libyal / **libfsntfs**

Code Issues 8 Pull requests 3 Actions Projects

Applied updates and worked on format support

+4,151 −902

joachimmetz committed on Jul 6, 2023

libfsntfs / documentation / New Technologies File System (NTFS).asciidoc

joachimmetz Applied updates and worked on format support 5d479d8 · 2 years ago

3674 lines (2924 loc) · 138 KB

Preview Code Blame Raw

New Technologies File System (NTFS)

Summary

NTFS is the primary file system for Microsoft Windows versions that are based on Windows NT. This specification is based on publicly available work on the format and was enhanced by analyzing test data.

This document is intended as a working document of the data format specification for the libfsntfs project.

Document information

Author(s):	Joachim Metz <joachim.metz@gmail.com>
Abstract:	This document contains information about the New Technologies File System (NTFS)

Classification:	Public
Keywords:	New Technologies File System, NTFS

License

```
Copyright (C) 2009-2023, Joachim Metz <joachim.metz@gmail.com>.
Permission is granted to copy, distribute and/or modify this document under the
terms of the GNU Free Documentation License, Version 1.3 or any later version
published by the Free Software Foundation; with no Invariant Sections, no
Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included
in the section entitled "GNU Free Documentation License".
```

Revision history

Version	Author	Date	Comments
0.0.1	J.B. Metz	August 2009 September 2010 October 2010 November 2010 December 2010	Initial version, based on earlier notes.
0.0.2	J.B. Metz	March 2011	Additional information about multi data run MFT.

Version	Author	Date	Comments
0.0.3	J.B. Metz	May 2012	Additional information.
0.0.4	J.B. Metz	February 2014	Additional information.
0.0.5	J.B. Metz	July 2014 August 2014	Textual changes and additional information.
0.0.6	J.B. Metz	March 2015	Additional information and switched to asciidoc format.
0.0.7	J.B. Metz	May 2015	Additional information.
0.0.8	J.B. Metz	July 2015	Textual changes.
0.0.9	J.B. Metz	August 2015	Additional information.
0.0.10	J.B. Metz	August 2015	Additional information regarding index and security descriptors.
0.0.11	J.B. Metz	January 2016	Additional information about common alternate data stream.
0.0.12	J.B. Metz	January 2016	Additional information about extended attributes.
0.0.13	J.B. Metz	March 2016	Small changes.

Version	Author	Date	Comments
0.0.14	J.B. Metz	April 2016	Small changes.
0.0.15	J.B. Metz	June 2018	Corrected typos.
0.0.16	J.B. Metz	January 2019	Changes to format.
0.0.17	J.B. Metz	August 2019	Additional information regarding volumes created by Windows 10 (1903) and mkntfs.
0.0.18	J.B. Metz	September 2019	Additional information regarding Windows Overlay Filter (WOF) compressed data.
0.0.19	J.B. Metz	November 2019	Additional information regarding short file names.
0.0.20	J.B. Metz	January 2020	Additional information regarding $STANDARD_INFORMATION and $ATTRIBUTE_LIST attributes.
0.0.21	J.B. Metz	April 2020	Additional information regarding compressed data runs.
0.0.22	J.B. Metz	May 2020	Additional information regarding LZNT1 compressed block corruption scenarios.
0.0.23	J.B. Metz	December 2020	Additional information regarding reparse point tag values.
0.0.24	J.B. Metz	October 2021	Small changes.

Version	Author	Date	Comments
0.0.25	J.B. Metz	February 2022	Additional information regarding index entry size.
0.0.26	J.B. Metz	December 2022	Additional information regarding $VOLUME_INFORMATION flags.
0.0.27	J.B. Metz	July 2023	Additional information file name UTF-16 strings.

1. Overview

NTFS is the primary file system for Microsoft Windows versions that are based on Windows NT.

TODO describe relation to OS2 HPFS

Characteristics	Description
Byte order	little-endian
Date and time values	FILETIME in UTC
Character strings	ASCII strings are Single Byte Character (SBC) or Multi Byte Character (MBC) string stored with a codepage. Sometimes referred to as ANSI string representation. Though technically maybe incorrect, this document will use term (extended) ASCII string. Unicode strings are stored in UTF-16 little-endian without the byte order mark (BOM).

1.1. Versions

There are multiple version of NTFS.

NTFS version	Remarks
1.0	Introduced in Windows NT 3.1
1.1	Introduced in Windows NT 3.5, also seen to be used by Windows NT 3.1
1.2	Introduced in Windows NT 3.51
3.0	Introduced in Windows 2000
3.1	Introduced in Windows XP

Need test images for version 1.0 and 1.1

Note	The versions mentioned above are the version as used by NTFS. Another common versioning schema uses the Windows version, e.g. NTFS 5.0 is the version of NTFS used on Windows XP which is version 3.1 in schema mentioned above.

1.2. Test version

The following version of programs were used to test the information within this document:

- Windows NT 3.1 (version 1.1)
- Windows NT4 (version 1.2)
- Windows 2000 (version 3.0)
- Windows XP SP3 (version 3.1)
- Windows 2003 (version 3.1)
- Windows Vista (version 3.1)
- Windows 2008 (version 3.1)

- Windows 7 (version 3.1)
- Windows 8 (version 3.1)
- Windows 10 (1809, 1903) (version 3.1)
- Windows 11 (21H2) (version 3.1)
- NTFS-3G

Note	Windows 10 (1809) has been observed to use NTFS version 1.2 for 64k cluster block size.

2. Terminology

2.1. Cluster

NTFS refers to it file system blocks as clusters. Note that these are not the same as the physical clusters of a harddisk. For clarity these are referred to as cluster blocks. In other sources they are also referred to as logical clusters which are numbered globally (or absolute).

Typically the cluster block is 8 sectors (8 x 512 = 4096 bytes) of size.

2.2. Virtual cluster

The term virtual cluster refers to cluster blocks which are numbered locally (or relative).

2.3. Long and short (file) name

In Windows terminology the name of a file (or directory) can either be short or long. The short name is an equivalent of the filename in the (DOS) 8.3 format. The long name is actual the (full) name of the file. The term long refers to the aspect that the name is longer than the short variant. Because most documentation refer to the (full) name as the long name, for clarity sake so will this document.

3. The volume

Everything on an NTFS volume is a file. There are two types of files:

- files that contain volume and file system metadata (referred to as metadata files);
- files that contain data (referred to as files).

3.1. The metadata files

NTFS uses the Master File Table (MFT) to store information about files and directories. The MFT entries reference the different volume and file system metadata. There are several predefined metadata files.

The following metadata files are predefined and use a fixed MFT entry index.

MFT entry index	Filename	Description
0	$MFT	Master File Table
1	$MFTMirr	Back up of the first 4 entries of the Master File Table
2	$LogFile	Metadata transaction journal
3	$Volume	Volume information
4	$AttrDef	MFT entry attribute definitions
5	.	Root directory
6	$Bitmap	Cluster block allocation bitmap
7	$Boot	Boot record (or boot code)
8	$BadClus	Bad clusters

MFT entry index	Filename	Description
9	$Quota	Quota information Last used in NTFS version 1.2
9	$Secure	Security and access control information Introduced in NTFS version 3.0
10	$UpCase	Table of uppercase characters used for ensuring case insensitivity in Windows and DOS name spaces.
11	$Extend	A directory containing extended metadata files
12-15		**Unknown (Reserved)** Marked as in use but empty
16-23		Unused Marked as unused
As of NTFS version 3.0		
24	$Extend\$Quota	Quota information Was MFT entry 9 in Windows NT 4
25	$Extend\$ObjId	Unique file identifiers for distributed link tracking
26	$Extend\$Reparse	Backreferences to reparse points
As of Windows Vista (**or server 2003?**) *Transactional NTFS metadata (See section: Transactional NTFS (TxF))*		
27	$Extend\$RmMetadata	Resource manager metadata directory
28	$Extend\$RmMetadata\$Repair	Repair information

MFT entry index	Filename	Description
29 or 30	$Extend\$RmMetadata\$TxfLog	Transactional NTFS (TxF) log metadata directory
30 or 31	$Extend\$RmMetadata\$Txf	Transactional NTFS (TxF) metadata directory
31 or 32	$Extend\$RmMetadata\$TxfLog\$Tops	TxF Old Page Stream (TOPS) file Used to store data that has been overwritten inside a currently active transaction
32 or 33	$Extend\$RmMetadata\$TxfLog\$TxfLog.blf	Transactional NTFS (TxF) base log metadata file
As of Windows 10		
29	$Extend\$Deleted	Temporary location for files that have an open handle but a request has been made to delete them
Common		
	...	A file or directory

The following metadata files are predefined, however the MFT entry index is commonly used but not fixed.

MFT entry index	Filename	Description
	$Extend\$UsnJrnl	USN change journal See section: USN change journal **Has this file been added in Windows XP SP3? Otherwise what are reasons for it to not be present?**

4. The volume header

The volume header is stored at the start of the volume (in the $Boot metadata file) and contains:

- the volume signature
- the BIOS parameter block
- the boot loader

The volume header is 512 bytes of size and consists of:

Offset	Size	Value	Description
0	3		Boot entry point Often contains: eb52 jmp 0x52 90 nop This is a jump instruction to the bootcode at offset 84 followed by a no-operation.
3	8	"NTFS\x20\x20\x20\x20"	File system signature (Also known as OEM and/or dummy identifier)
DOS version 2.0 BIOS parameter block (BPB)			
11	2		Bytes per sector Values supported by mkntfs: 256, 512, 1024, 2048 and 4096
13	1		Sectors per cluster block See below.
14	2	0x00	**Unknown (Reserved Sectors)** not used by NTFS `[POLLARD06]` and must be 0 `[MSDN]`

Offset	Size	Value	Description
16	1	0x00	Number of File Allocation Tables (FATs) not used by NTFS `[POLLARD06]` and must be 0 `[MSDN]`
17	2	0	Root directory entries not used by NTFS `[POLLARD06]` and must be 0 `[MSDN]`
19	2		**Unknown (Total number of sectors (16-bit))** **Used if the total of number of sectors fits in 16-bit?**
21	1		Media descriptor See section: Media descriptor
22	2	0x00	Sectors Per File Allocation Table (FAT) not used by NTFS `[POLLARD06]` and must be 0 `[MSDN]`
DOS version 3.4 BIOS parameter block (BPB)			
24	2	0x3f	Sectors per track Not used by NTFS `[MSDN]`
26	2	0xff	Number of heads Not used by NTFS `[MSDN]`
28	4	0x3f	Number of hidden sectors Not used by NTFS `[MSDN]`
32	4	0x00	**Unknown (Total number of sectors (32-bit))** **Used if the total of number of sectors fits in 32-bit?** Not used by NTFS must be 0 `[MSDN]`
NTFS version 8.0 BIOS parameter block (BPB) or extended BPB *Introduced in Windows NT version 3.1*			

Offset	Size	Value	Description
36	1	0x80	**Unknown (Disc unit number)** Not used by NTFS `[MSDN]`
37	1	0x00	**Unknown (Flags)** Not used by NTFS `[MSDN]`
38	1	0x80	**Unknown (BPB version signature byte)** Not used by NTFS `[MSDN]`
39	1	0x00	**Unknown (Reserved)** Not used by NTFS `[MSDN]`
40	8		Total number of sectors (64-bit)
48	8		Master File Table (MFT) cluster block number
56	8		Mirror MFT cluster block number
64	1		MFT entry size See below.
65	3		**Unknown** Not used by NTFS `[MSDN]`
68	1		Index entry size See below.
69	3		**Unknown** Not used by NTFS `[MSDN]`
72	8		NTFS volume serial number See below.

Offset	Size	Value	Description
80	4	0x00	Checksum not used by NTFS `[POLLARD06]` , `[MSDN]`
84	426		Bootcode **What is the exact end of the bootcode and are there no trailing values?**
510	2	0x55 0xaa	Sector signature

The sectors per cluster block value as used by mkntfs is defined as following:

- Values 0 to 128 represent sizes of 0 to 128 sectors.
- Values 244 to 255 represent sizes of `2^(256-n)` sectors.
- Other values are unknown.

The MFT entry size and index entry size are defined as following:

- Values 0 to 127 represent sizes of 0 to 127 cluster blocks.
- Values 128 to 255 represent sizes of `2^(256-n)` bytes; or `2^(-n)` if considered as a signed byte.
- Other values are not considered valid `[POLLARD06]` .

The cluster block size can be determined as following:

```
cluster block size = bytes per sector x sectors per cluster block
```

Note	Different NTFS implementations support different cluster block sizes.

Known supported cluster block size:

Cluster block size	Bytes per sector	Supported by
256	256	mkntfs
512	256 - 512	mkntfs, ntfs3g, Windows
1024	256 - 1024	mkntfs, ntfs3g, Windows
2048	256 - 2048	mkntfs, ntfs3g, Windows
4096	256 - 4096	mkntfs, ntfs3g, Windows
8192	256 - 4096	mkntfs, ntfs3g, Windows
16K (16384)	256 - 4096	mkntfs, ntfs3g, Windows
32K (32768)	256 - 4096	mkntfs, ntfs3g, Windows
64K (65536)	256 - 4096	mkntfs, ntfs3g, Windows
128K (131072)	256 - 4096	mkntfs, ntfs3g, Windows 10 (1903)
256K (262144)	256 - 4096	mkntfs, ntfs3g, Windows 10 (1903)
512K (524288)	256 - 4096	mkntfs, ntfs3g, Windows 10 (1903)
1M (1048576)	256 - 4096	mkntfs, ntfs3g, Windows 10 (1903)
2M (2097152)	512 - 4096	mkntfs, ntfs3g, Windows 10 (1903)

Note	Windows 10 (1903) requires the partition containing the NTFS file system to be aligned with the cluster block size. For example for a cluster block size of 128k the partition must 128 KiB aligned. The default partition partition alignment appears to be 64 KiB.

