

# Inside the Registry

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

The Registry is the centralized configuration database for Windows NT and Windows 2000 (Win2K), as well as for applications. The Registry stores information about tuning parameters, device configuration, and user preferences. Many books and articles can help you learn about the logical layout of the Registry. These resources describe where the Registry stores specific configuration settings, which values the settings can store, and what happens when you change the settings.

However, these resources don't describe how NT physically manages the Registry. In other words, how do NT and Win2K organize the Registry on disk, how do these OSs locate and retrieve configuration information when an application requests this information, and what measures do they employ to protect this database that's so crucial to their operation?

This month, I'll show you how the Configuration Manager—the kernel subsystem that implements the Registry—organizes the Registry's on-disk files. I'll discuss how the Configuration Manager manages the Registry as applications and other OS components read and change Registry keys and values. Finally, I'll discuss the mechanisms by which the Configuration Manager tries to ensure that the Registry is always in a recoverable state, even if the system crashes while you're modifying the Registry. To get the most from this column, you need to be familiar with the Registry's logical organization, including the concepts of root key, subkey, and values. If you don't have such knowledge, I recommend that you first read "Inside the Windows NT Registry," April 1997.

## Hives

On disk, the Registry isn't simply one large file but a set of discrete files called hives. Each hive contains a Registry tree, which has a key that serves as the root (i.e., starting point) of the tree. Subkeys and their values reside beneath the root. You might think that NT stores each root key you see when you run one of the Registry editors (regedit or regedt32) in a separate hive, but such is not the case. The correlation between the keys that regedit displays and the content of the hives isn't straightforward, as Table 1, page 68, shows. In fact, none of the root keys correlate to hives. Table 2, page 68, lists Registry hives and their on-disk filenames. Note that the names don't have extensions. The absence of corresponding files signifies that logical root keys are objects with no on-disk representation. The Configuration Manager creates the root keys, linking hives together to build the Registry structure you are familiar with and that regedit displays.

**TABLE 1: Regedit Root Keys**

| Key                 | Description  |
|---------------------|--|
| HKEY_CLASSES_ROOT   | Symbolic link to HKEY_LOCAL_MACHINE\SOFTWARE\Classes.  |
| HKEY_CURRENT_USER   | Symbolic link to a key under HKEY_USERS representing a user's profile hive.  |
| HKEY_LOCAL_MACHINE  | Placeholder with no corresponding physical hive. This key contains other keys that are hives.  |
| HKEY_USERS          | Placeholder that contains the user-profile hives of logged-on accounts.  |
| HKEY_CURRENT_CONFIG | Symbolic link to the key of the current hardware profile under HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\IDConfigDB\Hardware Profiles. |
| HKEY_DYN_DATA       | Placeholder for performance data lookups. This key has no corresponding physical hive.   |

# Inside the Registry

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

| Hive Registry Path              | Hive File Path                               |
|---------------------------------|--|
| HKEY_LOCAL_MACHINE\SYSTEM       | \winnt\system32\config\system                |
| HKEY_LOCAL_MACHINE\SAM          | \winnt\system32\config\sam                   |
| HKEY_LOCAL_MACHINE\SECURITY     | \winnt\system32\config\security              |
| HKEY_LOCAL_MACHINE\SOFTWARE     | \winnt\system32\config\software              |
| HKEY_LOCAL_MACHINE\HARDWARE     | Volatile hive                                |
| HKEY_LOCAL_MACHINE\SYSTEM\Clone | Volatile hive                                |
| HKEY_USERS\UserProfile          | Profile; usually under \winnt\profiles\usere |
| HKEY_USERS\DEFAULT              | \winnt\system32\config\default               |

When you look at Tables 1 and 2, you'll see that some hives are volatile and don't have associated files. The system creates and manages these hives entirely in memory; the hives are therefore temporary in nature. The system creates volatile hives every time the system boots. An example of a volatile hive is the HKEY\_LOCAL\_MACHINE\HARDWARE hive, which stores information regarding physical devices and the devices' assigned resources. Resource assignment and hardware detection occur every time the system boots, so not storing this data on disk is logical.

