# Finding Forensic Information on Creating a Folder in $LogFile of NTFS

Gyu-Sang Cho* and Marcus K. Rogers

Dept. Of Computer Information Warfare, Dongyang Univ.
1 Gyochon, Youngju, Kyoungbuk, Republic of Korea, 750-711
Dept. of Computer & Information Technology, Purdue Univ
401 N Grant St. W. Lafayette, IN, 47907, USA
cho@dyu.ac.kr, {chog*,rogersmk}@purdue.edu

**Abstract.** The NTFS journaling file($LogFile) is used to keep the file system clean in the event of a system crash or power failure. The log records operate on files or folders and leaves large amounts of information in the $LogFile. This information can be used to reconstruct operations and can also be used as forensic evidence. In this research, we present methods for collecting forensic evidence of timestamps and folder names relating to a folder's creation. In some of the related log records for creating a folder, four log records that have timestamps and folder name information that are 0x0E/0x0F(Redo/Undo op. code), 0x02/0x00, 0x08/0x00, and 0x14/0x14 were analyzed. Unfortunately, the structure of $LogFile is not well known or documented. As a result the researchers used reverse engineering in order to gain a better understanding of the log record structures. The study found that using basic information contained in the $LogFile, a forensic reconstruction of timestamp events could be created.

**Keywords:** computer forensics, timestamp, $LogFile, NTFS.

## 1 Introduction

The NTFS file system supports journaling in order to improve its reliability. Journaling is an advanced file system integrity feature that is not well exploited by most digital forensic tools. This feature is employed in virtually all modern file systems, including NTFS (Windows NT/2000/XP), HFSJ (Mac OS X), ext3 (Linux) and ReiserFS (Linux)[1]. The logged journal files have numerous log records that operate on their file systems. These records leave traces of evidence that can be of forensic importance.

Despite the importance of a journal file as a forensic evidence repository, its structure ($LogFile is a journal file name in NTFS) is not well documented[1,2]. It is therefore necessary to develop an understanding of the $LogFile in order to determine its significance. A solution to obtaining a more thorough knowledge of NTFS is

* Visiting scholar at Dept. of Computer & Information Technology, Purdue University.

P. Gladyshev and M.K. Rogers (Eds.): ICDF2C 2011, LNICST 88, pp. 211–225, 2012.

through reverse engineering. There has been some research related to the NTFS $LogFile. K. Dreher[3] wrote a thesis on NTFS in which he/she collected information about NTFS from previously written material and his/her experiments. The research was very helpful, but the paper included some incorrect analyses and unknowns. P. Singireddy[4] provided research on the category of log operation, that described the actions of the log operation. The author[5] used reverse engineering and showed that certain data areas of some log records corresponded to operations on files, and the reference in [6] showed that some series and structure of the log record corresponded to a file created operation.

Date and time evidence is a fundamental part of many forensic computing examinations[14]. Forensic examiners know that lawyers are often drawn to dates and times because they represent a concrete link between the real world and the less easily understood world of computer evidence[13]. Casey has indicated that MAC time analyses is necessary for the proper reconstruction of digital events[12]. Boyd and Forster discussed time structure and their use in Microsoft Internet Explorer with local and UTC time translation issues[13]. And they mentioned that experienced examiners are reluctant to draw their conclusions solely relying on the date and time information of a particular file, because such information contains many potential pitfalls[13, 14]. Chow et. al. conducted further research demonstrating the behavioral characteristics of MAC times on NTFS file system. This previous research provides the validation basis for the temporal analysis in event reconstruction models [14].

This paper will cover methods for collecting forensic evidence of timestamps and folder names corresponding to a folder creation in the $LogFile of the Windows NTFS file system. In section 2, we discuss an overview of $LogFile and structure of the log records. In section 3, we present a method to link the series of log records, in section 3.1, and we analyze four log records, i.e., 0x0E/0x0F, 0x02/0x00, 0x08/0x00, and 0x14/0x14, that have timestamps and folder name information in section 3.2. Finally, in section, we provide some concluding remarks.

## 2 Configuration of Journal File : $LogFile

### 2.1 Overview

In the event of a disk failure, NTFS runs a recovery procedure in order to restore the system to and maintain the consistency. NTFS guarantees that the file system is restored to a clean state [1,2]. The sequences of operations are recorded in the journal file. The journal file for the Windows operating system is called $LogFile. The $LogFile is used to recover from system crashes and unexpected conditions. NTFS is a transactional, or journaling, file system. It logs all file system metadata changes to the log file before attempting to make the changes. The log files contain redo and undo information used to recover from a system crash and maintain file system consistency[1,9].

The structure of $LogFile is not opened officially. The basic structure of $LogFile described below comes from some introductory literature[8,9,10]. And with the author's time-consuming reverse engineering. A significant portion of the structure of log record operations was derived from reference [5, 6]. In this section, we are going

to introduce essential portions of the structure to have a basic understanding of the proposed method.

The $LogFile consists of two parts: one is the restart area, the other is the logging area as shown in Fig. 1. The restart area contains information on how to start the recovery after a system failure. The information needed to recover the file system is stored in the logging area, which contains references to where in the logging area the recovery of the file system should start after a system failure[3]. The logging area contains two copies (for redundancy) of information describing the state of the log and a pointer for the last transaction that was known to be successful. The logging area also contains transaction records.

