Master File Table ($MFT)

on-disk Structures (NTFS 3.1)

Contents:

The size of the $MFT FILE records can be either 1024 or 4096 bytes[1] and is determined at Volume creation (format) time. Most common record size is 1024 bytes. There are two ways to check the record size:

1. From the NTFS Volume Boot Record's 'Clusters per File Record Segment' signed byte value (offset 0x40):

0	JMP instruction
3	System Name
B	Bytes_per_sector
D	Sectors_per_cluster
E	Reserved sectors
15	Media descriptor (hex)
18	Sectors per track
1A	Nr of Heads
1C	Nr of Hidden sectors
24	Drive Select (INT 13h drive Nr)
28	Total_sectors_excl_backup_boot_sector
30	LCN of $MFT
38	LCN of $MFTMirr
40	Clusters Per File Record Segment
44	Clusters Per Index Buffer
48	32-bit serial number (hex)
48	32-bit SN (hex, reversed)
48	64-bit serial number (hex)
50	Checksum
1FE	Boot Signature

   When this byte has a negative value, that value is the power of 2 (e.g., 2^{10} which is 1024). When this byte is 0x01, the $MFT record size is 1 cluster long (4096 bytes).

   To quote Microsoft[2], "*If this number is positive (up to 0x7F), it represents Clusters per MFT record. If the number is negative (0x80 to 0xFF), the size of the File Record is 2 raised to the absolute value of this number.*"

2. The second way is to examine the first $MFT FILE record's Header at offset 0x1C (4 bytes) which shows the Physical Size of the record.

[1] Windows Internals 7th Edition, Part 2, page 656 (Master File Table)
[2] https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc781134(v=ws.10)?redirectedfrom=MSDN#ntfs-boot-sector

Record Header

Every $MFT FILE record has a 56 (0x38) byte long Header:

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	File Record Signature	4
4	4	Offset to MFT Record update sequence number (*relative to start of record*)	2
6	6	Number of fix up byte pairs	2
8	8	$LogFile Sequence Number (LSN)	8
16	10	FILE Record Sequence Number[3]	2
18	12	HardLink Count	2
20	14	Offset to 1st attribute (*relative to start of record*)	2
22	16	Allocation Status Flags	2
24	18	Logical Size of MFT record	4
28	1C	Physical Size of MFT record	4
32	20	Base Record number	6
38	26	Base Record Sequence Number	2
40	28	Next available attribute ID	2
42	2A	*MFT record Number *(high part)*[4]	2
44	2C	*MFT record Number *(low part)*	4
48	30	MFT Record update sequence Number	2
50	32	Update sequence Array - FixUp1	2
52	34	Update sequence Array - FixUp2	2
54	36	Reserved/Unused	2

When the $MFT FILE record size is 1024 bytes long, the 'Number of fix up byte pairs' equals to 3: one is the 'MFT Record update sequence Number', and 2 are the FixUp values (*one for each 512 bytes*). When the $MFT FILE record size is 4096 bytes long, the 'Number of fix up byte pairs' is 9: one is the 'MFT Record update sequence Number', and 8 are the FixUp values (*8*512=4096*). In those cases, the $MFT FILE record Header is 66 bytes long:

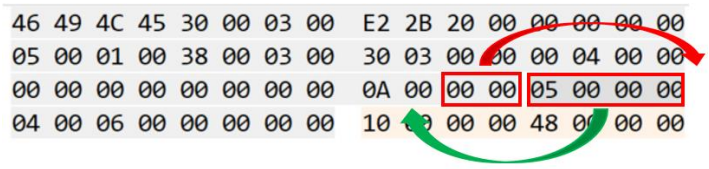
50	32	Update sequence Array – FixUp1	2
52	34	Update sequence Array – FixUp2	2
54	36	Update sequence Array – FixUp3	2
56	38	Update sequence Array – FixUp4	2
58	3A	Update sequence Array – FixUp5	2
60	3C	Update sequence Array – FixUp6	2
62	3E	Update sequence Array – FixUp7	2
64	40	Update sequence Array – FixUp8	2

[3] Record Sequence numbers start at 1. A value of 0 should only be found in unused records.
[4] https://dfir.ru/2019/01/19/ntfs-today/

*The full 48-bit MFT Record Number is obtained as follows:
(*after converting each to Little Endian*)

MFT record Number (low part)	MFT record Number (high part)

For example, in the following record header:



the MFT record number is: 0x 05 00 00 00 00 00 → 0x000000000005 → 5

The 64-bit 'File Record Number' or 'File Reference Number' or 'File Record ID' as it is also referred to, is the combination of the MFT Record Number and MFT Record sequence number:

bytes 8 - 7	***bytes 6 - 0***
MFT Record sequence nr	MFT Record Nr

The same applies to Base Record/Base Record sequence number and Parent Record/Parent Record sequence number.

For example, using fsutil.exe to view the MFT record information of a file with the 'fsutil file layout file-full-path' command will show the 64-bit File Reference Number instead of the $MFT File record number & sequence numbers:

```
********* File 0x0005000000000005 *********
File reference number   : 0x0005000000000005
File attributes         : 0x00000036: Hidden | System | Directory | Archive
File entry flags        : 0x00000000
Link (ParentID: Name)   : 0x0005000000000005: NTFS+DOS Name: \
Creation Time           : 14/7/2009 02:38:56
Last Access Time        : 4/9/2021 09:47:15
Last Write Time         : 4/9/2021 08:41:10
Change Time             : 4/9/2021 08:41:10
LastUsn                 : 103.181.310.128
```

In this example, the File Reference Number 0x0005000000000005 is MFT record 0x000000000005 (5) with MFT record sequence number 0x0005 (5).

The 'Allocation Status Flags' can be one:

- 0x0001 = MFT_RECORD_**IN_USE**
- 0x0002 = MFT_RECORD_**IS_DIRECTORY** (and has an $Index_Root Attribute)
- 0x0004 = MFT_RECORD_**IN_EXTEND** (i.e., File/Directory is in the $Extend directory)
- 0x0008 = MFT_RECORD_**IS_VIEW_INDEX** (not set if file has/is index $I30[5])

[5] https://opensource.apple.com/source/ntfs/ntfs-52/kext/ntfs_layout.h

or a combination of the above, thus are determined by which bits are set to 1.

Note: An interesting feature of the first 15 MFT records after the record for the $MFT itself (*record 0*), is that their MFT Record number and their MFT Record Sequence number are the same, and these records are not reused as other records. Ie:

FileName	MFT Rec. Nr.	MFT Seq.Nr.
$MFTMirr	1	1
$LogFile	2	2
$Volume	3	3
$AttrDef	4	4
. (Root Dir)	5	5
$Bitmap	6	6
$Boot	7	7
$BadClus	8	8
$Secure	9	9
$UpCase	10	10
$Extend	11	11
Reserved	12	12
Reserved	13	13
Reserved	14	14
Reserved	15	15

Attributes

Every record has at least one Attribute. The supported Attributes[6] on a an $MFT file can be seen by examining the **$AttrDef** NTFS Metafile in the same Volume, which also provides their general properties. The order in which Attributes appear in a FILE record is the same with the order they appear in the $AttrDef file.

The following screenshots are from a WinHex template[7] applied to a current $AttrDef file.
(*Minimum size of 0 means there is no minimum and a maximum size of -1 means no maximum.)*

[6] How NTFS Works (NTFS Boot Sector: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-fscc/a82e9105-2405-4e37-b2c3-28c773902d85)
[7] https://github.com/kacos2000/WinHex_Templates/blob/master/NTFS%20-%20%24AttrDef%20Structure.tpl

0	Attribute Label	$STANDARD_INFORMATION
80	Attribute Type (Decimal)	16
80	Attribute Type (Hex)	10 00 00 00
84	Display rule	00 00 00 00
88	Collation rule	00 00 00 00
88	-> Binary	0
8C	Flags	40 00 00 00
8C	-> Always Resident	40
90	Minimum Attribute Size	48
98	Maximum Attribute Size	72

A0	Attribute Label	$ATTRIBUTE_LIST
120	Attribute Type (Decimal)	32
120	Attribute Type (Hex)	20 00 00 00
124	Display rule	00 00 00 00
128	Collation rule	00 00 00 00
128	-> Binary	0
12C	Flags	80 00 00 00
12C	-> Allowed to be Non-Resident	80
130	Minimum Attribute Size	0
138	Maximum Attribute Size	-1

140	Attribute Label	$FILE_NAME
1C0	Attribute Type (Decimal)	48
1C0	Attribute Type (Hex)	30 00 00 00
1C4	Display rule	00 00 00 00
1C8	Collation rule	00 00 00 00
1C8	-> Binary	0
1CC	Flags	42 00 00 00
1CC	-> Always Resident & Indexed	42
1D0	Minimum Attribute Size	68
1D8	Maximum Attribute Size	578

1E0	Attribute Label	$OBJECT_ID
260	Attribute Type (Decimal)	64
260	Attribute Type (Hex)	40 00 00 00
264	Display rule	00 00 00 00
268	Collation rule	00 00 00 00
268	-> Binary	0
26C	Flags	40 00 00 00
26C	-> Always Resident	40
270	Minimum Attribute Size	0
278	Maximum Attribute Size	256

280	Attribute Label	$SECURITY_DESCRIPTOR
300	Attribute Type (Decimal)	80
300	Attribute Type (Hex)	50 00 00 00
304	Display rule	00 00 00 00
308	Collation rule	00 00 00 00
308	-> Binary	0
30C	Flags	80 00 00 00
30C	-> Allowed to be Non-Resident	80
310	Minimum Attribute Size	0
318	Maximum Attribute Size	-1

320	Attribute Label	$VOLUME_NAME
3A0	Attribute Type (Decimal)	96
3A0	Attribute Type (Hex)	60 00 00 00
3A4	Display rule	00 00 00 00
3A8	Collation rule	00 00 00 00
3A8	-> Binary	0
3AC	Flags	40 00 00 00
3AC	-> Always Resident	40
3B0	Minimum Attribute Size	2
3B8	Maximum Attribute Size	256

3C0	Attribute Label	$VOLUME_INFORMATION
440	Attribute Type (Decimal)	112
440	Attribute Type (Hex)	70 00 00 00
444	Display rule	00 00 00 00
448	Collation rule	00 00 00 00
448	-> Binary	0
44C	Flags	40 00 00 00
44C	-> Always Resident	40
450	Minimum Attribute Size	12
458	Maximum Attribute Size	12

460	Attribute Label	$DATA
4E0	Attribute Type (Decimal)	128
4E0	Attribute Type (Hex)	80 00 00 00
4E4	Display rule	00 00 00 00
4E8	Collation rule	00 00 00 00
4E8	-> Binary	0
4EC	Flags	00 00 00 00
4F0	Minimum Attribute Size	0
4F8	Maximum Attribute Size	-1

500	Attribute Label	$INDEX_ROOT
580	Attribute Type (Decimal)	144
580	Attribute Type (Hex)	90 00 00 00
584	Display rule	00 00 00 00
588	Collation rule	00 00 00 00
588	-> Binary	0
58C	Flags	40 00 00 00
58C	-> Always Resident	40
590	Minimum Attribute Size	0
598	Maximum Attribute Size	-1

5A0	Attribute Label	$INDEX_ALLOCATION
620	Attribute Type (Decimal)	160
620	Attribute Type (Hex)	A0 00 00 00
624	Display rule	00 00 00 00
628	Collation rule	00 00 00 00
628	-> Binary	0
62C	Flags	80 00 00 00
62C	-> Allowed to be Non-Resident	80
630	Minimum Attribute Size	0
638	Maximum Attribute Size	-1

640	Attribute Label	$BITMAP
6C0	Attribute Type (Decimal)	176
6C0	Attribute Type (Hex)	B0 00 00 00
6C4	Display rule	00 00 00 00
6C8	Collation rule	00 00 00 00
6C8	-> Binary	0
6CC	Flags	80 00 00 00
6CC	-> Allowed to be Non-Resident	80
6D0	Minimum Attribute Size	0
6D8	Maximum Attribute Size	-1

6E0	Attribute Label	$REPARSE_POINT
760	Attribute Type (Decimal)	192
760	Attribute Type (Hex)	C0 00 00 00
764	Display rule	00 00 00 00
768	Collation rule	00 00 00 00
768	-> Binary	0
76C	Flags	80 00 00 00
76C	-> Allowed to be Non-Resident	80
770	Minimum Attribute Size	0
778	Maximum Attribute Size	16.384

780	Attribute Label	$EA_INFORMATION
800	Attribute Type (Decimal)	208
800	Attribute Type (Hex)	D0 00 00 00
804	Display rule	00 00 00 00
808	Collation rule	00 00 00 00
808	-> Binary	0
80C	Flags	40 00 00 00
80C	-> Always Resident	40
810	Minimum Attribute Size	8
818	Maximum Attribute Size	8

820	Attribute Label	$EA
8A0	Attribute Type (Decimal)	224
8A0	Attribute Type (Hex)	E0 00 00 00
8A4	Display rule	00 00 00 00
8A8	Collation rule	00 00 00 00
8A8	-> Binary	0
8AC	Flags	00 00 00 00
8B0	Minimum Attribute Size	0
8B8	Maximum Attribute Size	65.536