The heart of the Registry is the HKEY\_LOCAL\_MACHINE\SYSTEM hive. In particular, this hive's subkey \CurrentControlSet\Control contains settings that the Configuration Manager uses to initialize the Registry. When the Configuration Manager is initializing hives and needs to locate the hives' files, the Configuration Manager refers to the value:

**HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\hivelist**

A special type of key known as a symbolic link makes it possible for the Configuration Manager to link hives to organize the Registry. A symbolic link is a key that redirects the Configuration Manager to another key. Thus, the key HKEY\_LOCAL\_MACHINE\SAM is a symbolic link to the key at the root of the SAM hive.

## Hive Structure

The Configuration Manager logically divides a hive into allocation units called blocks in much the same way that a file system divides a disk into clusters. By definition, the Registry block size is 4096 bytes (4KB). When new data expands a hive, the hive always expands in block-granular increments. The first block of a hive is the base block. The base block includes global information about the hive, including a signature—regf—that identifies the file as a hive, updated sequence numbers, a timestamp that shows the last time a write operation initiated on the hive, the hive format version number, a checksum, and the hive file's full name (e.g., SystemRoot\CONFIG\SAM). I'll clarify the significance of the updated sequence numbers and timestamp when I describe how data writes to a hive file. The hive format version number specifies the data format within the hive. Hive formats changed from NT 3.51 to NT 4.0, so if you try to load an NT 4.0 hive in earlier NT versions, you'll fail.

# Inside the Registry

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

NT organizes the Registry data that a hive stores in containers called cells. A cell can hold a key, a value, a security descriptor, a list of subkeys, or a list of key values. A field at the beginning of a cell's data describes the data's type. Table 3 describes each cell data type in more detail. A cell's header is a field that specifies the cell's size. When a cell joins a hive and the hive must expand to contain the cell, the system creates an allocation unit called a bin. A bin is the size of the new cell rounded up to the next block boundary. The system considers any space between the end of the cell and the end of the bin free space that it can allocate to other cells. Bins also have headers that contain a signature, hbin, and a field that records the offset into the hive file of the bin and the bin's size.

**TABLE 3: Cell Data Types**

| Cell                     | Data Type  |
|--------------------------|--|
| Key cell                 | A cell that contains a Registry key, also called a key node. A key cell contains a signature (kn for a key, kl for a symbolic link), the timestamp of the most recent update to the key, the cell index of the key's parent key cell, the cell index of the sub-key-list cell that identifies the key's subkeys, a cell index for the key's security descriptor cell, a cell index for a string key that specifies the class name of the key, and the name of the key (e.g., CurrentControlSet). |
| Value cell               | A cell that contains information about a key's value. This cell includes a signature (kv), the value's type (e.g., REG_DWORD, REG_BINARY), and the value's name (e.g., Boot-Execute). A value cell also contains the cell index of the cell that contains the value's data.  |
| Subkey-list cell         | A cell composed of a list of cell indexes for key cells that are all subkeys of a common parent key.   |
| Value-list cell          | A cell composed of a list of cell indexes for value cells that are all values of a common parent key.  |
| Security-descriptor cell | A cell that contains a security descriptor. Security-descriptor cells include a signature (ks) at the head of the cell and a reference count that records the number of key nodes that share the security descriptor. Multiple key cells can share security-descriptor cells.  |

By using bins, instead of cells, to track active parts of the Registry, NT minimizes some management chores. For example, the system usually allocates and deallocates bins less frequently than it does cells, which lets the Configuration Manager manage memory more efficiently. When the Configuration Manager reads a Registry hive into memory, it can choose to read only bins that contain cells (i.e., active bins) and to ignore empty bins. When the system adds and deletes cells in a hive, the hive can contain empty bins interspersed with active bins. This situation is similar to disk fragmentation, which occurs when the system creates and deletes files on the disk. When a bin becomes empty, the Configuration Manager joins to the empty bin any adjacent empty bins to form as large a contiguous empty bin as possible. The Configuration Manager also joins adjacent deleted cells to form larger free cells.