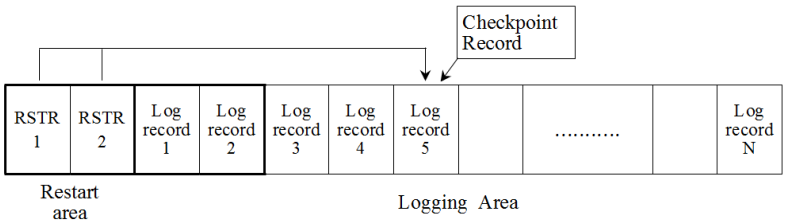


**Fig. 1.** Configuration of a journal file($LogFile): It consists of restart area and logging area. The restart area has two RSTRs and it contains information on how to start the recovery after a system failure. The logging area has a numerous log records to record transactions. It has two types of log record, i.e., a checkpoint record and an update record.

Important two types of log records are an update record and a checkpoint record. The update record is the most common record type. Each update record contains redo and undo attribute. The checkpoint record identifies where in the log file the OS should start from if it needs to verify the file system. The checkpoint records are periodically written to the $LogFile. The checkpoint records knows how far back the in the $LogFile a recovery of the file system must start. The LSN of the most recent checkpoint record is stored in the restart area for quick access upon recovery[3].

According to our observations, “Log record 1” and “Log record 2” are used for retaining the last page log record that is currently working in a memory. They have identical data. The last page is stored in the first two of the log record, temporarily, when the log record page is not fully paged up a 4KB-sized page. The last log record page is stored to the location which is written in the Last LSN(Logical Sequence Number) attribute of both the “Log record1” and “Log record2”. Only Log record1 and Log record2 are having that offset address. Except these two log records, the other log records use Last LSN attribute as its original purpose. If the system shuts down normally, the log record is stored at the last log record although the page is not fully paged up. The remaining empty part is padded at next operation.

## 2.2 Restart Area and Log Record Structure

The restart area is composed of three tables, which are the 1) restart page header, 2) restart area, and 3) log client[3]. The restart page header is 0x30 bytes in size and contains 10 attributes. The restart area follows right after restart header. The size of restart record is 0x2C bytes, and followed by 0x14 bytes reserved blank data. Total allocated size for this area is 0x30 bytes. The log client record has length of 0xA0 in bytes. This follows restart area. On Windows XP or later, this record usually starts from the offset at 0x70(Fig. 2).

The major role of the Restart Area is to hold information about the $LogFile. Forensically, LSN is the most important attribute. “Check Disk LSN” is used when the system shuts down abnormally for any reasons. The LSN is the start point for a disk recovery. “Current LSN” means the log record transacted lastly. “Client Restart LSN” has the same LSN as the “Current LSN”.

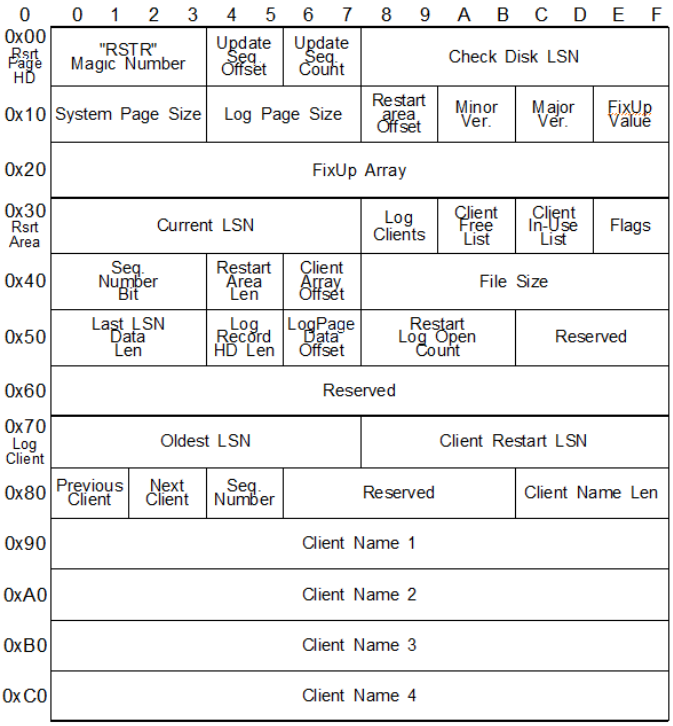


**Fig. 2.** Structures of Restart area: This area composed of three tables- the 1)restart page header begins at the offset 0x00, 2) restart area begins at the offset 0x30, and 3) log client locates at the offset 0x70.

### 2.3 Log Record Structure

The 4KB-sized log record page contains several log records. This has a fixed size header at the start location of each page with the magic number “RCRD”. The record page header is 0x40 bytes in size. 12 attributes are included in the structure (Fig. 3).

“Last LSN” Last LSN/File Offset(0x08, 8) is union of last LSN and the file offset. The “Last LSN” means the last record entry in this page. It is extended to the next log record page in a usual case. The File offset is used only where the first and the second log record as described in section 2.1. “Next Record Offset”(0x18, 2) is the offset from the beginning of the 4KB-sized log record to the last LSN is located in a page. “Last End LSN”(0x20, 8) is the last LSN that ends up logging fully in the log record, not extending to next page.