Note	mkntfs restricts the cluster size to: bytes per sector >= cluster size > 4096 * bytes per sector

The MFT offset can be determined as following:

```
MFT offset = volume header offset
           + ( MFT cluster block number x Cluster block size )
```

Note that the lower 32-bit part of the NTFS volume serial number is the WINAPI volume serial number. E.g. compare the output of:

```
fsutil fsinfo volumeinfo C:
fsutil fsinfo ntfsinfo C:
```

Often the volume will be smaller than the underlying partition. A (nearly identical) backup of the volume header is stored in last sector of cluster block, that follows the last cluster block of the volume. Often this is the 512 bytes after the last sector of the volume, but not necessarily. The backup volume header is not included in the volume size.

4.1. BitLocker Drive Encryption (BDE)

BitLocker Drive Encryption (BDE) uses the file system signature: "-FVE-FS-". Where FVE is an abbreviation of Full Volume Encryption.

The data structures of BDE on Windows Vista and 7 differ.

A Windows Vista BDE volume starts with:

```
eb 52 90 2d 46 56 45 26 46 53 2d
```

A Windows 7 BDE volume starts with:

```
eb 58 90 2d 46 56 45 26 46 53 2d
```

BDE is largely a stand-alone but has some integration with NTFS. For more information about BDE see [LIBBDE].

4.2. Volume Shadow Snapshots (VSS)

Volume Shadow Snapshots (VSS) uses the GUID 3808876b-c176-4e48-b7ae-04046e6cc752 (stored in little-endian) to identify its data. VSS is largely a stand-alone but has some integration with NTFS.

For more information about VSS see [LIBVSHADOW].

4.3. Media descriptor

Bit(s)	Identifier	Description
0		Sides: 0 ⇒ single-sided 1 ⇒ double-sided
1		Track size: 0 ⇒ 9 sectors per track 1 ⇒ 8 sectors per track
2		Density: 0 ⇒ 80 tracks 1 ⇒ 40 tracks
3		Type: 0 ⇒ Fixed disc 1 ⇒ Removable disc
4 – 7		Always set to 1

4.4. The boot loader

Offset	Size	Value	Description
512			**Windows NT (boot) loader NTLDR/BOOTMGR**

5. The Master File Table (MFT)

The MFT consist of an array of MFT entries. The offset of the MFT table can be found in the volume header and the size of the MFT is defined by the MFT entry of the $MFT metadata file.

Note	The MFT can consists of multiple data ranges, defined by the data runs in the $MFT metadata file.

5.1. MFT entry

Although the size of a MFT entry is defined in the volume header is commonly 1024 bytes of size and consists of:

- The MFT entry header
- The fix-up values
- An array of MFT attribute values
- Padding, which should contain 0-byte values

Note	The MFT entry can be filled entirely with 0-byte values. Seen in Windows XP for MFT entry indexes 16 - 23.

5.1.1. MFT entry header

The MFT entry header (FILE_RECORD_SEGMENT_HEADER) is 42 or 48 bytes of size and consists of:

Offset	Size	Value	Description
MULTI_SECTOR_HEADER			
0	4	"BAAD" "FILE"	Signature
4	2		The fix-up values offset Contains an offset relative from the start of the MFT entry According to `[MSDN]` this value is the update sequence array offset
6	2		The number of fix-up values According to `[MSDN]` this value is the update sequence array size.
8	8		Metadata transaction journal sequence number Contains a $LogFile Sequence Number (LSN)
16	2		Sequence (number)
18	2		Reference (link) count
20	2		Attributes offset (or first attribute offset) Contains an offset relative from the start of the MFT entry
22	2		Entry flags See section: MFT entry flags
24	4		Used entry size Contains the number of bytes of the MFT entry that are in use
28	4		Total entry size Contains the number of bytes of the MFT entry **Could this be used to store data larger than 1024 - header continuously?**

Offset	Size	Value	Description
32	8		Base record file reference See section: The file reference
40	2		First available attribute identifier
Version 3.0			
42	2		**Unknown (wfixupPattern)**
44	4		**Unknown**
Version 3.1			
42	2		**Unknown (wfixupPattern)**
44	4		The index

The base record file reference indicates if the MFT entry is used to store additional attributes for another MFT entry, e.g. for attribute list attributes.

According to `[MSDN]` the sequence number is incremented each time that a file record segment is freed; it is 0 if the segment is not used.

`[MSDN]` states that the update sequence array must end before the last USHORT value in the first sector. It also claims the update sequence array size value contains the number of bytes. It seems to be more likely to the number of words.

The "BAAD" signature presumably indicates a bad MFT entry. `[RUSSON05]` states that during chkdsk, if NTFS finds a multi-sector item where the multi-sector header does not match the values at the end of the sector, it marks the item as "BAAD" and fill it with 0-byte values except for a fix-up value at the end of the first sector. The "BAAD" signature has been seen to be used on Windows NT4 and XP.

In NT4 (version 1.2) the MFT entry is 42 bytes in size and the fix-up values are stored at offset 42. This is likely where the field name wfixupPattern originates from.

5.1.2. Notes

```
Live MFT header:
00000000: 02 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00000010: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00000020: 00 00 00 00 00 00 00 00  00 00                     ........ ..

00000000: 03 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00000010: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00000020: 00 00 00 00 00 00 00 00  00 00                     ........ ..
```

5.2. MFT entry flags

Value	Identifier	Description
0x0001	FILE_RECORD_SEGMENT_IN_USE MFT_RECORD_IN_USE	In use
0x0002	FILE_NAME_INDEX_PRESENT MFT_RECORD_IS_DIRECTORY	Has file name (or $I30) index When this flag is set the file entry represents a directory (that contains sub file entries)
0x0004	MFT_RECORD_IN_EXTEND	**Unknown** According to `[APPLE06]` this is set for all system files present in the $Extend directory
0x0008	MFT_RECORD_IS_VIEW_INDEX	Is index When this flag is set the file entry represents an index According to `[APPLE06]` this is set for all indices other than $I30

5.3. The file reference

The file reference (FILE_REFERENCE or MFT_SEGMENT_REFERENCE) is 8 bytes of size and consists of:

Offset	Size	Value	Description
0	6		MFT entry index Note that the index value in the MFT entry is only 32-bit of size.
6	2		Sequence number

5.4. The fix-up values

The fix-up values are variable of size and consists of:

Offset	Size	Value	Description
0	2		Fix-up placeholder value
2	2 x number of fix-up values		Fix-up (original) value array

On disk the last 2 bytes for each 512 bytes block is replaced by the fix-up placeholder value. The original value is stored in the corresponding fix-up (original) value array entry.

Note	There can be more fix-up values than the amount of sectors in the data.

See `[CARRIER05]` and/or `[RUSSON05]` for examples on applying the fix-up values.

5.5. MFT attribute

The MFT attribute consist of:

- the attribute header
- the attribute resident or non-resident data
- the attribute name

- **unknown data likely alignment padding (4-byte alignment)**
- the attribute data runs or data
- alignment padding (8-byte alignment), can contain remnant data

5.5.1. MFT attribute header

The MFT attribute header (ATTRIBUTE_RECORD_HEADER) is 16 bytes of size and consists of:

Offset	Size	Value	Description
0	4		Attribute type (or type code) See section: The attribute types
4	4		Size (or record length) The size of the attribute including the 8 bytes of the attribute type and size
8	1		Non-resident flag (or form code) 0 ⇒ RESIDENT_FORM 1 ⇒ NONRESIDENT_FORM
9	1		Name size (or name length) Contains the number of characters without the end-of-string character
10	2		Name offset Contains an offset relative from the start of the MFT entry
12	2		Attribute data flags See section: MFT attribute data flags
14	2		Attribute identifier (or instance) **An unique identifier to distinguish between attributes that contain segmented data.**

Notes

Size (or record length) upper 2 bytes overloaded or remnant data?

```
type                                    : 0x000000a0 ($INDEX_ALLOCATION)
size                                    : 458832 (0x70050)
non resident flag                       : 0x01
name size                               : 4
name offset                             : 64
data flags                              : 0x0000

identifier                              : 4
```

MFT attribute data flags

Value	Identifier	Description
0x0001		Is compressed
0x00ff	ATTRIBUTE_FLAG_COMPRESSION_MASK	
0x4000	ATTRIBUTE_FLAG_ENCRYPTED	Is encrypted
0x8000	ATTRIBUTE_FLAG_SPARSE	Is sparse

Does 0x0001 indicate the LZNT1 compression method? Do other values indicate other compression values?

5.5.2. Resident MFT attribute

The resident MFT attribute data is present when the non-resident flag is not set (0). The resident data is 8 bytes in size and consists of:

Offset	Size	Value	Description
0	4		Data size (or value length)

Offset	Size	Value	Description
4	2		Data offset (or value size) Contains an offset relative from the start of the MFT attribute
6	1		Indexed flag **Only the lower bit is used, do the other bits have any significance?**
7	1	0x00	Padding Contains an empty byte

Notes

What meaning has ATTRIBUTE_FLAG_COMPRESSION_MASK in $INDEX_ROOT attribute? The attribute data is uncompressed.

Seen on Windows 10 (NTFS version 3.1)

```
type                                    : 0x00000090 ($INDEX_ROOT)
size                                    : 88
non resident flag                       : 0x00
name size                               : 4
name offset                             : 24
data flags                              : 0x0001
        Is compressed
```

5.5.3. Non-resident MFT attribute

The non-resident MFT attribute data is present when the non-resident flag is set (1). The non-resident data is 48 or 56 bytes in size and consists of:

Offset	Size	Value	Description
0	8		First (or lowest) Virtual Cluster Number (VCN) of the data
8	8		Last (or highest) Virtual Cluster Number (VCN) of the data **Seen this value to be -1 in combination with data size of 0**
16	2		Data runs offset (or mappings pairs offset) Contains an offset relative from the start of the MFT attribute
18	2		Compression unit size Contains the compression unit size as `2^(n)` number of cluster blocks. This value is used for compressed data in the data runs. **A value of 0 indicates the attribute data is uncompressed.?** **Seen on XP, compressed MFT attribute data with compression unit size of 0.** **So it looks more the default compression unit size (16 cluster blocks) should be used.**
20	4		Padding Contains zero-bytes
24	8		Allocated data size (or allocated length) Contains the allocated data size in number of bytes. This value is not valid if the first VCN is nonzero.
32	8		Data size (or file size) Contains the data size in number of bytes. This value is not valid if the first VCN is nonzero.
40	8		Valid data size (or valid data length) Contains the valid data size in number of bytes. This value is not valid if the first VCN is nonzero.
If compression unit size > 0			

Offset	Size	Value	Description
48	8		Total allocated size Contains the total allocated size in number of cluster blocks.

Note	The total size of the data runs should be larger or equal to the data size.

Note	Windows will fill data ranges beyond the valid data size with 0-byte values. The data size remains unchanged. This applies to compressed and uncompressed data. If the first VCN is zero a valid data size of 0 represents a file entirely filled with 0-byte values.

5.5.4. Attribute name

The attribute name is variable of size and consists of:

Offset	Size	Value	Description
0	...		Name Contains an UTF-16 little-endian without end-of-string character

5.5.5. Data runs

The data runs are stored in a variable size (data) runlist. This runlist consists of runlist elements.

A runlist element is variable of size and consists of:

Offset	Size	Value	Description
0.0	4 bits		Number of cluster blocks value size Contains the number of bytes used to store the data run size
0.4	4 bits		Cluster block number value size Contains the number of bytes used to store the data run size

Offset	Size	Value	Description
1	Size value size		Data run number of cluster blocks Contains the number of cluster blocks
...	Cluster block number value size		Data run cluster block number See below.

The data run cluster block number is a singed value, where the MSB is the singed bit, e.g. if the data run cluster block contains \'dbc8' it corresponds to the 64-bit value 0xffffffffffffdbc8.

The first data run offset contains the absolute cluster block number where successive data run offsets are relative to the last data run offset.

Note	The cluster block number byte size is the first nibble when reading the byte stream, but here it is represented as the upper nibble of the first byte.

The last runlist element is an empty value size tuple; in other words a 0 byte.

Does a data run with a "number of cluster blocks value size" of 0 represent this as well?

The MFT attribute data flag (ATTRIBUTE_FLAG_SPARSE) indicates if the data stream is sparse or not.

A sparse data run has a "cluster block number value size" 0, representing there is no offset (cluster block number). A sparse data run should be filled with 0-byte values.

Note	Compressed files also define sparse data runs without setting the sparse flag.

TODO: what about data runs with a cluster block number value size of 0 but without the necessary flags? Seen in ADS: $BadClus:$Bad. Assuming for now the data run is sparse.

The MFT attribute data flags (0x00ff) indicate if the data stream is compressed or not. The currently known value for LZNT1 compression is 1.

