Chapter 4
QNX6

Conrad Meyer

Abstract The QNX6 filesystem is present in Smartphones delivered by Blackberry (e.g. Devices that are using Blackberry 10) and modern vehicle infotainment systems that use QNX as their operating system. In 2015 QNX as an OS was used in over 50 million vehicles [6] and can hence be considered as one of the most important operating systems in the automotive world. Today's digital forensics tools don't recover a lot from this filesystem, have difficulties with different block sizes, or even don't support the filesystem at all. So it's crucial for the forensic examiner to understand the principles of this filesystem used. This chapter gives an overview of how the filesystem generally stores the files and metadata to give the examiner the chance to get the most information out of the evidence.

4.1 Introduction

This chapter gives an insight into the different structures and principles of the QNX6 filesystem developed by QNX.The filesystem was first introduced within QNX Neutrino 6.4 real-time operating system, which today is owned and developed by Blackberry. It is a power-safe file system [7] and can withstand a sudden loss of power without corrupting or losing data. This property is especially useful for the forensic examiner, as it can easily happen that evidence (e.g. a vehicle or smartphone) loses its power supply due to a battery pack running empty.

Conrad Meyer
Central Office for Information Technology in the Security Sector (ZITiS), Zamdorfer Straße 88, Munich, Bavaria e-mail: conrad.meyer@zitis.bund.de

C. Hummert, D. Pawlaszczyk (eds.), *Mobile Forensics – The File Format Handbook*,
https://doi.org/10.1007/978-3-030-98467-0_4

Table 4.1: Standard Parameters of the QNX6 Filesystem

Parameter	Value	Remark
Max physical Size	2 TB 2	
Supported Standard Logical Blocksizes	512, 1024, 2048, 4096 Bytes	
Max Filename Length	510 bytes	UTF-8

Table 4.1 shows the standard values that are regularly used when formatting a volume with the QNX6 filesystem. Note, that especially in-car infotainment systems, those values can be different (e.g. larger blocksize). All the addressing inside the filesystem is based on the blocksize, extracted out of the superblock.

The following sections will give the reader an insight into the binary structures of the most important parts of the filesystem, like a superblock or inode and some basic knowledge about the mechanism when files are deleted.

4.2 QNX6 Filesystem Structure

To understand the principle behaviour and main functions of the QNX6 filesystem, the following chapter shows the structure of a volume and how files, directories and metadata are linked. Volumes can be formatted in QNX6 in little-endian or big-endian style. All the examples in the following show a QNX6 Volume formatted with little endianness. Fig. 4.1 shows the main parts of a QNX6 filesystem and their standard size and addresses. The system area contains the Bitmap of the allocated

Fig. 4.1: Layout of a QNX6 filesystem volume

and unallocated Blocks of the Filesystem. Each bit represents a Block. Suppose the volume is formatted in the standard way. In that case, the volume will start with a volume boot record, which contains standard ASCII coded bootloader messages (Fig. 4.2), already giving a hint that the Volume is formatted with QNX.

! Attention

Sometimes, on non standard volumes a partition directly starts with the Superblock.

```
Offset    0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F   ANSI ASCII
00000000  EB 10 90 00 20 60 D2 00  10 00 00 00 D8 3F 06 00  ë   `Ò    Ø?
00000010  00 80 FA 31 C0 8E D0 BC  00 20 B8 C0 07 50 B8 36   €ú1ÀŽÐ¼  ¸À P¸6
00000020  01 50 CB 00 00 00 00 00  00 00 00 00 00 00 00 00   PË
00000030  00 00 66 90 00 00 00 00  00 00 00 00 8D B4 00 00    f          ´
00000040  10 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000060  FF FF 00 00 00 93 00 00  FF FF 00 00 00 93 00 00  ÿÿ   "   ÿÿ   "
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000090  18 00 90 7C 00 00 00 00  FF FF 00 00 00 9B CF 00     |    ÿÿ   ›Ï
000000A0  FF FF 00 00 00 93 CF 00  0D 0A 51 4E 58 20 76 31  ÿÿ   "Ï   QNX v1
000000B0  2E 32 62 20 42 6F 6F 74  20 4C 6F 61 64 65 72 00  .2b Boot Loader
000000C0  0D 0A 55 6E 73 75 70 70  6F 72 74 65 64 20 42 49    Unsupported BI
000000D0  4F 53 00 0D 0A 52 41 4D  20 45 72 72 6F 72 00 0D  OS   RAM Error
000000E0  0A 44 69 73 6B 20 52 65  61 64 20 45 72 72 6F 72   Disk Read Error
000000F0  00 0D 0A 4D 69 73 73 69  6E 67 20 4F 53 20 49 6D     Missing OS Im
00000100  61 67 65 00 0D 0A 49 6E  76 61 6C 69 64 20 4F 53  age   Invalid OS
00000110  20 49 6D 61 67 65 00 0D  0A 55 6E 73 75 70 70 6F   Image   Unsuppo
00000120  72 74 65 64 20 4D 75 6C  74 69 2D 42 6F 6F 74 00  rted Multi-Boot
00000130  3A 20 00 0D 0A 00 0E 1F  88 16 11 00 FB FC F6 06  :        ˆ   ûüö
00000140  03 00 02 74 03 E8 3F 00  F6 06 03 00 01 75 06 BE     t è? ö    u ¾
00000150  A8 00 E8 3C 00 BB AA 55  B4 41 CD 13 72 21 81 FB  ¨ è< »ªU´AÍ r! û
00000160  55 AA 75 1B F6 C1 01 74  16 B8 00 02 50 8E C0 B8  Uªu öÁ t ¸  PŽÀ¸
00000170  00 02 50 66 31 C0 89 C7  BB 08 00 E8 40 00 CB BE    Pf1À‰Ç»  è@ Ë¾
00000180  C0 00 E8 0C 00 EB 53 B4  0F CD 10 83 E0 7F CD 10  À è  ëS´ Í ƒà Í
00000190  C3 AC 08 C0 74 09 BB 07  00 B4 0E CD 10 EB F2 C3  Ã¬ Àt »  ´ Í ëòÃ
000001A0  66 03 06 04 00 BE 40 00  89 5C 02 89 7C 04 8C 44  f    ¾@ ‰\ ‰| ŒD
000001B0  06 66 89 44 08 8A 16 11  00 B4 42 CD 13 C3 56 E8   f‰D Š   ´BÍ ÃVè
000001C0  DE FF 72 02 5E C3 BE DF  00 E8 C5 FF EB 0C E8 CF  Þÿr ^Ã¾ß èÅÿë èÏ
000001D0  FF 73 06 F6 C4 10 75 EE  F9 C3 F4 EB FD 00 00 00  ÿs öÄ uîùÃôëý
000001E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000001F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 AA                Uª
```

