

2nd Edition  
Covers Service Pack 2



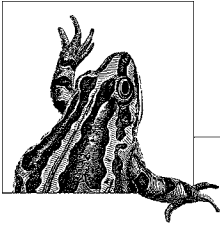
# WINDOWS XP

## IN A NUTSHELL

*A Desktop Quick Reference*

O'REILLY®

*David A. Karp,  
Tim O'Reilly & Troy Mott*



## The Registry

The Windows Registry is a database of settings used by Windows XP and the individual applications that run on it. Knowing how to access and modify the Registry effectively is important for troubleshooting, customizing, and unlocking hidden features in Windows XP.

An amazing amount of what one might assume to be “hardwired” into Windows—the locations of key directories, the titles of on-screen objects such as the Recycle Bin, and even the version number of Windows XP reported in Control Panel—is actually the product of data stored in the Registry. Change a setting in the Registry and key parts of your system can be affected; for this reason, Microsoft passively discourages tampering by providing only minimal user documentation on the Registry Editor, and no documentation at all on the structure of the Registry itself.



Despite the enormous potential for harm, the Registry is fairly robust, and for every entry that you can wreak havoc by changing, there are hundreds that you can change with impunity. Nonetheless, you should back up the Registry files before making significant changes with Registry Editor. See “Backing Up the Registry,” later in this chapter, for details.

The Registry is normally consulted silently by the programs (such as Explorer) that comprise the Windows user interface, as well as by nearly all applications. Programs also commonly write varying amounts of data to the Registry when they are installed, when you make changes to configuration settings, or just when they are run. For example, a game like FreeCell keeps statistics in the Registry on how many games you’ve won and lost. Every time you play the game, those statistics are updated. For that matter, every time you move an icon on your Desktop, its position is recorded in the Registry. All of your file type associations are stored in the Registry, as well as all of the network, hardware, and software settings for

Windows and all of the particular configuration options for most of the software you've installed. The settings and data stored by each of your applications and by the various Windows components vary substantially, but more often than not, a given Registry setting will appear in plain English, making it relatively easy to decipher. There are also several advanced techniques that not only help to identify more obscure settings, but allow you to use undocumented settings to uncover hidden functionality.

Microsoft provides the Registry Editor (*regedit.exe*), which is used to view and modify the contents of the Registry. Don't confuse the Registry with the Registry Editor; the Registry Editor merely reads and writes data in the Registry like any other Windows application. When you start Registry Editor, you'll see a window similar to the one in Figure 8-1.

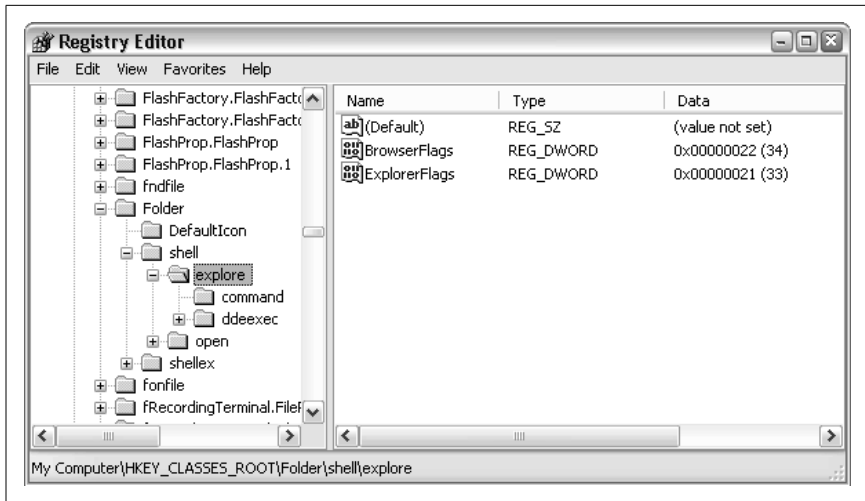


Figure 8-1. The Registry Editor uses a familiar interface to manipulate unfamiliar data

The organizational structure of the Registry is hierarchical, so Microsoft chose an interface familiar to anyone who has used Windows Explorer. As in Explorer, there are two panes: the folders (keys) are displayed in a cascading tree on the left, and the contents of the currently selected key appear on the right. Use the small plus (+) and minus (-) icons to expand and collapse the branches, respectively; cursor keys also work here.

While the interface elements might appear familiar, the data that is manipulated with Registry Editor is nothing like the files and folders we deal with in Explorer. Although you can certainly dive in and begin wading through the thousands of keys and values in the Registry, you're not likely to find anything of value until you arm yourself with a basic understanding of the way data is stored and organized in the Registry. And, of course, this is the focus of the next few sections.

# What's in the Registry

Data in the Registry is stored in individual pieces called *values*. Every value has a name and is capable of holding one of several types of data. Values are grouped and organized in *keys*, which are represented by Folder icons in Registry Editor. Keys can also contain other keys, thereby forming the basis for the hierarchy in the Registry. Like Explorer, Registry Editor arranges the keys in a collapsible tree structure, allowing you to navigate through the branches to locate a particular key, and hence, all the values contained therein.

Often, in order to view or modify a certain key or value, one must follow a *Registry path*. A path is merely a series of key names, separated by backslashes (\), used to specify an absolute location in the Registry. For example, to navigate to HKEY\_CURRENT\_USER\Control Panel\Keyboard, simply expand the HKEY\_CURRENT\_USER branch by clicking on the plus sign (+) next to it, then expand the Control Panel branch, and finally click on the Keyboard key name to display its contents. The path leading to the currently highlighted key is always shown at the bottom of the Registry Editor window.



It's easy to get confused about keys and values. In fact, value names sometimes appear at the end of a path, although this is mostly a holdover from the early days of the Registry. It's important to realize that only values can contain data, while keys are only used to organize values—just like files and folders in Explorer, respectively. Note that unlike folders in Explorer, keys never appear in the right pane of the Registry Editor window, even though keys can contain other keys.

Every key contains a value named (Default). If the default value contains no data, you'll see (value not set), as in Figure 8-2. If a given key contains other values, they will be listed below the default value. To modify the data stored in a value, simply double-click on the value name, or highlight it and select Modify from the Edit menu. To rename a value, which is not the same as changing its data, highlight it and press F2 or right-click it and select Rename.

For example, if I wanted to change the location of my Word Startup Folder, I could navigate to HKEY\_CURRENT\_USER\Software\Microsoft\Office\8.0\Word\Options, double-click on the Startup-Path value, and use the edit dialog box shown in Figure 8-2 to type new data.

The data stored in the Startup-Path value is a string of text, which means that Startup-Path is a *string value* (the most common type). There are seven types of values in all, each having a common name and a symbolic name (shown in parentheses in the following list). While all value types can be viewed and modified in Registry Editor, only three can be created.

## String values (REG\_SZ)

String values contain *strings* of characters, more commonly known as text. Most values of interest to us are string values; they're the easiest to edit and are usually in plain English. In addition to standard strings, there are two far less common string variants, used for special purposes:

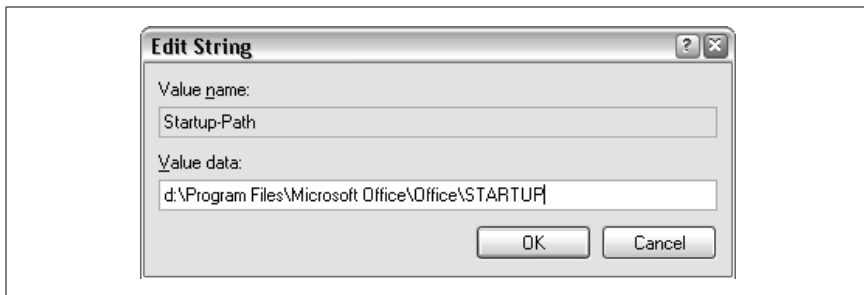


Figure 8-2. Editing a string value to change Microsoft Word Startup directory

#### String array value (REG\_MULTI\_SZ)

Contains several strings (usually representing a list of some sort), concatenated (glued) together and separated by null characters (ASCII code 00). You can't create these in the Registry Editor, but you can edit them. The dialog used to modify these values is the same as for binary values. Note that the individual characters in REG\_MULTI\_SZ keys are also separated by null characters, so you'll actually see three null characters in a row between multiple strings.

#### Expanded string value (REG\_EXPAND\_SZ)

Contains special variables into which Windows substitutes information before delivering to the owning application. For example, an expanded string value intended to point to a sound file may contain %SystemRoot%\media\startup.wav. When Windows reads this value from the Registry, it substitutes the full Windows path for the variable, %SystemRoot%; the resulting data then becomes (depending on where Windows is installed) c:\windows\media\startup.wav. This way, the value data is correct regardless of the location of the Windows folder. You can't create these in the Registry Editor, but you can edit them.