8C0	Attribute Label	$LOGGED_UTILITY_STREAM
940	Attribute Type (Decimal)	256
940	Attribute Type (Hex)	00 01 00 00
944	Display rule	00 00 00 00
948	Collation rule	00 00 00 00
948	-> Binary	0
94C	Flags	80 00 00 00
94C	-> Allowed to be Non-Resident	80
950	Minimum Attribute Size	0
958	Maximum Attribute Size	65.536

A summary of the above can be seen in the following table:

	Attribute Label	Type	Flags	Non-Resident	Indexed	Min Size	Max Size
1	$STANDARD_INFORMATION	10000000	0x40	Never	-	48	72
2	$ATTRIBUTE_LIST	20000000	0x80	Maybe	-	-	-
3	$FILE_NAME	30000000	0x42	Never	Yes	68	578
4	$OBJECT_ID	40000000	0x40	Never	-	-	256
5	$SECURITY_DESCRIPTOR	50000000	0x80	Maybe	-	-	-
6	$VOLUME_NAME	60000000	0x40	Never	-	2	256
7	$VOLUME_INFORMATION*	70000000	0x40	Never	-	12	12 *(24)*
8	$DATA	80000000	0x00	-	-	-	-
9	$INDEX_ROOT	90000000	0x40	Never	-	-	-
10	$INDEX_ALLOCATION	A0000000	0x80	Maybe	-	-	-
11	$BITMAP	B0000000	0x80	Maybe	-	-	-
12	$REPARSE_POINT	C0000000	0x80	Maybe	-	-	16384
13	$EA_INFORMATION	D0000000	0x40	Never	-	8	8
14	$EA**	E0000000	0x00	*(Maybe)*	-	-	65536
15	$LOGGED_UTILITY_STREAM	00010000	0x80	Maybe	-	-	65536

The content of Attributes with a back-color is Always Resident within the $MFT

* Ntfs.sys (*File version 10.0.21390.1, WinBuild.160101.0800*) from the same Windows installation contains a copy of the $AttrDef file (*offset 0x1C0074D60)*, in which the $Volume_Information attribute's Maximum Attribute Size has changed from 12 to 24 bytes long.

** Also, the $EA Attribute's flags have changed from 0x00000000 to 0x80000000 signifying that $EA is Allowed to be Non-Resident.

Attribute Header

All Attributes start with a common Header with the following structure:

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	Attribute Type (eg. 0x80000000 = $Data)	4
4	4	Attribute Length	4
8	8	Non-Resident Status flag	1
9	9	Length of Stream Name (*Number of Unicode characters*[8])	1
10	A	Offset to Stream Name (*from the start of the Attribute*)	2
12	C	Attribute Flags	2
14	E	Attribute Identifier (ID) *(starting from 0)*	2
16	10	Resident Content Size	4
20	14	Resident Content Offset (*from the start of the Attribute*)	2
22	16	Indexed flag[9]	1
23	17	Reserved/Unused	1
Variable		Stream Name	Variable[8]

The minimum Attribute Header length is 24 bytes long, whether it has a Stream Name or not.

The offset of the first Attribute in a FILE record is determined by the 'Offset to 1st attribute' in the Record Header.

To get to the 2nd Attribute, one must add the 'Attribute Length' of the 1st Attribute to the 'Offset to 1st attribute', etc. Eg:

'Offset to 1st attribute'+ '1st Attribute Length' → Start of 2nd Attribute
'Offset to 1st attribute'+ '1st Attribute Length' + '2nd Attribute Size' → Start of 3rd Attribute

Some known system Stream names:

Stream Name	Description
$I30	Filename index
Zone.Identifier[10]	URL security zone Info
$DSC[11]	Desired Storage Class
$TXF_DATA	Transactional NTFS data
$EFS	Encrypted File Stream
$EfsBackup	Online Encryption Backup

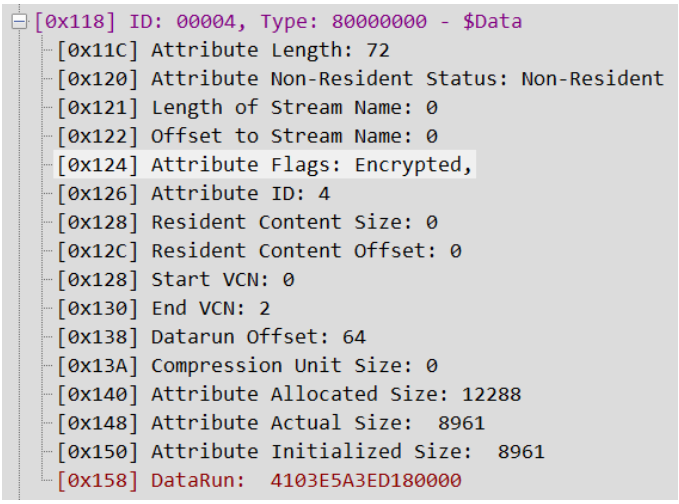
[8] Unicode characters are 2-byte values so the length in bytes is 2x this number.
[9] According to https://opensource.apple.com/source/ntfs/ntfs-52/kext/ntfs_layout.h (*line 696*), "*RESIDENT_ATTR_IS_INDEXED = 0x01, /* Attribute is referenced in an Index*".
I've seen this bit 'set to 1' only in $File_Name Attributes, so it is most likely linked to $I30.
[10] Defined at HKCU:\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones:
0 - Local Machine Zone
1 - Local intranet Zone
2 - Trusted sites Zone
3 - Internet Zone
4 - Restricted Sites Zone
[11] Windows Internals - Part#2 - CHAPTER 11 - Caching and file systems - p.663

$SDS	Security Data Stream
$SDH	Security Descriptor Hash
$SII	Security Identifier Index
$SRAT	Storage Reserve Allocation Table
$Q	Quota index
$O	ObjectID Index (in $ObjectID), OwnerID Index (in $Quota)
$R	Reparse Point index

The 'Attribute Flags' bytes have 3 currently observed[12] values:

0x0001	Attribute Content is Compressed
0x8000	Attribute Content is Sparse
0x4000	Attribute Content is Encrypted

These flags determine the Attribute content. For example, the content of the following $Data Attribute is Encrypted:



If the 'Non-Resident Status' flag has a value of 0x00, the Attribute content is **Resident** within the Attribute. Attributes with content always Resident within the $MFT Record are:

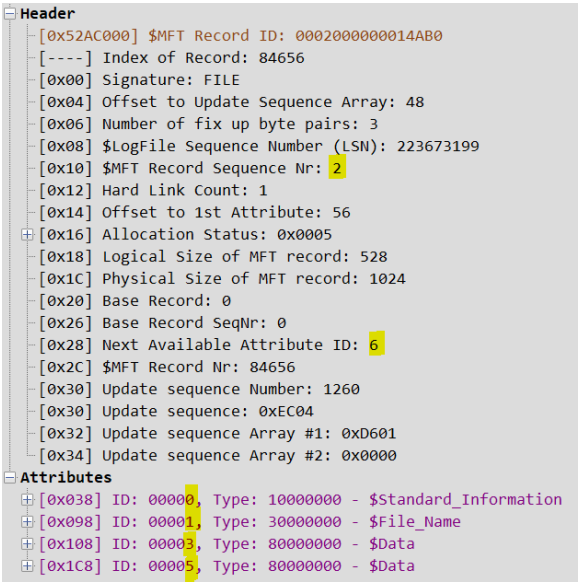
- Standard Information
- File Name
- Object ID
- Volume Information
- Volume Name
- Index Root
- Extended Attribute Information

Attributes with content that might be Resident are:

12 https://github.com/libyal/libfsntfs/blob/main/documentation/New%20Technologies%20File%20System%20(NTFS).asciidoc#mft_attribute_data_flags

- Attribute list
- Security descriptor
- Data
- Index Allocation
- Bitmap
- Reparse Point
- Extended attribute ($EA)
- Logged utility stream

If the 'Non-Resident Status' byte has a value of 0x01, the Attribute Content is too large to fit within the FILE record (**Non-Resident**), thus it is stored in other location(s) within the Volume. The 'map' of this (these) location(s) is the Attribute's 'Datarun'. The Non-Resident Status flag also determines the rest of the Attribute Structure.



Every Attribute is assigned an ID number, determined by the order the respective Attribute is written on the record. The number of Attributes a record has (*or has had*) can be determined by subtracting 1 from the Record Header's 'Next available attribute ID'. For example, this record of $UsnJrnl [13] on the left, shows that next available ID is 6, but only has 4 Attributes with IDs 0,1,3 and 5, meaning that Attributes with IDs 2 and 4 were deleted. The respective 'MFT Record Sequence number' is 2, indicating that this is the 2nd version of the record. The 'Update sequence' number shows that the record was updated 1260 times.

Note: *All Offsets after the Attribute header are from the 'Resident Content Offset' value of the Attribute header (unless noted otherwise).*

[13] The Case of the Stolen Szechuan Sauce (https://dfirmadness.com/the-stolen-szechuan-sauce/)

Non-Resident Attributes

The Non-Resident Attribute structure (*after the Attribute Header*) is:

Offset (D)	Offset (H)	Description	Length (bytes)
16	10	Start VCN	8
24	18	End VCN	8
32	20	DataRun offset	2
34	22	Compression[14] Unit Size[15]	2
36	24	Reserved/Unused	4
40	28	Attribute Content Allocated Size	8
48	30	Attribute Content Actual Size	8
56	38	Attribute Content Initialized Size	8
64	40	Attribute Total Allocated Size*	8
Variable	Variable	DataRun	Variable[16]

The 'Attribute Total Allocated Size'[17] is only found in attributes where the Attribute Flag 0x0001 (*Attribute Content is Compressed*) or 0x8000 (*Attribute Content is Sparse*) is set. The 'DataRun offset' in those Attributes is changed from the typical 0x40 (64) to 0x48 (72) to accommondate the 8 extra bytes:

[14] https://docs.microsoft.com/en-us/archive/blogs/ntdebugging/understanding-ntfs-compression

[15] Only seen in the $DATA Attribute and is used by native NTFS compression to store the 'compression unit' size as seen below:

Cluster Size	Compression Unit
512bytes	8kb (0x0200)
1kb	16kb (0x0400)
2kb	32kb (0x0800)
4kb	64kb (0x1000)
8kb	64kb (0x1000)
16kb	64kb (0x1000)
32kb	64kb (0x1000)
64kb	64kb (0x1000)

[16] To determine the Size of the 'Datarun', all that is needed is to Subtract the 'Datarun offset' from the 'Attribute Length'.

[17] The sum of the allocated clusters
https://docs.microsoft.com/en-us/windows/win32/devnotes/attribute-record-header

```
struct {
  VCN      LowestVcn;
  VCN      HighestVcn;
  USHORT   MappingPairsOffset;
  UCHAR    Reserved[6];
  LONGLONG AllocatedLength;
  LONGLONG FileSize;
  LONGLONG ValidDataLength;
  LONGLONG TotalAllocated;
} Nonresident;
```

Resident Attributes

The content structure of Resident Attributes depends on the Attribute type, so each will be examined separately. These structures follow the common Attribute Header.

Note: *Offsets below start from the 'Resident Content Offset' (unless specified otherwise).*

$STANDARD_INFORMATION

This Attribute is always Resident and must be present in every Base record.

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	File Create time[18]	8
8	8	File Modified time[18]	8
16	10	$MFT Record Modified time[18]	8
24	18	File Last Accessed time[18]	8
32	20	File Type Flags[19]	4
36	*24*	*Max Number of Versions**	*4*
40	*28*	*Version Number****	*4*
44	*2C*	*Class ID**	*4*
48	*30*	*Owner ID**	*4*
52	*34*	*Security ID**	*4*
56	*38*	*Quota Charged**	*8*
64	*40*	*Update sequence number (USN)**	*8*

***The existence (or not) of these entries depends on the Attribute content size and OS Version. If the 'Max Number of Versions' is 0 (0x00000000), Max Versions is disabled, and so is the 'Version Number'.*

[18] All Timestamps in the $MFT are in UTC and stored in the 64-bit Filetime format: https://docs.microsoft.com/en-us/windows/win32/sysinfo/file-times

[19] https://docs.microsoft.com/en-us/windows/win32/fileio/file-attribute-constants

Since Windows 10 version version 1803[20], part of the 'Version Number' bytes has been used for the "Is Case Sensitive" flag[21]:

40	*28*	*Is Case Sensitive*	*1*

A value of 1 means that the Directory is Case Sensitive.