Fig. 4.2: Sector 0 of a QNX6 Partition/Volume

In the following, we will have a closer look at all the structures above. We will follow those structures to construct a file and its metadata out of the filesystem information. The example filesystem is in little-endian mode.

4.2.1 Superblock

The filesystem maintains two Superblocks or global root blocks. One of those blocks, called the working Superblock, manages the modified data, while the other one, the stable Superblock, consists of the original version of all the blocks. Which Superblock is the active one is determined by the 64-bit long serial number. The Superblock with the higher serial is the active one. After all, active write operations are done, and the integrity is checked, the former working superblock becomes the new stable one by updating the serial number (old superblock serial +1).

The superblock contains the global information of the filesystem. Table 4.2 contains the offset address of the main features of the Superblock.

Table 4.2: Main Features and their Offset in the QNX6 superblock

Parameter	Offset in Superblock	Size (bytes)
Serialnumber	0x8	8
creation timestamp	0x10	8
last access timestamp	0x14	8
Volume ID	0x20	16
Blocksize	0x30	4
Root Inode Inodes	0x48	array 16 x 4 bytes
Root Inode bitmap	0x98	array 16 x 4 bytes
Root Inode longfilenames	0xE8	array 16 x 4 bytes

! Attention

When used with the standard driver issued by Blackberry and the default settings, you can determine the last access to the filesystem by selecting the stable superblock (highest serial) and checking the access timestamp (assuming that system time is used was valid). However, some non-standard drivers don't touch this timestamp, so for reliable results, you have to test the drivers from the System where the image originated in each case!

The superblock contains three root inodes that point to the main parts of the filesystem. The first array root inode contains the pointers to the inodes that contain the data (files, directories, data). The second one contains the pointers to the bitmap of the allocated blocks, and the third one is the pointers to the long filenames (filenames > 27 utf8 characters, up to 510 characters). The data inside those root inodes is shown in Table 4.3. Those root inodes contain pointers to the corresponding filesystem parts. If the level parameter is zero, the root inode has 16 direct pointers. By adding another level, indirect pointers are added, as shown in Fig. 4.4. Each indirect pointer then points to a block containing inodes or indirect 32-bit pointers, depending on the defined number of levels. The actual data is always at the lowest level of the tree. Given the value of blocks that such a tree can address is $16 * (\text{block size in bytes} / 4)^{level}$ So, for example, with a level value of 2, and a block size of 1024 bytes, already 1,048,576 blocks can be addressed.