The links that create the structure of a hive are cell indexes. A cell index is the offset into the hive file of a cell. Thus, a cell index is like a pointer from one cell to another cell that the Configuration Manager interprets relative to the start of a hive. For example, a cell that describes a key contains a field specifying the cell index of its parent key; a cell index for a subkey specifies the cell that describes the subkeys that are subordinate to the specified subkey. A subkey-list cell contains a list of

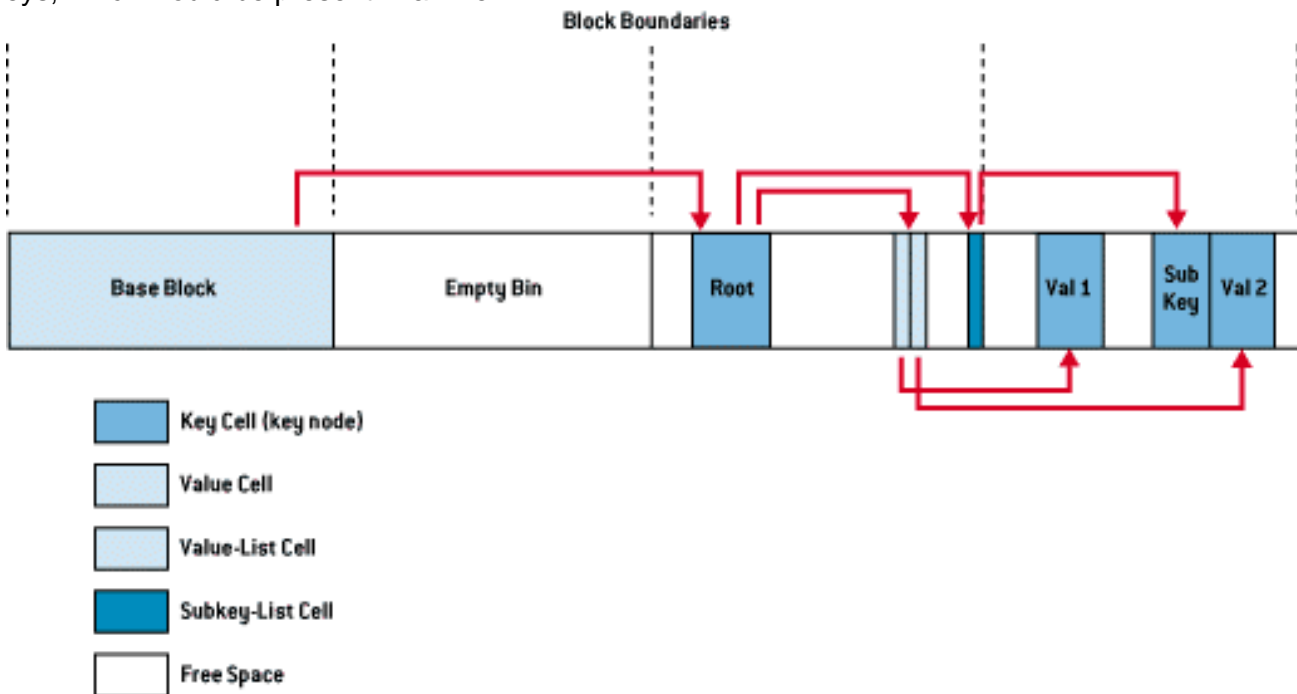
# Inside the Registry

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

cell indexes that refer to the subkey's key cells. Therefore, if you want to locate the key cell of a subkey that belongs to a particular key, you must first locate the cell containing the key's subkey list using the subkey-list cell index in the key's cell. Then, you locate each subkey cell using the list of cell indexes in the subkey-list cell. For each subkey cell, you check to see whether the subkey's name, which a key cell stores, matches the one you want to locate.