Note	Windows 10 supports Windows Overlay Filter (WOF) compressed data, which stores the LZXPRESS Huffman or LZX compressed data in alternate data stream named WofCompressedData and links it to the default data stream using a reparse point.

The data is stored in compression unit blocks. A compression unit typically consists of 16 cluster blocks. However the actual value is stored in the non-resident MFT attribute. See Compression for more information on how to determine which data runs store the compressed and which do not.

Note	Compression is supported upto NTFS file systems with a cluster block size of 4096 bytes or less.

The compression is specified on a pre attribute basis. Where an attribute chain can consists of attribute with compressed and uncompressed attribute data. **Note that it is unknown if mixing compressed and uncompressed attributes is supported by the Windows implementation.**

According to `[RUSSON05]` the size of the runlist is rounded up to the next multitude of 4 bytes. The size of the trailing data can be even larger than 3 and are not always zero-bytes.

See `[CARRIER05]` and/or `[RUSSON05]` for examples on reading the runlist.

6. The attributes

6.1. The attribute types

Technically the attribute types are stored in the `$AttrDef` metadata file. Also see section: The attribute definitions

Value	Identifier	Description
0x00000000		Unused
0x00000010	$STANDARD_INFORMATION	Standard information
0x00000020	$ATTRIBUTE_LIST	Attributes list

Value	Identifier	Description
0x00000030	$FILE_NAME	The file or directory name
0x00000040	$VOLUME_VERSION	Volume version Removed in NTFS version 3.0
0x00000040	$OBJECT_ID	Object identifier Introduced in NTFS version 3.0
0x00000050	$SECURITY_DESCRIPTOR	Security descriptor
0x00000060	$VOLUME_NAME	Volume name
0x00000070	$VOLUME_INFORMATION	Volume information
0x00000080	$DATA	Data stream
0x00000090	$INDEX_ROOT	Index root
0x000000a0	$INDEX_ALLOCATION	Index allocation
0x000000b0	$BITMAP	Bitmap
0x000000c0	$SYMBOLIC_LINK	Symbolic link Removed in NTFS version 3.0
0x000000c0	$REPARSE_POINT	Reparse point Introduced in NTFS version 3.0
0x000000d0	$EA_INFORMATION	(HPFS) extended attribute information
0x000000e0	$EA	(HPFS) extended attribute
0x000000f0	$PROPERTY_SET	Property set Removed in NTFS version 3.0

Value	Identifier	Description
0x00000100	$LOGGED_UTILITY_STREAM	Logged utility stream Introduced in NTFS version 3.0
0x00001000		First user defined attribute
0xffffffff		End of attributes marker

6.2. The standard information attribute

The standard information attribute ($STANDARD_INFORMATION) contains the basic file entry metadata. It is stored as a resident MFT attribute.

The standard information data (STANDARD_INFORMATION) is either 48 or 72 bytes of size and consists of:

Offset	Size	Value	Description
0	8		Creation date and time Contains a FILETIME
8	8		Last modification date and time (Also referred to as last written date and time) Contains a FILETIME
16	8		MFT entry last modification date and time Contains a FILETIME
24	8		Last access date and time Contains a FILETIME
32	4		File attribute flags See section: File attribute flags

Offset	Size	Value	Description
36	4		**Unknown (Maximum number of versions)** **What does it contain and what is it used for?**
40	4		**Unknown (Version number)** **What does it contain and what is it used for?** **On Windows 10 does a value of 1 indicate case-senstive folder?**
44	4		**Unknown (Class identifier)** **What does it contain and what is it used for?**
Introduced in NTFS version 3.0			
48	4		Owner identifier **What does it contain and what is it used for?**
52	4		Security descriptor identifier Contains the entry number in the security ID index ($Secure:$SII) See section: Access Control
56	8		Quota charged **What does it contain and what is it used for?** **Does this value correspond to StorageReservedID in fsutil layout output?**
64	8		Update Sequence Number (USN) **What does it contain and what is it used for?**

Note	MFT entries without a $STANDARD_INFORMATION attribute but with other attributes, such as $FILE_NAME, and an $I30 index have been observed.

6.3. The attribute list attribute

The attribute list attribute ($ATTRIBUTE_LIST) is a list of attributes in an MFT entry. The attributes stored in the list are placeholders for other attributes. Some of these attributes could not be stored in the MFT entry due to space limitations. The attribute list attribute can be stored as either a resident (for a small amount of data) and non-resident MFT attribute.

The attribute list data contains an array of attribute list entries and stored as a continous stream accross one or more cluster blocks.

Note that MFT entry 0 also can contain an attribute list and allows to store listed attributes beyond the first data run.

6.3.1. The attribute list entry

The attribute list entry consists of:

- the attribute list entry header
- the the attribute name
- alignment padding (8-byte alignment), can contain remnant data

The attribute list entry header

The attribute list entry header (ATTRIBUTE_LIST_ENTRY) is 26 bytes of size and consists of:

Offset	Size	Value	Description
0	4		Attribute type (or type code) See section: The attribute types
4	2		Size (or record length) The size of the attribute including the 6 bytes of the attribute type and size
6	1		Name size (or name length) Contains the number of characters without the end-of-string character

Offset	Size	Value	Description
7	1		Name offset Contains an offset relative from the start of the attribute list entry
8	8		Data first (or lowest) VCN
16	8		File reference (or segment reference) The file reference to the MFT entry that contains (part of) the attribute data See section: The file reference
24	2		Attribute identifier **An unique identifier to distinguish between attributes that contain segmented data.**

The data first VCN is used when the attribute data is stored. The attribute list contains an attribute list entry for every cluster block. The corresponding cluster block will contain an MFT attribute containing the attribute data. See `[CARRIER05]` pages 365 and 366 for more information.

Attribute name

The attribute name is variable of size and consists of:

Offset	Size	Value	Description
0	...		Name Contains an UTF-16 little-endian without end-of-string character

6.4. The file name attribute

The file name attribute ($FILE_NAME) contains the basic file system information, like the parent file entry, MAC times and filename. It is stored as a resident MFT attribute.

The file name data (FILE_NAME) is variable of size and consists of:

Offset	Size	Value	Description
0	8		Parent file reference See section: The file reference
8	8		Creation date and time Contains a FILETIME
16	8		Last modification date and time (Also referred to as last written date and time) Contains a FILETIME
24	8		MFT entry last modification date and time Contains a FILETIME
32	8		Last access date and time Contains a FILETIME
40	8		Allocated (or reserved) file size See below.
48	8		File size See below.
56	4		File attribute flags See section: File attribute flags
60	4		Extended data See below.
64	1		Name string size Contains the number of characters without the end-of-string character
65	1		Namespace of the name string

Offset	Size	Value	Description
66	...		Name string Contains an UTF-16 little-endian without an end-of-string character

Note	The name string is not strict UTF-16 since it allows for unpaired surrogates, such as "U+d800" and "U+dc00".

The extended data contains:

- the reparse point tag (see section Reparse point tag) if the reparse point file attribute flag (FILE_ATTRIBUTE_REPARSE_POINT) is set;
- **the extended attribute data size.**

The allocated file size and file size values do not always contain accurate values when stored in a MFT attribute, see `[CARRIER05]` page 363 for more information. [CARRIER05] also states that the file size values are accurate when 'used in a directory index' (stored in an index value), however this seems to be true for most files but not for all. At least the $MFT and $MFTMirr metadata file directory entries on a Windows Vista NTFS volume were found to contain the same value as the corresponding MFT entries, which were not equal to the size of the data stream.

An MFT attribute can contain multiple file name attributes, e.g. for a separate (long) name and short name.

In several cases on a Vista NTFS volume the MFT entry contained both a DOS & Windows and POSIX name space `$FILE_NAME` attribute. However the directory entry index (`$I30`) of the parent directory only contained the DOS & Windows name.

In case of a hard link the MFT entry will contain additional file name attributes with the parent file reference of each hard link.

6.4.1. Namespace

Value	Identifier	Description
0	POSIX	Case sensitive character set that consists of all Unicode characters except for: \0 (zero character), / (forward slash). The : (colon) is valid for NTFS but not for Windows.
1	FILE_NAME_NTFS (or WINDOWS)	A case insensitive sub set of the POSIX character set that consists of all Unicode characters except for: " * / : < > ? \ | Note that names cannot end with a . (dot) or ' ' (space).
2	FILE_NAME_DOS (or DOS)	A case insensitive sub set of the WINDOWS character set that consists of all upper case ASCII characters except for: " * + , / : ; < = > ? \ Note the name must follow the 8.3 format.
3	DOS_WINDOWS	Both the DOS and WINDOWS names are identical **Which is the same as the DOS character set, with the exception that lower case is used as well.**

Note	The Windows API function CreateFile allows to create case sensitive file names when the flag FILE_FLAG_POSIX_SEMANTICS is set.

6.4.2. Long to short name conversion

Basically the conversion from a long name to short name boils down to the approach mentioned below. Note that it differs from the approach mentioned in `[RUSSON05]` , in regard of the third case to make the short name unique.

In the long name:

- ignore Unicode characters beyond the first 8-bit (extended ASCII)
- ignore control characters and spaces (character < 0x20)
- ignore non-allowed characters (" * + , / : ; < = > ? \)
- ignore dots except the last one, which is used for the extension
- make all letters upper case

Additional observations:

- `[` or `]` are replaced by an underscore (_)

Make the name unique:

1. use the characters 1 to 6 add ~1 and if the long name has an extension add the a dot and its first 3 letters
2. if the name already exists try \~2 up to ~9
3. if the name already exists use **some 16-bit hexadecimal value** for characters 3 to 6 with ~1

`[MSDN]` Generates the next four letters of the short file name by mathematically manipulating the remaining letters of the long file name.

Note: behavior dependent on fsutil?

case 1: "Program Files" becomes "PROGRA~1" or " ~PLAYMOVIE.REG" becomes "~PLAYM~1.REG"

case 2: "Program Data", in the same directory as "Program Files", becomes "PROGRA~2"

case 3: "x86_microsoft-windows-r..ry-editor.resources_31bf3856ad364e35_6.0.6000.16386_en-us_f89a7b0005d42fd4", in a directory with a lot of filenames starting with "x86_microsoft", becomes "X8FCA6~1.163"

6.5. The volume version attribute

The volume version attribute ($VOLUME_VERSION) contains **TODO**

Need a pre NTFS 3.0 volume with this attribute. $AttrDef indicates the attribute to be 8 bytes of size.

6.6. The object identifier attribute

The object identifier attribute ($OBJECT_ID) contains distributed link tracker properties. It is stored as a resident MFT attribute.

The object identifier data is either 16 or 64 bytes of size and consists of:

Offset	Size	Value	Description
0	16		Droid file identifier Contains a GUID
16	16		Birth droid volume identifier Contains a GUID
32	16		Birth droid file identifier Contains a GUID
48	16		Birth droid domain identifier Contains a GUID

Droid in this context refers to CDomainRelativeObjId.

6.7. The security descriptor attribute

TODO: does this override any value in $Secure:$SDS?

The security descriptor attribute ($SECURITY_DESCRIPTOR) contains a Windows NT security descriptor. It can be stored as either a resident (for a small amount of data) and non-resident MFT attribute.

See: [LIBFWNT]

6.8. The volume name attribute

The volume name attribute ($VOLUME_NAME) contains the name of the volume. It is stored as a resident MFT attribute.

The volume name data is variable of size and consists of:

Offset	Size	Value	Description
0	...		Name string Contains an UTF-16 little-endian without an end-of-string character

The volume name attribute is used in the $Volume metadata file MFT entry.

6.9. The volume information attribute

The volume information attribute ($VOLUME_INFORMATION) contains information about the volume. It is stored as a resident MFT attribute.

The volume information data is 12 bytes of size and consists of:

Offset	Size	Value	Description
0	8		**Unknown (empty value?)**
8	1		Major version number
9	1		Minor version number
10	2		Volume flags

The volume information attribute is used in the $Volume metadata file MFT entry.

6.9.1. Volume flags

Value	Identifier	Description
0x0001	VOLUME_IS_DIRTY	**Is dirty**
0x0002	VOLUME_RESIZE_LOG_FILE	**Re-size journal (LogFile)**
0x0004	VOLUME_UPGRADE_ON_MOUNT	**Upgrade on next mount**
0x0008	VOLUME_MOUNTED_ON_NT4	**Mounted on Windows NT 4**
0x0010	VOLUME_DELETE_USN_UNDERWAY	**Delete USN underway**
0x0020	VOLUME_REPAIR_OBJECT_ID	**Repair object identifiers**
0x0080		**Unknown**
0x4000	VOLUME_CHKDSK_UNDERWAY	**chkdsk underway**
0x8000	VOLUME_MODIFIED_BY_CHKDSK	**Modified by chkdsk**

6.10. The data stream attribute

The data stream attribute ($DATA) contains the file data. It can be stored as either a resident (for a small amount of data) and non-resident MFT attribute.

Also note that multiple data attributes for the same data stream can be used in the attribute list to define different parts of the data stream data. The first data stream attribute will contain the size of the entire data stream data. Other data stream attributes should have a size of 0. Also see: Attribute chains.

6.11. The index root attribute

The index root attribute ($INDEX_ROOT) contains the root of the index tree. It is stored as a resident MFT attribute.