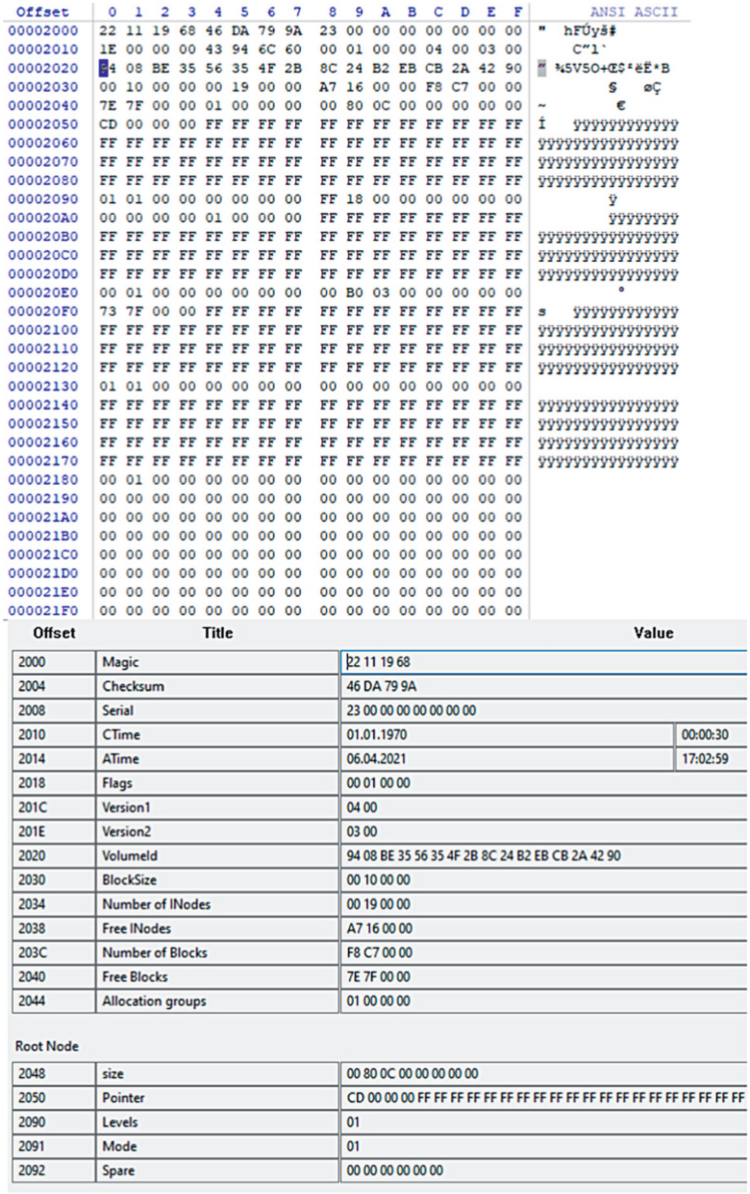


Fig. 4.3: An example of a QNX6 superblock.

Table 4.3: Structure of the root inodes

Parameter	Offset in root inode	Size (bytes)
Size	0x0	8
Pointer	0x8	array 16 x 4 bytes
Levels	0x48	1
Mode	0x49	1

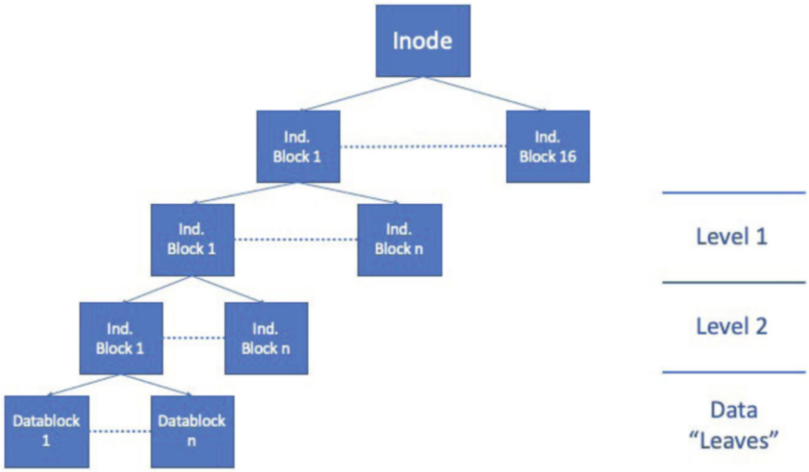


Fig. 4.4: Illustration of inode levels, here a level value of 3

4.2.2 Bitmap

The Bitmap block is used to determine whether a block in the filesystem is used or not. Each bit in the bitmap represents a block. A value of 0 means the Block is unused, 1 means that the Block is allocated. If the volume size is smaller than the bits available in the Bitmap Block, the unused bits are stuffed with ones. The bitmap incorporates two parts. First, system area 1 is split into two halves, where the upper half is used by superblock 1, and the lower half is used by superblock 2. This bitmap area contains the bitmap, inode and indirect addressing blocks of those structures. Second, the bitmap of the blocks that are not used for the filesystem structure (bitmap and inodes). The preallocation of the first system area block leads to the effect that each superblock always works on its own filesystem structure, and to the point that there is always a non-corrupted structure, even in the case of a sudden power loss (a superblock is just becoming the stable one, if all write operations are done, see sect. 4.2.1).

Fig. 4.5 depicts the end of the used space of the bitmap pointed to in the example superblock from Fig. 4.3. The bitmap comprises two blocks, starting at 0x3000, and the volume contains a total of 0xC7F8 blocks. In Fig 4.5, the stuffing of the unused space with ones therefore starts at 0x48FF: Bitmap starting address: 0x3000 + number of blocks 0xC8f8 divided by 8 (each Block represented by 1 bit).

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
000037F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00003800	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00003810	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00003820	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00003830	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00003840	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00003850	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00003860	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00003870	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00003880	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00003890	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000038A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000038B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000038C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000038D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000038E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000038F0	00	00	00	00	F0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ðÿÿÿÿÿÿÿÿÿÿÿ
00003900	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00003910	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00003920	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00003930	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00003940	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00003950	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00003960	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00003970	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00003980	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00003990	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000039A0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ

Fig. 4.5: An example of a QNX6 Bitmap

4.2.3 Inode

On the lowest level of the root inode tree, in the "leaves", the direct inode data is found. Depending on the level defined, also those inodes can address other indirect inode addressing blocks. An inode contains a vast amount of data useful for the forensic examiner, e.g. permissions, access time, change time, and modification time. Table 4.4 shows the offsets and the size of the various parameters in an inode.