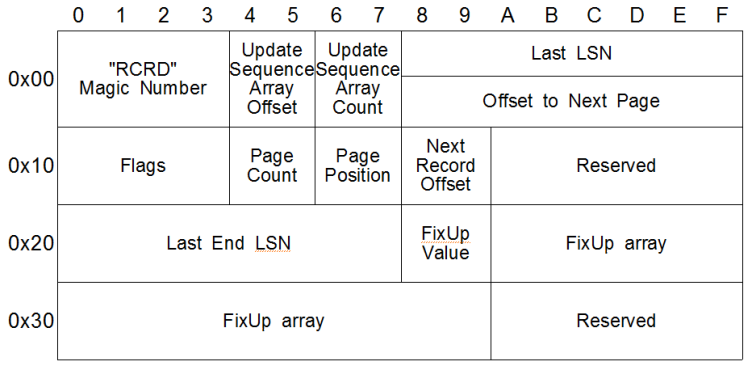


**Fig. 3.** Log record page header: The log record has fixed size header at start location of each page with the magic number “RCRD” at the offset 0x00. One of the important attribute is the “Last LSN” which is used for the last record entry in this page and the “Last End LSN” is for ending up logging fully in the log record.

The structure log record contains 18 attributes as in Fig. 4. “This LSN” is for LSN number for this log record. “Previous LSN” is located just before “This LSN”. “Undo Next LSN” is used for undoing log records, it is located before “This LSN” which acts as a backward link. The Data Length corresponds to “This LSN”. These are incremented by three LSNs which is used to represent sequential log record operations and related log operations between each LSNs. These LSNs are key elements to required solve NTFS forensic problems.

Sequence Number is set to a zero every time the $LogFile is restarted and it is incremented when the $LogFile is closed. The Client Index is always 1 for log record.

Record Type represents type of record that is either a transaction record/table or a checkpoint record. If the value is 1, this means the log record is a transaction record, if the value 2, then the log record is a checkpoint record. The Transaction ID represents the transaction ID for the log record. If a Flags has a value 1 it means the log record extends to the next log record page, otherwise it has 0.

The “Redo Operation” code represents log operation code for the redo operation. The “Undo Operation” code represents log operation code for undoing operations

when an abnormal shut down occurred. The “Redo Operation Offset” is an offset to redo the operation contents. “Redo Operation Length” corresponds to the length of the redo operation. “Undo Operation Offset” is an offset to the undo operation contents. “Undo Operation Length” corresponds to the length of the undo operation. These two operation codes and the LSN numbers are essential attributes for our method. Target Attribute is for the target attribute. “LCNs to Follow” means if associated records are to follow this record, the attributed set to 1. Data have different contents depending on redo and undo op codes.

The size of the “Data area” is directly dependent on the type of operation code. Every type of operation code has their specific data. In this paper, the proposed method uses the data in that area.

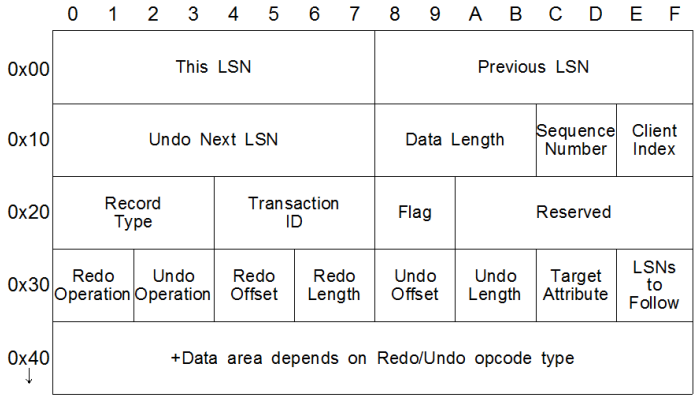


**Fig. 4.** General structure of log record: Data area of the log record has large amount of information, so it is important to analyze the log record. The contents of data is depends on the “Redo” operation code starts at the offset 0x30 in bytes and “Undo” operation code starts at the offset 0x32 in bytes. “This LSN” is a unique identification number of a log record. The “Previous LSN” indicates the just before the log record and it acts as a backward link. The “Undo Next LSN” is indicates the next backward log record for undoing log reocrd in the event of failure.

### 2.4 Log Operation Types

Table 1 shows codes for the log operation types[4]. The reference [4] is an informal document. But the lecture slide contains the log operation types, so it helpful for doing reverse engineering work. One of the operation codes is used in attributes of “Redo Operation” and “Undo Operation”. A series of log records for these codes is written to the $LogFile when a file operation is performed such as 1) Creating a file 2) Deleting a file 3) Extending a file 4) Truncating a file 5) Setting file information 6) Renaming a file 7) Changing the security applied to a file[1]. Whenever a Windows file operation is performed, we can find some regular patterns in a series of log records in $LogFile. These patterns are useful in order find evidences of what happened on the disk[3]. The author explained in reference [5] that a series of log

record forms a pattern for log record, so it can be used in the reconstruction of file operations and can include unchangeable time evidences.

We use the composite of operation code such as 0x0E/0x0F, 0x02/0x00, 0x08/0x00, and 0x14/0x14. The first part is for "Redo" operation code and the second part is for Undo operation code. These codes are unique, so we use these operation code composites for the name of log record operation.