See section: The index and The index root.

6.12. The index allocation attribute

The index allocation attribute ($INDEX_ALLOCATION) contains an array of index entries. It is stored as a non-resident MFT attribute.

Note that the index allocation attribute itself does not define which attribute type it contains in the index value data. For this information it needs the corresponding index root attribute.

Also note that multiple index allocation attributes for the same index can be used in the attribute list to define different parts of the index allocation data. The first index allocation attribute will contain the size of the entire index allocation data. Other index allocation attributes should have a size of 0. Also see: Attribute chains.

See section: The index.

6.13. The bitmap attribute

The bitmap attribute ($BITMAP) contains the allocation bitmap. It can be stored as either a resident (for a small amount of data) and non-resident MFT attribute.

It is used to maintain information about which entry is used and which is not. Every bit in the bitmap represents an entry. The index is stored byte-wise with the LSB of the byte corresponds to the first allocation element; the allocation element can represent several things, see below.

The allocation element is allocated if the corresponding bit contains 1 or unallocated if 0.

It is known to be used in:

- the MFT (nameless), where an allocation element represents a MFT entry;
- indexes ($I##), where an allocation element represents an index entry.

6.14. The symbolic link attribute

The symbolic link attribute ($SYMBOLIC_LINK) contains **TODO**

Need a pre NTFS 3.0 volume with this attribute. $AttrDef indicates the attribute is variable of size.

6.15. The reparse point attribute

The reparse point attribute ($REPARSE_POINT) contains information about a file system-level link. It is stored as a resident MFT attribute.

See section: The reparse point.

6.16. The (HPFS) extended attribute information

The (HPFS) extended attribute information ($EA_INFORMATION) contains information about the extended attribute ($EA).

The extended attribute information data is 8 bytes of size and consists of:

Offset	Size	Value	Description
0	2		Size of an extended attribute entry
2	2		Number of extended attributes which have NEED_EA set **TODO: determine what this flag is used for**
4	4		Size of the extended attribute ($EA) data

E.g.

```
00000000: 08 00 00 00 18 00 00 00                            ........
```

6.17. The (HPFS) extended attribute

The (HPFS) extended attribute ($EA) contains the extended attribute data.

The extended attribute data is variable of size and consists of:

Offset	Size	Value	Description
0	4		Offset to next extended attribute entry The offset is relative from the start of the extended attribute data
4	1		Flags 0x80 ⇒ NEED_EA (Need EA) flag
5	1		Number of characters of the extended attribute name
6	2		Value data size
8	...		The extended attribute name Contains an ASCII string **TODO: is this value 16-bit aligned?**
...	...		Value data
...	...		**TODO: unknown trailing data**

E.g.

```
00000000: 18 00 00 00 00 09 04 00  2e 55 4e 49 58 41 54 54   ........ .UNIXATT
00000010: 52 00 b6 01 00 00 03 87                            R.......
```

6.17.1. UNITATTR extended attribute value data

Offset	Size	Value	Description
0	4		**TODO: is this an equivalent of st_mode?**

6.18. The property set attribute

The property set attribute ($PROPERTY_SET) contains **TODO**

Need a pre NTFS 3.0 volume with this attribute. $AttrDef indicates does not seem to always define this attribute.

6.19. The logged utility stream attribute

attribute type for storing additional data for the files and directories

resident, known to cause problems when non-resident on Windows Vista

Value	Identifier	Description
$EFS		Encrypted NTFS (EFS)
$TXF_DATA		Transactional NTFS (TxF)

TODO add text

6.20. Attribute chains

Multiple attributes can make up a single attribute, e.g. the attributes:

1. $INDEX_ALLOCATION ($I30) VCN: 0
2. $INDEX_ALLOCATION ($I30) VCN: 596

The first attribute will contain the size of the data defined by all the attributes. Other attributes should have a size of 0.

It is assumed that the attributes in a chain must be continuous and defined in-order.

7. The attribute types

The attribute types are stored in the `$AttrDef` metadata file.

Offset	Size	Value	Description
0	128		Attribute name Contains an UTF-16 little-endian **with end-of-string character?** **The unused bytes are filled with 0-byte values**
128	4		Attribute type (or type code)
132	8		**Unknown (empty values?)**
140	4		**Unknown (flags?)** Seen: 0x40, 0x42, 0x80
144	8		**Unknown (minimum attribute size?)**
152	8		**Unknown (maximum attribute size?)** Seen: -1 (no maximum?), 48

```
00000000  24 00 53 00 54 00 41 00  4e 00 44 00 41 00 52 00  |$.S.T.A.N.D.A.R.|
00000010  44 00 5f 00 49 00 4e 00  46 00 4f 00 52 00 4d 00  |D._.I.N.F.O.R.M.|
00000020  41 00 54 00 49 00 4f 00  4e 00 00 00 00 00 00 00  |A.T.I.O.N.......|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000080  10 00 00 00 00 00 00 00  00 00 00 00 40 00 00 00  |............@...|
00000090  30 00 00 00 00 00 00 00  30 00 00 00 00 00 00 00  |0.......0.......|
000000a0  24 00 41 00 54 00 54 00  52 00 49 00 42 00 55 00  |$.A.T.T.R.I.B.U.|
000000b0  54 00 45 00 5f 00 4c 00  49 00 53 00 54 00 00 00  |T.E._.L.I.S.T...|
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000120  20 00 00 00 00 00 00 00  00 00 00 00 80 00 00 00  | ...............|
00000130  00 00 00 00 00 00 00 00  ff ff ff ff ff ff ff ff  |................|
00000140  24 00 46 00 49 00 4c 00  45 00 5f 00 4e 00 41 00  |$.F.I.L.E._.N.A.|
```

8. The index

The index structures are used for various purposes one of which are the directory entries.

The root of the index is stored in index root. The index root attribute defines which type of attribute is stored in the index and the root index node.

If the index is too large part of the index is stored in an index allocation attribute with the same attribute name. The index allocation attribute defines a data stream which contains index entries. Each index entry contains an index node.

See `[CARRIER05]` page 378 for an illustration how the index root and index allocation attribute relate.

An index consists of a tree, where both the branch and index leaf nodes contain the actual data. E.g. in case of a directory entries index, any node that contains index value data make up for the directory entries.

The index value data in a branch node signifies the upper bound of the values in the that specific branch. E.g. if directory entries index branch node contains the name 'textfile.txt' all names in that index branch are smaller than 'textfile.txt'. Note the actual sorting order is dependent on the collation type defined in the index root attribute.

The index allocation attribute is accompanied by a bitmap attribute with the corresponding attribute name. The bitmap attribute defines the allocation of virtual cluster blocks within the index allocation attribute data stream.

Note	The index allocation attribute can be present even though it is not used.

8.1. Common used indexes

Indexes commonly used by NTFS are:

Value	Identifier	Description
$I30		Directory entries (used by directories)
$SDH		Security descriptor hashes (used by $Secure)
$SII		Security descriptor identifiers (used by $Secure)
$O		Object identifiers (used by $ObjId)
$O		Owner identifiers (used by $Quota)
$Q		Quotas (used by $Quota)
$R		Reparse points (used by $Reparse)

8.2. The index root

The index root consists of:

- index root header
- index node header
- an array of index values

8.2.1. The index root header

The index root header is 16 bytes of size and consists of:

Offset	Size	Value	Description
0	4		Attribute type Contains the type of the indexed attribute or 0 if none

Offset	Size	Value	Description
4	4		Collation type Contains a value to indicate the ordering of the index entries See section: Collation type
8	4		Index entry size
12	4		Index entry number of cluster blocks

Note	`[CARRIER05]` and `[RUSSON05]` state that the last 3 bytes are unused (alignment padding). However it is highly probably that the last value is 32-bit of size.

Note	In NT4 (version 1.2) the index entry size does not have to match the index entry size in the volume header. The correct size seems to be the value in the index root header.

8.2.2. Collation type

Value	Identifier	Description
0x00000000	COLLATION_BINARY	Binary The first byte is most significant
0x00000001	COLLATION_FILENAME	Unicode strings case-insensitive
0x00000002	COLLATION_UNICODE_STRING	Unicode strings case-sensitive Upper case letters should come first
0x00000010	COLLATION_NTOFS_ULONG	Unsigned 32-bit little-endian integer
0x00000011	COLLATION_NTOFS_SID	NT security identifier (SID)
0x00000012	COLLATION_NTOFS_SECURITY_HASH	Security hash first, then NT security identifier

Value	Identifier	Description
0x00000013	COLLATION_NTOFS_ULONGS	An array of unsigned 32-bit little-endian integer values

8.3. The index entry

The index entry consists of:

- the index entry header
- the index node header
- the fix-up values
- alignment padding (8-byte alignment), contains zero-bytes
- an array of index values

8.3.1. The index entry header

The index entry header is 32 bytes of size and consists of:

Offset	Size	Value	Description
0	4	"INDX"	Signature
4	2		The fix-up values offset Contains an offset relative from the start of the index entry header.
6	2		The number of fix-up values
8	8		Metadata transaction journal sequence number Contains a $LogFile Sequence Number (LSN)
16	8		Virtual Cluster Number (VCN) of the index entry

Note	There can be more fix-up value than supported by the index entry data size.

8.4. The index node header

The index node header is 16 bytes of size and consists of:

Offset	Size	Value	Description
0	4		Index values offset The offset is relative from the start of the index node header
4	4		Index node size The value includes the size of the index node header. See below.
8	4		Allocated index node size The value includes the size of the index node header
12	4		Index node flags See section: The index node flags

Note that `[RUSSON05]` states that the last 3 bytes are unused (alignment padding), while `[CARRIER05]` states that the last value is 32-bit of size. Here we assume that the index node flags are a 32-bit value.

In an index entry (index allocation attribute) the index node size includes the size of the fix-up values and the alignment padding following it.

The remainder of the index node contains remnant data and/or zero-byte values.

8.4.1. The index node flags

Value	Identifier	Description
0x00000001		Is branch node Used to indicate if the node is a branch node that has sub nodes

8.5. The index value

The index value is variable of size and consists of:

Offset	Size	Value	Description
0	8		File reference See section: The file reference
8	2		Index value size
10	2		Index key data size
12	4		Index value flags
If index key data size > 0			
16	...		Index key data
...	...		Index value data
If index value flag 0x00000001 (is branch node) is set			
...	8		Sub node Virtual Cluster Number (VCN)

Note	The index values ared stored 8 byte aligned.

Note	Some sources define the index value flags as a 16-bit value followed by 2 bytes of padding.

8.5.1. The index value flags

Value	Identifier	Description
0x00000001		Has sub node If set the index value contains a sub node Virtual Cluster Number (VCN)
0x00000002		Is last If set the index value is the last in the index values array

8.6. Index key and value data

8.6.1. Directory entry index value

The MFT attribute name of the directory entry index is: $I30.

The directory entry index value contains a file name attribute in the index key data. See section: The file name attribute.

The index value data seems to contain remnant data.

Note	Both the short and long names of the same file have a separate index value. The short name uses the DOS name space and the long name the WINDOWS name space.

Index values with a single name use either the POSIX or DOS_WINDOWS name space?

A hard link to a file in the same directory will also have a separate index value.

Does the hard link always has POSIX name space?

8.6.2. Security descriptor hash index value

The MFT attribute name of the security descriptor hash index is: $SDH. It appears to only to be used by the $Secure metadata file.

See section: The security descriptor hash index value

8.6.3. Security descriptor identifier index value

The MFT attribute name of the security descriptor identifier index is: $SII. It appears to only to be used by the $Secure metadata file.

See section: The security descriptor identifier index value

9. Compression

Typically NTFS compression groups 16 cluster blocks together. This group of 16 cluster blocks also named a compression unit is either \'compressed' or uncompressed data. The term compressed is quoted here because, as you will see below, the group of cluster blocks can also contain uncompressed data. A group of cluster blocks is \'compressed' when it is compressed size is smaller than its uncompressed data size.

Note	The actual compression unit size is stored in the non-resident MFT attribute.

Within a group of cluster blocks each of the 16 blocks is \'compressed' individually see Block based storage. The maximum uncompressed data size is always the cluster size (in most cases 4096).

The data runs in the $DATA stream define cluster block ranges. A simple example:

```
21 02 35 52
```

This data run defines 2 data blocks starting at block number 21045 followed by 14 sparse blocks. The total number of blocks is 16 which is the size of the compression unit. The data is stored compressed in the first 2 blocks and the 14 sparse blocks are only there to make sure the data runs add up to the compression unit size. They do not define actual sparse data.

Another example:

```
21 40 37 52
```

This data run defines 64 data blocks starting at block number 21047. Since this data run is larger than the compression unit size the data is stored uncompressed.

If the data run was e.g. 60 data blocks followed by 4 sparse blocks the first 3 compression units (blocks 1 to 48) would be uncompressed and the last compression unit (blocks 49 to 64) would be compressed.

Also "sparse data" and "sparse compression unit" data runs can be mixed. If in the previous example the 60 data blocks would be followed by 20 sparse blocks the last compression unit (blocks 65 to 80) would be sparse.

Note	A compression unit can consists of multiple compressed data runs, e.g. 1 data block followed by 4 data blocks followed by 11 sparse blocks. Data runs have been observed where the last data run size does not align with the compression unit size.

Note	The sparse blocks data run can be stored in a subsequent attribute in an attribute chain and can be stored in multiple data runs.

