The Windows NT* Registry File Format Version 0.4

Timothy D. Morgan
tim-registry(α)sentinelchicken.org

June 9, 2009

Abstract

The Windows registry serves as a primary storage location for system configurations and other information. Numerous third-party commercial and open source tools have been released to interpret and manipulate registry hives, but a comprehensive description of the registry's data structures seems to be missing from the public domain. This document attempts to shed light on the details of the registry format and will be updated as more information is made available.

1 Introduction

The Windows registry stores a wide variety of information, including core system configurations, user-specific configuration, information on installed applications, and user credentials. Little information has been published by Microsoft related to the specifics of how registry information is organized into data structures on disk. Fortunately, various open source developers have worked to understand and publish these technical details in order to write software compatible with Microsoft's registry format. However, these sources are by and large incomplete and fragmented, making tool implementation difficult and tedious at best. Here we attempt to combine the available public information, along with additional knowledge gleaned from testing, to provide a comprehensive reference on Windows NT-based registry data structures. *This should be considered a living document and will be updated as new information becomes available. Please contact the author with any errata or new information pertaining to data structure specifics.*

*Throughout this paper, note that Windows, Microsoft, Windows 95, Windows 98, Windows ME, Windows NT, Windows 2000, Windows XP, Windows Vista, and Windows Server are registered trademarks of Microsoft Corporation.

2 Previous Work

Registry internal structures have been outlined by Mark Russinovich [15] and David Probert [14], which provide a good overview of how Windows interacts with registry components. Further detailed work has been published by unknown authors in [3] and [2], which lays the groundwork for a detailed understanding of registry data structures. Numerous open source tools provide access to NT registry internals [12, 16, 18, 20] and have expanded on the public's knowledge of technical specifics.

3 Registry Structure Overview

Here, we briefly provide an overview of the internal data structures of the registry. Later sections provided additional details about specific groups of data structures. Finally, a reference on the specific layout of each structure may be found in Appendix A.

The Windows registry is organized in a tree structure and is analogous to a filesystem. For instance, registry values are similar to files in a filesystem as they store name and type information for discrete portions of raw data. Registry keys are closely analogous to filesystem directories, acting as parent nodes for both subkeys and values. Finally, individual registry files (or "hives") are presented to users in Windows under a set of virtual top-level keys in much the same way that multiple filesystems in UNIX[1] are mounted under the same root directory.

The internal structure of Windows registry hives does, however, differ a great deal from typical filesystems. One major difference is that keys reference values differently than subkeys, whereas most filesystems reference both using the same structures. Additionally, due to the type of storage (a binary file), the allocation storage for data structures is done in a way as to minimize fragmentation and linear space utilization.

[1] UNIX is a registered trademark of the Open Group.

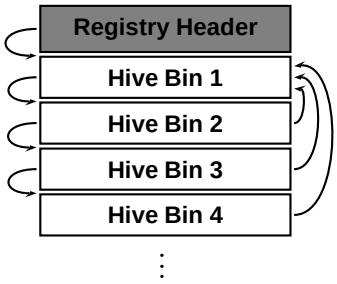

Figure 1: Top-level Registry Structure

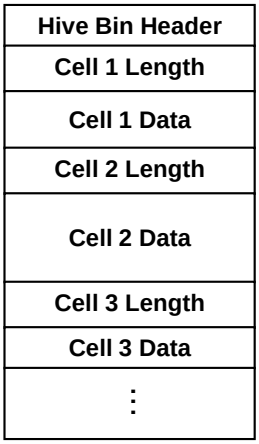

Figure 2: Hive Bin Structure

4 Hive Bins

Registry hive files are allocated in 4096-byte blocks starting with a header, or base block, and continuing with a series of hive bin blocks. Each hive bin (HBIN) is typically 4096 bytes, but may be any larger multiple of that size. HBINs are linked together through length and offset parameters as shown in Figure 1. Each HBIN references the beginning of the next HBIN in addition to indicating its distance from the first HBIN.

Within each HBIN can be found a series of variable length cells. These cells are stored in simple length-prefix notation where each cell's total length (including the 4-byte length header) is a multiple of 8 bytes. Figure 2 illustrates the layout of a typical HBIN. The data portion of each cell contains either value data or one of several different record types. Possible record types include: key (NK) records, subkey-lists, value-lists, value (VK) records, security (SK) records, big data records, and big data indirect offset cells.