The distinction between cells, bins, and blocks can be confusing, so let me give you an example of a simple Registry hive layout. The sample Registry hive file in Figure 1 contains a base block and two bins. The first bin is empty, and the second bin contains several cells. Logically, the hive has only two keys: the root key Root, and a subkey of Root, Sub Key. Root has two values, Val 1 and Val 2. A subkey-list cell locates the root key's subkey, and a value-list cell locates the root key's values. The free spaces in the second bin are empty cells. The figure doesn't show the security cells for the two keys, which would be present in a hive.

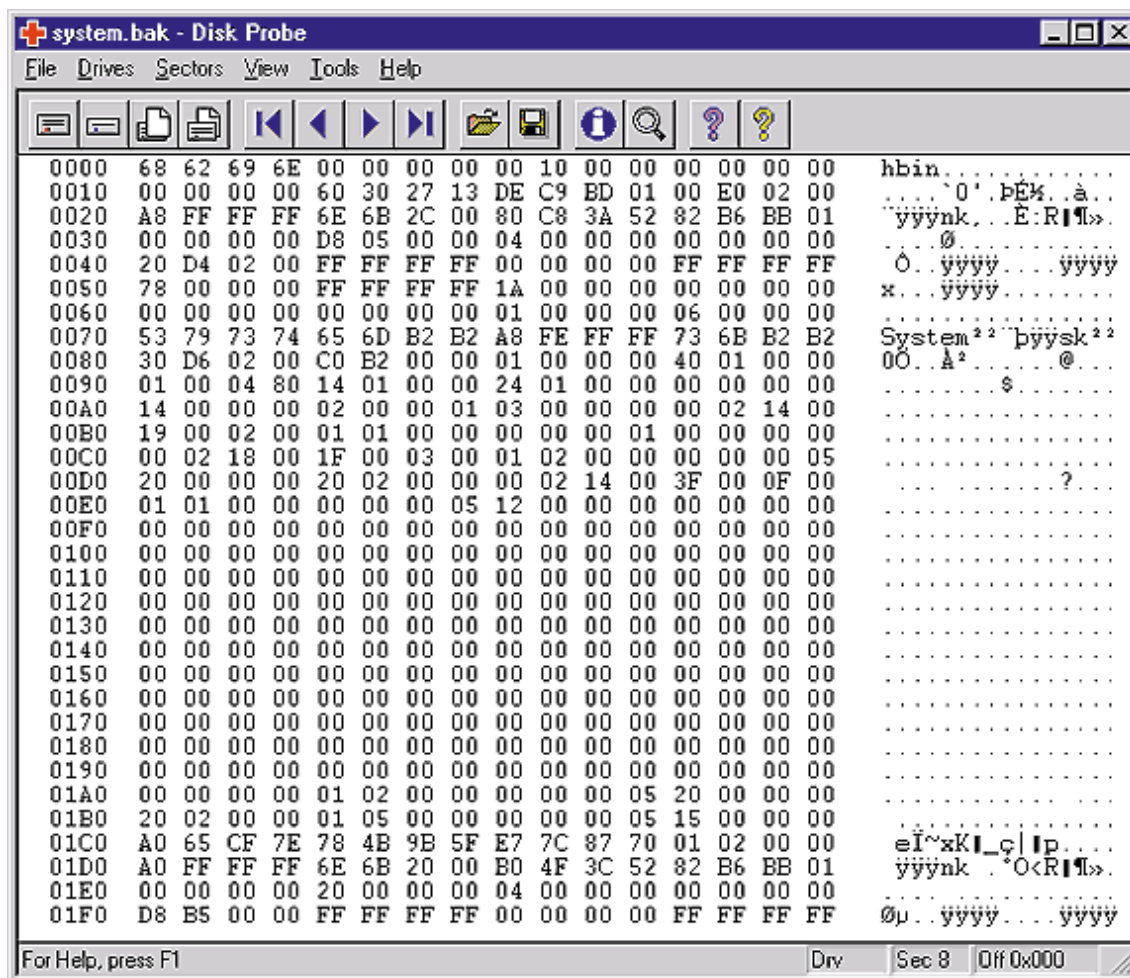


In Screen 1, you can see an image of the Microsoft Windows NT Server 4.0 Resource Kit Diskprobe utility examining the first bin in a SYSTEM hive. Note the bin's signature, hbin, at the top right side of the image. Look beneath the bin signature and you'll see the signature nk. This signature is the signature of a key cell. The signature displays backward because of the way x86 computers store data. The cell is the SYSTEM hive's root cell, which the name that follows the nk signature—System—denotes.

# Inside the Registry

Mark Russinovich

(Reprinted from WindowsItPro Magazine)



To optimize searches for both values and subkeys, the Configuration Manager sorts subkey-list cells and value-list cells alphabetically. Then, the Configuration Manager can perform a binary search when it looks for a subkey within a list of subkeys. The Configuration Manager examines the subkey in the middle of the list, and if the name of the subkey the Configuration Manager is looking for is alphabetically before the name of the middle subkey, the Configuration Manager knows that the subkey is in the first half of the subkey list; otherwise, the subkey is in the second half of the subkey list. This splitting process continues until the Configuration Manager locates the subkey or finds no match.

## Cell Maps

The Configuration Manager doesn't access a hive's image on disk every time a Registry access occurs. Instead, NT keeps a version of every hive in the kernel's address space. When a hive initializes, the Configuration Manager determines the size of the hive file, allocates enough memory from the kernel's paged pool to store the hive file, and reads the hive file into memory. The paged pool is a portion of the kernel's address map that NT reserves for device drivers and the kernel to use. NT can move any memory the system allocates from the paged pool to a paging file when the memory isn't in use.

If hives never grew, the Configuration Manager could perform all its Registry management on the in-memory version of a hive as if the hive were a file. Given a cell index, the Configuration Manager