Since Windows 10 version version 1809, part of the 'Version Number' bytes has been used for the "Reserve Storage ID" flag:

41	*29*	*Reserve Storage ID*	*1*

Possible 'Reserve Storage ID' values[22] are:

Hex	Binary	Reserve ID	Use
00	00000000	None	File or directory is not associated with any particular storage reserve area
01	00000001	Hard	Used by the system to help ensure there is sufficient disk space to download and install updates
02	00000010	Soft	Used by the system to help ensure there is sufficient disk space to install updates
03	00000011	Max	Marks the limit of predefined storage reserve IDs currently understood by the system

The Reserve Storage IDs can be queried with Windows's fsutil.exe tool:

```
fsutil storagereserve query c:
```

will show the reserved space by each Storage Reserve ID:

```
PS C:\> fsutil storagereserve query  c:

Reserve ID:       1
Flags:            0x00000001
Space Guarantee:  0x15ed3d000       (5613 MB)
Space Used:       0xf9c4b000        (3996 MB)

Reserve ID:       2
Flags:            0x00000000
Space Guarantee:  0x60000000        (1536 MB)
Space Used:       0x9766000         (151 MB)

Reserve ID:       3
Flags:            0x00000001
Space Guarantee:  0x0               (0 MB)
Space Used:       0x0               (0 MB)
```

[20] https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/ntifs/nf-ntifs-ntsetinformationfile (*FileCaseSensitiveInformation*)

[21] https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/ntifs/ns-ntifs-_file_case_sensitive_information

[22] https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/ntifs/ne-ntifs-_storage_reserve_id

And to get a list of MFT file records associated with a specific Storage Reserve ID:

```
fsutil storagereserve FindById c: X
```

where X is either 1,2 or 3.

The 'File Type flags[19]' can be any or a combination of the following:

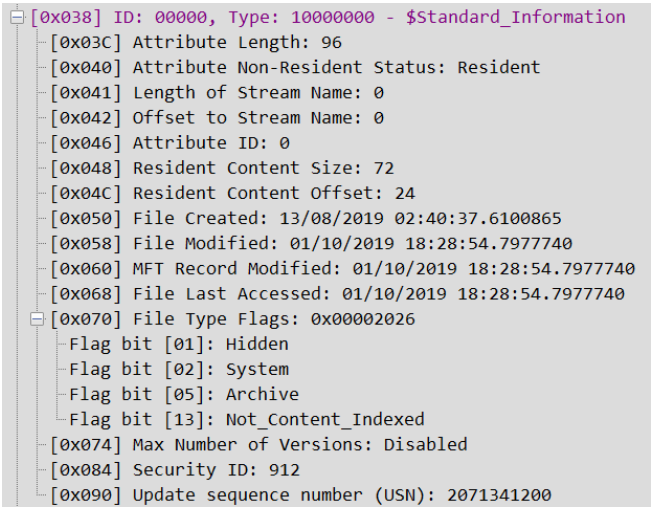
Hex	Binary	Description
0x00000001	0000-0000-0000-0000-0000-0000-0000-0001	ReadOnly
0x00000002	0000-0000-0000-0000-0000-0000-0000-0010	Hidden
0x00000004	0000-0000-0000-0000-0000-0000-0000-0100	System
0x00000010	0000-0000-0000-0000-0000-0000-0001-0000	Directory
0x00000020	0000-0000-0000-0000-0000-0000-0010-0000	Archive
0x00000040	0000-0000-0000-0000-0000-0000-0100-0000	Device
0x00000080	0000-0000-0000-0000-0000-0000-1000-0000	Normal
0x00000100	0000-0000-0000-0000-0000-0001-0000-0000	Temporary
0x00000200	0000-0000-0000-0000-0000-0010-0000-0000	Sparse_File
0x00000400	0000-0000-0000-0000-0000-0100-0000-0000	Reparse_Point
0x00000800	0000-0000-0000-0000-0000-1000-0000-0000	Compressed
0x00001000	0000-0000-0000-0000-0001-0000-0000-0000	Offline
0x00002000	0000-0000-0000-0000-0010-0000-0000-0000	Not_Content_Indexed
0x00004000	0000-0000-0000-0000-0100-0000-0000-0000	Encrypted
0x00008000	0000-0000-0000-0000-1000-0000-0000-0000	Integrity Stream
0x00010000	0000-0000-0000-0001-0000-0000-0000-0000	Virtual
0x00020000	0000-0000-0000-0010-0000-0000-0000-0000	No_Scrub_Data
0x00040000	0000-0000-0000-0100-0000-0000-0000-0000	Recall_On_Open
0x00400000	0000-0000-0100-0000-0000-0000-0000-0000	Recall_On_DataAccess
0x80000000	1000-0000-0000-0000-0000-0000-0000-0000	TxF related flag

A short description of their meaning can be seen in the following table:

Flag	Description
ReadOnly	A file that is read-only. Applications can read the file but cannot write to it or delete it.
Hidden	The file or directory is hidden. It is not included in an ordinary directory listing.
System	A file or directory that the operating system uses a part of or uses exclusively.
Directory	The handle that identifies a directory
Archive	A file or directory that is an archive file or directory. Applications typically use this attribute to mark files for backup or removal
Device	reserved for system use
Normal	A file that does not have other attributes set. This attribute is valid only when used alone.
Temporary	A file that is being used for temporary storage.
Sparse_File	A file that is a sparse file
Reparse_Point	A file or directory that has an associated reparse point, or a file that is a symbolic link.

Compressed	A file or directory that is compressed. For a file, all the data in the file is compressed. For a directory, compression is the default for newly created files and subdirectories.
Offline	The data of a file is not available immediately. This attribute indicates that the file data is physically moved to offline storage.
Not_Content_Indexed	The file or directory is not to be indexed by the content indexing service.
Encrypted	A file or directory that is encrypted. For a file, all data streams in the file are encrypted. For a directory, encryption is the default for newly created files and subdirectories.
Integrity Stream	The directory or user data stream is configured with integrity (only supported on ReFS volumes)
Virtual	reserved for system use
No_Scrub_Data	The user data stream not to be read by the background data integrity scanner (AKA scrubber). When set on a directory it only provides inheritance. This flag is only supported on Storage Spaces and ReFS volumes.
Recall_On_Open	When this attribute is set, it means that the file or directory has no physical representation on the local system; the item is virtual.
Recall_On_DataAccess	When this attribute is set, it means that the file or directory is not fully present locally.
TxF related flag	Undocumented

Example of a $Standard_Information Attribute:



$ATTRIBUTE_LIST

When a single FILE record cannot contain all the Attributes for that record, NTFS will offload these attributes to other FILE records, linked to the original FILE record by the Base MFT Record/sequence numbers. This is done with an '$Attribute_List' Attribute in the **Base** record.

The '$Attribute_List' Attribute can be either Resident or Non-Resident (*in which case, while the contents are not in the $MFT, the structure is the same*):

Offset (D)	Offset (H)	Description	Length (bytes)

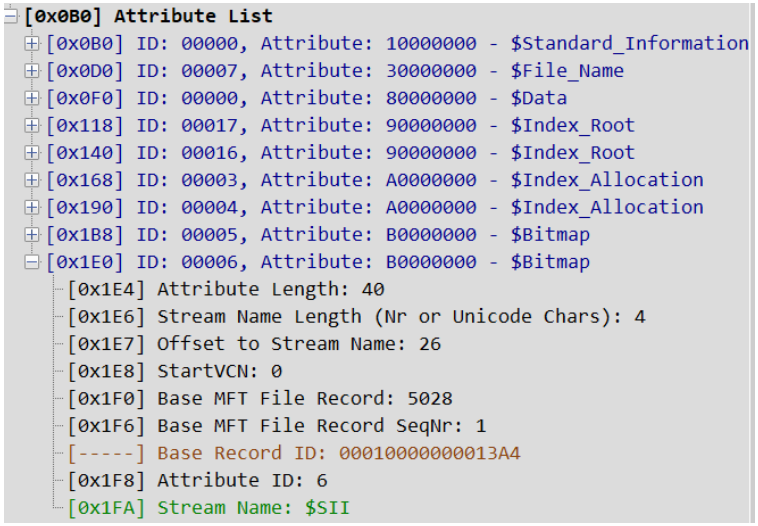
0	0	Attribute Type	4
4	4	Attribute Length	2
6	6	Stream Name Length *(Nr of Unicode chars)*	1
7	7	Offset to Stream Name	1
8	8	Attribute StartVCN	8
16	10	Attribute Base MFT record	6
22	16	Attribute Base MFT record Sequence Number	2
24	18	Attribute ID	2
Variable		Stream Name	Stream Name Length*2

The Offset to the start of the 1st '$Attribute_List' Attribute entry, starts from the 'Resident Content Offset'. The Offset to the start of the subsequent Attributes is variable, as seen below. To get all possible Attributes listed in an Attribute list, one must follow this simple process:

'*Resident Content Offset'+ '1st Attribute Length' = Offset to start of 2nd Attribute*
'Resident Content Offset'+ '1st Attribute Length'+'2nd Attribute Length' = Offset to start of 3rd Attribute ...

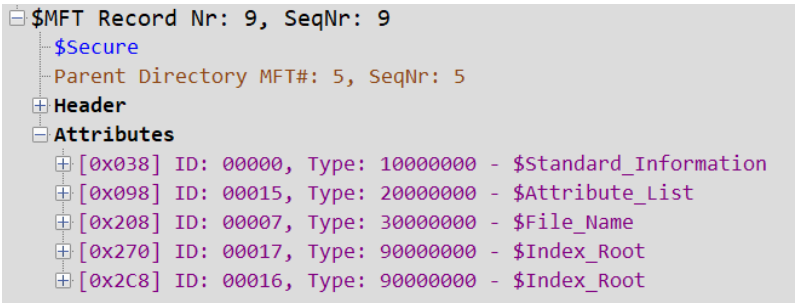
and keep going, until either the 'Resident Content Size' is reached by the above Sums (*in case of Resident $Attribute_List Attributes*) or the next 'Attribute Type' is either 0xFFFFFFFF or 0x00000000 (*for Non-Resident $Attribute_List Attributes)*.

Example of the Resident $Attribute_List of a $Secure NTFS Metadata file record:



In the above $Attribute_List Attribute with ID 6 points to Base record 5028 with Base record sequence number 1:

So, the Base record of the $Secure file *($MFT record 9, Sequence nr. 9)* has 5 Attributes:



And the rest of the Attributes are in the Child or Extended record 5028:

As a rule, if an $MFT Record Header has Base Record & Base Record Sequence numbers other than 0, this record is an extension of another record, and the Base Record will have an $Attribute_List Attribute.

$FILE_NAME

This Attribute is always Resident. A FILE record can have multiple Filename Attributes, for each type of Namespace or for Hard Links. Their structure is shown below:

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	$MFT Record number of the parent directory	6
6	6	Sequence number of the parent directory	2
8	8	File Create time[23]	8
16	10	File Modified time[23]	8
24	18	$MFT Record Modified time[23]	8
32	20	File Last Accessed time[23]	8
40	28	File Allocated Size	8
48	30	File Real Size	8
56	38	File Type Flags	4
60	3C	Used by $EA and $Reparse_Point	4
64	40	File name length *(Number of Unicode characters)*	1
65	41	File name type *(Namespace)*	1
66	42	File name	File name length *2

The 'File Type flags[24]' can be none, one or a combination of the following:

Hex	Binary	Description
0x00000001	0000-0000-0000-0000-0000-0000-0000-0001	ReadOnly
0x00000002	0000-0000-0000-0000-0000-0000-0000-0010	Hidden
0x00000004	0000-0000-0000-0000-0000-0000-0000-0100	System
0x00000010	0000-0000-0000-0000-0000-0000-0001-0000	Directory

[23] All Timestamps in the $MFT are in UTC and stored in the 64-bit Filetime format: https://docs.microsoft.com/en-us/windows/win32/sysinfo/file-times

[24] https://docs.microsoft.com/en-us/windows/win32/fileio/file-attribute-constants

0x00000020	0000-0000-0000-0000-0000-0000-0010-0000	Archive
0x00000040	0000-0000-0000-0000-0000-0000-0100-0000	Device
0x00000080	0000-0000-0000-0000-0000-0000-1000-0000	Normal
0x00000100	0000-0000-0000-0000-0000-0001-0000-0000	Temporary
0x00000200	0000-0000-0000-0000-0000-0010-0000-0000	Sparse_File
0x00000400	0000-0000-0000-0000-0000-0100-0000-0000	Reparse_Point
0x00000800	0000-0000-0000-0000-0000-1000-0000-0000	Compressed
0x00001000	0000-0000-0000-0000-0001-0000-0000-0000	Offline
0x00002000	0000-0000-0000-0000-0010-0000-0000-0000	Not_Content_Indexed
0x00004000	0000-0000-0000-0000-0100-0000-0000-0000	Encrypted
0x00008000	0000-0000-0000-0000-1000-0000-0000-0000	Integrity Stream
0x00010000	0000-0000-0000-0001-0000-0000-0000-0000	Virtual
0x00020000	0000-0000-0000-0010-0000-0000-0000-0000	No_Scrub_Data
0x00040000	0000-0000-0000-0100-0000-0000-0000-0000	Recall_On_Open
0x00400000	0000-0000-0100-0000-0000-0000-0000-0000	Recall_On_DataAccess
0x10000000	0001-0000-0000-0000-0000-0000-0000-0000	Directory
0x20000000	0010-0000-0000-0000-0000-0000-0000-0000	Index_view

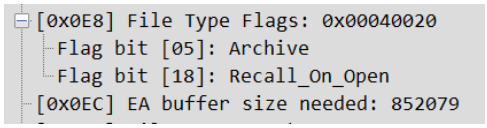
A short description[24] of their meaning can be seen in the following table:

Flag	**Description**
ReadOnly	A file that is read-only. Applications can read the file but cannot write to it or delete it.
Hidden	The file or directory is hidden. It is not included in an ordinary directory listing.
System	A file or directory that the operating system uses a part of or uses exclusively.
Directory	The handle that identifies a directory
Archive	A file or directory that is an archive file or directory. Applications typically use this attribute to mark files for backup or removal
Device	reserved for system use
Normal	A file that does not have other attributes set. This attribute is valid only when used alone.
Temporary	A file that is being used for temporary storage.
Sparse_File	A file that is a sparse file
Reparse_Point	A file or directory that has an associated reparse point, or a file that is a symbolic link.
Compressed	A file or directory that is compressed. For a file, all the data in the file is compressed. For a directory, compression is the default for newly created files and subdirectories.
Offline	The data of a file is not available immediately. This attribute indicates that the file data is physically moved to offline storage.
Not_Content_Indexed	The file or directory is not to be indexed by the content indexing service.
Encrypted	A file or directory that is encrypted. For a file, all data streams in the file are encrypted. For a directory, encryption is the default for newly created files and subdirectories.
Integrity Stream	*The directory or user data stream is configured with integrity (only supported on ReFS volumes)*
Virtual	reserved for system use
No_Scrub_Data	The user data stream not to be read by the background data integrity scanner (AKA scrubber). When set on a directory it only provides inheritance. This flag is only supported on Storage Spaces and ReFS volumes.