Table 4.4: Structure of an inode

Parameter	Offset	Size (bytes)
size	0x0	8
uid	0x8	4
gid	0xC	4
ftime	0x10	4
mtime	0x14	4
atime	0x18	4
ctime	0x1C	4
mode	0x20	2
blockpointer	0x24	array 16 x 4 bytes
Levels	0x54	1
status	0x49	1 (see table 4.5)

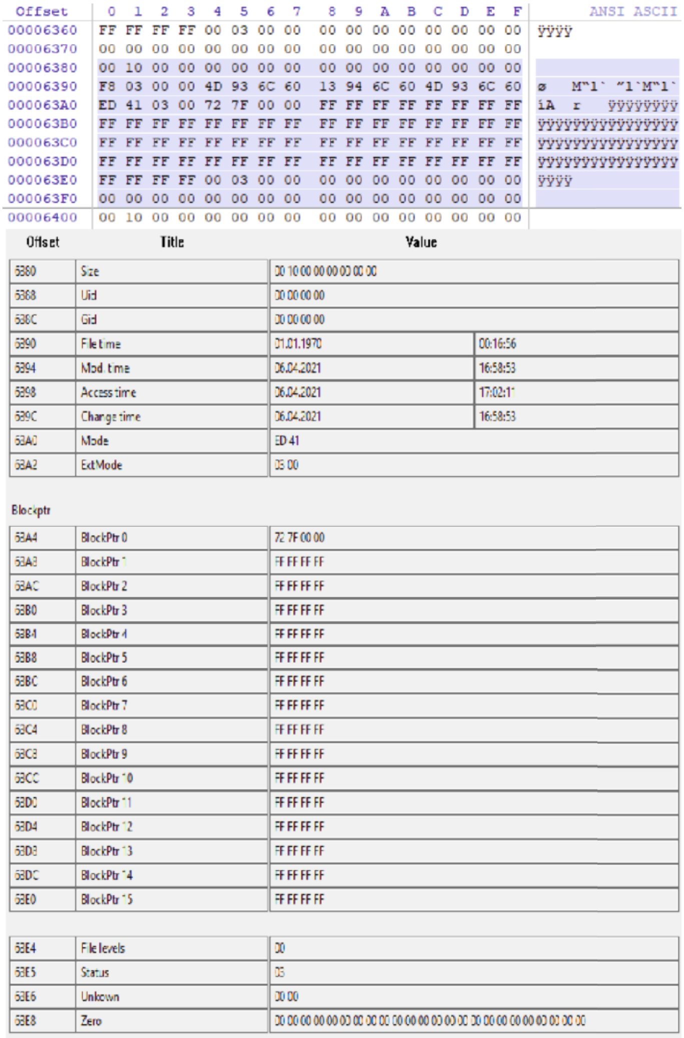
Table 4.5: inode status byte

Value	Status
0x1	directory
0x2	deleted
0x3	normal

As QNX OS is in line with the POSIX standards; also the timestamps are. The epoch is the standard POSIX (or UNIX) epoch, the 01.01.1970, 00:00 UTC. From that epoch, the timestamps are counted in seconds. The modified timestamp (mtime) is the time of the last write operation on this specific file. The access timestamp (atime) tells the examiner the time the file was last read. The change timestamp (ctime) is changed when the permissions of a file are changed. So ctime can be changed without a change in atime. The timestamp ftime is not fully referenced in the POSIX standard. Like in many other filesystems, it is the timestamp when the file was created. The inode 1 always contains the root directory, and inode counting starts with 1.

! Attention

When it comes to timestamps, the forensic expert has to pay attention to the reliability of the timestamps given. This is especially true for QNX6. Not all timestamps are actualised on some systems, as with QNX with the standard QNX6 file-system driver. Whenever possible, tests with the system you are examining should be performed (e.g. changing permissions, modifying files, etc.)!



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
00006360	FF	FF	FF	FF	00	03	00	00	00	00	00	00	00	00	00	00	ÿÿÿÿ
00006370	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00006380	00	10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00006390	F8	03	00	00	4D	93	6C	60	13	94	6C	60	4D	93	6C	60	ø M"l` "l`M"l`
000063A0	ED	41	03	00	72	7F	00	00	FF	FF	FF	FF	FF	FF	FF	FF	íA r ÿÿÿÿÿÿÿÿ
000063B0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000063C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000063D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000063E0	FF	FF	FF	FF	00	03	00	00	00	00	00	00	00	00	00	00	ÿÿÿÿ
000063F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00006400	00	10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Offset	Title	Value	
6380	Size	00 10 00 00 00 00 00 00	
6388	Uid	00 00 00 00	
638C	Gid	00 00 00 00	
6390	File time	01.01.1970	00:16:56
6394	Mod. time	06.04.2021	16:58:53
6398	Access time	06.04.2021	17:02:11
639C	Change time	06.04.2021	16:58:53
63A0	Mode	ED 41	
63A2	ExtMode	03 00	

Blockptr

Offset	Title	Value
63A4	BlockPtr 0	72 7F 00 00
63A8	BlockPtr 1	FF FF FF FF
63AC	BlockPtr 2	FF FF FF FF
63B0	BlockPtr 3	FF FF FF FF
63B4	BlockPtr 4	FF FF FF FF
63B8	BlockPtr 5	FF FF FF FF
63BC	BlockPtr 6	FF FF FF FF
63C0	BlockPtr 7	FF FF FF FF
63C4	BlockPtr 8	FF FF FF FF
63C8	BlockPtr 9	FF FF FF FF
63CC	BlockPtr 10	FF FF FF FF
63D0	BlockPtr 11	FF FF FF FF
63D4	BlockPtr 12	FF FF FF FF
63D8	BlockPtr 13	FF FF FF FF
63DC	BlockPtr 14	FF FF FF FF
63E0	BlockPtr 15	FF FF FF FF