#### Binary values (REG\_BINARY)

Similarly to string values, binary values hold strings of characters. The difference is the way the data is entered. Instead of a standard text box, binary data is entered with hexadecimal codes in an interface commonly known as a *hex editor*. Each individual character is specified by a two-digit number in base-16 (e.g., 6E is 110 in base 10), which allows characters not found on the keyboard to be entered. See Figure 8-3 for an example. Note that you can type hex codes on the left or normal ASCII characters on the right, depending on where you click with the mouse.

Binary values are often not represented by plain English and, therefore, should be left unchanged unless you either understand the contents or are instructed to do so by a solution in this book.

#### DWORD values (REG\_DWORD)

Essentially, a DWORD is a number. Often, the contents of a DWORD value are easily understood, such as 0 for no and 1 for yes, or 60 for the number of seconds in some timeout setting. A DWORD value is used only where numerical digits are allowed; string and binary types allow anything.

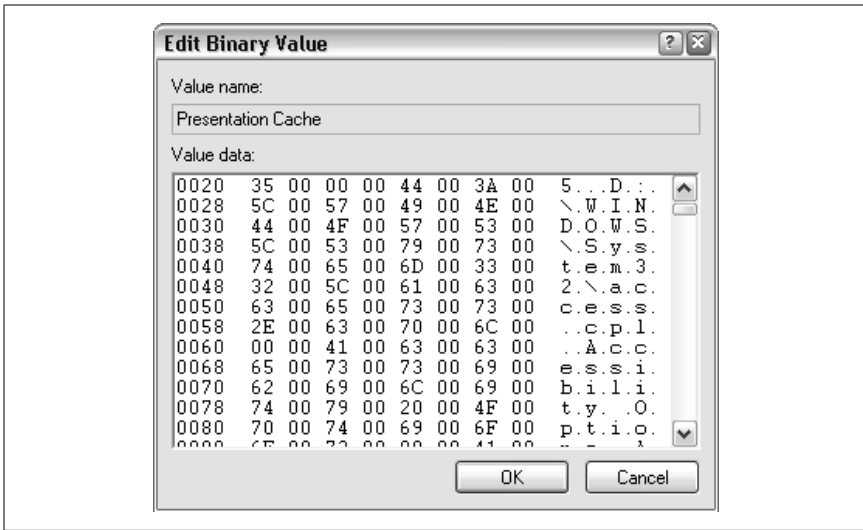
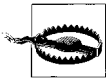


Figure 8-3. Binary values are entered differently from the common string values, but the contents are sometimes nearly as readable



In some circumstances, the particular number entered into a DWORD value is actually made up of several components, called bytes. The REG\_DWORD\_BIGENDIAN type is a variant of the DWORD type, where the bytes are in a different order. Unless you're a programmer, you'll want to stay away from these types of DWORD values.

The DWORD format, like the binary type, is a hexadecimal number, but this time in a more conventional representation. The leading 0x is a standard programmer's notation for a hex value, and the number is properly read from left to right. The equivalent decimal value is shown in parentheses following the hex value. What's more, when you edit a DWORD value, the edit dialog box gives you a choice of entering the new value in decimal or hex notation.

In general, if a value is stored in binary or DWORD format, you can guess that it was either programmatically generated or the program's author wished to make the value a little more obscure and difficult to edit. However, if you know what you are doing, you can edit binary or DWORD values almost as easily as you can string values. For example, if I want to lie to my friends to tell them I've won 435 games of FreeCell rather than just one, I simply need to double-click on "wins" and edit the value as shown in Figure 8-4.

Even if you're not a programmer, you can figure out hexadecimal values pretty easily with the Windows Calculator (*calc.exe*; see Chapter 4). Just enter the number you want to convert and click the Hex radio button to see the hexadecimal equivalent; 435 decimal is equal to 1B3 hex. Note, however, that hex values stored in binary Registry values are displayed in a somewhat unconventional format, in which the lowest-order digits appear first, followed by the next-higher pair of digits, and so on. In other words, the

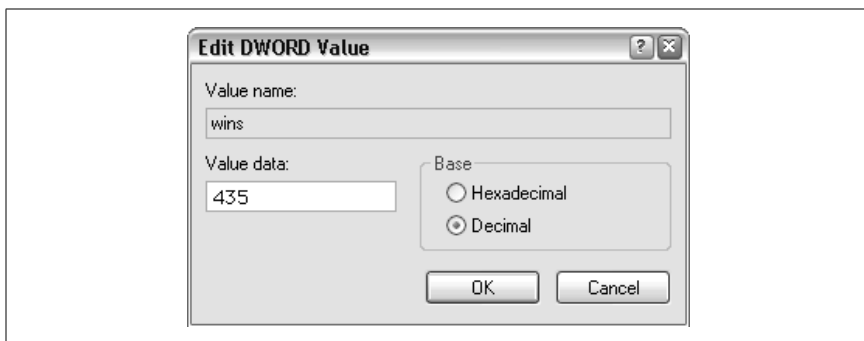


Figure 8-4. Editing a DWORD value to alter the number of games won in FreeCell

digits in a binary value are paired, and their order reversed: the hex value 1B3 thus needs to be entered as B3 01. If you want to convert a binary value shown in Registry Editor to decimal, you'll have to reverse this notation. For example, to find the decimal equivalent of 47 00 65 6e, set Calculator to hexadecimal mode and enter 6e650047, and then switch to decimal mode to display the decimal equivalent, 1,852,112,967.



If you aren't sure about the meaning of a specific Registry value, don't be afraid to experiment. Experimenting might include editing a value with Registry Editor, but it might be easier or safer to work from the other end: open the application whose data is stored there (e.g., a Control Panel applet), change a setting, and watch how the Registry data changes. In this way, you can derive the meaning of many binary-encoded values. Note that while the Registry data will often change immediately, you may need to press F5 (Refresh) to force Registry Editor to display the newly affected data.

It's a good idea, though, to make a backup copy of a Registry key before making any changes. See "Adding and Deleting Registry Keys and Values" and "Exporting and Importing Registry Data with Patches" later in this chapter for details.

Figure 8-4 shows an additional value, called "wins," which I entered into the FreeCell key using Registry Editor → Edit → New → DWORD Value. This example illustrates a very important point: a Registry entry is superfluous unless a program actually reads it. You can enter new keys and values all you like, with the only consequence being that you've bloated your Registry. (Note that there are sometimes undocumented Registry values that are meaningful to a program but that are not normally present; adding them to the Registry can make useful changes; see *Windows Me Annoyances* by David Karp [O'Reilly] for several examples.) The chief concern is in deleting or modifying existing entries; the odds of randomly creating a value that an application might be looking for are extremely small.

I take advantage of this fact by occasionally leaving myself notes in new Registry values. For example, before modifying a value, I might place a backup of its data in a new value in the same key. The application will ignore it, and it has sure come in handy for me to have a record of the original value!

A final note: any changes made in Registry Editor are saved automatically and immediately; there's no "undo" command in Registry Editor, and the automatic Registry backups made by Windows are of little use when small changes are made. The saving grace is the use of Registry patches, discussed later in this chapter.

## Adding and Deleting Registry Keys and Values

The Registry Editor, as mentioned earlier, is the primary tool for viewing, modifying, and deleting data in the Registry. And as you'll see later in this chapter, it also allows you to conveniently import and export data (via Registry patches), which can be thought of as another form of data entry.

Basic data entry in Registry Editor is fairly simple. In order to type data, you must first create a value to hold it. Depending on your goal, you may also need to create a new key in which to place the value.

To create a new key or value, use Edit → New. The key or value then appears within the currently selected key, with the name New Key or New Value #1, respectively. A new string value will have the null string as its value; a new binary value will show the following message in parentheses: "(zero length binary value)." A new DWORD value will show up as zero: 0x00000000 (0). You can then edit that value (see later in this chapter) to change it. New keys aren't created empty, either. They all contain the (Default) value described in the previous section.

To delete a key or value, select it and click Edit → Delete, or simply press the Del key. But be warned, there's no undelete, so you might want to first write out the branch containing the key you're about to delete as a .reg file (see "Backing Up the Registry" and "Exporting and Importing Registry Data with Patches," later in this chapter). Or, you can use Edit → Rename to rename the value or key. Since applications access values and keys by their names, renaming has the effect of disabling or "hiding" the item from the application that uses it, while preserving its data.