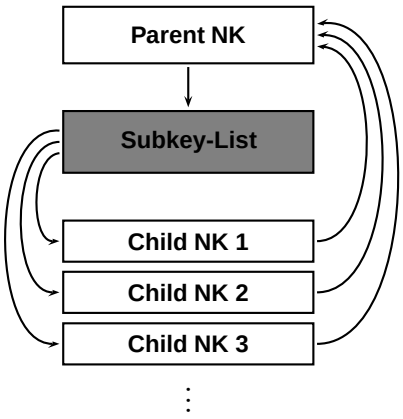

Figure 3: Key-Related Pointers

5 Keys

The data structure which ties all of these elements together is the key record. NK records contain a number of offset[2] fields to other data structures. These referenced structures may exist in any HBIN. In order to keep track of a key's subkeys, NK records reference subkey-lists which in turn reference a set of other NK records. NK records also store the offset of their parent NK record. These key-related pointers are illustrated in Figure 3. NK records also contain pointers to value-lists which in turn reference value (VK) records.

A final significant detail related to NK records is the inclusion of a modification time (MTIME) field. NK records are the only known record type, aside from the hive header, to contain any kind of time stamp. This field appears to be updated any time the NK record itself is updated (with some exceptions, detailed later), which includes changes to values and immediate subkeys.

6 Subkey and Value Lists

Subkey-lists are simple lists of pointers/hash tuples, sorted in order by the hash value, which is based on the referenced subkey names. Multiple types of subkey-lists have been used in different versions of Windows, but they appear to retain the same basic structure. In early versions, including Windows 2000, subkey-lists use records with a magic number of "`lf`", where the hash in each element is calculated simply by taking the first four characters of the associated subkey's name. "`lh`" records appear to be identical except that they use a more intelligent hash algorithm which is detailed in Appendix C.

Sometimes, when a large number of subkeys exist, Windows uses an "`ri`" subkey list type which implements an

[2]Nearly all offset values stored in cell records (whether in the NK records or elsewhere) are measured in bytes from the beginning of the first HBIN, not from the beginning of the file.

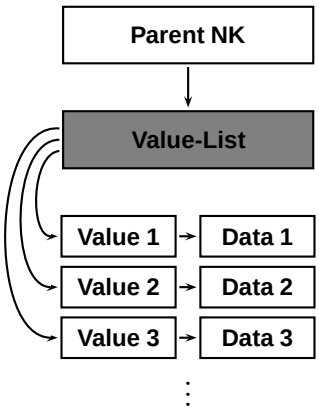

Figure 4: Value-Related Pointers

indirect block system, similar to what is found in some filesystems. These subkey-lists do not include a hash value in each list element and only reference additional subkey-lists in a tree structure. The leaf elements of these trees tend to be "lh" or "li" record types. The "li" record type seems to be identical to "ri" records in structure except that they reference keys rather than additional subkey lists.

Value-lists are similar to subkey-lists, but do not have hash values associated with them and are not sorted in any particular order. Finally, VK records contain minimal metadata about a single value and store the offset to yet another cell which contains the value's data. See Figure 4 for a sample illustration.

7 Security Records

A small number of security records are stored in a given registry hive and are referenced by NK records. SK records include a short header followed by a Windows security descriptor which defines permissions and ownership for local values and/or subkeys. (See [1, 7] for more information on security descriptors.) Multiple NK records may reference a single SK record which in turn stores a reference count to simplify deallocation.

8 Example: Keys and Values

Let us present an example to tie together some of the data structures discussed thus far. Suppose we had a simple registry hive rooted at a key named "parent", which has subkey named "child". Also suppose this subkey has a value stored under it named "item" which is a string, and this value's data is the string "datum". The user perspective of this structure is illustrated in Figure 5. The registry records needed to support this simple path are illustrated in Figure 6. In order to look up \parent\child's