**Table 1.** Log Operation Codes

| Log operation type | Op code |
|---|---|
| No Operation | 0x00 |
| Compensation Log Record | 0x01 |
| Initialize File Record Segment | 0x02 |
| Deallocate File Record Segment | 0x03 |
| Write End of File Record Segment | 0x04 |
| Create Attribute | 0x05 |
| Delete Attribute | 0x06 |
| Update Resident Value | 0x07 |
| Update Nonresident Value | 0x08 |
| Update Mapping Pairs | 0x09 |
| Delete Dirty Clusters | 0x0A |
| Set New Attribute Sizes | 0x0B |
| Add Index Entry Root | 0x0C |
| Delete Index Entry Root | 0x0D |
| Add Index Entry Allocation | 0x0E |
| Delete Index Entry Allocation | 0x0F |
| Set Index Entry VCN Allocation | 0x12 |
| Update File Name Root | 0x13 |
| Update File Name Allocation | 0x14 |
| Set Bits in Nonresident Bitmap | 0x15 |
| Clear Bits in Nonresident Bitmap | 0x16 |

## 3 Analyses on Log Records for Folder Creation

### 3.1 Series of Log Records

When we create a folder, numbers of log records are written in the $LogFile. Related log records for creating folder are as follows:

Series 1: 15/16→00/03→0e/0f→0e/0f→02/00→0b/0b→08/00→0b/0b→1b/01
Series 2: 07/07→14/14→1b/01
Series 3: 0b/0b→08/00→0b/0b→07/07→1b/01

The example of the log record series is composed of three series of log records. These are created by using the "mkdir NewDirectory" command in a command prompt window. If we create the same folder in an explorer window (due to the process including change in a folder name, the result is slightly different. There is a

0x1b/0x01 log record at the end of a series of log records. The opcode 0x1b means "Forget Transaction" and the opcode "0x01" means "Compensation Log" as listed in Table 1. Other file operations such as deletion, copy, move etc. have several series of log records like the creation operation.

To explain the log record series, we take an example such as "Series 2". The 0x07/0x07 log record is the first log record has the LSN number of 0x2F3B9656 and the "Previous LSN" and the "Undo LSN" have numbers of zero, respectively. The two zeros mean NULL links, so it has no backward log records links (Fig. 5).

The 0x14/0x14 log record is next to the 0x07/0x07 log record. The "LSN" of this log record is 0x2F3B9671. The number of the "Previous LSN" and the number of the "Undo LSN" have the same LSN number of 0x2F3B9656. This means that the previous record of this log record is the one with the LSN number of 0x2F3B9656. A log record with numbers of the "Previous LSN" and the "Undo LSN", generally, has the two LSN numbers the same (Fig. 6).

The 0x1B/01 log record has two LSN numbers in the "LSN" attribute of 0x2F3B968A and the "Previous LSN" attribute of 0x2F3B9671. In the third LSN attribute of the "Undo LSN" there is no LSN number. This log record means the end of a series of log records.

```
01DCB2B0  56 96 3B 2F 00 00 00 00  00 00 00 00 00 00 00 00  V▌;/............
01DCB2C0  00 00 00 00 00 00 00 00  A8 00 00 00 00 00 00 00  ........¨.......
01DCB2D0  01 00 00 00 18 00 00 00  00 00 00 00 00 00 00 00  ................
01DCB2E0  07 00 07 00 28 00 3F 00  68 00 3F 00 18 00 01 00  ....(.?.h.?.....
01DCB2F0  38 00 21 00 02 00 02 00  19 59 00 00 00 00 00 00  8.!......Y......
01DCB300  C8 06 EE 00 00 00 00 00  4A 0E C0 02 26 CC 01 F0  È.î.....J.À.&Ì.ð
```

**Fig. 5.** The 0x07/07 log record has only one LSN number in the "LSN" attribute. The other two LSN attribute have zero LSN. The log record stands for the first log record of one of the log record series.

```
01DCB380  94 D2 03 00 00 00 00 00  71 96 3B 2F 00 00 00 00  ▌Ò......q▌;/....
01DCB390  56 96 3B 2F 00 00 00 00  56 96 3B 2F 00 00 00 00  V▌;/....V▌;/....
01DCB3A0  98 00 00 00 00 00 00 00  01 00 00 00 18 00 00 00  ▌...............
01DCB3B0  00 00 00 00 00 00 00 00  14 00 14 00 28 00 38 00  ............(.8.
01DCB3C0  60 00 38 00 44 00 01 00  00 00 10 05 00 00 08 00  `.8.D...........
01DCB3D0  06 00 00 00 00 00 00 00  00 09 03 01 00 00 00 00  ................
```

**Fig. 6.** The 0x14/14 log record has three LSN numbers. The first one is "LSN" attribute, the second one is the "Previous LSN" attribute, and the third one is "Undo LSN" attribute.

```
01DCB450  8A 96 3B 2F 00 00 00 00  71 96 3B 2F 00 00 00 00  ▌▌;/....q▌;/....
01DCB460  00 00 00 00 00 00 00 00  28 00 00 00 00 00 00 00  ........(.......
01DCB470  01 00 00 00 18 00 00 00  00 00 00 00 00 00 00 00  ................
01DCB480  1B 00 01 00 28 00 00 00  28 00 00 00 18 00 00 00  ....(...(.......
01DCB490  00 00 00 00 00 00 02 00  00 00 00 00 00 00 00 00  ................
01DCB4A0  68 C9 8A 84 00 00 00 00  95 96 3B 2F 00 00 00 00  hÉ▌▌....▌▌;/....
```

**Fig. 7.** The 0x1B/01 log record has two LSN number in the "LSN" attribute and the "Previous LSN" attribute. It has no LSN number in the "Undo LSN" attribute. It is the last log record of a series of log records.