To edit a value, double-click on its icon or name, or highlight it and use Edit → Modify. You will see an edit box appropriate to the value type. When editing a string value, Registry Editor provides a standard text box, in which you can type text, as well as copy and paste (by right-clicking on the text, or by using Ctrl-C, Ctrl-X, and Ctrl-V). Binary values have a more complicated and less familiar edit dialog, which allows data entry via ASCII codes on the left, or plain text on the right. Finally, DWORD values have a simpler edit interface, providing only for entry of a single number (either in Hex or Decimal format). Cut, Copy, and Paste work in DWORD and Binary dialogs as well. See the previous section, "What's in the Registry," for more details on the different types of values.

Unfortunately, automation in Registry Editor is virtually nonexistent. For example, you can't copy and paste whole keys or values like you might expect (given the familiar Explorer-like interface), but you can copy key and value *names* to the clipboard by pretending you're going to rename them, and then pressing Ctrl-C to copy. Another useful tool is the Copy Key Name command on the Edit menu, which copies the full path to the selected key to the Clipboard (very handy

for writing this book, for example). It doesn't copy the contents of the key, nor does it include the selected value, however.

If you want to duplicate an existing value, double-click it and select all of the data in the Edit window (see Figure 8-5).

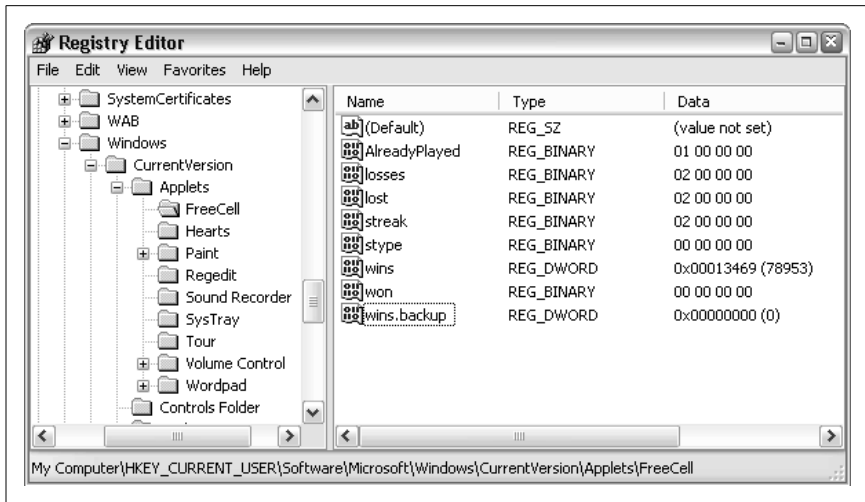


Figure 8-5. Copying an existing Registry value

Ctrl-C will copy the data to the clipboard. Then create a new value, being sure to match the type (string, binary, or DWORD) of the original value. Type the desired value name, double-click the new value to edit it, and then use Ctrl-V to paste the copied data into the edit window.

Duplicating values can be handy not only when using an existing value as a template for a new value, but also whenever you're going to make changes to an existing value. You can make little "inline backups" by creating a new value (*whatever.bak*, for instance) and pasting in the old value data before you change it. This might seem a little tedious, but it might prevent future headaches if you're about to change a complex value whose format you aren't completely sure you understand, or even if you anticipate having to roll back a value to its previous state for some reason.

Unfortunately, there's no easy way to copy a key and all of its contents in Registry Editor. If you want to copy an entire key and all its values, you'll have to do it one value at a time. It's usually much easier to export the key, edit the resulting file with a text editor, and then import the edited file. (See "Exporting and Importing Registry Data with Patches" later in this chapter.)

In addition to the Edit menu, you may find Registry Editor's context menus convenient. Right-clicking on a key in the left pane gives a context menu with Expand or Collapse, New, Find, Delete, Rename, and Copy Key Name. (Expand displays a key's subkeys. It will be grayed out if there are no subkeys to display, and it will be replaced with Collapse if said subkeys are already showing.) Right-clicking with a value selected in the right pane gives a context menu with Modify,

Delete, and Rename. Right-clicking in the right pane with no value selected gives a context menu with New (to create a new string, binary, or DWORD value). Press Shift-F10 to open a context menu without having to use the mouse.

## Organization of the Registry

The Registry is enormous and complex; a full Registry might easily contain 15,000 keys and 35,000 values. Entire books have been written about it, and we can't do it justice here. Our purpose in this section is to arm you with a basic understanding of how the Registry is organized, not to document individual values in detail or suggest changes you might want to make with Registry Editor.



David A. Karp's book *Windows Me Annoyances* provides many tips and tricks that rely on the Registry. If you can still find Ron Petrusha's out-of-print book, *Inside the Windows 95 Registry*, you'll find lots of useful information there. While aimed primarily at software developers and covering Windows 95, it contains several chapters aimed at experienced users. In particular, see Chapters 1 to 3, which give a good overview of the Registry and a detailed description of how to use Registry Editor. Also see Chapter 8.

The top level of the Registry is organized into five main *root* branches. By convention, the built-in top-level keys are always shown in all caps, even though the keys in the Registry are not case-sensitive. (For example, HKEY\_CURRENT\_USER\SOFTWARE\MICROSOFT\Windows is identical to HKEY\_CURRENT\_USER\Software\Microsoft\Windows.) Their purposes and contents are listed in the following summaries. Note that the root keys are sometimes abbreviated for convenience in documentation (although never in practice); these abbreviations are shown in parentheses. Subsequent sections go into the contents of the root keys in more detail.

### HKEY\_CLASSES\_ROOT (HKCR)

Contains file types, filename extensions, URL protocol prefixes, and registered classes. The information in this branch can be thought of as the “glue” that binds Windows with the applications and documents that run on it. It is critical to drag-and-drop operations, context menus, double-clicking, and many other familiar user interface semantics. The actions defined here tell Windows how to react to every file type available on the system.

This entire branch is a mirror (or symbolic link) of HKEY\_LOCAL\_MACHINE\SOFTWARE\Classes, provided as a root key purely for convenience.

### HKEY\_CURRENT\_USER (HKCU)

Contains user-specific settings for the currently logged-in user. This entire branch is a mirror (or symbolic link) of one of the subkeys HKEY\_USERS (see below). This allows Windows and all applications to access and store information for the current user without having to determine which user is currently logged in.

An application that keeps information on a per-user basis should store its data in `HKEY_CURRENT_USER\Software`, and put information that applies to all users of the application in `HKEY_LOCAL_MACHINE\SOFTWARE`. However, it is somewhat arbitrary what Windows applications seem to consider user-specific and what is for all users on the machine. Like many aspects of Windows, the Registry provides a mechanism for applications to store configuration data, but does little to enforce any policies about how and where it will actually be stored.

#### `HKEY_LOCAL_MACHINE (HKLM)`

Contains information about hardware and software on the machine not specific to the current user.

#### `HKEY_USERS (HKU)`

Stores underlying user data from which `HKEY_CURRENT_USER` is drawn. Although several keys will often appear here, only one of them will ever be the active branch. See the discussion of `HKEY_USERS`, below, for details.

#### `HKEY_CURRENT_CONFIG (HKCC)`

Contains hardware configuration settings for the currently loaded hardware profile. This branch works similarly to `HKEY_CURRENT_USER`, in that it is merely a mirror (or symbolic link) of another key. In this case, the source is `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Hardware Profiles\XXXX`, in which `XXXX` is a key representing the numeric value of the hardware profile currently in use. On a system with only a single hardware profile, its value will most likely be `0001`.

## **`HKEY_CLASSES_ROOT (HKCR)`**

At first glance, Windows XP seems very object-oriented. Files, folders, and devices are represented by icons that respond differently to various actions such as single or double-clicks, right-clicks, and left-clicks. But in a true object-oriented system, the object itself contains the knowledge of how to respond to events such as mouse clicks.

By contrast, Windows XP performs much like the Wizard of Oz, not with true object-oriented magic, but with a complex machinery hidden behind a screen. The knowledge of how the Explorer should treat each object is stored in the Registry in a complex chain of interrelated keys.

Much of the system's behavior depends on filename extensions. A *filename extension*, the string of letters that appear after the last dot in a filename, is the primary mechanism Windows uses to determine a file's type. Therefore it's essential that each filename extension accurately reflect the file type. For example, a file named *stuff.txt* will be treated by Windows as a text file. If one were to rename the file to *stuff.old*, it would still be a text file, but Windows would treat it differently: it would have a different icon in Explorer, a different description in Explorer's Details view, and a different action when double-clicked. (The exceptions are certain Microsoft Office documents, which are handled by Windows in a consistent manner regardless of the filename extension.) This illustrates how fragile and fallible Windows' file types system is, how some applications can so easily register themselves as the default for any given file type, and how stupid it is that filename extensions are hidden in Explorer by default.