Offset	Title	Value
63E4	File levels	00
63E5	Status	03
63E6	Unkown	00 00
63E8	Zero	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

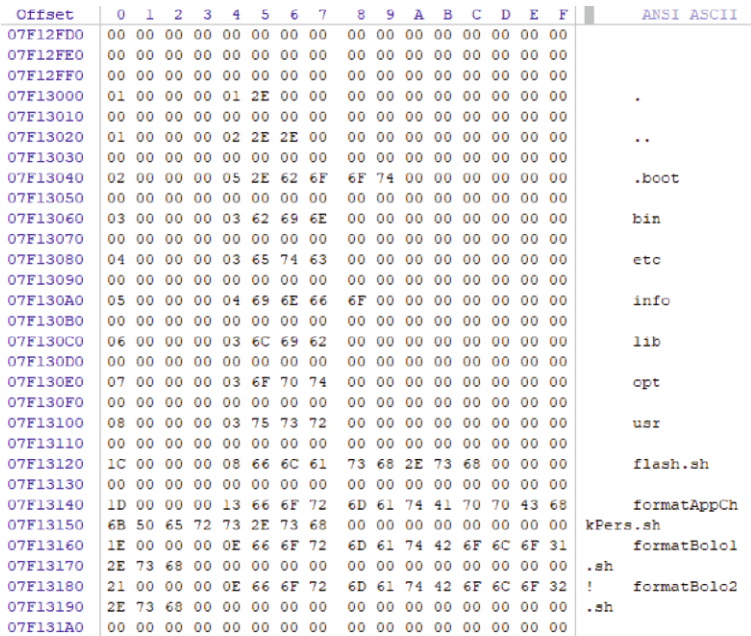
Fig. 4.6: An example of a QNX6 Inode.

4.2.4 Directories

Inodes with the status 0x3 point to a directory file system object that contains subdirectories and file entries with names shorter than 27 UTF-8 characters. An entry starts with the inode number of that entry, where you can find the metadata like timestamps and the pointers to the Data or other directories, followed by a name length field and the actual name. A directory always contains a "." and a ".." entry. The "." entry contains the inode number of the directory inode, and the ".." entry contains the inode number of the parent directory inode. In the example Fig. 4.7, those entries are both pointing to the same inode number because the directory shown is the root directory.