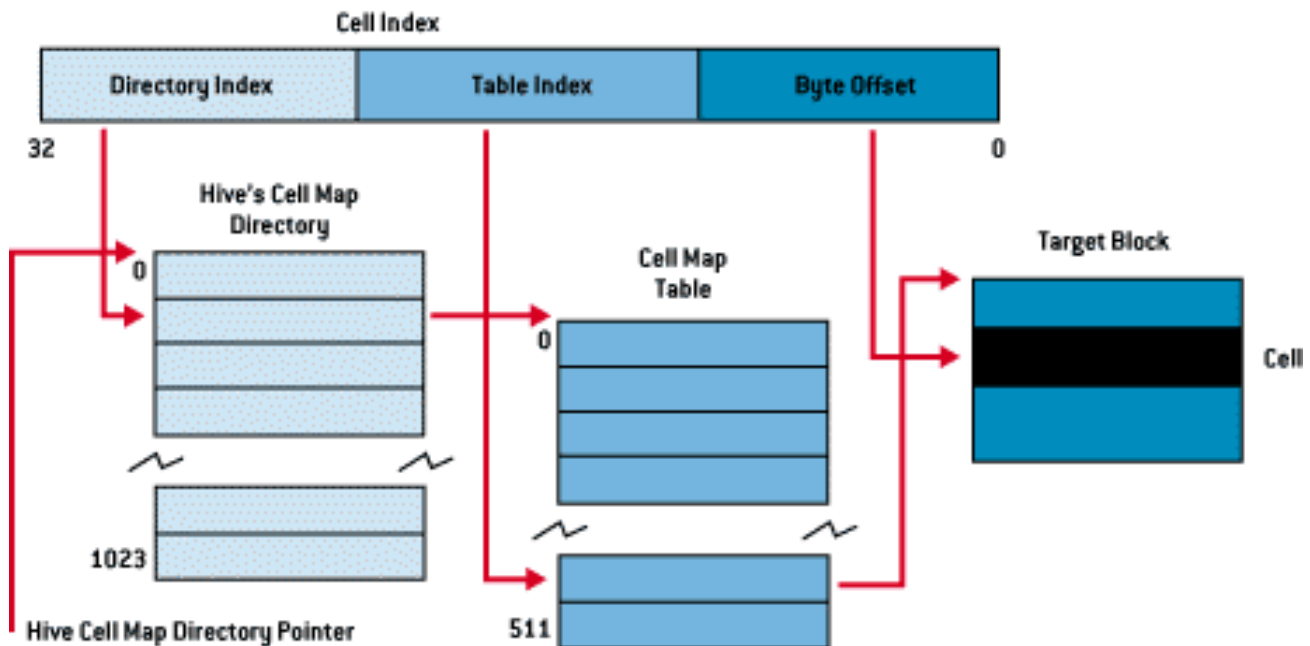
# Inside the Registry

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

could calculate the location in memory of a cell simply by adding the cell index, which is a hive file offset, to the base of the in-memory hive image. Early in the system boot, this process is exactly what Ntldr does with the SYSTEM hive: Ntldr reads the entire SYSTEM hive into memory as a read-only hive and adds the cell indexes to the base of the in-memory hive image to locate cells. Unfortunately, hives grow as they take on new keys and values, which means the system must allocate paged pool memory to store the new bins that contain added keys and values. Thus, the paged pool that keeps the Registry data in memory isn't necessarily contiguous.

To deal with noncontiguous memory buffers storing hive data in memory, the Configuration Manager adopts a strategy similar to what NT's Memory Manager uses to map virtual memory addresses to physical memory addresses. The Configuration Manager employs a two-level scheme, which Figure 2 illustrates, that takes as input a cell index (i.e., a hive file offset) and returns as output both the address in memory of the block the cell index resides in and the address in memory of the bin the cell resides in. Remember that a bin can contain one or more blocks and that hives grow in bins, so NT always represents a bin with a contiguous memory buffer. Therefore, all blocks within a bin occur within the same portion of a paged pool.



To implement the mapping, the Configuration Manager divides a cell index logically into fields, in the same way that the Memory Manager divides a virtual address into fields. NT interprets a cell index's first field as an index into a hive's cell map directory. The cell map directory contains 1024 entries, each of which refers to a cell map table that contains 512 map entries. The second field in the cell index specifies the entry in the cell map table that the first index field identified. That entry locates the bin and block memory addresses of the cell. In the final step of the translation process, the Configuration Manager interprets the last field of the cell as an offset into the identified block to precisely locate a cell in memory. When a hive initializes, the Configuration Manager dynamically creates the mapping tables, designating a map entry for each block in the hive, and adds and deletes tables from the cell directory as the changing size of the hive requires.

## The Registry Namespace and Operation

# Inside the Registry

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

The Configuration Manager defines a key object type to integrate the Registry's namespace with the kernel's general namespace. The Configuration Manager inserts a key object named REGISTRY into the root of the NT namespace, which serves as the entry point to the Registry. Regedit shows key names in the form HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet, but the Win32 subsystem translates such names into their object namespace form (e.g., \Registry\Machine\System\CurrentControlSet). When the NT Object Manager parses this name, it encounters the key object of the name Registry first, and hands the rest of the name to the Configuration Manager. The Configuration Manager takes over the name parsing, looking through its internal hive tree to find the desired key or value.

Before I describe the flow of control for a typical Registry operation, I want to discuss key objects and key control blocks. Whenever an application opens or creates a Registry key, the Object Manager gives a handle to the application with which to reference the key. The handle corresponds to a key object that the Configuration Manager allocates with the help of the Object Manager. By using the Object Manager's object support, the Configuration Manager takes advantage of the security and reference-counting functionality that the Object Manager provides.