When you open `HKEY_CLASSES_ROOT`, the first thing you'll see is a very long list of file extensions known to the system, from something like `.ai` (Adobe Illustrator Document) to `.zip` (Zip archive). What follows is a list of *document type keys*, which typically contain the actual file type information. These two sets of keys make up the file types in Windows XP.

Here's how it works: a file extension key (one which has a dot at the beginning of its name) has its default value set to the name of a document type key (and thus "points" to that key). For example, `HKEY_CLASSES_ROOT\.txt`, which corresponds to all files with the `.txt` extension, has its default value set to `txtfile`. Lower down, the `HKEY_CLASSES_ROOT\txtfile` key contains several keys that describe files of this type, and instruct Windows how to handle double-clicks, right-clicks, and other operations.

You also may notice that `HKEY_CLASSES_ROOT\.log` key also has its default value set to `txtfile`; in this way, many extension keys can point to a single file type key, and hence, a single file type can encompass several different filename extensions.

Applications frequently add new file types to the Registry, registering themselves with certain filename extensions. In the case of the `txtfile` file type, Notepad is registered as the default application when Windows XP is first installed; thus, when one double-clicks on a `.txt` file or a `.log` file, the file is opened in Notepad.

A common conflict occurs when two or more applications find themselves fighting to be the default application. For example, should a file with the `.htm` extension be opened by Netscape Navigator or Microsoft Internet Explorer? If you use Navigator, the `.htm` key might have the value name (Default) with the data `NetscapeMarkup`. If you use Internet Explorer, the value name (Default) will have the value data `htmlfile`. If you then look at either of those two class definition keys (`NetscapeMarkup` or `htmlfile`), you'll see a different chain of subkeys. While both Navigator and Internet Explorer know how to handle HTML files, they use a different set of internal instructions for figuring out how to display or edit the files, which icon to display for the file, and so on.

The detailed subkeys and values that appear under the class definition and document type keys start to get really confusing. (See *Windows Me Annoyances* for an in-depth examination of file types.)

Because each program may record and retrieve different keys, it's very hard to generalize about them. The best we can do is mention some of the kinds of keys you might see associated with a particular file extension subkey or class definition subkey. Here are some of the most common keys and values you may find in `HKEY_CLASSES_ROOT`, sorted alphabetically:

#### *CLSID*

Class identifier for an ActiveX component, a unique, 16-bit 16-byte number in the following format: {aaaaaaaa-bbbb-cccc-dddd-ffffffffffff}, in which each letter represents a hexadecimal digit. (That's a sequence of eight, four, four, four, and twelve hex digits, with a hyphen between each group of digits and the whole thing enclosed in curly braces.)

CLSID appears both as a subkey of many file type definition keys and as a class definition key in its own right. That is, the key `HKCR\NetscapeMarkup` might have a subkey `CLSID` with the data value `{61D8DE20-CA9A-11CE-9EA5-0080C82BE3B6}`, but there's also a key called `HKCR\CLSID` with the subkey `{61D8DE20-CA9A-11CE-9EA5-0080C82BE3B6}`, which in turn has the data value `Netscape Hypertext Document`. The first entry is simply a pointer to the second, which contains the actual class data. You must always be on the lookout for this kind of indirection.

CLSID keys don't necessarily correspond to filename extensions or file types. The `HKEY_CLASSES_ROOT\CLSID` branch, for example, contains a huge list of class ID keys, which each represent a different component. Most of the components are of little interest to mere mortals, but some correspond to visual elements in Windows XP. If you search for "Recycle Bin" here, for example, you'll find the key that describes the Recycle Bin Desktop icon. Try changing the contents of the Default value to "Trash," refresh your Desktop, and see what happens!

### *Content Type*

The data in this value is the Multipurpose Internet Mail Extension (MIME) descriptor for the corresponding file type. This key will typically appear in the file type key for Internet-related file types such as GIF and JPEG. It's used by email programs that support attachments and web browsers, such as Netscape Navigator and Internet Explorer, to help them identify downloaded files.

### *DefaultIcon*

The location (usually a pathname and an optional "icon index" within the file) of the file containing the default icon to use for a file type or CLSID. If you see the value data "%1", it means that the file will act as its own icon (a neat effect when used with BMP files).

Note that there may be more than one default icon for a given file type. A good example is the Recycle Bin, which shows a different icon when it is empty and when it is full. In cases like this, the program knows to copy its icon for the appropriate state to the `DefaultIcon` (Default) value. In other cases, though, a `DefaultIcon` may be specified in more than one place (e.g., under a document type key and under an associated CLSID key).

### *Shell*

Contains subkeys that define actions (open, edit, print, play, and so forth) appropriate to the object. These actions appear on the context menu for the associated file type, among other things. The easiest way to edit these actions is with the File Types dialog in Explorer (Tools → Folder Options → File Types), but you sacrifice a pretty interface for increased flexibility.

A common structure that uses the Shell key is `Shell\Open\Command`, where the default value in the Command key is the executable filename for a registered application. The command line often ends with "%1" (including the quotes), which represents a command-line parameter passed to the application (familiar to those who use batch files). For example, when you double-click on a .txt file (say, `c:\stuff\junk.txt`), Windows replaces "%1" in the command line with the name of the file, resulting in the following command being run:

notepad c:\stuff\junk.txt. If you were to select Run from the Start menu, and type that command, Notepad would appear and open the file. Note that the quotes around the %1 accommodate any spaces in the filename; otherwise, Windows would interpret a single filename with a space as two distinct filenames, and you'd get an error.

You might also find the key, `Shell\Open\ddeexec`, which contains information necessary for a DDE (Dynamic Data Exchange) conversation. DDE is the mechanism with which Windows communicates with applications that are already open. For example, Windows might send a DDE message to an application to tell it to print a file after opening it. Microsoft insists that DDE is obsolete, but you'd never know it from the important role DDE plays in Windows file types.

You'll see the same split between command-line options and DDE using the Explorer interface to file associations, via Tools → Folder Options → File Types. Some actions will list a command line; others will use DDE. If you're not a programmer with access to the DDE documentation for a particular application, you may find this difficult to follow at times.

You may find that a particular Shell branch doesn't contain all the actions for a particular file type. This is because these items may be specified in three other places. First, the ShellEx key contains more advanced actions. Second, the `HKEY_CLASSES_ROOT\*` key contains additional actions that apply to all file types. Third, some actions available to all files, such as the Delete and Properties actions, can't be removed, and therefore don't appear in this key.

### *ShellEx*

This is short for Shell Extensions. These keys contain entries that supplement a file's context menu (via the `ContextMenuHandlers` subkey), a file's Properties sheet (via the `PropertySheetHandlers` subkey), and a file's drag-drop behavior (via the `CopyHookHandlers` subkey). These extra features are too complex to be simple `Shell\Command` structures; instead, these keys simply point to registered CLSIDs (described earlier) and special programs that perform advanced features. For example, if you install the WinZip utility (<http://www.winzip.com>), all ZIP files will have extra items in their context menus (click with the right mouse button) and their drag-drop menus (drag with the right mouse button) that handle certain ZIP operations. Also, movie files (`.avi`, `.asf`, and `.mpg`) and Word Documents (`.doc`) all have an extra Summary tab in their Properties sheets that displays additional information about the contents of the file.

### *ShellNew*

Defines whether the file type will appear on Explorer's New menu. If the `ShellNew` key is present inside a file type or file extension key, the file type will appear in the list when you select New from Explorer's File menu. In most cases, this key will be empty, if it exists at all. (Contrast the enormous number of file types defined in the Registry with the much smaller number that appear on the New menu.)

### Command

Contains a command line to create the new file, used only for Briefcases and Windows Shortcuts (the *.bfc* and *.lnk* extensions, respectively), as well as any other file type that can't be created merely by copying a template (see next).

### Filename

Contains the name of a file “template” to copy to the new location. Its value data may contain a complete pathname, but if it's just a filename (e.g., *netscape.html*, *winword.doc*, or *winword8.doc*), it will be found in the directory `\Windows\ShellNew`. If the Filename value is not present, Windows will create an empty, zero-byte file.

### Nullfile

If present, instructs Windows not to create the file at all, but instead to launch a program that will create the file when first saved. Some file types (such as *.bmp* files, which may contain data in any one of a number of related formats, as specified by binary header data within the file itself) are described by the NullFile value. NullFile has the empty string (“”) as its value data.

### Data

Contains binary data that needs to be written to the new file. This might, for example, be some kind of binary header data.

Before leaving `HKEY_CLASSES_ROOT`, two other keys are worthy of note.

`HKCR\*` contains information that will be applied to all files, regardless of their extension.