Table 4.6: Directory entry

Parameter	Offset	Size (bytes)
Inode number	0x0	4
Namelength	0x4	1
Name	0x5	up to 27



```
Offset    0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F   ANSI ASCII
07F12FD0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
07F12FE0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
07F12FF0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
07F13000  01 00 00 00 01 2E 00 00  00 00 00 00 00 00 00 00       .
07F13010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
07F13020  01 00 00 00 02 2E 2E 00  00 00 00 00 00 00 00 00       ..
07F13030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
07F13040  02 00 00 00 05 2E 62 6F  6F 74 00 00 00 00 00 00       .boot
07F13050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
07F13060  03 00 00 00 03 62 69 6E  00 00 00 00 00 00 00 00       bin
07F13070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
07F13080  04 00 00 00 03 65 74 63  00 00 00 00 00 00 00 00       etc
07F13090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
07F130A0  05 00 00 00 04 69 6E 66  6F 00 00 00 00 00 00 00       info
07F130B0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
07F130C0  06 00 00 00 03 6C 69 62  00 00 00 00 00 00 00 00       lib
07F130D0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
07F130E0  07 00 00 00 03 6F 70 74  00 00 00 00 00 00 00 00       opt
07F130F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
07F13100  08 00 00 00 03 75 73 72  00 00 00 00 00 00 00 00       usr
07F13110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
07F13120  1C 00 00 00 08 66 6C 61  73 68 2E 73 68 00 00 00       flash.sh
07F13130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
07F13140  1D 00 00 00 13 66 6F 72  6D 61 74 41 70 70 43 68       formatAppCh
07F13150  6B 50 65 72 73 2E 73 68  00 00 00 00 00 00 00 00  kPers.sh
07F13160  1E 00 00 00 0E 66 6F 72  6D 61 74 42 6F 6C 6F 31       formatBolo1
07F13170  2E 73 68 00 00 00 00 00  00 00 00 00 00 00 00 00  .sh
07F13180  21 00 00 00 0E 66 6F 72  6D 61 74 42 6F 6C 6F 32  !    formatBolo2
07F13190  2E 73 68 00 00 00 00 00  00 00 00 00 00 00 00 00  .sh
07F131A0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

Fig. 4.7: An example of a QNX6 directory. Here, the root directory is shown.

A long directory entry has a different structure (Table 4.7). It includes the Inode, in which the timestamps and pointers to the data are. Furthermore, the long filenames inode Number, where the entry's name is found, is noted in this structure. An example of a long filename/directory entry is displayed in Fig. 4.8.

Table 4.7: Long Directory entry

Parameter	Offset	Size (bytes)
Inode number	0x0	4
size	0x4	1
Long Filenames Inode Number	0x8	4
checksum	0x12	checksum

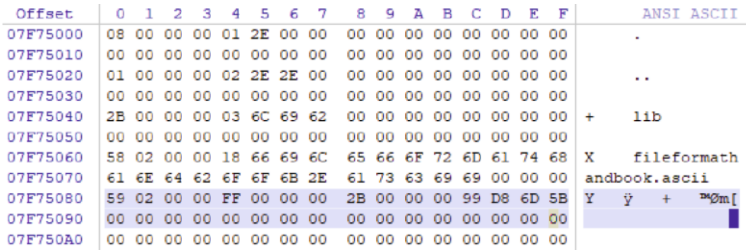


Fig. 4.8: An example of a QNX6 inode entry of a long filename

4.2.5 Long Filenames Inode

If a file or directories length is longer than 27 UTF-8 characters, the name is stored in the long filenames node. Long filenames Inodes start counting with zero. The structure is shown in Table 4.8, an example is Fig. 4.9.

Table 4.8: Long Filenames Inode

Parameter	Offset	Size (bytes)
filename length	0x0	2
filename	0x2	up to 510 bytes

4.3 Example: Construction of a file

To understand how a file can be retrieved from the filesystem data, we will manually find the file /usr/fileformathandbook.ascii with its content and metadata by using the

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
03221000	24	00	66	69	6C	65	66	6F	72	6D	61	74	68	61	6E	64	$ fileformathand
03221010	62	6F	6F	6B	76	65	72	79	6C	6F	6E	67	6E	61	6D	65	bookverylongname
03221020	2E	61	73	63	69	69	00	00	00	00	00	00	00	00	00	00	.ascii
03221030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Fig. 4.9: An example QNX6 long filenames entry

filesystem information. We will begin the reconstruction from the root directory. As already mentioned in the previous chapter, inode 1 contains the root directory. From there, we will start finding the file in the filesystem structure. The first step is to determine the valid stable superblock by the serial number. The superblocks inode root block is shown in Fig. 4.10

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
00002000	22	11	19	68	46	DA	79	9A	23	00	00	00	00	00	00	00	" hFÚyš#
00002010	1E	00	00	00	43	94	6C	60	00	01	00	00	04	00	03	00	C"l`
00002020	94	08	BE	35	56	35	4F	2B	8C	24	B2	EB	CB	2A	42	90	" ¾5V5O+Œ$²ëË*B
00002030	00	10	00	00	00	19	00	00	A7	16	00	00	F8	C7	00	00	§ øÇ
00002040	7E	7F	00	00	01	00	00	00	00	80	0C	00	00	00	00	00	~ €
00002050	CD	00	00	00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	Í ÿÿÿÿÿÿÿÿÿÿÿÿ
00002060	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00002070	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00002080	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00002090	01	01	00	00	00	00	00	00	FF	18	00	00	00	00	00	00	ÿ

Fig. 4.10: Inode Root block used in the file reconstruction example

The root block tree has one level, meaning that we go on with the indirect inode block in the next step. The formula can easily calculate the physical address of those blocks:

$$blockaddress = blocknumber * blocksize + offset$$

On standard QNX6 Volumes, the offset is the superblock size + the offset of the beginning of the superblock. Thus, the first indirect inode block is located at 0xCD * 0x1000 + 0x3000 = 0xD0000, where 0xCD is the block number, 0x1000 the blocksize and 0x3000 the global offset due to the superblock with size 0x1000 and start at 0x2000. From the indirect inode (Fig. 4.11), we can retrieve the number 0x03, and by this, the address of the first inode block, which is located at 0x6000.

The first inode in this block is the root inode. If we take the first block pointer, 0x7F10, of this inode, we get the address of the root directory: 0x7F13000. This root directory, Fig. 4.13 is already familiar to us, as the second version of it is shown in Fig. 4.7, but this time, it is the root directory maintained by the first superblock.
In the root directory, we take the inode number for the /usr directory, 0x08. With this number, we go back to the first Inode Block, where the inode 8 is located at 0x6380 (0x6000, where inode 1 is located plus 7 * 0x80 offset, for the preceding inodes). From that inode (Fig. 4.14) we can then calculate the /usr directory offset in the way we already did for the root directory. The /usr directory is defined at block 0x7F72

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
000D0000	03	00	00	00	CF	00	00	00	D0	00	00	00	D1	00	00	00	Ï Ð Ñ
000D0010	D2	00	00	00	D3	00	00	00	D4	00	00	00	D5	00	00	00	Ò Ó Ô Õ
000D0020	0B	00	00	00	D7	00	00	00	0D	00	00	00	0E	00	00	00	×
000D0030	DA	00	00	00	DB	00	00	00	DC	00	00	00	DD	00	00	00	Ú Û Ü Ý
000D0040	13	00	00	00	DF	00	00	00	E0	00	00	00	16	00	00	00	ß à
000D0050	17	00	00	00	18	00	00	00	19	00	00	00	1A	00	00	00	
000D0060	1B	00	00	00	1C	00	00	00	1D	00	00	00	1E	00	00	00	

Fig. 4.11: Indirect inode block

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
00006000	00	10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00006010	1E	00	00	00	CC	43	6D	38	10	94	6C	60	0C	44	6D	38	ÌCm8 ”l` Dm8
00006020	FD	41	09	00	10	7F	00	00	FF	FF	FF	FF	FF	FF	FF	FF	ýA ÿÿÿÿÿÿÿÿ
00006030	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00006040	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00006050	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00006060	FF	FF	FF	FF	00	01	00	00	00	00	00	00	00	00	00	00	ÿÿÿÿ
00006070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00006080	00	10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00006090	1E	00	00	00	1E	00	00	00	1E	00	00	00	0C	44	6D	38	Dm8

Fig. 4.12: inode 1 which contains the pointers to the root diretory

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
07F13000	01	00	00	00	01	2E	00	00	00	00	00	00	00	00	00	00	.
07F13010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
07F13020	01	00	00	00	02	2E	2E	00	00	00	00	00	00	00	00	00	..
07F13030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
07F13040	02	00	00	00	05	2E	62	6F	6F	74	00	00	00	00	00	00	.boot
07F13050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
07F13060	03	00	00	00	03	62	69	6E	00	00	00	00	00	00	00	00	bin
07F13070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
07F13080	04	00	00	00	03	65	74	63	00	00	00	00	00	00	00	00	etc
07F13090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
07F130A0	05	00	00	00	04	69	6E	66	6F	00	00	00	00	00	00	00	info
07F130B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
07F130C0	06	00	00	00	03	6C	69	62	00	00	00	00	00	00	00	00	lib
07F130D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
07F130E0	07	00	00	00	03	6F	70	74	00	00	00	00	00	00	00	00	opt
07F130F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
07F13100	08	00	00	00	03	75	73	72	00	00	00	00	00	00	00	00	usr
07F13110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
07F13120	1C	00	00	00	08	66	6C	61	73	68	2E	73	68	00	00	00	flash.sh
07F13130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Fig. 4.13: Root Directory

which is at offset 0x7F75000. Here we see now our filename and the corresponding inode Number, where the metadata and pointer to the file content is.

We see that the *fileformathandbook.ascii* file has the inode number 0x258. Knowing this, we have to find the offset where this inode is defined. With a block size of 0x1000 and an inode size of 0x80, each inode block contains 0x20 inodes, so the inode we are looking for is the 24th inode in inode block number 19. Going back to Fig. 4.11, the 19 inode block is at physical block 0xE0, calculated address 0xE3000

Offset	Title	Value	
6380	Size	00 10 00 00 00 00 00 00	
6388	Uid	00 00 00 00	
638C	Gid	00 00 00 00	
6390	File time	01.01.1970	00:16:56
6394	Mod. time	06.04.2021	16:58:53
6398	Access time	06.04.2021	17:02:11
639C	Change time	06.04.2021	16:58:53
63A0	Mode	ED 41	
63A2	ExtMode	03 00	