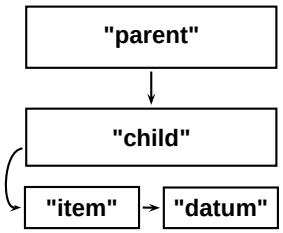

Figure 5: Simple Example: Logical View

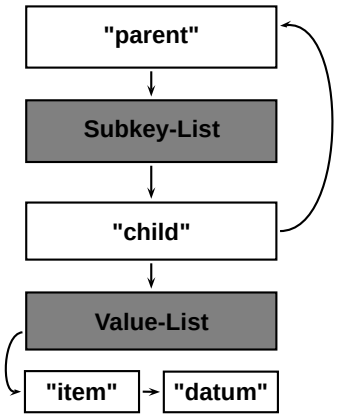

Figure 6: Simple Example: Physical View

item value, one would first need to find parent, and locate its subkey-list. The hash value for "child" would be calculated and used to quickly narrow the list of NK records needing to be checked (i.e., the set of all colliding hashes). Searching this reduced list of NK offsets would then yield an NK record which had the proper name. One would then traverse the child record's value list sequentially, checking each referenced VK record to locate the proper value. If the item value's data was desired, the data pointer would be followed to the data record, unless a specific flag is set indicating that the data is stored in the offset field of the VK record, in which case it would be retrieved from there.

9 Value Data Storage

In most cases, value data is stored very simply in a cell with no real structure, other than that dictated by the data type (discussed later). However, if a value is four or fewer bytes long, Windows may choose to store the data in the offset field of the VK record, rather than allocating a new data cell to store it. If this occurs, the highest bit of the data size field (stored in the VK record) will be set to 1.

In addition, starting with Windows XP, value data records may be fragmented in to multiple cells [8] using "big data" records. According to [17], Windows XP and later will look for a big data record under the following conditions: the registry major/minor version is 1.4 or later and the data size is greater than 16344 bytes in length. At that

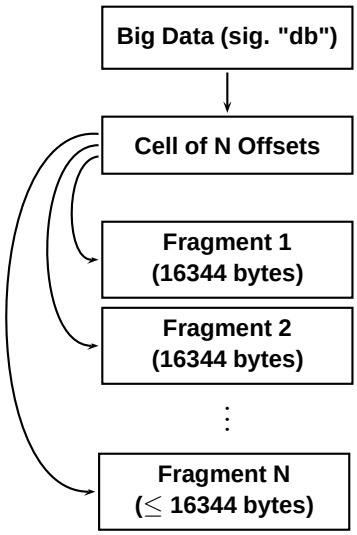

Figure 7: Big Data Linking

point, Windows will attempt to validate the cell referenced by the value record as a big data record. The big data record will indicate the number of data chunk fragments are stored on disk as well as a pointer to an indirect block of offsets for the fragment cells. Figure 7 demonstrates how the data fragments may be located and Appendix A contains more precise information on how to interpret the big data records themselves.

10 Value Data Types

Registry values can be one of several different types. The data type of a value is stored as a 32-bit integer in the VK record, and based on this data type, the data cell should adhere to a specific format. The known registry data types are named as follows: REG_NONE, REG_SZ, REG_EXPAND_SZ, REG_BINARY, REG_DWORD, REG_DWORD_BIG_ENDIAN, REG_LINK, REG_MULTI_SZ, REG_RESOURCE_LIST, REG_FULL_RESOURCE_DESCRIPTOR, REG_RESOURCE_REQUIREMENTS_LIST, and REG_QWORD. However, note that in some instances, Windows and third-party software does not honor this convention and instead uses the data type field in the VK record for other purposes. (One example is in the Windows SAM hive, where this field is used to store user IDs.)

The REG_NONE and REG_BINARY types are used to store arbitrary data without structure or with unspecified structures. The REG_DWORD, REG_DWORD_BIG_ENDIAN, and REG_QWORD are all integer types which store values as 32-bit little endian, 32-bit big endian, and 64-bit little endian, respectively[3]. The REG_SZ, REG_EXPAND_SZ, and REG_LINK types are all stored as UTF-16 little endian strings. REG_SZ is a basic string type, REG_EXPAND_SZ is similar to REG_SZ, except that it may contain references to environment variables with a "`%VARIABLENAME%`" syntax. Finally, the REG_LINK type is used to store symbolic links.

More structured data types include the REG_MULTI_SZ, REG_RESOURCE_LIST, REG_FULL_RESOURCE_DESCRIPTOR, and REG_RESOURCE_REQUIREMENTS_LIST types. REG_MULTI_SZ is a list of strings where each is stored in UTF-16 little endian and is NUL terminated. (Because each character in UTF-16 is at least two bytes wide, one NUL character is represented as "`\x00\x00`".) The end of a REG_MULTI_SZ list is also marked with a NUL character, resulting in a characteristic four byte sequence of zero bytes (two to terminate the final string, and two more to terminate the list). The REG_RESOURCE_LIST, REG_FULL_RESOURCE_DESCRIPTOR, and REG_RESOURCE_REQUIREMENTS_LIST types are used to store hardware information, in a series of nested lists [9] whose formats are currently unknown.