Does the sparse flag needs to be set for sparse compressed files?

Is resident data is always uncompressed?

Also see `[RUSSON05]` for more detailed examples.

9.1. Block based storage

NTFS compression stores the \'compressed' data in blocks. Each block has a 2 byte block header.

The block is variable of size and consists of:

Offset	Size	Value	Description
0	2		Block size
2	(compressed data size)		

The upper 4 bits of the block size are used as flags.

Bit(s)	Description
0 - 11	Compressed data size
12 - 14	**Unknown flags**
15	Data is compressed

9.2. LZNT1 compression method

For more information about LZNT1 see: [LIBFWNT]

Also see section: Corruption scenarios

9.3. Windows Overlay Filter (WOF) compressed data

A MFT entry that contains Windows Overlay Filter (WOF) compressed data has the following attributes:

- reparse point attribute with tag 0x80000017, which defines the compression method
- a nameless data attribute that is sparse and contains the uncompressed data size
- a data attribute named WofCompressedData that contains LZXPRESS Huffman or LZX compressed data

Offset	Size	Value	Description
Chunk offset table			
0	...		Array of 32-bit of 64-bit compressed data chunk offsets The offset is relative from the start of the data chunks
Data chunks			
...	...		One or more compressed or uncompressed data chunks

Note	If the chunk size equals the size of the uncompressed data the chunk is stored (as-is) uncompressed.

The size of the chunk offset table is:

```
number of chunk offsets = uncompressed size / compression unit size
```

The offset of the first compressed data chunk is at the end of the chunk offset table and is not stored in the chunk offset table.

For more information about the compression methods see section Windows Overlay Filter (WOF) compression method

10. The reparse point

The reparse point is used to create file system-level links. Reparse data is stored in the reparse point attribute. The reparse point data (REPARSE_DATA_BUFFER) is variable of size and consists of:

Offset	Size	Value	Description
0	4		Reparse point tag
4	2		Reparse data size

Offset	Size	Value	Description
6	2	0	**Unknown (Reserved)**
8	...		Reparse data

What about the GUID mentioned in `[RUSSON05]` in third party reparse points.

`[MSDN]` ReparseGuid: A 16-byte GUID that uniquely identifies the owner of the reparse point. Reparse pointGUIDs are assigned by the implementer of a file system, the file system filter driver, or the minifilter driver. The implementer must generate one GUID to use with their assigned reparse point tag, and must always use this GUID as the ReparseGuid for that tag.

10.1. Reparse point tag

Offset	Size	Value	Description
0.0	16 bits		Type
2.0	12 bits		**Unknown (Reserved)**
3.4	4 bits		Flags

10.1.1. Predefined reparse point tag values

Predefined reparse point tag values according to `[MSDN]`:

Value	Identifier	Description
0x00000000	IO_REPARSE_TAG_RESERVED_ZERO	**Unknown (Reserved)**
0x00000001	IO_REPARSE_TAG_RESERVED_ONE	**Unknown (Reserved)**
0x00000002	IO_REPARSE_TAG_RESERVED_TWO	**Unknown (Reserved)**

Value	Identifier	Description
0x80000005	IO_REPARSE_TAG_DRIVE_EXTENDER	Used by Home server drive extender
0x80000006	IO_REPARSE_TAG_HSM2	Used by Hierarchical Storage Manager Product
0x80000007	IO_REPARSE_TAG_SIS	Used by single-instance storage (SIS) filter driver
0x80000008	IO_REPARSE_TAG_WIM	Used by the WIM Mount filter
0x80000009	IO_REPARSE_TAG_CSV	Used by Clustered Shared Volumes (CSV) version 1
0x8000000a	IO_REPARSE_TAG_DFS	Used by the Distributed File System (DFS)
0x8000000b	IO_REPARSE_TAG_FILTER_MANAGER	Used by filter manager test harness
0x80000012	IO_REPARSE_TAG_DFSR	Used by the Distributed File System (DFS)
0x80000013	IO_REPARSE_TAG_DEDUP	Used by the Data Deduplication (Dedup)
0x80000014	IO_REPARSE_TAG_NFS	Used by the Network File System (NFS)
0x80000015	IO_REPARSE_TAG_FILE_PLACEHOLDER	Used by Windows Shell for placeholder files
0x80000016	IO_REPARSE_TAG_DFM	Used by Dynamic File filter
0x80000017	IO_REPARSE_TAG_WOF	Used by Windows Overlay Filter (WOF), for either WIMBoot or compression See section: Windows Overlay Filter (WOF) reparse data
0x80000018	IO_REPARSE_TAG_WCI	Used by Windows Container Isolation (WCI) See section: Windows Container Isolation (WCI) reparse data
0x8000001b	IO_REPARSE_TAG_APPEXECLINK	Used by Universal Windows Platform (UWP) packages to encode information that allows the application to be launched by CreateProcess

Value	Identifier	Description
0x8000001e	IO_REPARSE_TAG_STORAGE_SYNC	Used by the Azure File Sync (AFS) filter
0x80000020	IO_REPARSE_TAG_UNHANDLED	Used by Windows Container Isolation (WCI)
0x80000021	IO_REPARSE_TAG_ONEDRIVE	**Unknown (Not used)**
0x80000023	IO_REPARSE_TAG_AF_UNIX	Used by the Windows Subsystem for Linux (WSL) to represent a UNIX domain socket
0x80000024	IO_REPARSE_TAG_LX_FIFO	Used by the Windows Subsystem for Linux (WSL) to represent a UNIX FIFO (named pipe)
0x80000025	IO_REPARSE_TAG_LX_CHR	Used by the Windows Subsystem for Linux (WSL) to represent a UNIX character special file
0x80000036	IO_REPARSE_TAG_LX_BLK	Used by the Windows Subsystem for Linux (WSL) to represent a UNIX block special file
0x9000001c	IO_REPARSE_TAG_PROJFS	Used by the Windows Projected File System filter, for files managed by a user mode provider such as VFS for Git
0x90001018	IO_REPARSE_TAG_WCI_1	Used by Windows Container Isolation (WCI)
0x9000101a	IO_REPARSE_TAG_CLOUD_1	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive
0x9000201a	IO_REPARSE_TAG_CLOUD_2	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive

Value	Identifier	Description
0x9000301a	IO_REPARSE_TAG_CLOUD_3	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive
0x9000401a	IO_REPARSE_TAG_CLOUD_4	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive
0x9000501a	IO_REPARSE_TAG_CLOUD_5	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive
0x9000601a	IO_REPARSE_TAG_CLOUD_6	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive
0x9000701a	IO_REPARSE_TAG_CLOUD_7	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive
0x9000801a	IO_REPARSE_TAG_CLOUD_8	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive
0x9000901a	IO_REPARSE_TAG_CLOUD_9	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive
0x9000a01a	IO_REPARSE_TAG_CLOUD_A	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive
0x9000b01a	IO_REPARSE_TAG_CLOUD_B	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive

Value	Identifier	Description
0x9000c01a	IO_REPARSE_TAG_CLOUD_C	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive
0x9000d01a	IO_REPARSE_TAG_CLOUD_D	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive
0x9000e01a	IO_REPARSE_TAG_CLOUD_E	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive
0x9000f01a	IO_REPARSE_TAG_CLOUD_F	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive
0xa0000003	IO_REPARSE_TAG_MOUNT_POINT	Junction or mount point See section: Junction or mount point reparse data
0xa000000c	IO_REPARSE_TAG_SYMLINK	Symbolic link See section: Symbolic link reparse data
0xa0000010	IO_REPARSE_TAG_IIS_CACHE	Used by Microsoft Internet Information Services (IIS) caching
0xa0000019	IO_REPARSE_TAG_GLOBAL_REPARSE	Used by NPFS to indicate a named pipe symbolic link from a server silo into the host silo
0xa000001a	IO_REPARSE_TAG_CLOUD	Used by the Cloud Files filter, for files managed by a sync engine such as Microsoft OneDrive
0xa000001d	IO_REPARSE_TAG_LX_SYMLINK	Used by the Windows Subsystem for Linux (WSL) to represent a UNIX symbolic link

Value	Identifier	Description
0xa000001f	IO_REPARSE_TAG_WCI_TOMBSTONE	Used by Windows Container Isolation (WCI)
0xa0000022	IO_REPARSE_TAG_PROJFS_TOMBSTONE	Used by the Windows Projected File System filter, for files managed by a user mode provider such as VFS for Git
0xa0000027	IO_REPARSE_TAG_WCI_LINK	Used by Windows Container Isolation (WCI)
0xa0001027	IO_REPARSE_TAG_WCI_LINK_1	Used by Windows Container Isolation (WCI)
0xc0000004	IO_REPARSE_TAG_HSM	Used by Hierarchical Storage Manager Product
0xc0000014	IO_REPARSE_TAG_APPXSTRM	**Unknown (Not used)**

10.1.2. Notes

single-instance storage (SIS): An NTFS feature that implements links with the semantics of copies for files stored on an NTFSvolume. SIS uses copy-on-close to implement the copy semantics of its links.

```
Is this documentation wrong or are these alternative values?

Flag    Description
0x68000005      NSS
0x68000006      NSS recover
0x68000007      SIS
0x68000008      DFS
0x88000003      Mount point
0xA8000004      HSM
0xE8000000      Symbolic link
```

10.1.3. Reparse point tag flags

Value	Identifier	Description
0x1		**Unknown (Reserved)** Reserved according to `[MSDN]`
0x2		Is alias (Name surrogate bit) If this bit is set, the file or directory represents another named entity in the system.
0x4		Is high-latency media Reserved according to `[MSDN]`
0x8		Is native (Microsoft-bit) **Does this flag influence the reparse point GUID?**

10.2. Junction or mount point reparse data

A reparse point with tag IO_REPARSE_TAG_MOUNT_POINT (0xa0000003) contains junction or mount point reparse data. The junction or mount point reparse data is variable of size and consists of:

Offset	Size	Value	Description
0	2		Substitute name offset The offset is relative from the start of the reparse name data
2	2		Substitute name size Value in bytes, the size of the end-of-string character is not included
4	2		Print name offset The offset is relative from the start of the reparse name data
6	2		Print name size Value in bytes, the size of the end-of-string character is not included

Offset	Size	Value	Description
Reparse name data			
8	...		Substitute name Contains an UTF-16 little-endian **with the end-of-string character?**
...	...		Print name Contains an UTF-16 little-endian **with the end-of-string character?**

TODO: what do byte values like 0x02 represent in the substitute name?

```
00000010: 5c 00 3f 00 3f 00 02 00  43 00 3a 00 5c 00 55 00   \.?.?... C.:.\.U.
00000020: 73 00 65 00 72 00 73 00  5c 00 74 00 65 00 73 00   s.e.r.s. \.t.e.s.
00000030: 74 00 5c 00 44 00 6f 00  63 00 75 00 6d 00 65 00   t.\.D.o. c.u.m.e.
00000040: 6e 00 74 00 73 00 00 00                            n.t.s...
```

10.3. Symbolic link reparse data

A reparse point with tag IO_REPARSE_TAG_SYMLINK (0xa000000c0) contains symbolic link reparse data. The symbolic link reparse data is variable of size and consists of:

Offset	Size	Value	Description
0	2		Substitute name offset The offset is relative from the start of the reparse name data
2	2		Substitute name size Value in bytes
4	2		Print name offset The offset is relative from the start of the reparse name data

Offset	Size	Value	Description
6	2		Print name size Value in bytes
8	4		Symbolic link flags
Reparse name data			
12	...		Substitute name Contains an UTF-16 little-endian without end-of-string character
...	...		Print name Contains an UTF-16 little-endian without end-of-string character

10.3.1. Symbolic link flags

Value	Identifier	Description
0x00000001	SYMLINK_FLAG_RELATIVE	The substitute name is a path name relative to the directory containing the symbolic link.

10.4. Windows Overlay Filter (WOF) reparse data

A reparse point with tag IO_REPARSE_TAG_WOF (0x80000017) contains Windows Overlay Filter (WOF) reparse data. The Windows Overlay Filter (WOF) reparse data is 16 bytes of size and consists of:

Offset	Size	Value	Description
External provider information			
0	4	1	**Unknown (WOF version)**
4	4	2	**Unknown (WOF provider)**

Offset	Size	Value	Description
Internal provider information			
8	4	1	**Unknown (file information version)**
12	4		Compression method See section Windows Overlay Filter (WOF) compression method

10.4.1. Windows Overlay Filter (WOF) compression method

Value	Identifier	Description
0		LZXPRESS Huffman with 4k window (compression unit)
1		LZX with 32k window (compression unit)
2		LZXPRESS Huffman with 8k window (compression unit)
3		LZXPRESS Huffman with 16k window (compression unit)

For more information about LZXPRESS Huffman see: [LIBFWNT]

10.5. Windows Container Isolation (WCI) reparse data

A reparse point with tag IO_REPARSE_TAG_WCI (0x80000018) contains Windows Container Isolation (WCI) reparse data. The Windows Container Isolation (WCI) reparse data is variable of size and consists of:

Offset	Size	Value	Description
0	4	1	Version
4	4	0	**Unknown (reserved)**

Offset	Size	Value	Description
8	16		Look-up identifier Contains a GUID
24	2		Name size Value in bytes
26	...		Name Contains an UTF-16 little-endian without end-of-string character

11. The allocation bitmap

The metadata file $Bitmap contains the allocation bitmap.

