, starts. WinLogon allows the user to log on and use Windows XP.

Once Windows XP is booted, both the operating system and applications use the registry. The registry is dynamic, but usage of the registry may be dynamic or static. That is, some registry items are read one time and never reread until the system is restarted. Other items are read every time they are referenced. There is no fixed rule as to what is read each time it is needed and what is not, but to be on the safe side, follow these guidelines:

- ◆ Application-related data is probably read when the application starts. If you change application-based data, restart the application. In fact, the best path to follow is this: do not change application-based data while the application is running.
- ◆ User-interface data is sometimes dynamic, sometimes static. With user-interface data, the way to go is to change the data and wait to see the results of the change. If the change doesn't appear, try logging on again.
- ◆ System data is usually either static or otherwise buffered. Many system-related registry changes won't become effective until the system is restarted. Some system data is rewritten, or created, at startup time, precluding changes by users. Many of the items in HKEY_LOCAL_MACHINE may be reset at system boot time, especially those items that are hardware related.

A Note on Terminology

The registry is made up of hives, keys, subkeys, and value entries. Well, actually, depending on the source, you may be faced with hives and data keys, or keys and items, or just data keys, or who knows what else.

There is some indication that Microsoft wants to drop the original term for a registry section—the *hive*—and replace this term with the word *key*. In the Windows NT Resource Kit, Microsoft makes the following definition:

The registry is divided into parts called *hives*. A hive is a discrete body of keys, subkeys, and values rooted at the top of the registry hierarchy. Hives are distinguished from other groups of keys in that they are permanent components of the registry; they are not created dynamically when the system starts and deleted when it stops. Thus, HKEY_LOCAL_MACHINE\Hardware, which is built dynamically by the Hardware Recognizer when Windows NT starts, is not a hive.

In the Windows XP documentation, Microsoft says a hive is:

A section of the registry that appears as a file on your hard disk...

These definitions are absolute and state exactly what is a hive and what is not. However, in the real world, no one follows this exact definition. Many authors call all holders of information *hives* (or *sub-hives*) and call data objects *keys*. Others never refer to hives at all, and instead call all holders *keys*, or *subkeys*, and refer to data objects as *values*.

Virtually every definition leaves something to be desired. To call the thing that holds data a “value entry” sometimes makes it awkward to refer to the contents. Consider these examples:

The value entry named `asdf` contains the value 1234.

The value called `asdf` contains the value 1234.

The following example is much more readable:

The value entry `asdf` is a `REG_DWORD` with a value of 1234.

Is there a need to distinguish between what Microsoft calls a “hive” (a top-level, permanent, registry component) and what Microsoft calls a “key”? When does a hive become a key, and is this important? I can’t think of any context in which anything is gained by making this distinction. Referring to the top-level objects as *hives* certainly frees up the term *key* to be used elsewhere, but why not stick to one term?

Table 1.1 compares registry terminology against the terminology used for the Windows file system—and gives the terminology I’ll be using in this book.

TABLE 1.1: REGISTRY TERMINOLOGY EXPLAINED

CONTEXT	ROOT COLLECTIONS	SUBCOLLECTIONS	OBJECTS	DATA
Disks	Root directories		Files	Data
Older registry terminology	Hives	Subhives	Data keys	Data
Newer registry terminology	Hives	Keys/subkeys	Value entry	Data
Registry terminology used in this book	Hives	Keys/subkeys*	Value entry	Data

**Just to keep things easy to read, I’ll use the term key to refer to both keys and subkeys.*