11 Registry Deletion Behaviors

The most basic behavior to understand in analyzing registry deletions is how the registry manages unallocated cells. As new records are added and free space is allocated, existing empty cells may be split if they are much larger than the required space. However, as cells are later deallocated, any adjacent unallocated cells would need to be merged in order to prevent serious fragmentation. Indeed, this is how the registry manages unallocated space. When a given cell is decommissioned, the cells directly before and after are checked. If either (or both) of these cells are already unallocated, the cells are merged together by updating the header length value of the earliest cell. The other cells' lengths are not updated. This process makes recovery somewhat complicated, since structures cannot be found at specific offsets within a cell.

We have found that the majority of structures stored in cells are preserved when they are deallocated; however, certain key pieces of information are explictly destroyed or partially corrupted during deletion, and these behaviors vary from record type to record type. Here we outline the changes that take place for each record.

Since registry keys act as the glue that ties registry elements together, they are of primary importance. When a key is deleted, its NK record is changed in a number of ways. For one, the pointer which references subkey-lists is destroyed (overwritten with `0xFFFFFFFF`) and the stored

[3] It is not known if the endianness of any of the integer types, or even the UTF-16 strings, would be different on a big endian Windows architecture, such as NT on Alpha.

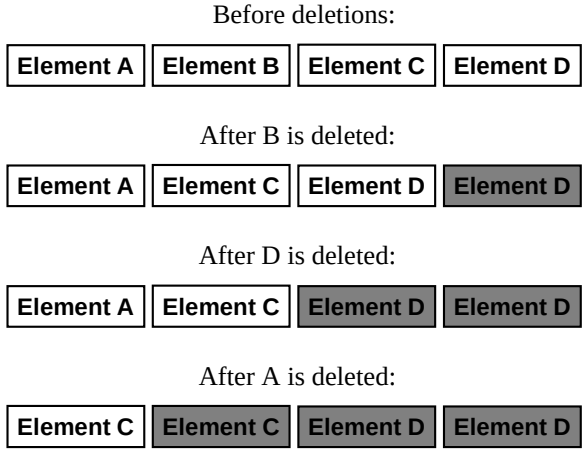

Figure 8: Hypothetical Subkey Deletion Sequence

number of subkeys is set to 0. In addition, the pointer to a key's security record is similarly destroyed. If a key has subkeys, the record's modification time is updated, otherwise it is not. The only known exception to this rule is found on Windows 2000 where a key with subkeys does not have its modification time updated at all when it is deleted.

When analyzing changes to subkey-lists, we must consider two cases: first, where the parent key (and therefore all subkeys) is deleted; and second, where some number of subkeys are deleted. As it turns out, these two cases are actually very similar. When a single subkey is deleted, the element is removed from the list and the resulting list is rewritten to the cell. The remaining free space in the cell is not wiped and in no tests was the cell shortened to conserve wasted space. Consequently, a number of subkey-list elements (each 8 bytes in size) can be found at the end of a subkey-list that has been shortened. Unfortunately, this information is typically not very useful because in most cases the last element in the subkey-list will be repeated over and over, unless it was deleted midway through a set of deletions, at which point the second to last element would begin the repetitions as internal elements continue to be deleted. Figure 8 illustrates how a subkey-list would look at each step if elements B, D, and A were removed, in that order, from an original list of: (A,B,C,D). When it comes to deletion of a parent key, our experiments indicated that all children are merely deleted in sequence with some unknown or arbitrary order. This causes the subkey-list to be repeatedly rewritten with each successive deletion, corrupting the majority of records in most cases. The number of elements in the subkey-list is also reduced to 0 upon deletion of a parent key.

The changes which occur to value-lists during deletion differ somewhat from those of subkey-lists, even though their structures are almost identical. As with subkey-lists, when values are deleted from a key the elements are removed and the list is simply rewritten. Here, slack space is also not wiped and value-list cells do not appear to be shortened as elements are removed, which matches the general behavior of subkey-lists. However, when a value-list's parent NK record is deleted, value-lists are not modified beyond having their holding cell deallocated; all links to the (now deleted) VK records are left intact.