### 3.2 Acquiring Forensic Information about a Folder Name and Timestamps in Log Records

Windows supports both long and short file names in NTFS file system. The $FILE_NAME attribute(0x30) is used to store the file's name and parent directory information in an MFT entry in $MFT metafile, and is used in a directory index. The structure is shown in Fig. 8. The $FILE_NAME attribute is always a resident attribute, and will be the second attribute.

When we create a long file name, NTFS creates a second file entry that has a 8.3 DOS format file name. A long file name is called Win32 name, and a short file name is called DOS name. The Win32 name is case insensitive and allows Unicode characters except for special characters such as '/', '\', ':','>','<', and '?'. And the file name can contain spaces, multiple periods, and special characters that are not allowed with DOS file names. The DOS name is case insensitive, upper case, and no special characters. The name contains eight or fewer characters and has a file name extension containing three or less in the extension which are separated by a period[2].

In the Fig. 8, the namespace byte is used to identify the name rule. The value 0 means the name space uses POSIX, the value 1 means Win32, the value 2 means DOS, and the value 3 means Win32 & DOS[2]. If a file's name has over 8 characters, the file's name space will be the value 3(Win32 & DOS), which has both 8.3 file name and the long file name. But, if a file's name is less than 8 characters, it has only the DOS name.

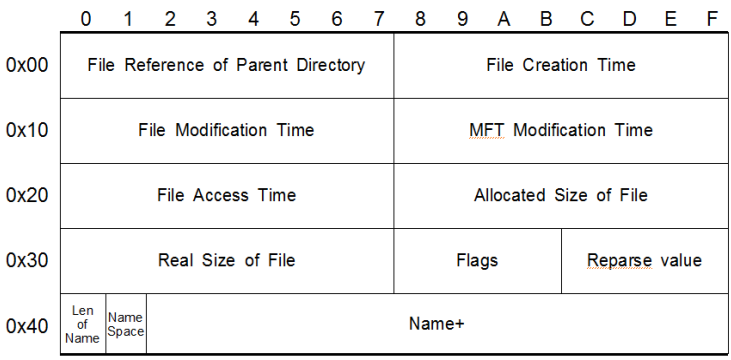


**Fig. 8.** This is the $FILE_NAME attribute structure, which is used to store the file's name and parent directory information in an MFT entry, and is used in a directory index. The name space byte has one of 0(POSIX), 1(Win32), 2(DOS) and 3(Win32 & DOS) value.

There are four kinds of log records having a folder/file name in a folder; 0x0E/0x0F(add index entry allocation), 0x02/0x00(initialize file record segment), 0x08/0x00(update non-resident value), and 0x14/0x14(update a file name allocation). In this case, the log record 0e/0f is used for newly adding index entry, the log record 0x02/0x00 is used for a new file allocation with the initial value of the MFT entry,

and the log record 0x08/0x00 is used for adding the USN journal log to the $UsnJrnl file, and the log record 0x14/0x14 is used for updating timestamps of the parent folder of a directory. We explain getting the folder name and timestamps from the four log records in detail as follows:

**0x0e/0x0f Log Record.** This 0x0e/0x0f log record is used for adding/deleting index information of a file or a folder in the index entry. It has a "Redo" and a "Undo" data from offset 0x58 bytes. The data size is dependent upon file name length. In the case of a file or folder whose name is below 8 characters, this log record shows up once in the series of the log record, but in the case of long file name, this log record is written twice in a row; one is for DOS name and the other is for Win32 name. The data structure for "Redo" is depicted in the Fig. 9. There is, generally, no data for the "Undo" data part. In a directory index entry, it only includes a $FILE_NAME attribute(Fig. 8). So, the length of this log record is dependent upon a file or a folder name.

The log record 0x0e/0x0f for a folder named "NewDirectory" is shown in Fig. 10. The "Redo" data of the log record begins at 0x58 bytes offset with a 0x70 bytes long. "Undo" begins at 0x98 bytes offset but it has no actual data. The byte 0x0C marked "A" means the length of name string, i.e. it has a 12 byte-long folder name. The byte 0x01 marked "B" means the namespace, which stands for Win32 namespace. The long folder name "NewDirectory" begins at the place marked "C". Timestamps for this record, marked with "T1"-"T4", have the same time value, which is the folder creation.

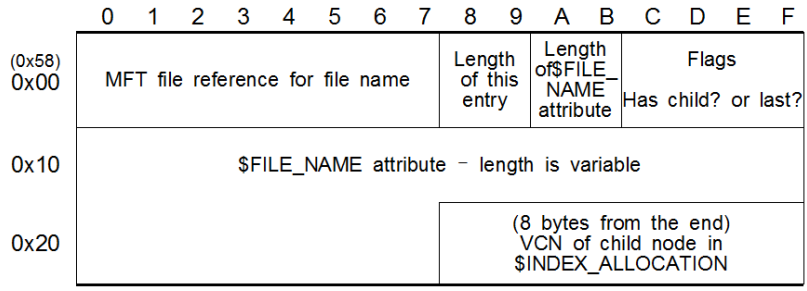


**Fig. 9.** This is a structure of a directory index entry. This is included in 0x0e/0x0f log record at 0x58 bytes offset. A $FILE_NAME attribute structure is in this structure. Length of this structure is depended upon the length of a file or folder name.