`HKCR\Unknown` describes, via its `Shell\OpenAs\Command` subkey, what will happen to a file whose type is unknown, either because the file has no extension, or because the particular extension has not yet been registered with Windows. By default, if you double-click on a file Windows categorizes as unknown, the Open With dialog appears, allowing you to choose a new association for the filename extension, thereby automatically creating new file type keys in the Registry box. Here you find the *rundll32* command line to bring up that dialog box. *rundll32* is merely a program that allows a particular function in a *.dll* or *.exe* file to be run via a command-line interface.

## HKEY\_CURRENT\_USER (HKCU)

The Registry separates settings specific to individual users from global Windows settings applicable to all users. In the FreeCell example earlier in this chapter, each user of the machine can have his or her own separate won/lost statistics because the program keeps these statistics in the `HKEY_CURRENT_USER` branch of the Registry. If it instead used `HKEY_LOCAL_MACHINE`, all users would share the same statistics.

This entire branch is a mirror (or symbolic link) of `HKEY_USERS\DEFAULT` (see below), and its contents always correspond to those of the currently logged-in user.

By default, there are twelve top-level subkeys in `HKCU`: AppEvents, Console, Control Panel, Environment, Identities, Keyboard Layout, Printers, RemoteAccess, SessionInformation, Software, UNICODE Program Groups, and Volatile Environment.

## HKCU\AppEvents

The associations between events and system sounds are kept here. (See “Sounds” in Chapter 5.) There are two branches: EventLabels and Schemes. EventLabels contains the labels that will be used for the sounds; Schemes contains the pointers to the actual sounds.

Schemes has two main subkeys: Apps and Names.

Applications that use sounds can create their own subkey under Schemes\Apps, or they can add sounds into the default list, which is kept in the subkey Apps\.Default. If they add their own subkey, the sounds will show up in a separate section of the sounds list in Control Panel → Sounds. So you might see a subkey such as Mplayer or Office97, since these applications add some of their own sound events in addition to the default sounds. Note that unless Windows or an application is specifically designed to look for an event listed here, any new events you might add will have no effect.

Schemes\Names is where Windows stores the settings for each of the sound schemes, such as Utopia. When you change the sound scheme using the drop-down Scheme list on Control Panel → Sounds, the appropriate scheme is copied into .Default.

## HKCU\ControlPanel

Data from several of the Control Panel applets is stored here, particularly Accessibility and some of the Display settings. The names don’t match up cleanly to the names used in the Control Panel, but you can usually figure out what’s what by going back and forth between Registry Editor and the target Control Panel applet. For example, HKCU\ControlPanel\Accessibility maps directly to Control Panel → Accessibility, but HKCU\ControlPanel\Cursors maps to Control Panel → Mouse → Pointers.

As is typical in the convoluted world of the Registry, some entries point somewhere else entirely. For example, HKCU\ControlPanel\International simply defines a Locale value, such as 00000409, which is the standard code for what the Control Panel calls “English (United States).” If you use Registry Editor’s Find function to trace this value, you’ll eventually find the scattered locations of many of the individual values that Control Panel → Regional Settings brings together in one place.



This example illustrates a key point: there’s usually little reason to poke around in the Registry for values that have a convenient user interface in the application. The exception is where the interface has limited which values can be entered, making the Registry a tool you can use for greater control (at the expense of the convenience of a user interface), as well as where there is simply no interface for some of the more obscure settings.

## HKCU\Environment

The *environment* is a small chunk of memory devoted to storing a few system-wide settings, primarily for use with older Windows and DOS applications, but still used by Windows XP. In Windows 9x and Windows Me, information was added to the environment via the *AUTOEXEC.BAT* file (now

obsolete). Like Windows 2000, Environment variables in Windows XP are set via Control Panel → System → Advanced → Environment Variables. The upper section of this dialog box (user variables for *username*) is where user-specific variables are entered, and are thus stored in the HKCU\Environment Registry key. The lower box (system variables) is for system-wide, user-independent variables and is stored in HKLM\SYSTEM\CurrentControlSet\Control\SessionManager\Environment. Note also the HKCU\Volatile Environment key, which contains temporary environment variables, resets each time Windows is started.

#### HKCU\keyboard layout

This key is used only if you have installed more than one keyboard layout via Control Panel → Keyboard → Language. A Preload subkey lists a separate subkey for each installed language, with subkey 1 specifying the default language.

#### HKCU\RemoteAccess

This key lists various types of information used by Dial-Up Networking, including the default connection to be used by Internet applications. The Implicit subkey lists the UNC path of any shared folders or printers that are accessed over a particular Dial-Up Networking connection. The Profile subkey stores information specific to each connection, such as the saved login name that will be supplied automatically when you make the connection.

#### HKCU\Software

Probably the most useful key in HKEY\_CURRENT\_USER, this key contains subkeys for each vendor whose software is loaded onto the machine, and, within each vendor's area, subkeys for each product. The keys stored here are supposed to contain only user-specific settings for each software application. Other settings, which are common to all users of software on the machine, are stored in HKLM\SOFTWARE.

The structure of this branch (and particularly of the Microsoft\Windows\CurrentVersion branch under both) is described later in this chapter, in the section “HKCU\Software and HKLM\SOFTWARE.”

## HKEY\_LOCAL\_MACHINE (HKLM)

HKLM contains hardware settings and global software settings that apply to all users. It has five top-level subkeys: HARDWARE, SAM, SECURITY, SOFTWARE, and SYSTEM. Each of these keys is stored in a separate hive file (see “Hives,” later in this chapter).

#### HKLM\HARDWARE

The data stored in this branch is used by Windows XP to load drivers and initialize resources for the various hardware components of your computer. All of the settings here are more easily accessible through Device Manager (Control Panel → System → Hardware → Device Manager); there's little need to edit them directly. However, you may find it interesting to snoop around in this branch and see the various pieces of information that are stored for your CD Writer, your scanner, your hard disk, and your processor. The HKLM\HARDWARE\DESCRIPTION\System\CentralProcessor\0 key tells me that my CPU is “GenuineIntel.” I hope the thought police aren't checking this key.

## HKLM\SAM

This key stores data for the Security Accounts Manager (SAM), used only if your Windows XP system is providing domain services. You'll find little reason to ever mess with the settings in this branch, and as most of the data is in binary format, you'll have a hard time deciphering it anyway. The information stored in this key is managed primarily through the User Manager for Domains (on NT Server) and User Manager (on NT Workstation). All of the settings here can be accessed through the Local Security Policy Editor (*secpol.msc*).

## HKLM\SECURITY

The Security branch is where Windows stores the local security policy, which includes user accounts, permissions, passwords, and group membership information. Like HKLM\SAM, there is little reason to mess with this key. All of the settings here can be accessed through Control Panel → User Accounts.

## HKLM\SOFTWARE

Probably the most useful key in HKEY\_LOCAL\_MACHINE, this key contains subkeys for each vendor whose software is loaded onto the machine, and, within each vendor's area, subkeys for each product. The keys stored here are supposed to contain global system settings for each application, common to all users on the machine. Other settings, specific to each user, are stored in HKCU\SOFTWARE.

The structure of this branch (and particularly of the Microsoft\Windows\CurrentVersion branch under both) is described later in this chapter, in the section "HKCU\Software and HKLM\SOFTWARE."

## HKLM\SYSTEM

The settings in this branch primarily handle multiple hardware profiles. Windows uses this data together with HKLM\HARDWARE to handle drivers and Plug-&-Play management for all hardware on the system. All of the settings stored in this branch are accessible via Control Panel → System → Hardware → Hardware Profiles.

## HKCU\Software and HKLM\SOFTWARE

As noted earlier, both HKEY\_CURRENT\_USER\Software and HKEY\_LOCAL\_MACHINE\SOFTWARE are structured similarly. Each of these areas has a branch for each manufacturer who has software installed on the system, and most vendors will have keys that appear in both areas. In each manufacturer key, there will be one or more subkeys, corresponding to each of that manufacturer's applications that are installed. For example, under the Adobe key, you might see an entry for Photoshop and one for Illustrator, assuming both of those applications are installed on your system.

In theory, the HKCU branch should include information that is configurable on a per-user basis (which is the case, for instance, with a software package with a per-user license or per-user customization). The HKLM branch should include software that is standard for all users. In practice, though, it doesn't seem to be as consistent as that. Some data might seem to be placed in wrong branch, while other data might be placed in both branches. Fortunately, this doesn't pose much

of a problem in practice, partly because the vast majority of systems will only have a single user account, but more importantly, because the only practical rule as to the location and organization of data in the Registry is that it is consistent with the application that uses it. For example, since WordPerfect knows where to look for its own settings, it doesn't really matter that they aren't in a place the casual user would expect. Basically, if you're looking for something, look in both branches (HKLM and HKCU).