In general, VK records and the data cells they reference are not altered when they are deleted. The only exception to this rule is on Windows 2000 where the first 4 bytes of these cells (for both VK records and data cells) are overwritten with `0xFFFFFFFF`. In the case of the VK record, this corrupts both the two-byte magic number and the two-byte length for the value's name. In the case of a data cell, there's no way of knowing what data would be lost. Fortunately, this behavior was only observed on Windows 2000 and may be indicative of a bug on that platform.

Finally, there are few changes associated with the deletion of security records. Of course since these records may be referenced by multiple keys, they are only deleted when all keys referencing them are also deleted or are set to reference other SK records. Our observations indicate that nothing changes in SK records when this occurs. In fact, not even the reference count (which would store a value of 1 before the final parent key deletion) was updated to 0 when the SK record was deallocated.

Information on proposed methods for recovering deleted registry data may be found in [11, 19].

12 Future Work

While the majority of the registry data structures are largely understood, there are always nagging details that remain unexplained. Here we list a number of them and invite readers to help us complete our knowledge of the registry format.

- A recent, detailed source of information in the master's thesis by Peter Norris[13] should be compared against the results here and any differences sorted out.
- Determine purpose of remaining NK record flags.
- Reveal format of free cell hive structure mentioned in [8].
- Investigate any changes brought by Windows 7.

13 Acknowledgements

We would like to thank the following individuals for their invaluable contributions to this paper: Harlan Carvey, Jason DeMent, Brendan Dolan-Gavitt, George Gal, Jason Morgan, Joan Morgan, Jeffrey Muir, Matthieu Suiche, and Jolanta Thomassen.

14 Revision History

DATE	VERSION	COMMENTS
2009-06-09	0.4	Expanded information on big data records. More references. Added algorithms to appendices.
2009-05-22	0.3.1	Fixed a mistake regarding big data elements. Added a bit of info about volatile keys in the appendix.
2009-05-22	0.3	Thanks to Matthieu Suiche, Jeffrey Muir, and Jolanta Thomassen with help improving subkey list structure, key flags, value flags, and regf header information.
2008-12-01	0.2	Improved descriptions of subkey-lists and added section on values formats.
2008-08-08	0.1	Initial Release.

References

[1] Keith Brown. *What Is A Security Descriptor*. pluralsite.com, 2005. Ported: 2005-01-18. Accessed: 2008-03-09. Available at: http://www.pluralsight.com/wiki/default.aspx/Keith.GuideBook/WhatIsASecurityDescriptor.html.

[2] clark@hushmail.com. *NT Security - Registry Structure*. beginningtoseethelight.org, http://www.beginningtoseethelight.org/ntsecurity/37AB35307A7D52ED, 2005. Available at: http://www.beginningtoseethelight.org/ntsecurity/37AB35307A7D52ED.

[3] B. D. *WinReg.txt*. Available at: http://home.eunet.no/%7epnordahl/ntpasswd/WinReg.txt.

[4] Brendan Dolan-Gavitt. *Forensic analysis of the Windows registry in memory*. DFRWS, 2008. Available at: http://dfrws.org/2008/proceedings/p26-dolan-gavitt.pdf.

[5] Scott Dorman. *Volatile Registry Keys*. 2007. Available at: http://geekswithblogs.net/sdorman/archive/2007/12/24/volatile-registry-keys.aspx.

[6] Stefan Kuhr. *Registry Symbolic Links*. The Code Project, 2005. Available at: http://www.codeproject.com/KB/system/regsymlink.aspx.

[7] Microsoft. *SECURITY_DESCRIPTOR Structure*. Last Updated: 2008-02-19. Available at: http://msdn2.microsoft.com/en-us/library/aa379561.aspx.

[8] Microsoft. *Kernel Enhancements for XP – Registry Enhancements*. Microsoft, 2003. Available at: http://www.microsoft.com/whdc/archive/XP_kernel.mspx#ELC.

[9] Microsoft. *Windows Registry Information for Advanced Users*. Microsoft, february 4, 2008 edition, 2008. Available at: http://msdn.microsoft.com/en-us/library/ms724836%28VS.85%29.aspx.

[10] Microsoft. *Predefined Keys*. Microsoft, http://msdn.microsoft.com/en-us/library/ms724836 Available at: http://msdn.microsoft.com/en-us/library/ms724836

[11] Timothy D. Morgan. *Recovering Deleted Data From the Windows Registry*. Available at: http://www.sentinelchicken.com/research/registry_recovery/.