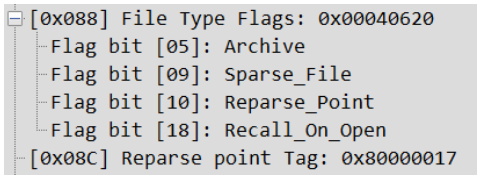
Recall_On_Open	When this attribute is set, it means that the file or directory has no physical representation on the local system; the item is virtual.
Recall_On_DataAccess	When this attribute is set, it means that the file or directory is not fully present locally.
Directory	copy from corresponding bit in MFT record header[25] flag (IS_DIRECTORY)
Index_view	copy from corresponding bit in MFT record header[25] flag (IS_VIEW_INDEX)

The 'Used by $EA and $Reparse_Point' bytes value & usage depend on the whether either 'Reparse_Point' or 'Recall_On_Open' $File_Name Attribute flag bits are set:

- If the 'Recall_On_Open' flag bits are set and the 'Reparse_Point' bits are not, the 'Used by $EA and $Reparse_Point' bytes contain the size of the $EA buffer needed. The FILE record will also have an $EA Attribute:



- If the 'Reparse_Point' flag bits are set, the 'Used by $EA and $Reparse_Point' bytes contain the Reparse Point Tag[26]. The FILE record will also have a $Reparse_Point attribute:



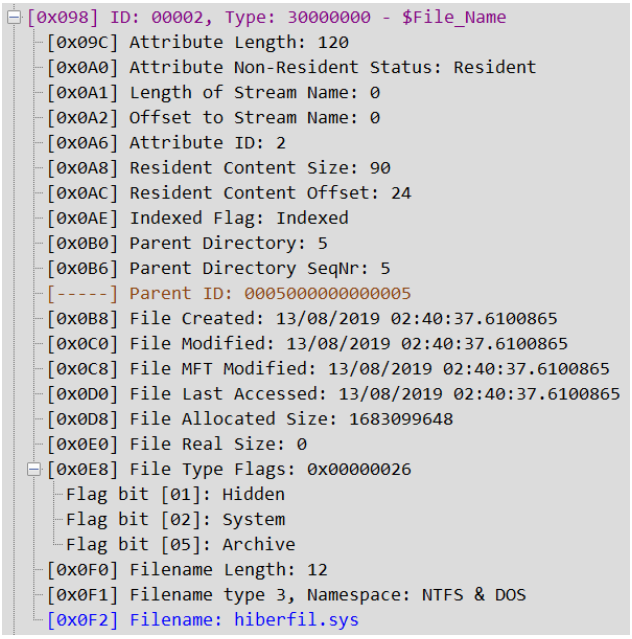
Note: According to Microsoft, "*Reparse points and extended attributes are mutually exclusive. The NTFS file system cannot create a reparse point when the file contains extended attributes, and it cannot create extended attributes on a file that contains a reparse point.*"[27]

The 'File name type (Namespace)' byte can have any of the following values:

Hex	Description
0x00	POSIX
0x01	NTFS (LFN)
0x02	DOS (SFN)
0x03	NTFS & DOS

[25] http://inform.pucp.edu.pe/~inf232/Ntfs/ntfs_doc_v0.5/attributes/file_name.html
[26] Reparse point tag: A unique identifier for a file system filter driver stored within a file's optional reparse point data that indicates the file system filter driver that performs additional filter-defined processing on a file during I/O operations. https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-fscc/c8e77b37-3909-4fe6-a4ea-2b9d423b1ee4
[27] https://docs.microsoft.com/en-us/windows/win32/fileio/reparse-points

Example of a $File_Name Attribute:



$OBJECT_ID

The Structure[28.29] of the $Object_ID Attribute depends on the OS Version and its Resident content length. Most $Object_ID Attributes only have the first 16 bytes[30] (*Object ID type2*) of the structure below:

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	Object Id	16
16	10	Birth Volume Id	16
32	20	Birth Object Id	16
48	30	Domain Id	16

Each 16 bytes represent a GUID. A GUID is organized in 5 byte groups:

[4bytes (*LE*)] - [2-bytes (*LE*)] - [2bytes (*LE*)] - [2bytes (*BE*)] - [6bytes (*BE*)][31], as seen below:

Byte group	**A**					**B**			**C**			**D**			**E**					
Raw bytes	0	1	2	3	-	4	5	-	6	7	-	8	9	-	10	11	12	13	14	15

[28] https://docs.microsoft.com/en-us/windows/win32/api/winioctl/ns-winioctl-file_objectid_buffer
[29] ObjectID type1 (https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-fscc/34a727a2-960a-4825-9cd2-6100c84e3a81)
[30] ObjectID type2 (https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-fscc/e9f94ce6-c61c-4f05-b772-af90f9d59d8f)
[31] LE = Little-Endian, BE = Big-Endian

GUID order	3	2	1	0	-	5	4	-	7	6	-	8	9	-	10	11	12	13	14	15

For example, the Object ID bytes 0x3FDB4D9A-16FA-EA11-80BF-000C29E184E6 are translated to GUID: 9A4DDB3F-FA16-11EA-80BF-000C29E184E6:

Byte group	A					B			C			D			E					
Raw bytes	3F	DB	4D	9A	-	16	FA	-	EA	11	-	80	BF	-	00	0C	29	E1	84	E6
GUID order	9A	4D	DB	3F	-	FA	16	-	11	EA	-	80	BF	-	00	0C	29	E1	84	E6

The GUID Version, is found in the 3rd nimble of the 'C' byte group (*Raw Byte order*), or the 1st nimble of the same byte group (*GUID byte order*):

C
EA 11
11 EA

When this nimble is 4, the GUID is randomly or pseudo-randomly generated:

```
[0x118] ObjectID: 7A80E88E-C216-47F5-9C0D-47F4525ECF66
    GUID Version: 4
    GUID Variant: 2
    GUID Sequence: 7181
```

When the nimble is 1, then the GUID is time-based and generated using the date, time and MAC address of the computer node generating it:

```
[0x1A0] ObjectID: 9A4DDB3F-FA16-11EA-80BF-000C29E184E6
    GUID Version: 1
    GUID Variant: 2
    GUID Sequence: 191
    GUID created at: 19/09/2020 01:22:37.3239615
    MAC Address: 00:0C:29:E1:84:E6
```

More detailed, the possible GUID (*UUID*) versions are:

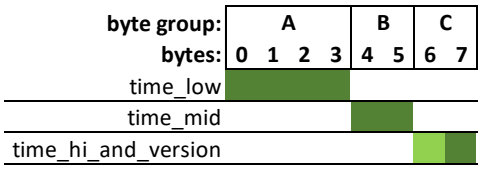
Hex	Binary	Version
1	0001	Time based*
2	0010	DCE Security version, with embedded POSIX UIDs
3	0011	Name based
4	0100	Randomly or pseudo-randomly generated*
5	0101	Name based with SHA-1 hashing

(** only versions 1 and 4 are used in the $Object_ID Attribute*)

The raw byte group '**D**' contains the GUID Variant and the GUID Sequence Number. In the above example (0x3FDB4D9A-16FA-EA11-80BF-000C29E184E6), byte group D is: 0x80BF, or in binary: 1000000010111111. The 2 left-most (*most significant bits*) comprise are the GUID Variant, and the rest 14 bits are the GUID Sequence Number. So, in this example

GUID Variant = **10**000000010111111 or 2 in Decimal, and the
GUID Sequence number = 10**00000010111111** or 191 in Decimal.

The first 8 raw bytes of the GUID contain the Timestamp of its creation (*in UTC*). This is in the form of a 60-bit value of the number of 100 nanosecond intervals since 00:00:00.00, 15 October 1582[32]. This 60-bit value is stored as:



To retrieve the timestamp, first the bytes must be reversed (*Little-endian*). Using the previous GUID example, the first 8 raw bytes are: 0x3FDB4D9A-16FA-EA11. But this is still a 64-bit value.

To get the 60-bit value, either the last nimble (*4 bits == GUID Version*) must be removed, or even simpler, the 1 can be replaced with a 0 (*4 bits equal to 0000*). Converted to Little-Endian the resulting 60-bit value in Hex is 0x01EAFA169A4DDB3F or 138197713573239615 in decimal.

In order for this value to be converted to FILETIME[33], the difference between 12:00 A.M. October 15, 1582, and 12:00 A.M. January 1, 1601 (*which FILETIME is using*) must be found.

Using PowerShell:

```
PS C:\> $1582offset = (New-Object DateTime(1582, 10, 15, 0, 0, 0)).Ticks
PS C:\> $1601offset = (New-Object DateTime(1601, 1, 1, 0, 0, 0)).Ticks
PS C:\> $1601offset - $1582offset
5748192000000000
```

So, the GUID's FILETIME value is: 138197713573239615 – 5748192000000000 = 132449521573239615. To translate this number to timestamp using PowerShell:

```
PS C:\> [datetime]::FromFileTimeUtc(132449521573239615).ToString("dd/MM/yyyy HH:mm:ss.fffffff")
19/09/2020 01:22:37.3239615
```

and Command Prompt:

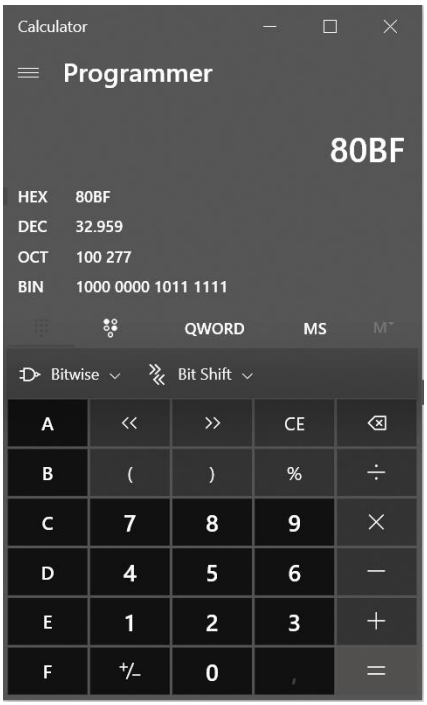
```
C:\>w32tm /ntte 132449521573239615
153298 01:22:37.3239615 - 9/19/2020 4:22:37 AM
```

In Time-based GUIDs (*version 1*), byte Group '**E**'s 6-bytes comprise the MAC Address. In the above example, byte group 'E' is 0x000C29E184E6, which translates to MAC Address: 00:0C:29:E1:84:E6. (*Using a site such as https://macaddress.io/ can provide more information on the NIC card manufacturer using this MAC address blocks.*)

[32] https://www.ietf.org/rfc/rfc4122.txt pages 5+
[33] 64-bit Filetime format: https://docs.microsoft.com/en-us/windows/win32/sysinfo/file-times

Note: For number conversions you can use Windows PowerShell or the Windows Calculator app (*Programmer mode*):

```
PS C:\> [Convert]::ToInt32("00000010111111", 2)
191
PS C:\> [convert]::tostring("0x80bf",2)
1000000010111111
PS C:\> 0x80BF
32959
PS C:\> [convert]::tostring("32959",16)
80bf
```



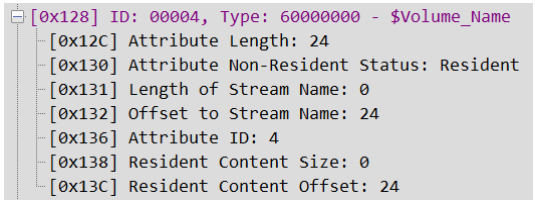
Further reading:
https://www.sciencedirect.com/science/article/pii/S1742287619300234
http://computerforensics.parsonage.co.uk/downloads/TheMeaningofLIFE.pdf
https://www.ietf.org/rfc/rfc4122.txt

$SECURITY_DESCRIPTOR

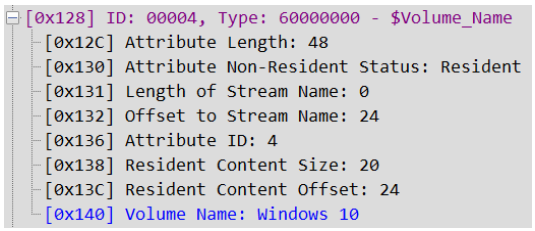
This Attribute is rarely used in NTFS 3.1 and is mostly provided for backward compatibility. A $Security_Descriptor Attribute can be found in the FILE record of NTFS Metafile $Boot.