Because this is a book about Windows XP and not about the third-party applications that might be installed in it, the primary focus of this discussion is on the `Microsoft\Windows\CurrentVersion` branch, located in both HKCU and HKLM. There is a ton of information in these two areas, and the following keys represent the more useful and intelligible data:

`..\Microsoft\Windows\CurrentVersion`

This key contains a dozen-or-so values describing some basic Windows settings, such as the folder location for Program Files, and the 20-digit Product ID number. Note the use of `REG_EXPAND_SZ` values, described earlier in this chapter.

`..\Microsoft\Windows\CurrentVersion\App Paths (HKLM only)`

This branch lists a path for many application executables (Microsoft and otherwise) that are installed in nonstandard locations (i.e., not in `\`, `\Windows`, or `\Windows\Command`). It is the reason why you can successfully type a command name like `excel` or `winword` at the Run prompt, but not at the Command prompt, unless you add `\Program Files\Microsoft Office\Office` to your search path. They have listed their path individually under this key.



If you have an application that installs a shortcut on the Start menu, but doesn't let you type its name at the Run prompt, add a key for it in the App Paths key (using an existing entry as a template). For example, I added a `PHOTOSHOP.EXE` key, with the values:

```
(Default) "C:\Program Files\Adobe Photoshop\Photoshop.exe"  
Path "C:\Program Files\Adobe Photoshop"
```

The end result is something like the Path environment variable, except that the target is a specific executable rather than an entire folder.

`..\Microsoft\Windows\CurrentVersion\Explorer\ShellFolders`

`..\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders`

Specifies the locations of many of the standard Windows system folders, including Desktop, Programs, Send To, Start Menu, Startup, and Templates.

This branch really brings home the extent of Windows' mutability. Even the directory names that Explorer relies on, such as `\Windows\Desktop`, are not hard-wired. So Explorer doesn't know anything about `c:\Windows\Desktop`. All it knows is that it can get the name of the folder it's supposed to use as the Desktop from the Registry. Most of these values probably shouldn't be changed.

The ShellFolders and User Shell Folders keys each exist in the HKCU and HKLM branches. If you're looking for a particular setting, make sure you look in all four keys; whenever a key seems to be duplicated in more than one place, it's good practice to make changes in both places. Note also the use of the REG\_EXPAND\_SZ data type (explained earlier in this chapter) is used for some of the values.

- .. \Microsoft\Windows\CurrentVersion\Explorer\Desktop\NameSpace  
Contains keys named with the CLSID of system icons that appear on the Desktop such as the Recycle Bin, My Documents, and My Network Places. Since these are simply pointers to objects defined elsewhere in the Registry (like Windows Shortcuts), they can be safely deleted (one method of removing the respective icons from your Desktop).
- .. \Microsoft\Windows\CurrentVersion\Explorer\DontShowMeThisDialogAgain  
Every time you click the "Don't show me this again" option on some warning dialogs, Windows records your choice in this key. If you'd like to restore a warning you've previously dismissed, you can remove the corresponding value here. Too bad you can't add new values here to get rid of all the annoying Windows messages!
- .. \Microsoft\Windows\CurrentVersion\Explorer\FindExtensions  
Contains keys corresponding to the various entries in the Start menu's Search menu and in the Explorer Search Bar. While you can't indiscriminately add items here (unless you're a programmer), you can remove unwanted entries.
- .. \Microsoft\Windows\CurrentVersion\Explorer\MyComputer  
Contains several keys that relate to the My Computer window. The NameSpace subkey, for example, contains keys pointing to CLSIDs of various optional components that might appear in your My Computer window. To identify any unlabeled CLSID that shows up here or anywhere else, copy the entire CLSID string to the clipboard and paste it into Registry Editor's search box.
- .. \Microsoft\Windows\CurrentVersion\Explorer\StartMenu (*HKLM only*)  
Contains several keys that relate to the seemingly hardcoded entries that appear in the Start Menu → My Computer window. The NameSpace subkey, for example, contains keys pointing to CLSIDs of various optional components that might appear in your My Computer window. To identify any unlabeled CLSID that shows up here or anywhere else, copy the entire CLSID string to the clipboard and paste it into Registry Editor's search box.
- .. \Microsoft\Windows\CurrentVersion\Internet Settings  
Contains a whole slew of settings for Internet Explorer and the Internet Options dialog in Control Panel. You'll find settings in this key for just about everything from Passport settings to the filename of the bitmap used for the background of Internet Explorer's toolbar.
- .. \Microsoft\Windows\CurrentVersion\Run (*HKLM only*)
- .. \Microsoft\Windows\CurrentVersion\RunOnce  
In these keys, you'll find a list of programs that Windows loads at start up. Note that these aren't the same as those found in the Startup folder in your Start menu, but they work similarly. The format is simple enough: the data in

each value contains the full path and filename of a program to be launched, and the name of the value is merely a reminder as to the purpose of the entry. You can safely add any program you like, or remove entries you'd like to stop loading automatically. Note that values placed in either of the RunOnce keys (HKCU or HKLM) will only be run the next time Windows starts, and will be deleted immediately thereafter. There are several advantages of these keys over the Startup folder. For example, they're more concealed, and therefore more tamper-resistant. Also, items placed here (HKLM only) are run before the login screen appears, while items in the Startup folder are run only after a user logs in.

In addition to the previous specific keys, there are a few paradigms that show up again and again, such as the following:

#### ../MRU

Stands for Most Recently Used. Any time Windows shows you a “history” of the last few things you've typed into a field, such as the Start menu's Run dialog, those items are stored in an MRU key in the Registry. A quick search for MRU in the Registry Editor will yield dozens of instances.

Knowing the location and use of a particular MRU list has several advantages. For example, you can write a script to clear a given list to erase your “footprints,” so to speak. Or, perhaps you could create a Registry patch that would preload the Find drop-down list with a set of long names you wanted to search for repeatedly.

In MRU keys, the value names are a series of letters; there is also a value called MRUList that specifies the order in which the entries should be displayed. That is, if you've just typed in the filenames *\*.dll* and *\*.exe* into the Find dialog box, they might appear as items *h* and *i* in the list. But then, if you picked *\*.dll* again from the list, the MRUList would show *ijabcdehgh*. The list is updated only when you actually execute a find, not when you type in a new filename to search for or pick an item from the drop-down list. In addition, when you quit Find, value *j*, which always contains an empty string, rotates to the front of the list.

#### ../Namespace

Keys named Namespace usually contain values or subkeys that point to CLSIDs (16-bit identifiers to registered program components), which, in effect, instructs Windows to load those components. For example, if a CLSID for the Recycle Bin is listed under the Desktop\Namespace key, it corresponds to the Recycle Bin object appearing on the Desktop. Sometimes, removing these entries will have the effect of removing the corresponding objects, but not always. I recommend experimentation with some degree of caution.

## HKEY\_USERS (HKU)

Despite its name, this branch does not contain the Registry entries for all users configured on the system. This is because user information is loaded when a user logs in, and only one user can be logged in at any given time. Rather, it contains only information for the currently logged-in user as well as a template for new user profiles. (For those of you familiar with this key in Windows 9x and Windows Me, it does not quite work in the same way in Windows XP.)

This branch contains several keys, although only one of them is mirrored as HKEY\_CURRENT\_USER (discussed above). Usually, it's the one that looks like 5-x-x-xx-xxxxxxxx-xxxxxxxx-xxxx, where the long string of x's is the Windows serial number. If it's not clear which key is the active one, just create a new key or value in one of the branches called test. Then, move up to HKEY\_CURRENT\_USER, and press F5 to refresh the view. If the test entry you just added shows up here as well, you've found the active key; if not, delete test and try again in a different key.

The .DEFAULT key is used as a template for creating new users, and unless you want to affect new user settings, you should leave this key alone. In fact, there's little reason to ever play with the HKEY\_USERS branch, as all applicable settings for the active user are more easily accessible through HKEY\_CURRENT\_USER.

This design prevents one user from easily viewing or changing another user's settings.

## HKEY\_CURRENT\_CONFIG (HKCC)

As noted earlier, HKCC is mirrored from the HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Hardware Profiles\XXXX, in which XXXX is a key representing the numeric value of the hardware profile currently in use. On a system with only a single hardware profile, its value will most likely be 0001. This branch contains hardware configuration settings for the currently loaded hardware profile, but there's little reason to edit this branch directly. Instead, go to Control Panel → System → Hardware → Hardware Profiles.

## Hives

HKEY\_USERS and HKEY\_LOCAL\_MACHINE can be thought of as the only true root keys, since the Registry's three other root keys are simply symbolic links, or mirrors, of different portions of these two. This means that these two branches are the only ones that actually need to be stored on your hard disk, and this is where *hives* come into play.