Every bit in the allocation bitmap represents a block the size of the cluster block, where the LSB is the first bit in a byte.

TODO determine and describe what the $SRAT data stream is used for.

12. Access control

The $Secure metadata file contains the security descriptors used for access control.

Type	Name	Description
Data	$SDS	Security descriptor data stream Contains all the Security descriptors on the volume
Index	$SDH	Security descriptor hash index

Type	Name	Description
Index	$SII	Security descriptor identifier index Contains the mapping of the security descriptor identifier (in $STANDARD_INFORMATION) to the offset of the security descriptor data (in $Secure:$SDS)

TODO add text

In the $SII index do the index values contain a 32-bit checksum?

12.1. Security descriptor hash ($SDH) index

12.1.1. The security descriptor hash index value

Offset	Size	Value	Description
Key data			
0	4		Security descriptor hash
4	4		Security descriptor identifier
Value data			
8	4		Security descriptor hash
12	4		Security descriptor identifier
16	8		Security descriptor data offset (in $SDS)
24	4		Security descriptor data size (in $SDS)
28	4		**Alignment padding (8-byte alignment)** **Contains string "I\x00I\x00" ?**

12.2. Security descriptor identifier ($SII) index

12.2.1. The security descriptor identifier index value

Offset	Size	Value	Description
Key data			
0	4		Security descriptor identifier
Value data			
4	4		Security descriptor hash **TODO describe the hash algorithm**
8	4		Security descriptor identifier
12	8		Security descriptor data offset (in $SDS)
20	4		Security descriptor data size (in $SDS)

12.3. Security descriptor ($SDS) data stream

Offset	Size	Value	Description
0	4		Security descriptor hash
4	4		Security descriptor identifier
12	8		Security descriptor data offset (in $SDS)
20	4		Security descriptor data size (in $SDS)
24	...		Security descriptor data See: [LIBFWNT]
...	...		16-bit alignment padding.

13. The object identifiers

TODO add text

13.1. $ObjID:$O

Offset	Size	Value	Description
Key data			
0	16		File (or object) identifier Contains a GUID
Value data			
4	8		File reference See section: The file reference
12	16		Birth droid volume identifier Contains a GUID
28	16		Birth droid file (or object) identifier Contains a GUID
44	16		Birth droid domain identifier Contains a GUID

```
00000000  00 00 00 00 13 00 00 00  00 10 00 00 01 00 00 00  |................|
00000010  10 00 00 00 88 00 00 00  88 00 00 00 01 00 00 00  |................|
00000020  20 00 38 00 00 00 00 00  60 00 10 00 01 00 00 00  | .8.....`.......|

OBJECT_ID: 43ecee59-e2b3-11dc-ad7e-001c2582598f of root directory

00000030  59 ee ec 43 b3 e2 dc 11  ad 7e 00 1c 25 82 59 8f  |Y..C.....~..%.Y.|
MFT file reference
OBJECT_ID: e6a67b60-c0b5-4b53-b8fe-94470c83df89 of $Volume

00000040  05 00 00 00 00 00 05 00  60 7b a6 e6 b5 c0 53 4b  |........`{....SK|
00000050  b8 fe 94 47 0c 83 df 89  59 ee ec 43 b3 e2 dc 11  |...G....Y..C....|
00000060  ad 7e 00 1c 25 82 59 8f  00 00 00 00 00 00 00 00  |.~..%.Y.........|
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  18 00 00 00 03 00 00 00  |................|
00000090  01 00 00 00 00 00 00 00                           |........|
```

14. Metadata transaction journal (log file)

The metadata file $LogFile contains the metadata transaction journal.

TODO add text.

- Log File Service restart page header
- fix-up values

14.1. Log File service restart page header

The Log File service restart page header (LFS_RESTART_PAGE_HEADER) is 30 bytes of size and consists of:

Offset	Size	Value	Description
MULTI_SECTOR_HEADER			
0	4	"CHKD" "RCRD" "RSTR"	Signature

Offset	Size	Value	Description
4	2		The fix-up values offset Contains an offset relative from the start of the restart page header According to `[MSDN]` this value is the update sequence array offset
6	2		The number of fix-up values According to `[MSDN]` this value is the update sequence array size.
8	8		Checkdisk last LSN
16	4		System page size
20	4		Log page size
24	2		Restart offset
26	2		Minor format version
28	2		Major format version -1 ⇒ Beta Version 0 ⇒ Transition 1 ⇒ Update sequence support

14.2. Notes

Log File Service record header (LFS_RECORD_HEADER)

Offset	Size	Value	Description
0	8		Metadata transaction journal sequence number Contains a $LogFile Sequence Number (LSN)

Offset	Size	Value	Description
8	8		Previous metadata transaction journal sequence number Contains a $LogFile Sequence Number (LSN)
16	8		Undo next metadata transaction journal sequence number Contains a $LogFile Sequence Number (LSN)

```
LFS_RECORD_HEADER {

    //
    //  The following field is the size of data area for this record.  The
    //  log record header will be padded if necessary to fill to a 64-bit
    //  boundary, so the client data will begin on a 64-bit boundary to
    //  insure that all of his data is 64-bit aligned.  The below value
    //  has not been padded to 64 bits however.
    //

    ULONG ClientDataLength;

    //
    //  Client ID.  This identifies the owner of this log record.  The owner
    //  is uniquely identified by his offset in the client array and the
    //  sequence number associated with that client record.
    //

    LFS_CLIENT_ID ClientId;

    //
    //  This the Log Record type.  This could be a commit protocol record,
    //  a client restart area or a client update record.
    //

    LFS_RECORD_TYPE RecordType;

    //
    //  Transaction ID.  This is used externally by a client (Transaction
    //  Manager) to group log file entries.
    //

    TRANSACTION_ID TransactionId;

    //
    //  Log record flags.
    //

    USHORT Flags;

    //
    //  Alignment field.
    //

    USHORT AlignWord;
```

15. USN change journal

The metadata file $Extend\$UsnJrnl contains the USN change journal. It is a sparse file in which NTFS stores records of changes to files and directories. Applications make use of the journal to respond to file and directory changes as they occur, like e.g. the Windows File Replication Service (FRS) and the Windows (Desktop) Search service.

The USN change journal consists of:

- the $UsnJrnl:$Max data stream, containing metadata like the maximum size of the journal
- the $UsnJrnl:$J data stream, containing the update (or change) entries. The $UsnJrnl:$J data stream is sparse.

15.1. USN change journal metadata

The USN change journal metadata is 32 bytes of size and consists of:

Offset	Size	Value	Description
0	8		Maximum size Contains the number of bytes
8	8		Allocation delta Contains the number of bytes
16	8		Update (USN) journal identifier Contains a FILETIME
24	8		**Unknown (empty)**

15.2. USN change journal entries

The $UsnJrnl:$J data stream consists of an array of USN change journal entries. The USN change journal entries are stored on a per block-basis and 64-bit aligned. Therefore the remainder of the block can contain 0-byte values.

TODO: The journal block size seems to be 4096 bytes, but could this be defined by the index entry size? It does not match the cluster block size.

Once the stream reaches maximum size the earliest USN change journal entries are removed from the stream and replaced with a sparse data run.

15.2.1. USN change journal entry

USN change journal entry version 2

The USN change journal entry version 2 (USN_RECORD or USN_RECORD_V2) is variable of size and consists of:

Offset	Size	Value	Description
0	4		Entry (or record) size

Offset	Size	Value	Description
4	2	0x0002	Major version
6	2	0x0000	Minor version
8	8		File reference
16	8		Parent file reference
24	8		Update sequence number (USN) Contains the file offset of the USN change journal entry which is used as a unique identifier
32	8		Update date and time Contains a FILETIME
40	4		Update reason flags See section: Update reason flags
44	4		Update source flags See section: Update source flags
48	4		Security descriptor identifier Contains the entry number in the security ID index ($Secure:$SII) See section: Access Control
52	4		File attribute flags See section: File attribute flags
56	2		Name size Contains the byte size of the name
58	2		Name offset The offset is relative from the start of the USN change journal entry

Offset	Size	Value	Description
Common			
60	(name size)		Name
...	...	0x00	Padding

USN change journal entry version 3

TODO need a sample to confirm this version of the USN record is actually used by NTFS. For now it is assumed that it is likely used by ReFS.

The USN change journal entry version 3 (USN_RECORD_V3) is variable of size and consists of:

Offset	Size	Value	Description
0	4		Entry (or record) size
4	2	0x0003	Major version
6	2	0x0000	Minor version
8	**16**		File reference
24	**16**		Parent file reference
40	8		Update sequence number (USN) Contains the file offset of the USN change journal entry which is used as a unique identifier
48	8		Update date and time Contains a FILETIME

Offset	Size	Value	Description
56	4		Update reason flags See section: Update reason flags
60	4		Update source flags See section: Update source flags
64	4		Security descriptor identifier Contains the entry number in the security ID index ($Secure:$SII) See section: Access Control
68	4		File attribute flags See section: File attribute flags
72	2		Name size Contains the byte size of the name
74	2		Name offset The offset is relative from the start of the USN change journal entry
Common			
78	(name size)		Name
...	...	0x00	Padding

The file reference has changed from a 64-bit to a 128-bit value which consists of:

Offset	Size	Value	Description
0	8		MFT entry index Note that the index value in the MFT entry is only 32-bit of size.
8	8		Sequence number

15.2.2. Update reason flags

Value	Identifier	Description
0x00000001	USN_REASON_DATA_OVERWRITE	The data in the file or directory is overwritten. **The default (unnamed) $DATA attribute was overwritten**
0x00000002	USN_REASON_DATA_EXTEND	The file or directory is extended **The default (unnamed) $DATA attribute was extended**
0x00000004	USN_REASON_DATA_TRUNCATION	The file or directory is truncated. **The default (unnamed) $DATA attribute was truncated**
0x00000010	USN_REASON_NAMED_DATA_OVERWRITE	One or more named data streams ($DATA attributes) of file were overwritten
0x00000020	USN_REASON_NAMED_DATA_EXTEND	One or more named data streams ($DATA attributes) of file were extended
0x00000040	USN_REASON_NAMED_DATA_TRUNCATION	One or more named data streams ($DATA attributes) of a file were truncated
0x00000100	USN_REASON_FILE_CREATE	The file or directory was created
0x00000200	USN_REASON_FILE_DELETE	The file or directory was deleted
0x00000400	USN_REASON_EA_CHANGE	The extended attributes of the file were changed
0x00000800	USN_REASON_SECURITY_CHANGE	The access rights (security descriptor) of a file or directory were changed
0x00001000	USN_REASON_RENAME_OLD_NAME	The name changed The USN change journal entry contains the old name

Value	Identifier	Description
0x00002000	USN_REASON_RENAME_NEW_NAME	The name changed The USN change journal entry contains the new name
0x00004000	USN_REASON_INDEXABLE_CHANGE	Content indexed status changed the file attribute: FILE_ATTRIBUTE_NOT_CONTENT_INDEXED was changed
0x00008000	USN_REASON_BASIC_INFO_CHANGE	Basic file or directory attributes changed One or more file or directory attributes were changed e.g. read-only, hidden, system, archive, or sparse attribute, or one or more time stamps.
0x00010000	USN_REASON_HARD_LINK_CHANGE	A hard link was created or deleted
0x00020000	USN_REASON_COMPRESSION_CHANGE	The file or directory was compressed or decompressed
0x00040000	USN_REASON_ENCRYPTION_CHANGE	The file or directory was encrypted or decrypted
0x00080000	USN_REASON_OBJECT_ID_CHANGE	The object identifier of a file or directory was changed
0x00100000	USN_REASON_REPARSE_POINT_CHANGE	The reparse point that in a file or directory was changed, or a reparse point was added to or deleted from a file or directory.
0x00200000	USN_REASON_STREAM_CHANGE	A named data stream ($DATA attribute) is added to or removed from a file, or a named stream is renamed
0x00400000	USN_REASON_TRANSACTED_CHANGE	**Unknown found in TxF USN change journal entry list**
0x80000000	USN_REASON_CLOSE	The file or directory was closed

15.2.3. Update source flags

Value	Identifier	Description
0x00000001	USN_SOURCE_DATA_MANAGEMENT	The operation added a private data stream to a file or directory. The modifications did not change the application data.
0x00000002	USN_SOURCE_AUXILIARY_DATA	The operation was caused by the operating system. Although a write operation is performed on the item, the data was not changed.
0x00000004	USN_SOURCE_REPLICATION_MANAGEMENT	The operation was caused by file replication

16. Alternate data streams

Data stream name	Description
♣BnhqlkugBim0elg1M1pt2tjdZe ♣SummaryInformation {4c8cc155-6c1e-11d1-8e41-00c04fb9386d}	Used to store property sets Where ♣ is Unicode character U+2663 also known as black club
{59828bbb-3f72-4c1b-a420-b51ad66eb5d3}.XPRESS	Used during remote differential compression
AFP_AfpInfo AFP_Resource	Used to store Macintosh operating system property lists
encryptable	Used to store attributes relating to thumbnails in the thumbnails database
favicon	Used to store favorite icons for web pages.
ms-properties	**Unknown (Used to store property sets)**
OECustomProperty	Used to store custom properties related to email files