$VOLUME_NAME

This Attribute only exists only in the FILE record of the $Volume NTFS Metafile, and its always Resident content is the Volume Name (*in Unicode*), if that has been set. Example of an attribute without a Volume Name – note that the Resident content size of this attribute is 0:



Example of an Attribute with Volume Name set – note that the Resident content size of this attribute is 20 (*10 Unicode characters*):



$VOLUME_INFORMATION

This Attribute exists only in the FILE record of the $Volume NTFS Metafile.
Its always Resident content has the following structure[34]:

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	Reserved/Unused	8
8	8	NTFS Major Version	1
9	9	NTFS Minor Version	1
10	20	Volume Flags	2
12	C	Reserved/Unused	4

34 https://github.com/libyal/libfsntfs/blob/main/documentation/New%20Technologies%20File%20System%20(NTFS).asciidoc#69-the-volume-information-attribute

For reference[35]:

Operating System	NTFS Version
Windows NT 4.0	1.2
Windows 2000	3.0
Windows XP+	3.1

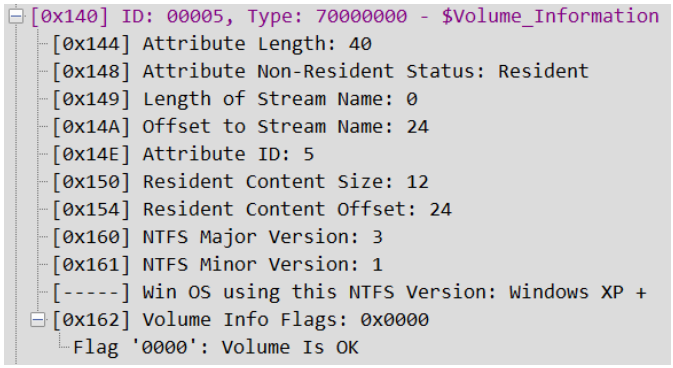
The 'Volume Flags' can have any or a combination of the following values[36]:

Hex	Binary	Description
0x0000	0000000000000000	Volume is OK
0x0001	0000000000000001	Is dirty
0x0002	0000000000000010	Re-size journal ($LogFile)
0x0004	0000000000000100	Upgrade Volume version underway
0x0008	0000000000001000	Mounted on Windows NT4
0x0010	0000000000010000	Delete USN underway
0x0020	0000000000100000	Repair Object IDs underway
0x0040	0000000001000000	Volume is corrupt and caused a bug check
0x0080	0000000010000000	Tunneling cache, short filenames disabled*
0x0100	0000000100000000	Full Chkdsk scan underway
0x0200	0000001000000000	Proactive scan underway
0x0400	0000010000000000	TxF feature is disabled
0x0800	0000100000000000	Volume scrub disabled
0x1000	0001000000000000	$Verify and $Corrupt disabled
0x2000	0010000000000000	Heat gathering disabled
0x4000	0100000000000000	Chkdsk underway
0x8000	1000000000000000	Modified by Chkdsk

***** *Since Windows 8.1, by default all the NTFS nonbootable volumes have short filenames disabled.*

Example of a $Volume_Information Attribute:

[35] https://en.wikipedia.org/wiki/NTFS#Versions
[36] https://dfir.ru/2019/01/19/ntfs-today/



$DATA

The $Data Attribute's Resident content has the full contents of a File (*if that file is small enough to fit within the record*). However, if the Attribute has a Stream Name, its Resident content may be further interpreted.

$UpCase:$Info

Since Windows 8[37], the $UpCase NTFS Metafile, has a second $Data Attribute with the 'Stream Name' "$Info". The **$UpCase:$Info:$Data** content is Resident, has a fixed size of 32 bytes, and the following structure[38]:

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	Length	4
4	4	Reserved/Unused	4
8	8	CRC (*64-bit*)[39]	8
16	10	Windows Major Version	4
20	14	Windows Minor Version	4
24	18	Windows Build number[40]	4
28	1C	Service Pack Major	2
30	1E	Service Pack Minor	2

Examples of such an Attribute:

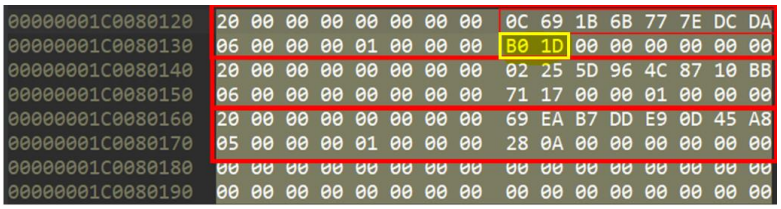
[37] https://twitter.com/errno_fail/status/1258900141479809025?s=20
[38] https://github.com/vitalif/ntfs-3g/blob/04d4b37a9a6c992d89c93193d0abdd13ab1f2931/ntfsprogs/mkntfs.c#L168
[39] https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-fciads/2cfdd8d0-8730-4340-b04e-8d588669fb35
[40] https://en.wikipedia.org/wiki/List_of_Microsoft_Windows_versions

```
[0x148] ID: 00004, Type: 80000000 - $Data
    [0x14C] Attribute Length: 72
    [0x150] Attribute Non-Resident Status: Resident
    [0x151] Length of Stream Name: 5
    [0x152] Offset to Stream Name: 24
    [0x156] Attribute ID: 4
    [0x158] Resident Content Size: 32
    [0x15C] Resident Content Offset: 40
  [0x160] Stream Name: $Info
      [0x170] Stream Length: 32
      [0x178] CRC64: 0xDADC7E776B1B690C
      [0x180] OS Version (Major): 6
      [0x184] OS Version (Minor): 3
      [0x188] Windows build number: 9600
      [0x18C] Pack (Major): 0
      [0x18E] Pack (Minor): 0
```

```
[0x148] ID: 00004, Type: 80000000 - $Data
    [0x14C] Attribute Length: 72
    [0x150] Attribute Non-Resident Status: Resident
    [0x151] Length of Stream Name: 5
    [0x152] Offset to Stream Name: 24
    [0x156] Attribute ID: 4
    [0x158] Resident Content Size: 32
    [0x15C] Resident Content Offset: 40
  [0x160] Stream Name: $Info
      [0x170] Stream Length: 32
      [0x178] CRC64: 0xDADC7E776B1B690C
      [0x180] OS Version (Major): 10
      [0x184] OS Version (Minor): 0
      [0x188] Windows build number: 22000
      [0x18C] Pack (Major): 0
      [0x18E] Pack (Minor): 0
```

The Windows/OS Version information shown, refers to the version used to create the $UpCase file *(ie. format the Volume)*. The 64-bit CRC value is used to check if the $Upcase character mapping table is valid. On boot or volume mount, NTFS, compares the known values stored in 'ntfs.sys' against the values stored in the volume's $UpCase:$Info:

Raw hexadecimal values from 'ntfs.sys''s 'NtfsKnownUpcaseInfoArray':



The current CRC64 value has been the same since Windows 7 Build 7600 (or 0xB01D as seen above)

Raw hexadecimal values from an $UpCase:$Info attribute content:

20	00	00	00	00	00	00	00	0C	69	1B	6B	77	7E	DC	DA
0A	00	00	00	00	00	00	00	39	38	00	00	00	00	00	00

If they don't match *(error: 'The upcase file content is incorrect')*, Chkdsk starts scanning the Volume for errors, and the $Volume metafile FILE record $Volume_Information attribute 'Volume Info Flags' get updated to:

```
[0x162] Volume Info Flags: 0x0101
    Flag bit [15]: Is Dirty
    Flag bit [07]: Full Chkdsk scan underway
```

The 3 entries in the 'NtfsKnownUpcaseInfoArray' are translated το:

Windows OS[41]	**Windows 7**	**Windows Vista**[42]	**Windows XP**
CRC-64	DADC7E776B1B690C	BB10874C965D2502	A8450DE9DDB7EA69
Windows Major Version	6	6	5

[41] https://en.wikipedia.org/wiki/List_of_Microsoft_Windows_versions

[42] https://en.wikipedia.org/wiki/List_of_Microsoft_Windows_versions#cite_note-Windows_Vista_build_number-6

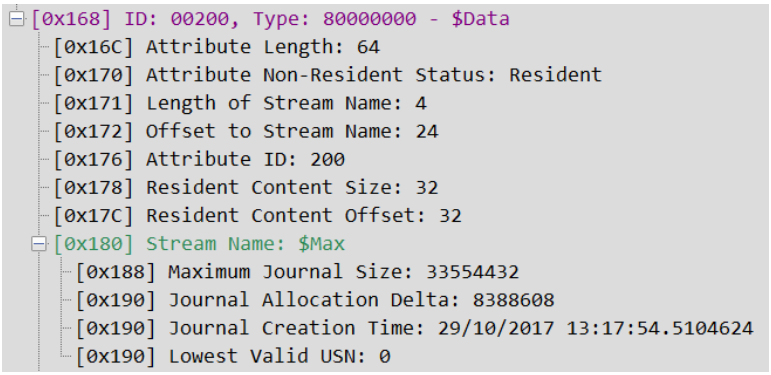
Windows Minor Version	1	0	0
Windows Build Number	7600	6001	2600
Service Pack Major	0	1	0
Service Pack Minor	0	0	0

$UsnJrnl:$Max

The $UsnJrnl NTFS Metafile has two $Data Attributes with Stream Names $J and $Max[43]. The $UsnJrnl:$J:$DATA[44] content is Non-Resident (*due to its size*), but the **$UsnJrnl:$Max:$Data[45]** content is Resident, has a fixed size of 32 bytes and the following structure:

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	Maximum Size of the Change Journal	8
8	8	Allocation Delta Size of the Change Journal	8
16	10	Creation Time[23] of the Journal	8
24	18	Lowest Valid USN	8

Example of such an Attribute:



The same information can be obtained (*on a live Volume*) with the fsutil[46] command:

[43] http://forensicinsight.org/wp-content/uploads/2013/07/F-INSIGHT-Advanced-UsnJrnl-Forensics-English.pdf
[44] Change Journal
[45] Change Journal Matadata
[46] https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/fsutil-usn

```
PS E:\> fsutil usn queryjournal g:
Usn Journal ID   : 0x01d350b8547610f0
First Usn        : 0x0000000024000000
Next Usn         : 0x00000000266f0a80
Lowest Valid Usn : 0x0000000000000000
Max Usn          : 0x00000fffffff0000
Maximum Size     : 0x0000000002000000 (32,0 MB)
Allocation Delta : 0x0000000000800000 ( 8,0 MB)
Minimum record version supported : 2
Maximum record version supported : 4
Write range tracking: Disabled
```

Where:
Maximum Size = 0x0000000002000000 = 33554432,
Allocation Delta = 8388608 and
Lowest Valid Usn = 0x0000000000000000 = 0

The Journal Creation time coincides with the $Standard_Information File Created time:

```
[0x050] File Created: 29/10/2017 13:17:54.5104624
[0x058] File Modified: 29/10/2017 13:17:54.5104624
[0x060] MFT Record Modified: 29/10/2017 13:17:54.5104624
[0x068] File Last Accessed: 29/10/2017 13:17:54.5104624
```

$Bitmap: $SRAT

Since Windows 10, version 1809, the $MFT FILE record of the $Bitmap NTFS Metafile, might have a second $Data Attribute with the Stream Name "$SRAT"[47], and Resident content. "*The $SRAT data stream contains a data structure that tracks each Reserve area, including the number of reserved and used clusters … It also stores and updates the on-disk $STANDARD_INFO attribute of the target file. The latter maintains 4 bits to store the Storage Reserve area ID.* "[47]

[47] Windows Internals, 7th Edition, CHAPTER 11 Caching and file systems, p686

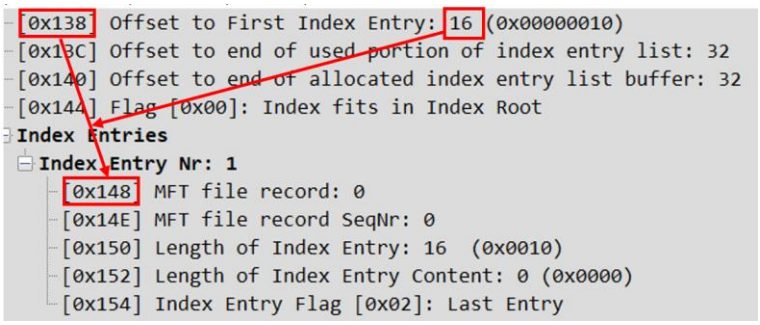
$INDEX_ROOT

The Resident content of this Attribute starts with this header structure:

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	Attribute Type	4
4	4	Collation Sorting Rule	4
8	8	Size of Index Record in bytes	4
12	C	Number of Clusters of Index Record	4
16	10	Offset* to 1st Entry	4
20	14	Offset* to end of used portion of index entry list	4
24	18	Offset* to end of allocated index entry list buffer	4
28	1C	Flag	4

**These 3 offsets are relative to 'Offset to 1st Entry' (the first byte of the field).*

For example, in the screenshot below 0x138 + 0x10 (decimal 16) = 0x148



The 'Collation Sorting Rule' indicates the type of sorting applied to the Index and its value[48] can be any of the following:

Hex	Description
00000000	Binary
00000001	File Name
00000002	Unicode String
00000010	Unsigned Long
00000011	SID
00000012	Security Hash
00000013	Multiple Unsigned Longs

The 'Flag' value can be either:

Hex	Description
0x00000000	Index fits in $Index_Root Attribute
0x00000001	Index points to $Index_Allocation Child Nodes

[48] https://opensource.apple.com/source/ntfs/ntfs-52/kext/ntfs_layout.h (Lines 502-547)

The rest of the Resident content structure is variable. After the header structure, follow the Index Entries (*at least one*), which have this common structure:

Offset (D)	Offset (H)	Description	Length (bytes)
0	*0*	*MFT file reference for file name or Unused (index)*	6
6	*6*	*MFT file SeqNr or Unused (index)*	2
8	8	Length of Entry	2
10	*A*	*Length of of Filename attribute ($I30 index only)*	*2*
12	C	Index Flags	1

where the 'Index Flags' can have any of the following values:

Hex	Description
0x00	Child node
0x01	Child node in `$Index_Allocation
0x02	Last Entry
0x03	Last Entry, Child node in `$Index_Allocation

Which determine if there are more Index node entries following or if this is the last entry. The 'Attribute Type' also determines the Structure of the Index Entries.