[12] Petter Nordahl-Hagen. *Offline NT Password & Registry Editor*. Available at: http://home.eunet.no/%7epnordahl/ntpasswd/.

[13] Peter Norris. *The Internal Structure of the Windows Registry*. 2009. Available at: http://amnesia.gtisc.gatech.edu/%7emoyix/suzibandit.ltd.uk/MSc/.

[14] David B. Probert. *Windows Kernel Internals: NT Registry Implementation*. Available at: http://www.i.u-tokyo.ac.jp/edu/training/ss/lecture/new-documents/Lectures/09-Registry/Registry.pdf.

[15] Mark Russinovich. *Inside the Registry*. Windows NT Magazine, Microsoft, may 1999 edition, May 1999. Available at: http://www.microsoft.com/technet/archive/winntas/tips/winntmag/inreg.mspx?mfr=true.

[16] Richard Sharpe. *editreg.c*. 2002. Available at: http://websvn.samba.org/cgi-bin/viewcvs.cgi/trunk/source/utils/editreg.c?rev=2&view=markup.

[17] Matthieu Suiche. *Undocumented Windows Vista and later registry secrets*. June 2009. Available at: http://www.msuiche.net/2009/06/07/windows-vista-and-later-registry-secrets/.

[18] Samba Development Team. *Samba GIT Sources: Registry Library*. www.samba.org, 2008. Available at: http://gitweb.samba.org/?p=samba.git;a=tree;f=source/lib/registry;h=21934b5f658009ff0383f6aed41b102013b5b046;hb=v4-0-stable.

[19] Jolanta Thomassen. *Forensic Analysis of Unallocated Space in Windows Registry Hive Files*. University of Liverpool, 2008. available at: http://www.sentinelchicken.com/research/thomassen_registry_unallocated_space/.

[20] Nigel Williams. *dosreg.c*. c. 2000. Available at: http://www.wednesday.demon.co.uk/dosreg.html.

Appendix A: Registry Data Structures

REGISTRY HEADER/BASE BLOCK

OFFSET	SIZE	TYPE	DESCRIPTION
0x0	4	String (“regf”)	Magic number
0x4	4	Unsigned Integer	Sequence Number 1: matches next field if hive was properly synchronized.
0x8	4	Unsigned Integer	Sequence Number 2: matches previous field if hive was properly synchronized.
0xC	8	Unsigned Integer	64-bit NT time stamp
0x14	4	Unsigned Integer	Major version
0x18	4	Unsigned Integer	Minor version
0x1C	4	Unknown	Unknown (type?)
0x20	4	Unknown	Unknown (format?)
0x24	4	Offset	Pointer to the first key record
0x28	4	Offset	Pointer to start of last hbin in file
0x2C	4	Unknown	Unknown (always 1)
0x30	64	String	Hive file name?
0x70	16	GUID	Unknown
0x80	16	GUID	Unknown
0x90	4	Unsigned Integer	Unknown (flags?)
0x94	16	GUID	Unknown
0xA4	4	Unsigned Integer	Unknown
0xA8	340	Unknown	Unknown (reserved?)
0x1FC	4	Unsigned Integer	Checksum of data to this point in header. See Appendix C.
0x200	3528	Unknown	Unknown (reserved?)
0xFC8	16	GUID	Unknown
0xFD8	16	GUID	Unknown
0xFE8	16	GUID	Unknown
0xFF8	4	Unknown	Unknown
0xFFC	4	Unknown	Unknown

HIVE BINS

OFFSET	SIZE	TYPE	DESCRIPTION
0x0	4	String (“hbin”)	Magic number
0x4	4	Unsigned Integer	This bin’s distance from the first hive bin
0x8	4	Unsigned Integer	This hive bin’s size (multiple of 4096)
0xC	16	Unknown	Unknown
0x1C	4	Unsigned Integer	Relative offset of next hive bin (should be the same value as at offset 0x8)
0x20..[bin size]	variable	Structure List	List of cells used to store various records (see below)

CELLS

OFFSET	SIZE	TYPE	DESCRIPTION	DELETION NOTES
0x0	4	Signed Integer	Cell length (including these 4 bytes)	Negative if allocated, positive if free. If a cell becomes unallocated and is adjacent to another unallocated cell, they are merged by having the earlier cell’s length extended.
0x4	variable	varies	Contains one of: NK record, VK record, SK record, subkey-list, value-list, or raw data blocks (see below)	

SECURITY (SK) RECORDS

OFFSET	SIZE	TYPE	DESCRIPTION	DELETION NOTES
0x0	2	String (“sk”)	Magic number	
0x2	2	Unknown	Unknown	
0x4	4	Offset	Pointer to previous SK record	
0x8	4	Offset	Pointer to next SK record	
0xC	4	Unsigned Integer	Reference count	Not set to 0 when deleted, typically left at 1
0x10	4	Unsigned Integer	Size of security descriptor	
0x14	varies	Windows security descriptor	Data structure which contains owner SID, group SID, DACL, SACL, and control flags. More information can be found in [1, 7].	