For every branch in HKEY\_LOCAL\_MACHINE, a corresponding hive file is stored in your \Windows\System32\config folder. For example, HKEY\_LOCAL\_MACHINE\Software is stored in a file called *software* (no filename extension). Since new branches can be added to HKEY\_LOCAL\_MACHINE, new hives can be generated at any time. Most systems will have the following hives: *sam*, *security*, *software*, and *system*.

Not all Registry data is stored on your hard disk, however. Some keys are dynamic, in that they are held only in memory, and are forgotten when you shut down. An example of a dynamic branch is HKEY\_LOCAL\_MACHINE\HARDWARE, which is built up each time Windows is started (an artifact of plug-and-play). Only non-dynamic branches are stored in hives, so you won't see a hive called *hardware*.

The branches in HKEY\_USERS, one for each configured user, are similarly stored in hives. The hive file for each user is called *ntuser.dat*, and is located in \Documents and Settings\{username}. For example, the hive for the Administrator user is stored in the file \Documents and Settings\Administrator\ntuser.dat.

Knowing which files comprise the Registry is important only for backup and emergency recovery procedures (see “Backing Up the Registry,” next), and for troubleshooting (and so you don’t accidentally delete them). The storage mechanism is quite transparent to the Registry Editor and the applications that use the Registry; there’s no reason to ever edit the hive files directly. If you want to migrate a key or a collection of keys from one computer to another, don’t even think about trying to copy the hive files. Instead, use Registry patches, discussed later in this chapter.

## Backing Up the Registry

Given that the Registry is an essential component of Windows, and a damaged Registry can make Windows totally inaccessible, a good backup of the Registry is one of the most important safeguards you can employ.

Unlike Windows 98 and Windows Me, Windows XP does not come with a distinct mechanism that automatically backs up the Registry, which means you’ll have to implement one of your own to fully safeguard your Windows environment.

The Registry is stored in certain files (see “Hives” earlier in this chapter) on your hard disk, so you can create a backup by simply copying the appropriate files to another location.

When you start Windows, the information in the Registry is loaded into memory. While Windows is running, some changes may not be physically written to the Registry files until you shut down your computer; others, such as those made by the Registry Editor, are usually written immediately. For this reason, if you’ve made any substantial changes to the contents of the Registry, you may want to restart Windows *before* backing up the Registry to ensure that the files on the disk reflect the most recent changes.

The other consequence of using the Registry files is that you may not be able to simply use Explorer to copy them while Windows is running, and you certainly won’t be able to overwrite them. The workaround is to attempt these measures when Windows isn’t running, which means starting with the Emergency Recovery Console or with a set of boot disks, and then using DOS commands or a DOS batch file to copy the files (see Appendix C for details).

Although it’s very useful to make backups of the Registry on your hard disk, it certainly can’t prepare your computer for an actual disaster. If your hard disk crashes or gets infected with a virus or if your computer is stolen or dropped out of a eight-story building, those Registry backups on your hard disk won’t do you much good. The most effective Registry backup is simply a matter of making a copy of all hives on your hard disk and keeping that copy somewhere other than inside your computer.

If you back up your entire system regularly, such as to a tape drive or other backup device, you should ensure that the backup software you use specifically supports safeguarding the Registry. Although your Registry certainly won’t be compact enough to fit on a single floppy, it will fit easily on a removable drive

(recordable CD, Zip disk, etc.). In addition, most modern backup software, such as the Backup utility that comes with Windows XP (see Chapter 4), includes a feature to back up the Registry.

One useful shortcut is a *local backup*. If you plan on modifying a specific value or key, it's wise to back up just that key, because restoring it in the event of a problem is much less of a hassle than attempting to restore the entire Registry. See the subsequent section for details.

## Exporting and Importing Registry Data with Patches

Hives have an arcane format, making direct editing all but futile. Fortunately, the Registry Editor conveniently supports the importing and exporting of any number of keys and values with *Registry patches*. Patches (.reg files) are ordinary ASCII text files that can contain anything from a single key to a dump of the entire Registry.

Registry patches can be created with Registry Editor or a standard text editor, such as Notepad. You can also use Notepad to view and modify patches, and then use Registry Editor to reimport the patch.

Patches have many practical uses, including creating local backups of portions of the Registry as a preventative measure before editing keys (see the previous section). You can create a Registry key on one computer and apply it on another, useful for migrating a single setting or a whole group of settings to any number of Windows systems. Patches can allow easier editing than with Registry Editor, and certainly afford quicker and more flexible searches.

To create a Registry patch, highlight the key you want to export and select File → Export. Once you've chosen a filename, the selected key, any subkeys, and all their values and respective data will be saved in a single file with the .reg extension. In most cases, you wouldn't want to select My Computer to export the entire Registry, since, for no other reason, HKLM is enormous and you wouldn't want to reimport it in any case.



Before making any changes to a Registry key, do a quick backup by exporting the key. Depending on what changes you've made, the Registry might not be identical after reimporting the key, but at least you'll have a record of what the key looked like before the changes.

Importing .reg files isn't quite as simple as creating them, partly because of the concept of merging, and partly because of the potential for harm. It's important to note that the contents of a Registry patch are merged with existing keys; they don't simply overwrite them. So, if a given key contains four keys (apple, pear, banana, and peach), and you apply a Registry patch (pointing to the same key) that contains four keys (apple, pear, banana, and pomegranate), the resulting key will have five keys (apple, pear, banana, peach, and pomegranate). The existing values and keys will indeed be overwritten with those in the patch, but any additional values and keys in the Registry will remain intact.

## Stop! Do Not Double-Click on This File!

The default action for double-clicking on a *.reg* file is not to edit the file, as you might expect, but to merge it into the Registry. Registry Editor does warn you before committing a patch, and then informs you that the patch was successful (both messages can be turned off with the */s* command-line parameter—see Appendix A). However, if you work with Registry patches often, and are concerned about accidentally applying one, you may want to change the default action so that *.reg* files are edited when double-clicked. Open the File Types dialog (Control Panel → Folder Options → File Types), highlight REG|Registration Entries in the list of file types, and click Advanced. Highlight the Edit action, click Set Default (making it appear bold), and click OK.

The format of *.reg* files is similar to *.ini* files, rather than anything resembling the way data is displayed in Registry Editor. A section begins with the full path of a key in square brackets, like this: [section name]. This is followed by any number of values, each of the format name="data" (the default value appears as @="value"). Then, the next key (if any) is listed, followed by all the values *it* contains. The order of the keys, as well as the order of the values in each key, is irrelevant. However, if you were to move a key from one section to another, it would, in effect, be moving the value from one key to another.

The quotes around value data are only used for string values; binary values and REG\_EXPAND\_SZ values are prefixed with the keyword hex: and appear without quotes. Similarly, DWORD values are preceded by dword: and appear without quotes. Any backslashes in value data (found most often in folder paths and Registry paths) are doubled to distinguish them from the backslashes in key names.

Lastly, the first line in every Registry patch created in Windows XP and Windows 2000 will be Windows Registry Editor Version 5.00, followed by a blank line. Patches created in Windows 95, 98, and Me will instead begin with Registry Editor4, also followed by a blank line. There doesn't appear to be any difference in the treatment of these two types of patches by Registry Editor.

Here is an excerpted Registry patch:

```
Windows Registry Editor Version 5.00
```

```
[HKEY_CURRENT_USER\Software\mozilla.org\Mozilla\0.9.2 (en)\Main]
"Program Folder Path"="C:\\Documents and Settings\\All Users\\Start Menu\\
Programs\\Mozilla\\"
"Install Directory"="C:\\Program Files\\Mozilla\\"
```

```
[HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main]
"NoUpdateCheck"=dword:00000001
"Show_ChannelBand"="No"
"Display Inline Images"="yes"
"Show_ToolBar"="yes"
"Check_Associations"="no"
"SmoothScroll"=dword:00000001
```

```
"Play_Animations"="yes"  
"Play_Background_Sounds"="yes"  
"Display_Inline_Videos"="yes"  
"Print_Background"="no"
```

You may notice that these two keys are in different manufacturer branches, and wouldn't appear next to one another in the Registry. If you think about the way Registry patches are created, you'll realize that the one shown above couldn't have been created in a single step. Instead, two different Registry Patches were created, and the contents of one were cut and pasted into the other using a plain text editor. As long as you remove the redundant header, this is perfectly acceptable. In fact, it can be a very convenient way to implement several Registry changes in one step.

Extra credit question: given the structure of file types discussed earlier in this chapter, how would you create a single Registry patch that contains all the necessary information to register a new file association on any computer? Could you use such a patch to restore your preferred file type settings if another application were to ever overwrite them?