$I30

The most common 'Attribute Type' is 0x30000000 ($File_Name) is paired to Attribute Stream Name "$I30". In these $Index_Root attributes, if the Index Flag 'Last Entry' is not set, the Index node entry will have the following structure:

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	MFT file reference for file name or Unused (index)	6
6	6	MFT file SeqNr or Unused (index)	2
8	8	Length of Node Entry	2
10	A	Length of content *(length of Filename attribute)*	2
12	C	Index Flags	1
16	10	$MFT Record number of the parent directory	6
22	16	$MFT Record Sequence number of the parent directory	2
24	18	File Create time[18]	8
32	20	File Modified time[18]	8
40	28	$MFT Record Modified time[18]	8
48	30	File Last Accessed time[18]	8
56	38	File Allocated Size	8
64	40	File Real Size	8
72	48	File Type Flags	4
76	4C	$EA buffer size needed or $Reparse_Point Tag	4
80	50	Filename Length *(Nr of Unicode chars)*	1
81	51	File name type *(Namespace)*	1

82	52	File name	Filename Length*2

Example $I30 entry:

```
Index Entries
  Index Entry Nr: 1
    [0x160] MFT file record: 87867
    [0x166] MFT file record SeqNr: 1
    [-----] MFT record ID: 000100000001573B
    [0x168] Length of Index Entry: 136
    [0x16A] Length of Index Entry Content: 110
    [0x16C] Index Entry Flag [0x01]: Child node in $Index_Allocation
    [0x170] Parent Directory MFT record Nr.: 5
    [0x176] Parent Directory MFT record SeqNr.: 5
    [-----] Parent MFT record ID: 0005000000000005
    [0x178] File Created: 10/02/2017 09:42:04.7729127
    [0x180] File Modified: 10/02/2017 09:42:04.7729127
    [0x188] MFT Record Modified: 10/02/2017 09:42:04.7729127
    [0x190] Last Accessed: 10/02/2017 09:42:04.7729127
    [0x198] File Allocated Size: 0
    [0x1A0] File Real Size: 0
    [0x1A8] File Type Flags: 0x10002406
      Flag bit [01]: Hidden
      Flag bit [02]: System
      Flag bit [10]: Reparse_Point
      Flag bit [13]: Not_Content_Indexed
      Flag bit [28]: Directory
    [0x1AC] Reparse point Tag: 0xA0000003
    [-----] Reparse point Filter: IO_REPARSE_TAG_MOUNT_POINT
    [0x1B0] Filename Length (Nr of Unicode Chars): 22
    [0x1B1] Filename type: NTFS (LFN)
    [0x1B2] Filename: Documents and Settings
    [0x1E0] VCN of child node: 0
  Index Entry Nr: 2
    [0x1E8] MFT file record: 0
    [0x1EE] MFT file record SeqNr: 0
    [0x1F0] Length of Index Entry: 24
    [0x1F2] Length of Index Entry Content: 0
    [0x1F4] Index Entry Flag [0x03]: Last Entry, Child node in $Index_Allocation
```

Note: Other indexes (*NTFS Metadata files $Quota, $ObjId, $Secure and $Reparse*) have records with $Index_Root 'Attribute Type' of 0x00000000, and Stream Names ($Q and $O (*$Quota*), $O (*$ObjId*), $SII and $SDH (*$Secure*) and $R (*$Reparse*). The respective $Index_Root Attributes present in those $MFT FILE records have node entries with a different structure. (*Their Record Header 'Allocation Status flag' IS_VIEW_INDEX is also set).* Examples of such indexes:

$ObjId:$O

```
Index Entry Nr: 1
  [0x140] Index Entry Offset to Data: 32
  [0x142] Index Entry Data Size: 56
  [0x148] Index Entry Length: 96
  [0x14A] Index Entry Key Size: 16
  [0x14C] Index Entry Flag [0x01]: Child node in $Index_Allocation
  [0x150] Index Entry Key: DD6763C4-F9F7-11EA-95EC-000C2914C295
  [-----] Reference MFT record ID: 000100000001573B
  [0x160] Reference MFT record Nr: 87867
  [0x166] Reference MFT Sequence Nr: 1
  [0x168] Birth Volume ID: 51C576BE-87D2-4F7D-8CE8-D5DB07D72495
  [0x178] Birth ObjectID: DD6763C4-F9F7-11EA-95EC-000C2914C295
```

Where each ID is a GUID with the usual structure:

```
[0x150] Index Entry Key: DD6763C4-F9F7-11EA-95EC-000C2914C295
    GUID Version: 1
    GUID Variant: 2
    GUID Sequence: 5612
    GUID created at: 18/09/2020 21:42:35.5000260
    MAC Address: 00:0C:29:14:C2:95
```

$Quota:$O

Owner details for built-in account:

```
Index Entry Nr: 3
    [0x190] Index Entry Offset to Data: 32
    [0x192] Index Entry Data Size: 4
    [0x198] Index Entry Length: 40
    [0x19A] Index Entry Key Size: 16
    [0x19C] Index Entry Flag [0x00]: Child node
    [0x1A0] Index Entry Key: S-1-5-32-544
    [0x1A0] Index Entry Key: BUILTIN\Administrators
    [0x1B0] Owner ID: 256
```

Owner details for User:

```
Index Entry Nr: 4
    [0x1B8] Index Entry Offset to Data: 44
    [0x1BA] Index Entry Data Size: 4
    [0x1C0] Index Entry Length: 48
    [0x1C2] Index Entry Key Size: 28
    [0x1C4] Index Entry Flag [0x00]: Child node
    [0x1C8] Index Entry Key: S-1-5-21-3852348952-3567372440-4181163953-1001
    [0x1E4] Owner ID: 259
```

$Quota:$Q

Quota details for the Volume (*Index Entry Key 1*):

```
Index Entry Nr: 1
    [0x2A8] Index Entry Offset to Data: 20
    [0x2AA] Index Entry Data Size: 48
    [0x2B0] Index Entry Length: 72
    [0x2B2] Index Entry Key Size: 4
    [0x2B4] Index Entry Flag [0x00]: Child node
    [0x2B8] Index Entry Key: 1
    [0x2BC] Quota Version: 2
    [0x2C0] Quota Flag: Default Limits
    [0x2C0] Quota Flag: Tracking Enabled
    [0x2C0] Quota Flag: Enforcement Enabled
    [0x2C0] Quota Flag: Tracking Requested
    [0x2C4] Quota Bytes Used: 0
    [0x2CC] Quota Changed Time: 12/10/2021 14:48:10.8035176
    [0x2D4] Quota Warning Limit: 524288000
    [0x2DC] Quota Hard Limit: 734003200
    [0x2E4] Quota Exceeded Time: None
```

Quota for built-in account:

```
Index Entry Nr: 2
    [0x2F0] Index Entry Offset to Data: 20
    [0x2F2] Index Entry Data Size: 64
    [0x2F8] Index Entry Length: 88
    [0x2FA] Index Entry Key Size: 4
    [0x2FC] Index Entry Flag [0x00]: Child node
    [0x300] Index Entry Key: 256
    [0x304] Quota Version: 2
    [0x308] Quota Flag: Default Limits
    [0x30C] Quota Bytes Used: 361893888
    [0x314] Quota Changed Time: 12/10/2021 14:51:48.0415216
    [0x31C] Quota Warning Limit: No-Limit
    [0x324] Quota Hard Limit: No-Limit
    [0x32C] Quota Exceeded Time: None
    [0x334] Quota for Sid: S-1-5-32-544
    [0x334] Quota for Account: BUILTIN\Administrators
```

Quota for user:

```
Index Entry Nr: 5
    [0x3E8] Index Entry Offset to Data: 20
    [0x3EA] Index Entry Data Size: 76
    [0x3F0] Index Entry Length: 96
    [0x3F2] Index Entry Key Size: 4
    [0x3F4] Index Entry Flag [0x00]: Child node
    [0x3F8] Index Entry Key: 259
    [0x3FC] Quota Version: 2
    [0x400] Quota Flag: Default Limits
    [0x404] Quota Bytes Used: 815009792
    [0x40C] Quota Changed Time: 12/10/2021 14:51:12.5570258
    [0x414] Quota Warning Limit: 524288000
    [0x41C] Quota Hard Limit: 1073741824
    [0x424] Quota Exceeded Time: None
    [0x42C] Quota for Sid: S-1-5-21-3852348952-3567372440-4181163953-1001
```

Quota Entries for (E:)

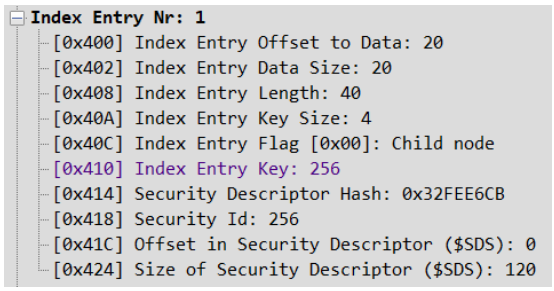
Quota Edit View Help

Status	Name	Logon Name	Amount Used	Quota Limit	Warning Level	Percent Used
Warning	Cos...	WIN11_4K_MFT_VM\kacos	771,67 MB	1 GB	500 MB	75
OK		BUILTIN\Administrators	345,12 MB	No Limit	No Limit	N/A
OK		NT AUTHORITY\NETWOR...	200 KB	1 GB	500 MB	0
OK		NT AUTHORITY\SYSTEM	20,14 MB	1 GB	500 MB	1
OK		NT SERVICE\TrustedInstaller	12 KB	1 GB	500 MB	0
OK		NT SERVICE\MapsBroker	4 KB	1 GB	500 MB	0

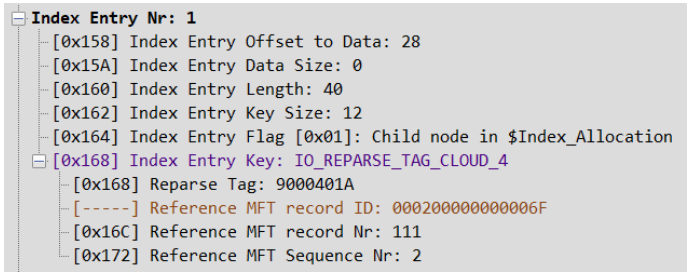
$Secure:$SDH

```
Index Entry Nr: 1
    [0x1A0] Index Entry Offset to Data: 24
    [0x1A2] Index Entry Data Size: 20
    [0x1A8] Index Entry Length: 48
    [0x1AA] Index Entry Key Size: 8
    [0x1AC] Index Entry Flag [0x00]: Child node
    [0x1B0] Index Entry Key Sid: 260
        [0x1B0] Key - Security Descriptor Hash: 0x068E5891
        [0x1B4] Key - Security Id: 260
    [0x1B8] Security Descriptor Hash: 0x068E5891
    [0x1BC] Security Id: 260
    [0x1C0] Offset in Security Descriptor ($SDS): 512
    [0x1C8] Size of Security Descriptor ($SDS): 116
```