KEY (NK) RECORDS

OFFSET	SIZE	TYPE	DESCRIPTION	DELETION NOTES
0x0	2	String (“nk”)	Magic number	
0x2	2	Flags	See Observed Key Flags table below	
0x4	8	Unsigned Integer	64-bit NT time stamp	Only updated if this key has sub-keys. On Win2K, not updated even in that case.
0xC	4	Unknown	Unknown	
0x10	4	Offset	Parent NK record	
0x14	4	Unsigned Integer	Number of subkeys (stable)	Set to 0
0x18	4	Unsigned Integer	Number of subkeys (volatile [4])	
0x1C	4	Offset	Pointer to the subkey-list (stable)	Set to 0xFFFFFFFF
0x20	4	Offset	Pointer to the subkey-list (volatile [4])	
0x24	4	Unsigned Integer	Number of values	
0x28	4	Offset	Pointer to the value-list for values	
0x2C	4	Offset	Pointer to the SK record	Set to 0xFFFFFFFF
0x30	4	Offset	Pointer to the class name	
0x34	4	Unsigned Integer	Maximum number of bytes in a subkey name (unconfirmed)	Set to 0
0x38	4	Unsigned Integer	Maximum subkey class name length (unconfirmed)	
0x3C	4	Unsigned Integer	Maximum number of bytes in a value name (unconfirmed)	
0x40	4	Unsigned Integer	Maximum value data size (unconfirmed)	
0x44	4	Unknown	Unknown (possibly some sort of run-time index)	
0x48	2	Unsigned Integer	Key name length	
0x4A	2	Unsigned Integer	Class name length	
0x4C	variable	String	The key name; stored in ASCII and is typically NUL terminated	

SUBKEY-LISTS

OFFSET	SIZE	TYPE	DESCRIPTION	DELETION NOTES
0x0	2	String	Magic number (“lf”, “lh”, “ri”, or “li”)	
0x2	2	Unsigned Integer	Number of elements in this subkey-list	Set to 0
0x4	4 or 8 (each)	Structure List	Multiple subkey-list elements; see below for contents	List of elements deleted in some sequence, causing many old elements to be lost.

SUBKEY-LIST ELEMENTS FOR LF AND LH TYPES

OFFSET	SIZE	TYPE	DESCRIPTION	DELETION NOTES
0x0	4	Offset	Pointer to NK record	
0x4	4	Unsigned Integer	Hash value computed differently depending on subkey-list type (“lf” or “lh”)	

SUBKEY-LIST ELEMENTS FOR RI AND LI TYPES

OFFSET	SIZE	TYPE	DESCRIPTION	DELETION NOTES
0x0	4	Offset	If the type is “ri” then this is a pointer to another subkey-list record. Otherwise, it points to a subkey.	

VALUE (VK) RECORDS

OFFSET	SIZE	TYPE	DESCRIPTION	DELETION NOTES
0x0	2	String (“vk”)	Magic number	Under Win2K, typically overwritten with 0xFFFF
0x2	2	Unsigned Integer	Value name length	Under Win2K, typically overwritten with 0xFFFF
0x4	4	Unsigned Integer	Data length	
0x8	4	Offset	Pointer to data	
0xC	4	Enumeration	Value type; one of: REG_NONE (0), REG_SZ (1), REG_EXPAND_SZ (2), REG_BINARY (3), REG_DWORD (4), REG_DWORD_BIG_ENDIAN (5), REG_LINK (6), REG_MULTI_SZ (7), REG_RESOURCE_LIST (8), REG_FULL_RESOURCE_DESCRIPTOR (9), REG_RESOURCE_REQUIREMENTS_LIST (10), REG_QWORD (11). In some cases, this convention is not followed and other nonstandard values are used.	
0x10	2	Flags	If the 0 bit is set, the value name is in ASCII, otherwise it is in UTF-16LE.	
0x12	2	Unknown	Unknown	
0x14	variable	String	The value name; stored in ASCII and is typically NUL terminated	

VALUE-LISTS

OFFSET	SIZE	TYPE	DESCRIPTION	DELETION NOTES
0x0..[4*(num. values)]	4	Offset	List of pointers to VK records; Appear in order of value creation	List left intact if parent key is deleted. List simply rewritten over the top of original when elements are removed.