Data stream name	Description
Zone.Identifier	Used to store URL security zones

16.1. ms-properties

```
00000000  e9 00 00 00 8c 00 00 00  31 53 50 53 53 f1 ef fc  |........1SPSS...|
00000010  39 e8 f3 4c a9 e7 ea 22  83 20 94 b8 25 00 00 00  |9..L...". ..%...|
00000020  6f 00 00 00 00 1f 10 00  00 01 00 00 00 08 00 00  |o...............|
00000030  00 50 00 72 00 69 00 76  00 61 00 74 00 65 00 00  |.P.r.i.v.a.t.e..|
00000040  00 25 00 00 00 6c 00 00  00 00 1f 00 00 00 09 00  |.%...l..........|
00000050  00 00 53 00 6b 00 79 00  44 00 72 00 69 00 76 00  |..S.k.y.D.r.i.v.|
00000060  65 00 00 00 00 00 11 00  00 00 6d 00 00 00 00 13  |e.........m.....|
00000070  00 00 00 00 0d 04 00 15  00 00 00 6e 00 00 00 00  |...........n....|
00000080  15 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000090  59 00 00 00 31 53 50 53  d6 b9 f9 b2 c4 fe d5 4d  |Y...1SPS.......M|
000000a0  94 d7 89 57 48 8c 80 7b  3d 00 00 00 03 00 00 00  |...WH..{=.......|
000000b0  00 1f 00 00 00 15 00 00  00 37 00 46 00 30 00 39  |.........7.F.0.9|
000000c0  00 35 00 31 00 34 00 39  00 30 00 32 00 37 00 38  |.5.1.4.9.0.2.7.8|
000000d0  00 34 00 38 00 45 00 44  00 21 00 31 00 30 00 33  |.4.8.E.D.!.1.0.3|
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00           |.............|
000000ed
```

16.2. Zone.Identifier

```
00000000  5b 5a 6f 6e 65 54 72 61  6e 73 66 65 72 5d 0d 0a  |[ZoneTransfer]..|
00000010  5a 6f 6e 65 49 64 3d 33  0d 0a                    |ZoneId=3..|
0000001a
```

17. Transactional NTFS (TxF)

As of Vista (**or windows server 2003?**) Transactional NTFS (TxF) was added.

In TxF the resource manager (RM) keeps track of transactional metadata and log files. The TxF related metadata files are stored in the metadata directory:

```
$Extend\$RmMetadata
```

17.1. Resource manager repair information

The resource manager repair information metadata file: $Extend\$RmMetadata\$Repair consists of the following data streams:

- the default (unnamed) data stream, **purpose unknown**
- the $Config data stream, contains the resource manager repair configuration information

17.1.1. Resource manager repair configuration information

The $Repair:$Config data streams contains:

TODO

```
00000000  01 00 00 00 01 00 00 00                           |........|
```

Offset	Size	Value	Description
0	4		**Unknown**
4	4		**Unknown**

17.2. Transactional NTFS (TxF) metadata directory

The transactional NTFS (TxF) metadata directory: $Extend\$RmMetadata\$Txf is used to isolate files for delete or overwrite operations.

File format? All files seem to start with similar information

17.3. TxF Old Page Stream (TOPS) file

The TxF Old Page Stream (TOPS) file: $Extend\$RmMetadata\$TxfLog\$Tops consists of the following data streams:

- the default (unnamed) data stream, contains metadata about the resource manager, such as its GUID, its CLFS log policy, and the LSN at which recovery should start
- the $T data stream, contains the file data that is partially overwritten by a transaction as opposed to a full overwrite, which would move the file into the Transactional NTFS (TxF) metadata directory

17.3.1. TxF Old Page Stream (TOPS) metadata

The $Tops default (unnamed) data streams contains:

TODO

Offset	Size	Value	Description
0	2	0x000a	**Unknown**
2	2	0x0064	**Size of TOPS metadata**
4	4	0x0001	**Unknown** **Number of resource managers/streams?**
8	16		Resource Manager (RM) identifier Contains a GUID

Offset	Size	Value	Description
24	8		**Unknown (empty)**
32	8		**Base (or log start)** LSN of TxFLog stream
40	8		**Unknown**
48	8		**Last** flushed LSN of TxFLog stream
56	8		**Unknown**
64	8		**Unknown (empty)**
72	8		**Restart LSN?**
80	20		**Unknown**

17.3.2. TxF Old Page Stream (TOPS) file data

The $Tops:$T data streams contains the file data that is partially overwritten by a transaction. It consists of multiple pending transaction XML-documents.

Note that the start of each sector contains 0x0001, is this a value indication the sector is empty? Or are there fix-up values stored somewhere else?

A pending transaction XML-document starts with an UTF-8 byte-order-mark. Is roughly contains the following data:

```
<?xml version='1.0' encoding='utf-8'?>
<PendingTransaction Version="2.0" Identifier="...">
   <Transactions>
      <Transaction TransactionId="...">
      <Install Application="..., Culture=..., Version=..., PublicKeyToken=...,
                           ProcessorArchitecture=..., versionScope=..."
               RefGuid="..."
               RefIdentifier="..."
               RefExtra="..."/>
      ...
      </Transaction>
   </Transactions>
   <ChangeList>
      <Change Family="..., Culture=..., PublicKeyToken=...,
                     ProcessorArchitecture=..., versionScope=..."
              New="..."/>
      ...
   </ChangeList>
   <POQ>
      <BeginTransaction id="..."/>

      <CreateFile path="..."
                  fileAttribute="..."/>
      <DeleteFile path="..."/>
      <MoveFile source="..." destination="..."/>
      <HardlinkFile source="..." destination="..."/>
      <SetFileInformation path="..."
                          securityDescriptor="binary base64:..."
                          flags="..."/>

       <CreateKey path="..."/>
       <SetKeyValue path="..."
                    name="..."
                    type="..."
                    encoding="base64"
                    value="..."/>
      <DeleteKeyValue path="..."
                      name="..."/>

      ...
   </POQ>
   <InstallerQueue Length="...">
      <Action Installer="..."
              Mode="..."
              Phase="..."
```

```
                Family="..., Culture=..., PublicKeyToken=...,
                       ProcessorArchitecture=..., versionScope=..."
                Old="..."
                New="..."/>

        ...
    </InstallerQueue >
</PendingTransaction>
```

17.4. Transactional NTFS (TxF) Common Log File System (CLFS) files

TxF uses a Common Log File System (CLFS) log store and the logged utility stream attribute named $TXF_DATA.

See `[RUSSNOVICH09]`, `[MSDN]` and `[LIBFSCLFS]` for more information about CLFS.

The base log file (BLF) of the TxF log store is:

```
$Extend\$RmMetadata\$TxfLog\TxfLog.blf
```

Commonly the corresponding container files are:

```
$Extend\$RmMetadata\$TxfLog\TxfLogContainer00000000000000000001
$Extend\$RmMetadata\$TxfLog\TxfLogContainer00000000000000000002
```

TxF uses a multiplexed log store which contains two streams:

- the KtmLog stream used for Kernel Transaction Manager (KTM) metadata records
- TxfLog stream, which contains the TxF log records.

17.5. Transactional data logged utility stream attribute

The transactional data ($TXF_DATA) logged utility stream attribute is 56 bytes of size and consist of:

Offset	Size	Value	Description
0	6		**Unknown (remnant data)**
6	8		Resource manager root file reference Contains an NTFS file reference that refers to the MFT
14	8		**USN index?**
22	8		File identifier (TxID) Contains a TxF file identifier
30	8		Data LSN Contains a CLFS LSN of file data transaction records
38	8		Metadata LSN Contains a CLFS LSN of file system metadata transaction records
46	8		Directory index LSN Contains a CLFS LSN of directory index transaction records
54	2		**Flags?** **Seen: 0x0000, 0x0002**

Note there can be more than 1 per MTF entry

18. Windows definitions

18.1. File attribute flags

The file attribute flags consist of the following values:

Value	Identifier	Description
0x00000001	FILE_ATTRIBUTE_READONLY	Is read-only
0x00000002	FILE_ATTRIBUTE_HIDDEN	Is hidden
0x00000004	FILE_ATTRIBUTE_SYSTEM	Is a system file or directory
0x00000008		Is a volume label **Not used by NTFS**
0x00000010	FILE_ATTRIBUTE_DIRECTORY	Is a directory **Not used by NTFS**
0x00000020	FILE_ATTRIBUTE_ARCHIVE	Should be archived
0x00000040	FILE_ATTRIBUTE_DEVICE	Is a device **Not used by NTFS**
0x00000080	FILE_ATTRIBUTE_NORMAL	Is normal None of the other flags should be set
0x00000100	FILE_ATTRIBUTE_TEMPORARY	Is temporary
0x00000200	FILE_ATTRIBUTE_SPARSE_FILE	Is a sparse file
0x00000400	FILE_ATTRIBUTE_REPARSE_POINT	Is a reparse point or symbolic link
0x00000800	FILE_ATTRIBUTE_COMPRESSED	Is compressed
0x00001000	FILE_ATTRIBUTE_OFFLINE	Is offline The data of the file is stored on an offline storage.
0x00002000	FILE_ATTRIBUTE_NOT_CONTENT_INDEXED	Do not index content The content of the file or directory should not be indexed by the indexing service.

Value	Identifier	Description
0x00004000	FILE_ATTRIBUTE_ENCRYPTED	Is encrypted
0x00008000		**Unknown (seen on Windows 95 FAT)**
0x00010000	FILE_ATTRIBUTE_VIRTUAL	Is virtual
The following flags are mainly used in the file name attribute and sparsely in the standard information attribute; *it could be that they have a different meaning in both types of attributes or that the standard information flags are not updated.* *For now the latter is assumed.*		
0x10000000		**Unknown** **Is directory (or has $I30 index ?) used instead of 0x00000010 ? Seen $Extend directory without this flag**
0x20000000		**Is index view** **(copy from corresponding bit in MFT record)**

19. Corruption scenarios

19.1. $FILE_NAME attribute with non-proper UTF-16 name

An MFT entry contains an $FILE_NAME attribute that contains a name that contains non-proper UTF-16.

```
parent file reference            : 9711-3
creation time                    : Oct 04, 2014 14:44:40.703125000 UTC
modification time                : Oct 04, 2014 14:48:37.562500000 UTC
entry modification time          : Oct 04, 2014 14:48:37.750000000 UTC
access time                      : Oct 04, 2014 14:48:37.562500000 UTC
allocated file size              : 40960
file size                        : 38926
file attribute flags              : 0x00000020
        Should be archived (FILE_ATTRIBUTE_ARCHIVE)

extended data                     : 0x00000000
name size                         : 36
name space                        : 1 (Windows)

name data:
00000040:       2e 00 63 dc 79 dc  67 dc 30 00 30 00 32 00     ..c.y. g.0.0.2.
00000050: 65 00 30 00 30 00 30 00  30 00 30 00 30 00 30 00   e.0.0.0. 0.0.0.0.
00000060: 31 00 61 00 39 00 34 00  66 00 62 00 36 00 30 00   1.a.9.4. f.b.6.0.
00000070: 62 00 35 00 30 00 37 00  66 00 30 00 36 00 38 00   b.5.0.7. f.0.6.8.
00000080: 30 00 66 00 39 00 36 00  38 00                     0.f.9.6. 8.
```

This edge case was observed on Windows XP in the Recycler directory. The same name is stored in the corresponding $I30 entry.

19.2. Data steam with inconsistent data flags

An MFT entry contains an $ATTRIBUTE_LIST attribute that contains multiple $DATA attributes. The $DATA attributes define a LZNT1 compressed data stream though only the first $DATA attribute has the compressed data flag set.