The folder name "NewDirectory" has another name and is converted to a short name "NewDir~1", which is called the DOS name as shown in Fig. 11. It has a same log records as shown in Fig. 10, but the only difference is the name. "Undo" begins at 0x98 bytes offset, as well. The byte 0x08 marked "A" indicates the length of the name string, i.e. it has an 8 byte-long folder name. The byte 0x02 marked "B" means namespace, which stands for DOS namespace. The short folder name with "NewDir~1" begins at the location marked "C". Timestamps marked with "T1"-"T4" are "Creation", "Write", "MFT modified", and "Access" time, respectively. The timestamps are always the same as the log record for "Win32".

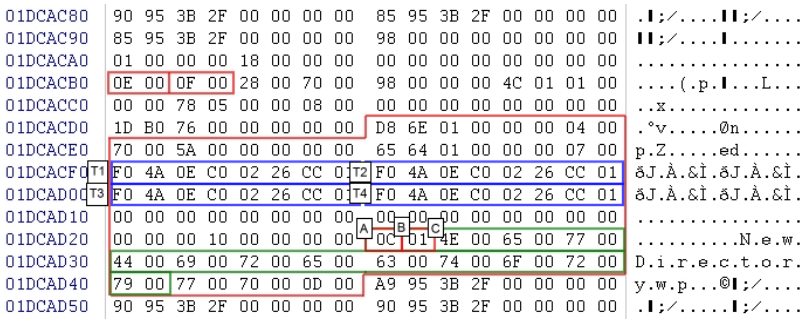


**Fig. 10. 0x0E/0x0F log record for Win32 namespace:** This log record has "Redo" data at 0x58 bytes offset, which has 0x70 bytes data length. "Undo" begins at 0x98 bytes offset but it has no actual data. The byte 0x0C marked "A" means length of name string, i.e. it has a 12 byte-long folder name. The byte 0x01 marked "B" means namespace, which stands for Win32 namespace. The long folder name "NewDirectory" begins at the location marked "C".

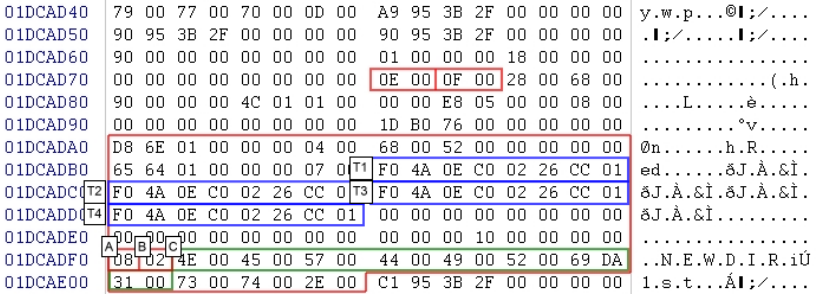


**Fig. 11. 0x0E/0x0F log record for DOS namespace**: This log record has "Redo" data at 0x58 bytes offset, which has 0x68 bytes data length. "Undo" begins at 0x90 bytes offset but it has no actual data. The byte 0x08 marked "A" means length of name string, i.e. it has an 8 byte-long folder name. The byte 0x02 marked "B" means namespace, which stands for DOS namespace. The short folder name "NewDir~1" begins at the location marked "C".

**0x02/0x00 Log Record.** 0x02 log record is used for initializing the file record segment. The "Redo" data of the log record has an initial MFT entry of the newly created file or folder allocated in $MFT metafile. We should keep in mind that it does not contain the final MFT entry information but only the initial MFT entry information. Therefore, a few attributes of the contents are subject to changed. In Fig. 12, the folder creation process has several steps, so the LSN and USN are changed after the creation of a folder. Even though, in this case, timestamps are not changed after creating a folder, when it comes to a file creation, the timestamp of the "MFT modified" time should change slightly. How the other timestamps are changed depends on the application.

**0x08/0x00 Log Record.** This 0x08 log record is used for updating the value of the non-resident attribute. In this instance, the log record is used for adding new

USN(Update Sequence Number) journal log into the $UsnJrrnl metafile. The $Data attribute name is $J, is sparse and non-resident. The $UsnJrnl is a file for change journaling, and it contains a list of the files that have changed. It is used to quickly identify the files that have been changed in a certain time frame, instead of looking at each file individually. Each record contains a file name, the time of change, and the type of change. The USN, (64-bits in size), is used to index the records in the journal, and the number is stored in the $STANDARD_INFORMATION attribute of the file that was modified lastly[2].

The “Redo” data of the log recordis0x58 bytes long, and has a single record for an entry for $J as shown in Fig. 13. We can check the record in the $UsnJrnl:$J as in when a folder is created. In this case two change journal entries are written in the $UsnJrnl:$J as in Fig. 14. The eight bytes long data marked with “T1” corresponds to the time for creating a folder. The four bytes long data marked with “C1” corresponds to the value for the change type filed in $J entry. The value of “0x00000100” means “The file or directory was created” marked as “C1” in Fig 13 and Fig. 14, and the value of “0x80000100” means “The file or directory was closed” marked with “C2” in Fig.14.