$Secure:$SII



and **$Reparse:$R**

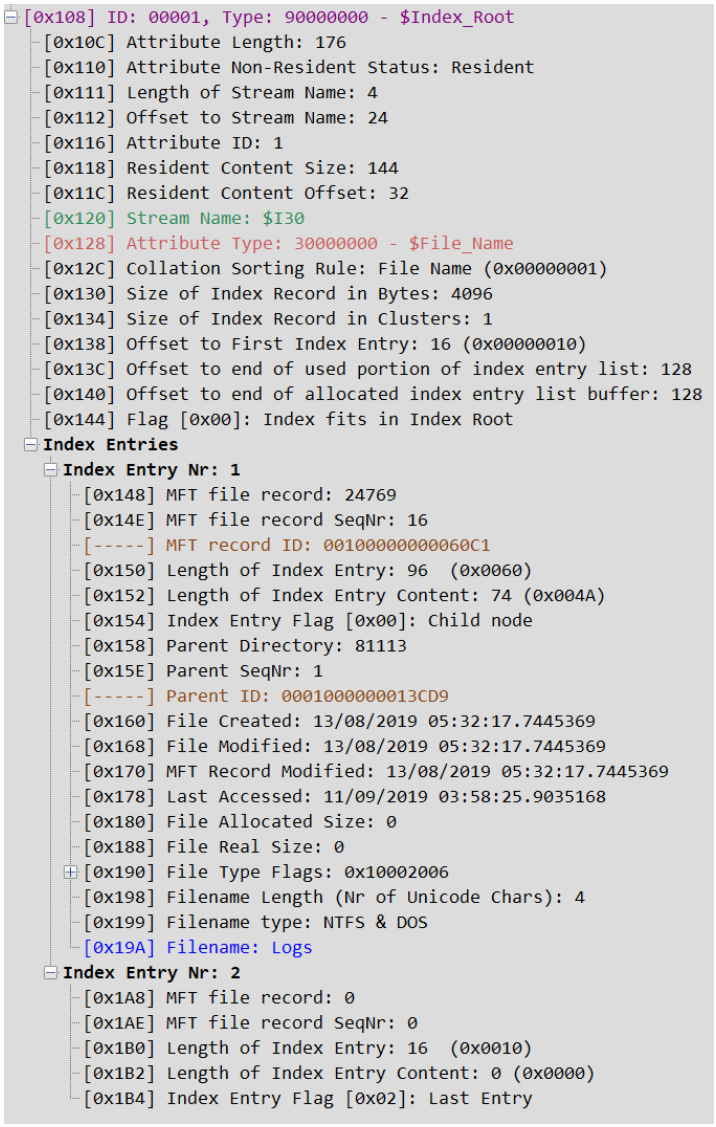


To get from the 1st Index Node Entry to any following ones, this simple rule can help:

Attribute's 'Resident content offset'+16 bytes+ 'Offset to 1st Entry' = 1st Index Entry
Attribute's 'Resident content offset'+16 bytes+ 'Offset to 1st Entry'+' 1st Index Entry's Length of Entry' = 2nd Index Entry

keep adding the 'Length of Node Entry' to the previous sum to get to the next Index Entry, until the Index Flag designates as the entry as the last one.

Example of an $index_Root Attribute with 'Attribute Type' Filename, and 2 Index node entries:

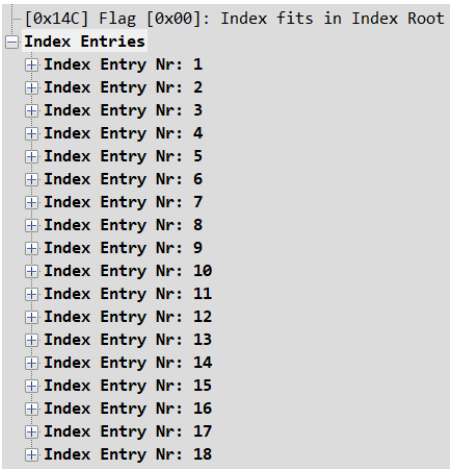


$INDEX_ALLOCATION

This attribute is never resident.

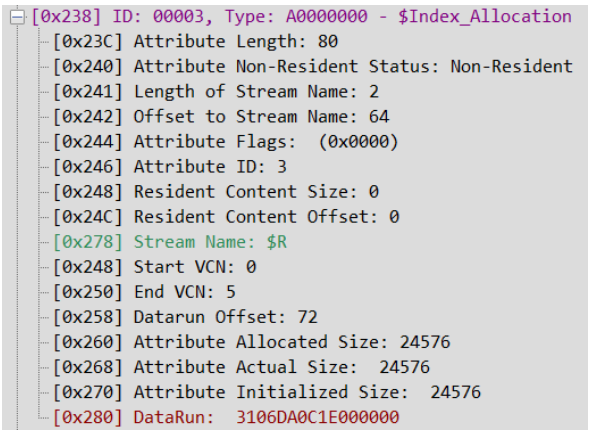
Records will only have an $Index_Allocation Attribute, if the $Index_Root attribute header has its flag set to '0x00000001' (*Index points to $Index_Allocation Child Nodes*).

For example, this Record's $Index_Root atribute's flag is '0x00000000' and does not have an $Index_Allocation attribute:



The DataRun of the attribute points to off-$MFT volume location(s) where Index records that do not fit in the $Index_Root Attribute are stored.

Example of such an Attribute:



$BITMAP

This attribute is usually Resident when it has a Stream Name. In Records with a Stream Name and an $Index_Allocation Attribute (*the Index is Non-Resident*), their $Bitmap Attribute content tracks which 4Kb Index blocks are in allocated. Such are the records of $Secure ($SFH), $Reparse ($R), $ObjectId ($O) and Records with Stream Name $I30.

To read the Attribute's content in its Resident form, one has to convert each Hexadecimal byte to Binary, and reverse the bit order. For example, 0x14FB is read as:

0x14 == 00010100 => reversed == 00101000
0xFB == 11111011 => reversed == 11011111

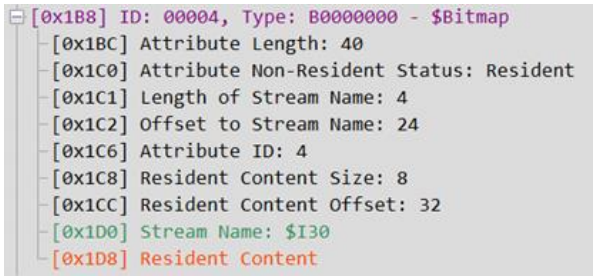
so that 0x14 FB becomes:

Bit	00	01	02	03	04	05	06	07		08	09	10	11	12	13	14	15
#	0	0	1	0	1	0	0	0		1	1	0	1	1	1	1	1

where (*starting from 0, left to right*) the bits equal to 1 are 2,4,8,9,11,12,13,14,15

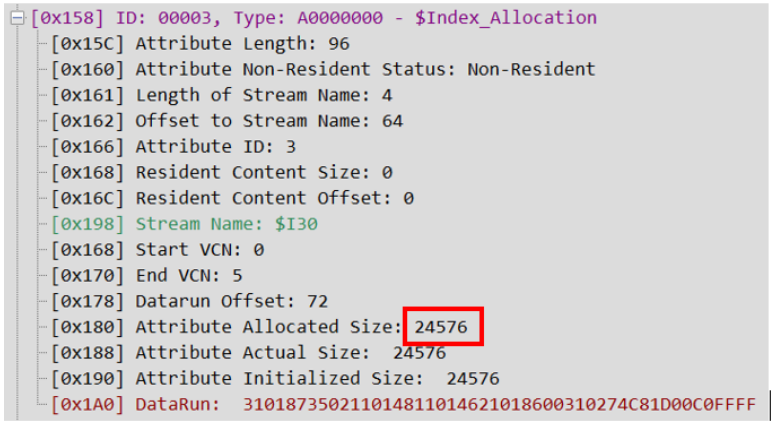
Each bit corresponds to a 4Kb Index block and the bits that equal to 1 are the Allocated ones.

Example of a $Bitmap Attribute with Stream Name $I30:
(*from the FILE record of a Directory with an $Index_Allocation Attribute*)

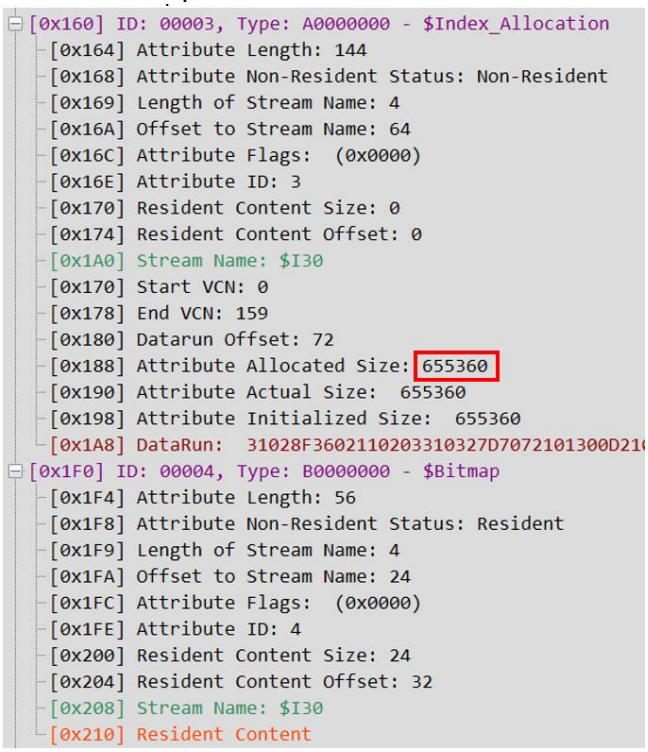


Resident content of the above example is: 0x3F00000000000000 == 0x3F => 11111100
Total Allocated 4Kb Index blocks: 6 (*Blocks 0,1,2,3,4,5*)
(*Total Allocated Index size = 4096Kb*6 blocks = 24.576 bytes*)

The $index_Allocation Attribute of the same Directory FILE record shows the same Allocation size for the $I30 Index:



Another example,



where the $Bitmap attribute's *(resident content)* value is:

0x FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 0F 00 00 00 00 00 00

which translates to 140 Index Blocks:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139

Indicating an actual allocated index size of 140*4096 *(index block size in bytes)* = 573440 bytes, smaller than the $Index_Allocation's allocated size of 655360.

$MFT file Record's $Bitmap Attribute

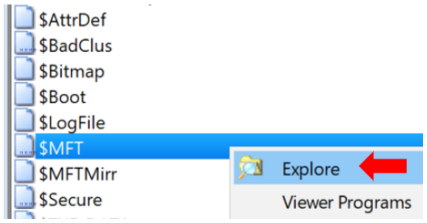
The $MFT file Record's *(Record Nr: 0*) $Bitmap Attribute is Non-Resident, doesn't have a Stream Name and contains the Allocated MFT segments[49]. That is, the DataRun of this $Bitmap Attribute points to an off-MFT file (*different from the NTFS $Bitmap Metafile*) which is read in a similar manner as described above, but each bit corresponds to an $MFT FILE record number. Bits equal to 1 signify used record numbers, while bits equal to 0 point to unused/free to use records.

Example $Bitmap Attribute of an $MFT record of itself:

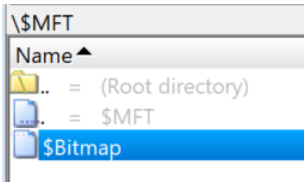
[49] Windows Internals 7th Edition Part 2: CHAPTER 11 Caching and file systems (p.662)

```
[0x148] ID: 00005, Type: B0000000 - $Bitmap
    [0x14C] Attribute Length: 80
    [0x150] Attribute Non-Resident Status: Non-Resident
    [0x151] Length of Stream Name: 0
    [0x152] Offset to Stream Name: 64
    [0x156] Attribute ID: 5
    [0x158] Resident Content Size: 0
    [0x15C] Resident Content Offset: 0
    [0x158] Start VCN: 0
    [0x160] End VCN: 1
    [0x168] Datarun Offset: 64
    [0x170] Attribute Allocated Size: 8192
    [0x178] Attribute Actual Size:  4104
    [0x180] Attribute Initialized Size:  4104
    [0x188] DataRun:  3101FF9E0131012661FE000000000000
```

Note: When viewing an NTFS Volume in WinHex, a right Mouse-Click on the $MFT entry and selection of the Explore option:



will open the Non-Resident content of this Attribute:



A double Mouse-Click on it (*or right click* + *Open*) will show its contents:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	FF	FF	00	FF	07	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	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

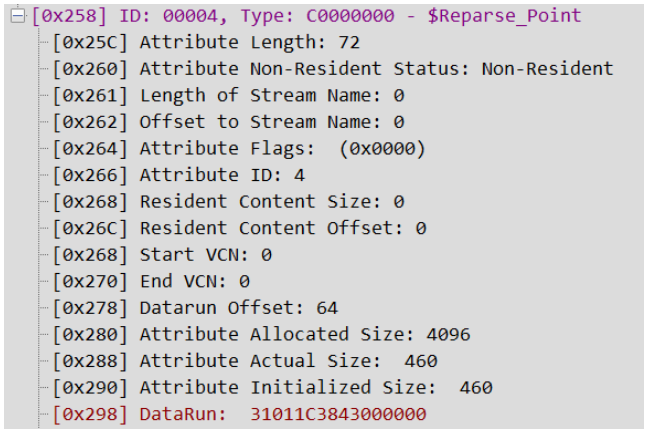
The above $Bitmap shows that a total of 27 $MFT records are used, and the Allocated (*used*) record numbers are:

0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,24,25,26,27,28,29,30,31,32,33,34.

Records 16,17,18,19,20,21,22,23,35,36,37,38,39 are marked as Not-Allocated.