Note	It is unclear if this is a corruption scenario or not.

```
MFT entry: 220 information:
        Is allocated                    : true
        File reference                  : 220-59
        Base record file reference      : Not set (0)
        Journal sequence number         : 51876429013
        Number of attributes            : 5

Attribute: 1
        Type                            : $STANDARD_INFORMATION (0x00000010)
        Creation time                   : Jun 05, 2019 06:56:26.032730300 UTC
        Modification time               : Oct 05, 2019 06:56:04.150940700 UTC
        Access time                       : Oct 05, 2019 06:56:04.150940700 UTC
        Entry modification time           : Oct 05, 2019 06:56:04.150940700 UTC
        Owner identifier                  : 0
        Security descriptor identifier    : 5862
        Update sequence number            : 11553149976
        File attribute flags              : 0x00000820
                Should be archived (FILE_ATTRIBUTE_ARCHIVE)
                Is compressed (FILE_ATTRIBUTE_COMPRESSED)

Attribute: 2
        Type                              : $ATTRIBUTE_LIST (0x00000020)

Attribute: 3
        Type                              : $FILE_NAME (0x00000030)
        Parent file reference             : 33996-57
        Creation time                     : Jun 05, 2019 06:56:26.032730300 UTC
        Modification time                 : Oct 05, 2019 06:56:03.510061800 UTC
        Access time                       : Oct 05, 2019 06:56:03.510061800 UTC
        Entry modification time           : Oct 05, 2019 06:56:03.510061800 UTC
        File attribute flags              : 0x00000020
                Should be archived (FILE_ATTRIBUTE_ARCHIVE)
        Namespace                         : POSIX (0)
        Name                              : setupapi.dev.20191005_085603.log

Attribute: 4
        Type                              : $DATA (0x00000080)
        Data VCN range                    : 513 - 1103
        Data flags                        : 0x0000

Attribute: 5
        Type                              : $DATA (0x00000080)
        Data VCN range                    : 0 - 512
        Data size                         : 4487594 bytes
        Data flags                        : 0x0001
```

19.3. Directory entry with outdated file reference

The directory entry: \ProgramData\McAfee\Common Framework\Task\5.ini

```
File entry:
        Path                            : \ProgramData\McAfee\Common Framework\Task\5.ini
        File reference                  : 51106-400
        Name                            : 5.ini
        Parent file reference           : 65804-10
        Size                            : 723
        Creation time                   : Sep 16, 2011 20:47:54.561041200 UTC
        Modification time               : Apr 07, 2012 21:07:02.684060000 UTC
        Access time                     : Apr 07, 2012 21:07:02.652810200 UTC
        Entry modification time         : Apr 07, 2012 21:07:02.684060000 UTC
        File attribute flags            : 0x00002020
                Should be archived (FILE_ATTRIBUTE_ARCHIVE)
                Content should not be indexed (FILE_ATTRIBUTE_NOT_CONTENT_INDEXED)
```

The corresponding MFT entry:

```
MFT entry: 51106 information:
        Is allocated                    : true
        File reference                  : 51106-496
        Base record file reference      : Not set (0)
        Journal sequence number         : 0
        Number of attributes            : 3

Attribute: 1
        Type                            : $STANDARD_INFORMATION (0x00000010)
        Creation time                   : Sep 16, 2011 20:47:54.561041200 UTC
        Modification time               : Apr 07, 2012 21:07:02.684060000 UTC
        Access time                     : Apr 07, 2012 21:07:02.652810200 UTC
        Entry modification time         : Apr 07, 2012 21:07:02.684060000 UTC
        Owner identifier                : 0
        Security descriptor identifier  : 1368
        Update sequence number          : 1947271600
        File attribute flags            : 0x00002020
```

```
                Should be archived (FILE_ATTRIBUTE_ARCHIVE)
                Content should not be indexed (FILE_ATTRIBUTE_NOT_CONTENT_INDEXED)

Attribute: 2
        Type                            : $FILE_NAME (0x00000030)
        Parent file reference           : 65804-10
        Creation time                   : Sep 16, 2011 20:47:54.561041200 UTC
        Modification time               : Apr 07, 2012 21:07:02.652810200 UTC
        Access time                     : Apr 07, 2012 21:07:02.652810200 UTC
        Entry modification time         : Apr 07, 2012 21:07:02.652810200 UTC
        File attribute flags            : 0x00002020
                Should be archived (FILE_ATTRIBUTE_ARCHIVE)
                Content should not be indexed (FILE_ATTRIBUTE_NOT_CONTENT_INDEXED)
        Namespace                       : DOS and Windows (3)
        Name                            : 1.ini

Attribute: 3
        Type                            : $DATA (0x00000080)
        Data size                       : 723 bytes
        Data flags                      : 0x0000
```

TODO look into using $LogFile

19.4. LZNT1 compressed block with data size of 0

Not sure if this is a corruption scenario or a data format edge case.

A compression unit (index 30) consisting of the following data runs:

```
reading data run: 60.
data run:
00000000: 11 01 01                                          ...

value sizes                             : 1, 1
number of cluster blocks                : 1 (size: 4096)
cluster block number                    : 687143 (1) (offset: 0xa7c27000)
```

```
reading data run: 61.
data run:
00000000: 01 0f                                               ..

value sizes                               : 1, 0
number of cluster blocks                  : 15 (size: 61440)
cluster block number                      : 0 (0) (offset: 0x00000000)
        Is sparse
```

Contains the following data:

```
a7c27000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
...
a7c27ff0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

This relates to an empty LZNT1 compressed block.

```
compressed data offset                    : 0 (0x00000000)
compression chunk header                  : 0x0000
compressed chunk size                     : 1
signature value                           : 0
is compressed flag                        : 0
```

It was observed in 2 differnt NTFS implementations that the entire block is filled with 0-byte values.

TODO: verify behavior of Windows NTFS implementation.

19.5. Truncated LZNT1 compressed block

Not sure if this is a corruption scenario or a data format edge case.

A compression unit (index 0) consisting of the following data runs:

```
reading data run: 0.
data run:
00000000: 31 08 48 d8 01                                     1.H..

value sizes                             : 1, 3
number of cluster blocks                : 8 (size: 32768)
cluster block number                    : 120904 (120904) (offset: 0x1d848000)

reading data run: 1.
data run:
00000000: 01 08                                              ..

value sizes                             : 1, 0
number of cluster blocks                : 8 (size: 32768)
cluster block number                    : 0 (0) (offset: 0x00000000)
        Is sparse
```

Contains the following data:

```
1d848000  bd b7 50 44 46 50 00 01  00 01 00 40 e0 00 07 0b  |..PDFP.....@....|
...
1d84c000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
1d84fff0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

This relates to a LZNT1 compressed block that appears to be truncated at offset 16384 (0x00004000).

```
compressed data offset                  : 16384 (0x00004000)
compression flag byte                   : 0x00
```

Different behavior was observed in 2 differnt NTFS implementations: * one implementation fills the compressed block with the uncompressed data it could read and the rest with with 0-byte values * another implementation seems to provide the data that was already in its buffer

TODO: verify behavior of Windows NTFS implementation.

20. Notes

Signatures as indicated in `[RUSSON05]` :

```
        "HOLE" == ??? (NTFS 3.0+?)
```

20.1. NTFS reserved file names

MFT entry index	Filename	Description
	\$Extend\$Config	Used for NTFS repair activity
	\$Extend\$Delete	Delete file name
	\$Extend\$Repair.log	Repair log name
	\$Extend\$Tops	
	\$Extend\$TxfLog	Transactional NTFS log

20.2. File system flags

```
fsutil fsinfo volumeinfo C:
```

```
FILE_CASE_PRESERVED_NAMES
0x00000002

The specified volume supports preserved case of file names when it places a name on disk.

FILE_CASE_SENSITIVE_SEARCH
0x00000001

The specified volume supports case-sensitive file names.

FILE_FILE_COMPRESSION
0x00000010

The specified volume supports file-based compression.

FILE_NAMED_STREAMS
0x00040000

The specified volume supports named streams.

FILE_PERSISTENT_ACLS
0x00000008

The specified volume preserves and enforces access control lists (ACL). For example, the NTFS file system preserves

FILE_READ_ONLY_VOLUME
0x00080000

The specified volume is read-only.

    Windows 2000:  This value is not supported.

FILE_SEQUENTIAL_WRITE_ONCE
0x00100000

The specified volume supports a single sequential write.
```

```
    Windows 2000:  This value is not supported.

FILE_SUPPORTS_ENCRYPTION
0x00020000

The specified volume supports the Encrypted File System (EFS). For more information, see File Encryption.

FILE_SUPPORTS_EXTENDED_ATTRIBUTES
0x00800000

The specified volume supports extended attributes. An extended attribute is a piece of application-specific metadata

    Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP/2000:  This value is not supported until

FILE_SUPPORTS_HARD_LINKS
0x00400000

The specified volume supports hard links. For more information, see Hard Links and Junctions.

    Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP/2000:  This value is not supported until

FILE_SUPPORTS_OBJECT_IDS
0x00010000

The specified volume supports object identifiers.

FILE_SUPPORTS_OPEN_BY_FILE_ID
0x01000000

The file system supports open by FileID. For more information, see FILE_ID_BOTH_DIR_INFO.

    Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP/2000:  This value is not supported until

FILE_SUPPORTS_REPARSE_POINTS
0x00000080
```

```
    The specified volume supports re-parse points.

    FILE_SUPPORTS_SPARSE_FILES
    0x00000040

    The specified volume supports sparse files.

    FILE_SUPPORTS_TRANSACTIONS
    0x00200000

    The specified volume supports transactions. For more information, see About KTM.

        Windows 2000:  This value is not supported.

    FILE_SUPPORTS_USN_JOURNAL
    0x02000000

    The specified volume supports update sequence number (USN) journals. For more information, see Change Journal Record

        Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP/2000:  This value is not supported until

    FILE_UNICODE_ON_DISK
    0x00000004

    The specified volume supports Unicode in file names as they appear on disk.

    FILE_VOLUME_IS_COMPRESSED
    0x00008000

    The specified volume is a compressed volume, for example, a DoubleSpace volume.

    FILE_VOLUME_QUOTAS
    0x00000020

    The specified volume supports disk quotas.
```

20.3. $MFT metadata file only parsing

What file system metadata is missing:

- Attribute lists are stored outside the $MFT
- $I30 indexes are stored outside the $MFT
- How to map MFT entry to parent? what if parent MFT entry has changed?

Appendix A: References

`[CARRIER05]`

Title	File System Forensic Analysis
Author(s)	Brian Carrier
Date	2005
ISBN-10	0-321-26817-2

`[RUSSON05]`

Title	NTFS Documentation
Author(s)	Richard Russon, Yuval Fiedel
Date	2005
URL	https://flatcap.github.io/linux-ntfs/ntfs/

`[APPLE06]`

Title	ntfs_layout.h - NTFS associated on-disk structures
URL	https://opensource.apple.com/source/ntfs/ntfs-65.2/kext/ntfs_layout.h.auto.html

`[POLLARD06]`

Title	All about BIOS parameter blocks
Author(s)	Jonathan de Boyne Pollard
URL	https://jdebp.eu/FGA/bios-parameter-block.html

`[RUSSNOVICH09]`

Title	Windows Internals 5 - Covering Windows Server 2008 and Windows Vista
Author(s)	Mark E. Russinovich and David A. Solomon
Date	June 17, 2009
ISBN-13	978-0735625303

`[LIBBDE]`

Title	BitLocker Drive Encryption (BDE) format specification - Analysis of theBitLocker Drive Encryption (BDE) vol
Date	March 2011
Author(s)	Joachim Metz
URL	https://github.com/libyal/libbde/blob/main/documentation/BitLocker%20Drive%20Encryption%20(BDE)%20fo

`[LIBFSCLFS]`

Title	Common Log File System – Analysis of the Windows ARIES log system
Date	November 2010
Author(s)	Joachim Metz
URL	https://github.com/libyal/libfsclfs/blob/main/documenation/Common%20Log%20File%20System%20(CLFS).as

`[LIBVSHADOW]`

Title	Volume Shadow Snapshot (VSS) - Analysis the Windows NT VSS format
Date	March 2011
Author(s)	Joachim Metz
URL	https://github.com/libyal/libvshadow/blob/master/documentation/Volume%20Shadow%20Snapshot%20(VSS)

`[MSDN]`

Title	Microsoft Developer Network
URL	http://technet.microsoft.com/en-us/library/cc781134%28WS.10%29.aspx

Subject	Master File Table
URL	http://msdn.microsoft.com/en-us/library/bb470206(v=vs.85).aspx

Subject	Reserved file names
URL	http://msdn.microsoft.com/en-us/library/ff469234%28v=PROT.10%29.aspx

Subject	NTFS attribute types
URL	http://msdn.microsoft.com/en-us/library/ff469236%28PROT.10%29.aspx

Subject	Reparse point
URL	http://msdn.microsoft.com/en-us/library/aa365740%28VS.85%29.aspx http://msdn.microsoft.com/en-us/library/aa365511%28v=VS.85%29.aspx http://msdn.microsoft.com/en-us/library/dd541667%28PROT.13%29.aspx http://msdn.microsoft.com/en-us/library/cc232005%28v=PROT.13%29.aspx http://msdn.microsoft.com/en-us/library/cc232006%28v=PROT.13%29.aspx http://msdn.microsoft.com/en-us/library/cc232007%28v=PROT.13%29.aspx

Subject	Update (or change) journal
URL	http://msdn.microsoft.com/en-us/library/aa363798.aspx http://msdn.microsoft.com/en-us/library/aa363803%28VS.85%29.aspx http://msdn.microsoft.com/en-us/library/aa365722%28VS.85%29.aspx

Subject	Known Alternate Stream Names
URL	https://msdn.microsoft.com/en-us/library/dn365326.aspx

Subject	transactional NTFS
URL	http://msdn.microsoft.com/en-us/library/bb968806%28v=VS.85%29.aspx http://msdn.microsoft.com/en-us/library/bb986748%28VS.85%29.aspx http://msdn.microsoft.com/en-us/library/bb540368%28VS.85%29.aspx

`[WIKI]`

URL	http://en.wikipedia.org/wiki/NTFS http://en.wikipedia.org/wiki/BIOS_parameter_block http://en.wikipedia.org/wiki/Transactional_NTFS

`[WIN32PROG]`

Title	The Win32 Programming Tutorials For Fun – Appendix E. NTFS On-Disk Structure
URL	http://www.installsetupconfig.com/win32programming/1996%20AppE_apnilife.pdf

Appendix B: GNU Free Documentation License

Version 1.3, 3 November 2008 Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. http://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

1. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

2. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

3. State on the Title page the name of the publisher of the Modified Version, as the publisher.

4. Preserve all the copyright notices of the Document.

5. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

6. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

7. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.

8. Include an unaltered copy of this License.

9. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
10. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
11. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
12. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
13. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
14. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
15. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.