For each open Registry key, the Configuration Manager also allocates a key control block. A key control block stores the full pathname of the key, includes the cell index of the key node that the control block refers to, and contains a flag that notes whether the Configuration Manager needs to delete the key cell that the key control block refers to when the last handle for the key closes. NT places all key control blocks into an alphabetized binary tree to enable quick searches for existing key control blocks by name. A key object points to its corresponding key control block, so if two applications open the same Registry key, each will receive a key object, and both key objects will point to a common key control block.

The flow of control when an application opens an existing Registry key starts with the application specifying the name of the key in a Registry API that invokes the Object Manager's name-parsing routine. The Object Manager, upon encountering the Configuration Manager's Registry key object in the namespace, hands the pathname to the Configuration Manager. The Configuration Manager uses the in-memory hive data structures to search through keys and subkeys to find the specified key. If the Configuration Manager finds the key cell, the Configuration Manager searches the key control block tree to determine whether the key is open (by the same or another application). If the key is open, the Configuration Manager increments the existing key control block's reference count. If the key isn't open, the Configuration Manager allocates a new key control block and inserts it into the tree. Then, the Configuration Manager allocates a key object, points the key object at the key control block, and returns control to the Object Manager, which returns a handle to the application.

When an application creates a new Registry key, the Configuration Manager first finds the key cell for the new key's parent. Then, the Configuration Manager searches the list of free cells for the hive in which the new key will reside to determine whether cells exist that are large enough to hold the new key cell. If not, the Configuration Manager allocates a new bin and uses it for the cell, placing any space at the end of the bin on the free cell list. The new key cell fills with pertinent information—including the key's name—and the Configuration Manager adds the key cell to the subkey list of the parent key's subkey-list cell. Finally, the system stores the cell index of the parent cell in the new subkey's key cell.

The Configuration Manager uses a key control block's reference count to determine when to delete the key control block. When all the handles that refer to a key in a key control block close, the reference count becomes zero, which denotes that the key control block is no longer necessary. If an

# Inside the Registry

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

application that calls an API to delete the key sets the delete flag, the Configuration Manager can delete the associated key from the key's hive, because it knows that no application is keeping the key open.

## Stable Storage

To make sure that a nonvolatile Registry hive (one with an on-disk file) is always in a recoverable state, the Configuration Manager makes use of log hives. Each nonvolatile hive has an associated log hive, which has the same base name and a .log extension. For example, if you look in your `\winnt\system32\config` directory, you'll see `system.log`, `sam.log`, and other .log files. When a hive initializes, the Configuration Manager allocates a bit array in which each bit represents a 512-byte portion, or sector, of the hive. This array is called the dirty sector array because an on bit in the array means that the system has modified the corresponding sector in the hive in memory and must write the sector back to the hive file. (An off bit means that the corresponding sector is up-to-date with the in-memory hive's contents.)

When an operation (such as the creation of a new key or value) or the modification of an existing key or value takes place, the Configuration Manager notes the sectors of the hive that change in the hive's dirty sector array. Then the Configuration Manager schedules a lazy write operation, or a hive sync. The hive lazy writer system thread wakes up 5 seconds after the request to synchronize the hive and writes dirty hive sectors for all hives from memory to the hive files on disk. Thus, the system will flush at the same time all the Registry modifications that take place between the time a hive sync is requested and the time the hive sync takes place. When a hive sync takes place, the next hive sync will occur no sooner than 5 seconds later.