NORMAL DATA BLOCKS

OFFSET	SIZE	TYPE	DESCRIPTION	DELETION NOTES
0x0	variable	Raw Data	Data type and structure depends on type indicated by VK record.	On Win2K, first 4 bytes overwritten with 0xFFFFFFFF.

BIG DATA RECORDS

OFFSET	SIZE	TYPE	DESCRIPTION	DELETION NOTES
0x0	2	String (“db”)	Magic number	Not yet studied.
0x2	2	Unsigned Integer	Number of data fragments	Not yet studied.
0x4	4	Offset	Pointer to big data indirect cell	Not yet studied.
0x8	4	Unknown	Unknown (unused?)	Not yet studied.

BIG DATA INDIRECT CELLS

OFFSET	SIZE	TYPE	DESCRIPTION	DELETION NOTES
0x0..[4*(num. fragments)]	4	Offset	To a data fragment	Not yet studied.

Appendix B: Record Flags and Constants

VALUE DATA TYPES

NAME	ENUM. VALUE	FORMAT SUMMARY
REG_NONE	0x0	Unknown. Apparently treated like REG_BINARY.
REG_SZ	0x1	UTF-16 little endian string
REG_EXPAND_SZ	0x2	UTF-16 little endian string with system path variable (e.g., “`%SYSTEMROOT%`”) escapes
REG_BINARY	0x3	Raw data
REG_DWORD	0x4	32 bit, little endian integer
REG_DWORD_LITTLE_ENDIAN	0x4	Microsoft alias for REG_DWORD, though it is not clear what endian format a big endian Windows system (e.g. NT on Alpha) would default to.
REG_DWORD_BIG_ENDIAN	0x5	32 bit, big endian integer
REG_LINK	0x6	A symbolic link, stored as a UTF-16 little endian string
REG_MULTI_SZ	0x7	A list of UTF-16 little endian strings. Each string is NUL (“`\x00\x00`”) terminated, and the list itself is NUL terminated as well (resulting in a total of four 0-bytes at the end of the data).
REG_RESOURCE_LIST	0x8	“A series of nested arrays” of unknown format. See [9].
REG_FULL_RESOURCE_DESCRIPTOR	0x9	“A series of nested arrays” of unknown format. See [9].
REG_RESOURCE_REQUIREMENTS_LIST	0xA	“A series of nested arrays” of unknown format. See [9].
REG_QWORD	0xB	64 bit, little endian integer

OBSERVED KEY FLAGS

FLAG	DESCRIPTION
0x4000	Unknown; shows up on normal-seeming keys in Vista and W2K3 hives.
0x1000	Unknown; shows up on normal-seeming keys in Vista and W2K3 hives.
0x0080	Unknown; shows up on root keys in some Vista "software" hives.
0x0040	Predefined handle; see: [10]
0x0020	The key name will be in ASCII if set; otherwise it is in UTF-16LE.
0x0010	Symlink key; see: [6]
0x0008	This key cannot be deleted.
0x0004	Key is root of a registry hive.
0x0002	Mount point of another hive.
0x0001	Volatile key; these keys shouldn't be stored on disk, according to: [5]

Appendix C: Algorithms

Base Block Hash Algorithm

The following algorithm is used in the "`regf`" header base block:

```
let B be the first 508 bytes of the registry base block
let H be a 32-bit value
H = 0
for each 32-bit group, C, in B do
    H = H ⊕ C
H = reverseByteOrder(H) /* interpret 4-byte groups as little-endian */
return H
```

"lh" Subkey-List Hash Algorithm

The following algorithm was extrapolated from Samba source code[18]. It has not been verified for correctness in all situations. In particular, it is not known precisely how subkeys with UTF-16LE names would be processed since it does not appear that Samba handles this case.

```
let N be the subkey name
let H be a 32-bit value
H = 0
N = uppercase(N)
for each byte, B, in N do
    H = (H × 37) mod 2^32
    H = (H + B) mod 2^32
return H
```