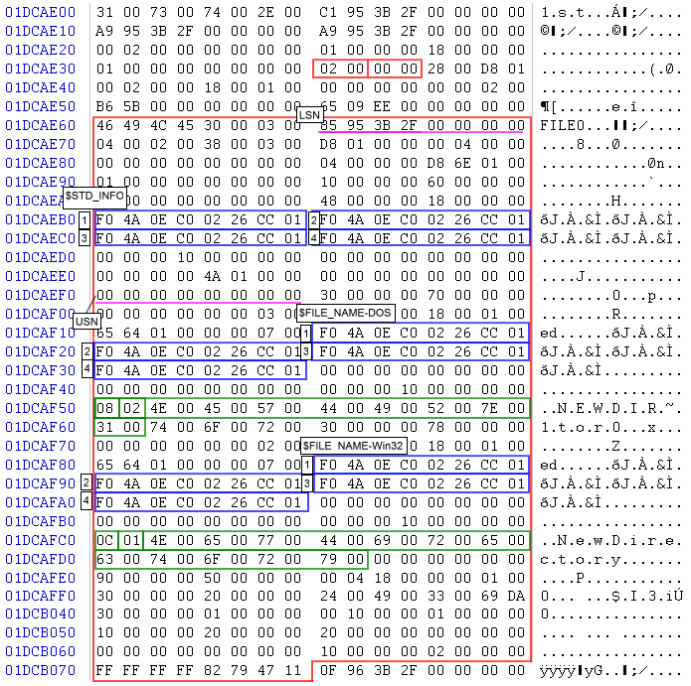


**Fig. 12. 0x02/0x00 log** This log record has “Redo” data at 0x58 bytes offset, which has 0x01D8 bytes data length. “Undo” begins at 0x0200 bytes offset but it has no actual data. It has an initial MFT entry of newly created file or folder allocated in $MFT metafile.

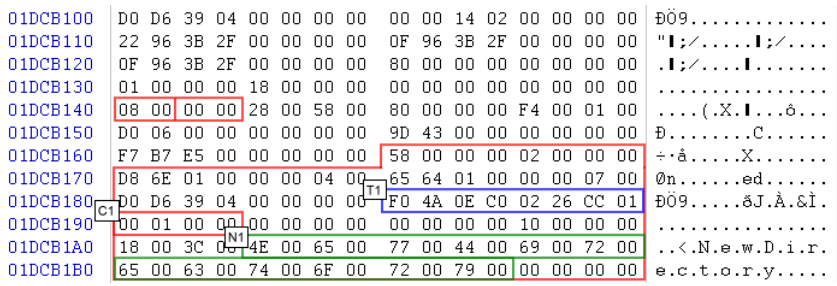


**Fig. 13. 0x08/0x00 log record for a change journal** This is the USN journal log entries for the folder creation of "NewDirecotry". "Redo" data has a single change journal entry and it has 0x58 bytes long. The location marked with "T1" indicates the timestamp for creating folder time, "C1" is a change type field, and "N1" is the name of the folder.

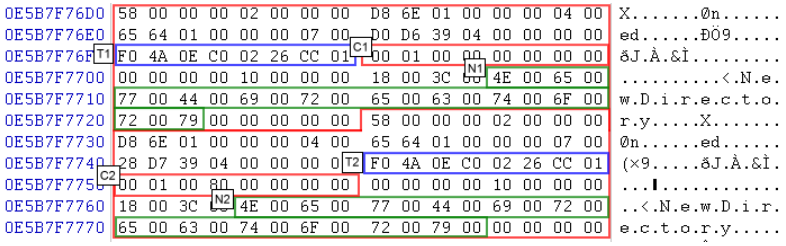


**Fig. 14. $UsnJrnl:$J** This is USN journal log entries for the folder creation of "NewDirecotry". The data part log record has "Redo" data at 0x58 bytes offset, which is 0x01D8 bytes of data length. "Undo" begins at 0x0200 bytes offset but it has no actual data. It has an initial MFT entry of the newly created file or folder allocated in $MFT.

**0x14/0x14 Log Record.** This 0x14/0x14 log record is used for updating the file name allocation. In this case, "Redo" opcode and "Undo" opcode use the same 0x14 code. "Redo" and "Undo" data are 0x38 bytes long. "Redo" data has four timestamps marked with "T1"-"T4", and "Undo" data has four timestamps marked with "T5"-"T8", respectively. The four timestamps are not related to the "NewDirecotry" folder, but to their parent folder named "TStamp". The 8 bytes data marked with "F1" is the MFT number of the "TStamp" folder. "F2" means parent of the "TStamp", which is actually the root. Timestamps marked with "T1"-"T4" are located in the $STANDARD_INFORMATION attribute in the MFT entry of the "TStamp" folder, which are changed from the timestamps marked with "T5"-"T8". The reason for the timestamp changes is due to the creation of the "NewDirectory" in the "TStamp" folder.