```
00006370  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00006380  00 10 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00006390  F8 03 00 00 4D 93 6C 60  13 94 6C 60 4D 93 6C 60  ø   M"l` "l`M"l`
000063A0  ED 41 03 00 72 7F 00 00  FF FF FF FF FF FF FF FF  íA  r   ÿÿÿÿÿÿÿÿ
000063B0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000063C0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000063D0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000063E0  FF FF FF FF 00 03 00 00  00 00 00 00 00 00 00 00  ÿÿÿÿ
000063F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

Fig. 4.14: Inode 8, which has the pointer to the /usr directory in our example

```
Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F      ANSI ASCII
07F75000  08 00 00 00 01 2E 00 00  00 00 00 00 00 00 00 00       .
07F75010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
07F75020  01 00 00 00 02 2E 2E 00  00 00 00 00 00 00 00 00       ..
07F75030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
07F75040  2B 00 00 00 03 6C 69 62  00 00 00 00 00 00 00 00  +    lib
07F75050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
07F75060  58 02 00 00 18 66 69 6C  65 66 6F 72 6D 61 74 68  X    fileformath
07F75070  61 6E 64 62 6F 6F 6B 2E  61 73 63 69 69 00 00 00  andbook.ascii
07F75080  59 02 00 00 FF 00 00 00  2B 00 00 00 99 D8 6D 5B  Y   ÿ   +   ™Øm[
07F75090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

Fig. 4.15: /usr directory with the entry of the file we are looking for

+ 0xB80 (24th inode in Block). In this inode, depicted in Fig. 4.16 we find all the relevant filesystem metadata for this file and the pointers to the filesystem content.

Following now the pointers to the content, beginning with 0x19D, we can retrieve the file block by block (Fig. 4.17).

After demonstrating the retrieval of the example file from the file system data, it is easy to understand the next section, which shows the possibilities to reconstruct deleted files.

4.4 Deleted Files

There are some possibilities to recover deleted files in a QNX6 Volume, depending, when the file or directory was deleted and what happened with the filesystem in the meanwhile. Deleting an entry (directory or file) in QNX6 means that the Status in

Offset	Title	Value	
E3B80	Size	9C 16 00 00 00 00 00 00	
E3B88	Uid	00 00 00 00	
E3B8C	Gid	00 00 00 00	
E3B90	File time	06.04.2021	16:57:31
E3B94	Mod. time	06.04.2021	17:02:39
E3B98	Access time	06.04.2021	17:02:56
E3B9C	Change time	06.04.2021	17:02:39
E3BA0	Mode	FD 81	
E3BA2	ExtMode	01 00	
Blockptr			
E3BA4	BlockPtr 0	9D 01 00 00	
E3BA8	BlockPtr 1	1C 32 00 00	
E3BAC	BlockPtr 2	FF FF FF FF	
E3BB0	BlockPtr 3	FF FF FF FF	