If the lazy writer simply wrote all of a hive's dirty sectors to the hive file and the system crashed in midoperation, the hive file would be in an inconsistent (corrupted) and unrecoverable state. To prevent such an occurrence, the lazy writer first dumps the hive's dirty sector array and all the dirty sectors to the hive's log file, increasing the log file's size if necessary. Then, the lazy writer updates a sequence number in the hive's base block and writes the dirty sectors to the hive. When the lazy writer is finished, it updates a second sequence number in the base block. Thus, if the system crashes during the write operations to the hive, at the next reboot the Configuration Manager will notice that the two sequence numbers in the hive's base block don't match. The Configuration Manager can update the hive with the dirty sectors in the hive's log file to roll the hive forward. The hive is then up-to-date and consistent.

To further protect the integrity of the crucial SYSTEM hive, the Configuration Manager maintains a passive mirror of the SYSTEM hive on disk. If you look in your `\winnt\system32\config` directory, you'll see three files with the base name System: `System`, `System.log`, and `System.Alt`. `System.Alt` is the alternate hive. Whenever a hive sync flushes dirty sectors to the SYSTEM hive, the hive sync also updates the `System.Alt` hive. If the Configuration Manager detects that the SYSTEM hive is corrupt when the system boots, the Configuration Manager attempts to load the hive's alternate.

## Registry Optimizations

The Configuration Manager makes a few noteworthy performance optimizations. First, virtually every Registry key has a security descriptor that protects access to the key. Storing a unique security-descriptor copy for every key in a hive would be highly inefficient, however, because the same security settings often apply to entire subtrees of the Registry. When the system applies security to a key, the Configuration Manager first checks the security descriptors associated with the key's parent key, then checks all the parent's subkeys. If any of those security descriptors match the security descriptor the system is applying to the key, the Configuration Manager simply shares the existing

# Inside the Registry

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

descriptors with the key, employing a reference count to track how many keys share the same descriptor.

The Configuration Manager also optimizes the way it stores key and value names in a hive. Although the Registry is fully Unicode-capable and specifies all names using the Unicode convention, if a name contains only ASCII characters, the Configuration Manager stores the name in ASCII form in the hive. When the Configuration Manager reads the name (e.g., when performing name lookups), it converts the name into Unicode form in memory. Storing the name in ASCII form can introduce significant savings in a hive's size.

Finally, over time a Registry hive can fragment. For example, the hive in Figure 1 is fragmented because it contains empty spaces. The Configuration Manager never tries to compact a Registry hive, but NT can achieve Registry compaction. When you use the Win32 API RegSaveKey to back up a Registry hive to another file (Emergency Repair Disk— ERD—creation uses this API), the system strips the copy of all free space, effectively condensing the hive. To switch to the compacted hive, you use the RegReplaceKey or RegReplaceKey Win32 APIs, which the ERD restore function uses.

## The End of the Tour

Win2K beta 3 contains no significant changes to the Configuration Manager or to the way NT manages Registry hives on disk or in memory. In fact, you can use regedt32's Load Hive capability to load Win2K Registry hives from an NT 4.0 system. However, Win2K beta 3 introduces significant memory and performance optimizations. First, key control blocks in Win2K don't store full key Registry pathnames, as happens in NT 4.0. Instead, Win2K key control blocks reference only a key's name. For example, a key control block that refers to \Registry\System\Control would refer to the name Control rather than to the full path. A further memory optimization is that the Configuration Manager uses key name control blocks to store key names, and all key control blocks for keys with the same name share the same key name control block. To optimize performance, the Configuration Manager stores the key control block names in a hash table for quick look-ups.

A second optimization is that the Configuration Manager stores frequently accessed key control blocks in the Cache Table, which Win2K configures as a hash table. When the Configuration Manager needs to look up a key control block, it first checks the Cache Table.

Finally, the Configuration Manager has another cache, the Delayed Close Table, for keeping key control blocks that applications close. The Configuration Manager would usually deallocate closed key control blocks but in Win2K keeps the blocks in the Delayed Close Table, in case an application needs to reopen a key it has recently closed. The Configuration Manager removes the oldest key control blocks from the Delayed Close Table as it adds the most recently closed blocks to the table.