## Ten Cool Things You Can Do in Your Registry

Armed with your new understanding of the Windows XP Registry, you're no doubt ready to get in there and start exploring. Hopefully, this chapter has provided the "lay of the land" you need to get and keep your bearings in the otherwise confusing wilderness of the Registry. While we don't have the kind of room in this book it takes to make you an expert, we would like to send you on your way by pointing out some interesting landmarks; i.e., ten cool changes you can make in your own Registry.

### 1. *Expand the scope of IE's AutoComplete feature.*

In Internet Explorer, you can enter an incomplete URL (i.e., *oreilly* instead of *www.oreilly.com*) and IE will attempt to complete the address itself by searching for all instances. However, IE only searches the *.com*, *.edu*, *.net*, and *.org* top-level domains (TLDs) by default, and only tries the *www* prefix. To add new domain suffixes and prefixes to search, go to:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Internet Explorer\Main\UrlTemplate
```

By default, this key has four values in Windows XP: 1, 2, 3, and 4, set to *www.%s.com*, *www.%s.net*, *www.%s.org*, and *www.%s.edu*, respectively. The value names (the numbers) specify the search order (lower numbers take precedence), and the data specifies the format. Feel free to rearrange the existing items, remove unwanted items, or add new TLDs, like *.gov* (for US government websites), *.co.nz* (for commercial web sites in New Zealand), or *.store* (one of the newly proposed TLDs, still on the drawing table at the time of this writing).

### 2. *Roll back any single setting to the Windows default.*

An entire branch in the Registry is used as a template with which to create new user profiles. As described earlier in this chapter, the path:

```
HKEY_USERS\DEFAULT
```

is duplicated for each new user that is created in Windows XP. If, down the road, you trace a certain problem to an incorrect Registry setting, you can just visit this branch and obtain the default value. For example, I recently ran into a problem caused by incorrect data in the `UserPreferenceMask` value, located in `HKEY_CURRENT_USER\Control Panel\Desktop`. I looked up the corresponding value in `HKEY_USERS\DEFAULT\Control Panel\Desktop`, and copied its data into the active `UserPreferenceMask` value. Problem solved!

Another use of this key, especially for those who need to configure a large number of users, is that any change made to the `.DEFAULT` branch will appear in each new user that is added to the system (existing users won't be affected). This can be a great way, for example, to disable the system sounds for each student account on a classroom computer.

### 3. *Disable the Shut Down command.*

If you're running a kiosk or demo system (or if you just don't want people shutting down your machine), you can disable the Shut Down command by going to:

```
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer
```

Create a new `DWORD` value in this key and name it `NoClose`. Double-click the new value, and set its data to 1. You'll have to log out and log back in (or restart Windows) for this change to take effect. Note that in order to shut down now, you'll have to press `Ctrl-Alt-Del`, and click Shut Down. To undo this change, just delete the `NoClose` value.

### 4. *Registry Editor remembers where you were.*

Each time you open the Registry Editor, it automatically expands the branch you had open the last time Registry Editor was used, but no others. So, if you find yourself repeatedly adjusting a particular setting and then closing Registry Editor (such as when implementing the previous tip), make sure the relevant key is highlighted just before Registry Editor is closed, and that key will be opened next time as well.

Note also the Favorites menu, which works very much like the one in Internet Explorer, allows you to bookmark frequently accessed Registry keys. While it's useful, I personally find the existence of such a feature in a troubleshooting tool like Registry Editor to be more than a little eerie.

### 5. *Change the registered user and company names for Windows XP.*

When Windows XP is installed, a user and company name are entered. Unfortunately, there is no convenient way to change this information after installation. Surprise—you can do it in the Registry! Just go to:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion
```

The values you need are `RegisteredOwner` and `RegisteredOrganization`, both of which can be changed to whatever you'd like. You may notice that the Registry key containing these values is in the Windows NT branch, rather than the more commonly used Windows branch. Don't worry, both branches are used in Windows XP. The less-used Windows NT branch contains more advanced settings, mostly those that differentiate the Windows 9x and Windows NT lines of operating systems (as described in Chapter 1).

## 6. Change your default installation path.

When you install Windows XP, the path to your installation files is set in the Registry. Unfortunately, this setting is not updated when drive letters change or when you point to a different location when optional components are added or removed. To change the default setup path, making subsequent configuration changes more convenient, go to:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Setup
```

Start by changing the `SourcePath` value to either the root directory of your CD drive (e.g. `d:\`), or to a path on your hard disk or network containing the Windows XP installation files. Note also the `Installation Sources` entry, which is a `REG_MULTI_SZ` value (see “What’s in the Registry,” earlier in this chapter, for details on this value type). It contains a list of all the folders displayed in Windows’ drop-down list, allowing you to quickly point to any one of several favorite installation paths.

## 7. Try something new with My Computer.

Double-click the My Computer icon, and the My Computer window appears. It doesn’t have to be this way. The program launched when you double-click My Computer is simply another value in the Registry. Start by navigating to:

```
HKEY_CLASSES_ROOT\CLSID\{20D04FE0-3AEA-1069-A2D8-08002B30309D}
```

(See the following tip for an easy way to locate this key)

You’ll notice that the structure of this key is very similar to standard file type keys (discussed earlier in this chapter), which means we can treat this object like a file type and create new actions for it. Open the `Shell` subkey, and create a new key named `open`; in the new `Open` key, create a new key named `command`. You should then be here:

```
HKEY_CLASSES_ROOT\CLSID\{20D04FE0-3AEA-1069-A2D8-08002B30309D}\shell\
open\command
```

Double-click on the default value of that new `Command` key and type the full path and filename of the program you wish to open. For example, I find it handy to have the My Computer icon open an Explorer window; to do this, just type `explorer.exe` for the value data. You’ll have to log out and log back in for the change to take effect.

## 8. Some handy Registry navigation shortcuts.

The previous tip involved navigating to the Registry key associated with the My Computer icon on the Desktop, which is located in the `HKCR\CLSID` branch. If you visit this branch, you’ll notice hundreds of Class ID keys, all sorted alphabetically (so to speak), which makes finding a single key rather laborious. Luckily, there are a few alternatives that will greatly simplify this task.

First, you can simply search the Registry for “My Computer.” Start by highlighting the key at the top of the tree (coincidentally named “My Computer”), which instructs Registry Editor to begin searching at the beginning. Then, use `Edit → Find`, type `My Computer`, make sure that all the “Look at” options are checked, and click `Find Next`. The first instance it finds will probably be the key you’re looking for, although it won’t always be this easy.

Another shortcut is to use the keyboard. Like Explorer, when you press a letter or number key, Registry Editor will jump to the first entry that starts with that character. Furthermore, if you press several keys in succession, they will all be used to spell the target item. For example, to navigate to:

```
HKEY_CLASSES_ROOT\CLSID\{20D04FE0-3AEA-1069-A2D8-08002B30309D}
```

start by expanding the HKEY\_CLASSES\_ROOT key. Then, press C + L + S quickly in succession, and Registry Editor will jump to the CLSID key. Next, expand that key by pressing the (+) button, or by pressing the right arrow key, and press { + 2 + 0 (the first three characters of the key name, including the curly brace), and you'll be in the neighborhood of the target key in seconds.

9. *Permanently remove many unwanted system tray icons.*

In the “HKCU\Software and HKLM\SOFTWARE” section, earlier in this chapter, the HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run key is described as listing many programs that are run automatically when Windows starts. Some of these entries are included in order to install icons in the system tray (the area on your Taskbar, by the clock). Since most of the tray icons that come with Windows can be toggled on and off in Control Panel, the more bothersome ones are usually installed by third-party programs. To disable one or more of these tray icons, preventing them from loading the next time Windows starts, you'll have to delete the corresponding value in this key (renaming isn't sufficient). Use caution, and certainly make a Registry patch to back up the entire key before fiddling with it.

10. *Alphabetize your Start menu in one step.*

The Windows XP Start menu allows you to rearrange shortcuts by dragging and dropping them; the unfortunate consequence of this feature is that new shortcuts and folders that appear when applications are installed are added to the end of the list. Now, you can sort a single Start Menu folder alphabetically by right-clicking any shortcut and selecting Sort by Name, but this can get tedious very quickly. The solution to this is in the Registry; just navigate to:

```
HKEY_CURRENT_USER\Microsoft\Windows\CurrentVersion\Explorer\MenuOrder\  
Start Menu
```

and you'll see subkeys and values that determine the sort order of the contents of the Start menu (and the Favorites menu, next door). Simply delete the entire Start menu key to sort all of the folders in your Start menu alphabetically, or selectively delete the desired subkeys to sort corresponding folders. Note that the next time you drag-drop a shortcut in the Start menu, Windows will recreate these keys automatically, so you may wish to write a WSH script (see Chapter 9) to automatically delete this key, say, every time Windows is started.