```
000E3B60  FF FF FF FF 00 03 00 00  00 00 00 00 00 00 00 00  ÿÿÿÿ
000E3B70  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000E3B80  9C 16 00 00 00 00 00 00  00 00 00 00 00 00 00 00  œ
000E3B90  FB 92 6C 60 2F 94 6C 60  40 94 6C 60 2F 94 6C 60  û'l`/"l`@"l`/"l`
000E3BA0  FD 81 01 00 9D 01 00 00  1C 32 00 00 FF FF FF FF  ý        2  ÿÿÿÿ
000E3BB0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000E3BC0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000E3BD0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000E3BE0  FF FF FF FF 00 03 00 00  00 00 00 00 00 00 00 00  ÿÿÿÿ
000E3BF0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

Fig. 4.16: Inode entry of our example file

```
Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F      ANSI ASCII
0019FFD0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0019FFE0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0019FFF0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
001A0000  54 68 69 73 20 69 73 20  61 20 54 65 73 74 66 69  This is a Testfi
001A0010  6C 65 20 66 6F 72 20 74  68 65 20 46 69 6C 65 20  le for the File 
001A0020  46 6F 72 6D 61 74 20 68  61 6E 64 62 6F 6F 6B 2E  Format handbook.
001A0030  20 54 68 69 73 20 54 65  73 74 66 69 6C 65 20 6A   This Testfile j
001A0040  75 73 74 20 72 65 70 65  61 74 73 20 74 68 65 20  ust repeats the 
001A0050  73 61 6D 65 20 74 65 78  74 20 6F 76 65 72 20 61  same text over a
001A0060  6E 64 20 6F 76 65 72 20  61 67 61 69 6E 2E 20 54  nd over again. T
001A0070  68 69 73 20 69 73 20 61  20 54 65 73 74 66 69 6C  his is a Testfil
001A0080  65 20 66 6F 72 20 74 68  65 20 46 69 6C 65 20 46  e for the File F
001A0090  6F 72 6D 61 74 20 68 61  6E 64 62 6F 6F 6B 2E 20  ormat handbook. 
001A00A0  54 68 69 73 20 54 65 73  74 66 69 6C 65 20 77 61  This Testfile wa
```

Fig. 4.17: Content of our example file

an Inode switches to "deleted" (see Table 4.5) and that the entries inode number is deleted from the directory as shown in Fig. 4.18. By this, it is not possible to recover a file by its name, because there is no link anymore between the filename and the inode containing the metadata and the pointers to the file content. If a directory is updated after a file was deleted (e.g. a new file is added), the filesystem driver moves the directory to another block. The filename is “lost” from the regular filesystem directory tree. Also, the blocks, which contain the content of the files are set to unused in the bitmap, which means, they are free to be overwritten by other data. Knowing this, there are still some possibilities to recover files, with and without their respective names.

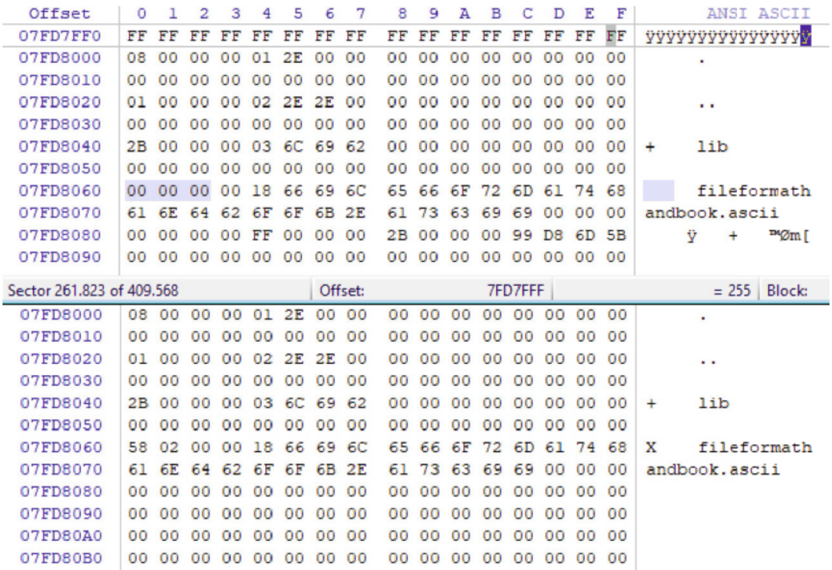


Fig. 4.18: Directory entry before (bottom) and after (top) deletion

The first possibility, if the file was just deleted recently, it may still be present in the non-active filesystem structure of the second superblock. If this is the case, the file can normally be fully recovered, even with its content (still, it is possible that the content is not original).

Second, you can parse the inodes to recover files with their metadata without the associated filename. This fact is quite problematic because the Blocks do not necessarily still contain the files original data.

In conclusion, we see that the reconstruction of files is sometimes possible. However, compared to some other filesystems (e.g. NTFS), there is a smaller possibility to recover deleted files from the filesystem information. In some special cases where you can prove the integrity of a file in another way (e.g. some packed/zipped files), it is still helpful to take advantage of the inode structure and the possibility to put together fragmented files from the pointers inside the inode.

4.5 Forensic Tools supporting QNX6 filesystems

The Linux kernel includes a read-only driver for QNX6 (and QNX4) file systems. Also, some mobile forensic tools like UFED physical analyzer support this file system to a certain degree. Until today, those tools just support volumes formatted with the standard values shown in Table 4.1. Lately, there have been some projects in the Autopsy / Sleuthkit community to support QNX6, but until today, none of the projects has come to an end.