From a computer forensic perspective, we should keep in mind that if there are any operations made on a file or a folder in the folder, these cause the timestamps change to the time of the current operation time. Among the four timestamps, three timestamps are changed to the current operation time simultaneously, but the "Creation" time retains its original creation time unchanged. In the Fig 15, we can see the changes for the timestamps. There is no change in the current "Creation" time

which is “T1” and the past “Creation” time “T5”. The other three timestamps are changed from “T6”, “T7”, and “T8” to “T2”, “T3”, and “T4”,

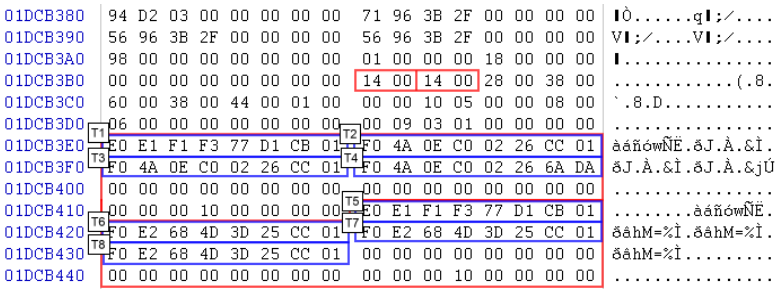


**Fig. 15. 0x14/0x14 log record** This log record has “Redo” data 0x58 bytes long, which has four timestamps of the parent folder “TStamp” of “NewDirectory”. The “Undo” data has four timestamps written before the creation of the folder operated on.

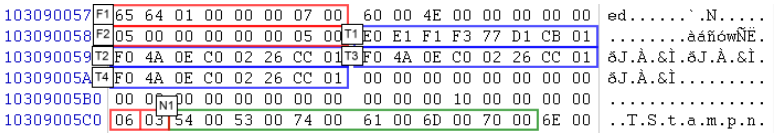


**Fig. 16. Index entry for the parent of “NewDirectory” folder** This index entry is for the parent(“TStamp”) of the “NewDirectory” folder. Any file operations are performed in the “TStamp” folder, The timestamps are changed to the operation time, except for the “Creation” time.

## 4 Conclusion

In this research, we presented methods for collecting forensic evidence from timestamps and folder names relating to a folder’s creation. Among a number of related log records for creating a folder, we analyzed only four log records that have timestamps and folder name information; 0x0E/0x0F, 0x02/0x00, 0x08/0x00, and 0x14/0x14. Unfortunately, the structure of $LogFile is not well known or documented. Therefore the authors used reverse engineering in order to gain a better understanding of the $LogFile structure.

The log record 0x0E/0x0F contains the $FILE_NAME attribute, so we located the folder name and four timestamps, i.e. “Creation”, “Write”, “MFT entry modified”, and “Access” time. In case where the name is over 8 characters, the log record has two log record named as “DOS” and “Win32” format, respectively. In the log record 0x02/0x00, we could find complete MTF entry contents except for non-resident data. However, we should, be aware that the contents of the log record are not up to date, and that some data might have additional updates. The log record 0x08/0x00 is used for writing to the change journal in the $UsnJrnl file. But, the change journal does not leave behind contents, but it does contain records of file operations that have occurred. Using information from the log record in conjunction with the $LogFile, we

are able to forensically reconstruct events. And finally we found that the log record 0x14/0x14, has timestamp that are used for the $STANDARA_INFORMATION attribute of parents of the folder. The moment for file operations in this folder is written in the timestamps of the folder to the time of the current operation time.

This research has demonstrated that basic information and knowledge about log records and the $LogFile can greatly assist in a forensic investigation. However further research is required in order to complete a more detailed forensic reconstruction using the NTFS file system.

## References


1. Russinovich, M.E., Solomon, D.A.: Microsoft Windows Internals, 4th edn., pp. 733–774. Microsoft Press (2005)
2. Carrier, B.: File System Forensic Analysis, pp. 273–396. Addison-Wesley (2005)
3. Dreher, K.: NTFS. Master Thesis of Department of Information Technology Institute of technology, Lund, Sweden (November 1998)
4. Singireddy, P.: Recoverability Support in NT File System (NTFS), http://www.eas.asu.edu/~cse532/ or http://www.docstoc.com/docs/28691891/ntfs_mod/
5. Cho, G.S.: An Analysis of NTFS Journal File for a Computer Forensic. Digital Forensic Research 3(1), 51–60 (2009)
6. Kim, T.H., Cho, G.S.: A Digital Forensic Method for File Creation using Journal File of NTFS. Journal of KSDIM 6(2), 107–118 (2010)
7. Data Integrity and Recoverability with NTFS, http://www.ntfs.com
8. Transaction log supports NTFS recoverability, http://support.microsoft.com/kb/101670
9. NTFS Documentation, http://www.linux-ntfs.org
10. Russon, R.: NTFS Documentation (2009), http://www.linux-ntfs.org
11. Naik, D.C.: Inside Windows Storage, ch. 6.5. Addison Wesley (July 2003)
12. Casey, E.: Uncertainty and Loss in Digital Evidence. International Journal of Digital Evidence 1(2) (Summer 2002)
13. Boyd, C., Forster, P.: Time and Date Issues in Forensic Computing – A Case Study. Digital Investigation 1(1), 18–23 (2004)
14. Chow, K.P., et al.: The Rules of Time on NTFS File System. In: SADFE, pp. 71–85 (March 2007)