$REPARSE_POINT

The $Reparse_Point Attribute is usually Resident, but there are occations where it might be Non-Resident e.g:



To quote Windows Internals, "*A reparse point is a file or directory that has a block of data called reparse data associated with it. Reparse data is user-defined data about the file or directory, such as its state or location that can be read from the reparse point by the application that created the data, a file system filter driver, or the I/O manager.*"[50]

Resident Attributes of this type start with a Header right after the common Attribute Header, shown below for reference:

Offset (D)	Offset (H)	Description	Length (bytes)
0	*0*	*Attribute Type (eg. 0x80000000 = $Data)*	*4*
4	*4*	*Attribute Length*	*4*
8	*8*	*Non-Resident Status flag*	*1*
9	*9*	*Length of Stream Name (Number of Unicode characters[51])*	*1*
10	*A*	*Offset to Stream Name (from the start of the Attribute)*	*2*
12	*C*	*Attribute Flags*	*2*
14	*E*	*Attribute Identifier (ID) (starting from 0)*	*2*
16	*10*	*Resident Content Size*	*4*
20	*14*	*Resident Content Offset (from the start of the Attribute)*	*2*
23	*17*	*Reserved/Unused*	*1*
Variable		*Stream Name*	*Variable[8]*
Reparse_Point 'Header'			
Resident Content Offset		Reparse Tag	4
+4	+4	Reparse Data Length	2

[50] Windows Internals 7th Edition, Part 2, CHAPTER 11 Caching and file systems, p.635
[51] Unicode characters are 2-byte values so the length in bytes is 2x this number.

+6	+6	*Reserved/Unused*	2
+8	+8	**Data Buffer** *(structures vary by Reparse Tag)*	Reparse Data Length

Reparse Point Tags

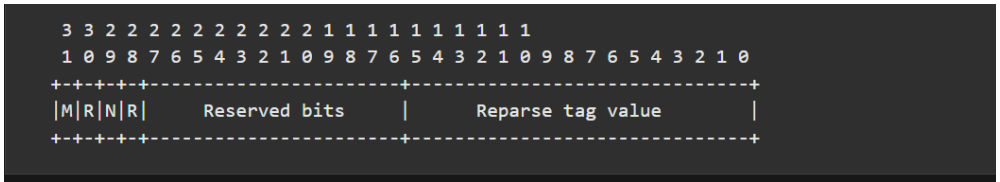
List of known Reparse Point Tags[52]:

Reparse Tag	Usage
0x00000000	RESERVED_ZERO
0x00000001	RESERVED_ONE
0x00000002	RESERVED_TWO
0x80000005	DRIVE_EXTENDER
0x80000006	HSM2
0x80000007	SIS
0x80000008	WIM
0x80000009	CSV
0x8000000A	DFS
0x8000000B	FILTER_MANAGER
0x80000012	DFSR
0x80000013	DEDUP
0x80000014	NFS
0x80000015	FILE_PLACEHOLDER
0x80000016	DFM
0x80000017	WOF
0x80000018	WCI
0x8000001B	APPEXECLINK
0x8000001E	STORAGE_SYNC
0x80000020	UNHANDLED
0x80000021	ONEDRIVE
0x80000023	AF_UNIX
0x80000024	LX_FIFO
0x80000025	LX_CHR
0x80000026	LX_BLK
0x9000001C	PROJFS
0x90001018	WCI_1
0x9000101A	CLOUD_1
0x9000201A	CLOUD_2
0x9000301A	CLOUD_3
0x9000401A	CLOUD_4
0x9000501A	CLOUD_5
0x9000601A	CLOUD_6
0x9000701A	CLOUD_7

[52] https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-fscc/c8e77b37-3909-4fe6-a4ea-2b9d423b1ee4

0x9000801A	CLOUD_8
0x9000901A	CLOUD_9
0x9000A01A	CLOUD_A
0x9000B01A	CLOUD_B
0x9000C01A	CLOUD_C
0x9000D01A	CLOUD_D
0x9000E01A	CLOUD_E
0x9000F01A	CLOUD_F
0xA0000003	MOUNT_POINT
0xA000000C	SYMLINK
0xA0000010	IIS_CACHE
0xA0000019	GLOBAL_REPARSE
0xA000001D	LX_SYMLINK
0xA000001F	WCI_TOMBSTONE
0xA0000022	PROJFS_TOMBSTONE
0xA0000027	WCI_LINK
0xA0001027	WCI_LINK_1
0xC0000004	HSM
0xC0000014	APPXSTRM

The ‘Reparse Tag’ value determines the rest of the Structure of the Attribute. This 4-byte Hex value, when converted to Binary shows the type of the Reparse point. The Tag structure according to Microsoft[53] is:



The low 16 bits determine the kind of reparse point. The high 16 bits have 12 bits reserved for future use and 4 bits that denote specific attributes of the tags and the data represented by the reparse point. The following table describes these bits.

Bit	Description
M	Microsoft bit. If this bit is set, the tag is owned by Microsoft. All other tags must use zero for this bit.
R	Reserved; must be zero for all non-Microsoft tags.
N	Name surrogate bit. If this bit is set, the file or directory represents another named entity in the system.

[53] https://docs.microsoft.com/en-us/windows/win32/fileio/reparse-point-tags

However, bit 1 has another meaning: "*High-latency bit. If set, accessing the first byte of data will be slow.*"[54] Let's examine a couple of Reparse Tags. Reparse Tag 0xA0000027 in Binary translates to **101-**0000000000000**-0000000000100111**

The first 3 left-most bits translate to:

- Microsoft owned[55] Tag (bit 0 set to 1)
- High-latency bit (bit 1 set to 0) is not set.
- It has the Name Surrogate bit set (bit 2 set to 1).

Similarly, Reparse Tag 0x80000021 in Binary is **100**-0000000000000-**0000000000010010**, so it translates to this being a Microsoft owned Tag (bit 0 set to 1), but the Name Surrogate bit is not set. Consequently, Tags starting with 0xA0, 0xB0, 0xE0 and 0xF0 are created by Microsoft and have the Name Surrogate bit set (the file or directory represents another named entity in the system). Such Reparse Tags are the two most common ones: 0xA0000003 (Mount Point) and 0xA000000C (Symbolic Link)[56].

Note1: *As of Windows 10 version 1607, a new flag bit was instroduced by Microsoft. Bit 4, dubbed the Directory bit, which when set to 1 indicates that a Directory with this Tag can have children.*[57] Such Reparse Tags start with 0x90 and 0xB0.

Note2: *The $Reparse:$R index records the $MFT records with Reparse Tags.(Indx2Csv64.exe can parse this index).*

0xA000000C (Symbolic Link)

Offset (D)	Offset (H)	Description	Length (bytes)
Resident Content Offset		Reparse Tag (0x**A000000C)**	4
4	4	Reparse Data Length	2
6	6	Reserved	2
8	8	Substitute Name Offset*	2
10	A	Substitute Name Length	2
12	C	Print Name Offset*	2
14	E	Print Name Length	2

[54] 1. The least significant 16 bits (i.e., bits 0 to 15) specifiy the type of the reparse point.
2. The 13 bits after this (i.e., bits 16 to 28) are reserved for future use.
3. "The most significant three bits are flags describing the reparse point. They are defined as follows:
bit 29: Name surrogate bit. If set, the filename is an alias for another object in the system.
bit 30: High-latency bit. If set, accessing the first byte of data will be slow.
(E.g., the data is stored on a tape drive.)
bit 31: Microsoft bit. If set, the tag is owned by Microsoft. User defined tags have to use zero here."
https://android.googlesource.com/kernel/hikey-linaro/+/android-hikey-linaro-4.1/fs/ntfs/layout.h

[55] https://docs.microsoft.com/en-us/windows/win32/fileio/reparse-point-tags

[56] https://download.microsoft.com/download/5/6/E/56E9388B-4568-444D-8860-81B37927E0B5/[MS-FSCC-Diff].pdf

[57] https://www.osr.com/nt-insider/2017-issue1/windows-10-whats-new-for-fs-filters/

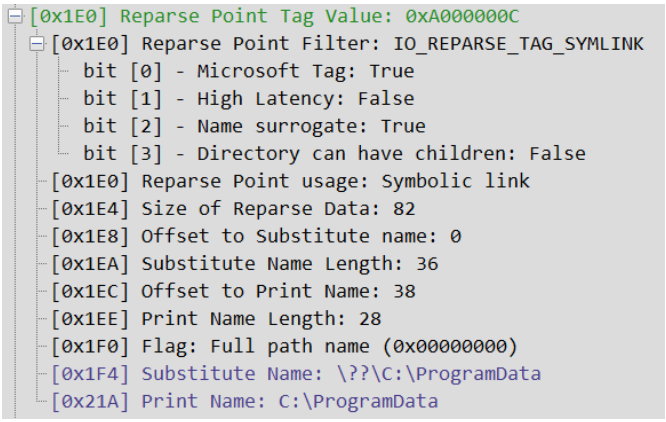
16	10	Flag	4
+ Substitute Name Offset		Substitute Name	Substitute Name Length
+ Print Name Offset		Print Name	Print Name Length

*****_These offsets start from byte 20, the end of the Reparse buffer structure above. In the few occations where either offset is 0, the respective Name starts from the end of the previous value._

The Names are in Unicode, and Name Lengths are number of bytes, not number of Unicode characters. The Flag can be:

0x00000000	The substitute name is a full path name
0x00000001	The substitute name is a path name relative to the directory containing the symbolic link

Example of such a Reparse Point:



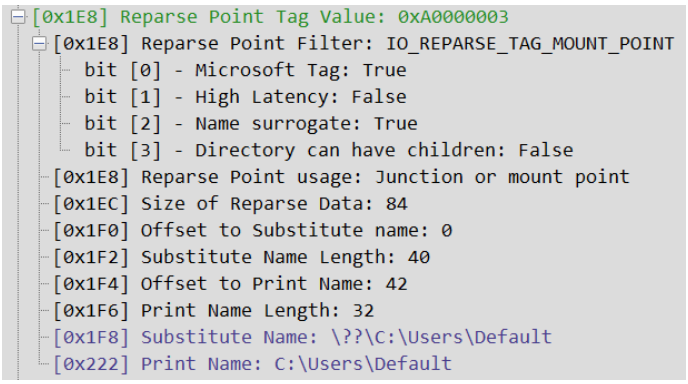
0xA0000003 (Junction or Mount Point)

Offset (D)	Offset (H)	Description	Length (bytes)
Resident Content Offset		Reparse Tag (0x**A0000003)**	4
4	4	Reparse Data Length	2
6	6	Reserved	2
8	8	Substitute Name Offset*	2
10	A	Substitute Name Length	2
12	C	Print Name Offset*	2
14	E	Print Name Length	2
Substitute Name Offset		Substitute Name	Substitute Name Length
+ Print Name Offset		Print Name	Print Name Length

*_These offsets start from byte 16, the end of the Reparse buffer structure above. In the few occations where either offset is 0, the respective Name starts from the end of the previous value._

The structure of the Mount Point Buffer is the same with Symbolic Link, with the exception that this Reparse Point does not have the 4 Flag bytes. The Names are in Unicode, and Name Lengths are number of bytes, not number of Unicode characters.

Example of such a Reparse Point:

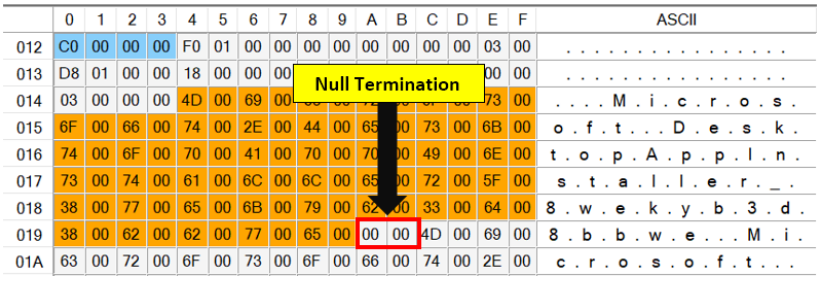


0x8000001B (AppExecutionAlias)

Since Windows 10 version 1709/RS3[58], this Tag is used by Universal Windows Platform (UWP) packages to encode information that allows the application to be launched from a command prompt.

Offset (D)	Offset (H)	Description	Length (bytes)
Resident Content Offset		Reparse Tag (0x**8000001B**)	4
4	4	Reparse Data Length	2
6	6	Reserved	2
8	8	Version	4
12	C	*Package ID*	*Variable*
Variable		*Entry Point*	*Variable*
Variable		*Executable*	*Variable*
Variable		*Application Type flag*	*2*

All the entries after the 'Version' bytes are Unicode strings NULL Terminated, meaning that each string ends with a Null Unicode character (\x00\x00), for example, this is how the 'Package ID' string is separated from the 'Entry Point':

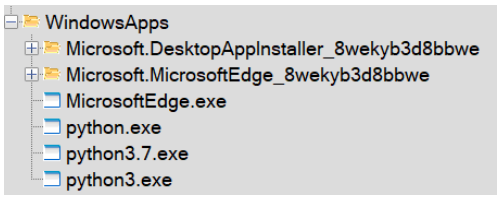


	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII
012	C0	00	00	00	F0	01	00	00	00	00	00	00	00	00	03	00	. . . . .
013	D8	01	00	00	18	00	00	00							00	00	. . . . .
014	03	00	00	00	4D	00	69	00							73	00	. . . . M . i . c . r . o . s .
015	6F	00	66	00	74	00	2E	00	44	00	65	00	73	00	6B	00	o . f . t . . . D . e . s . k .
016	74	00	6F	00	70	00	41	00	70	00	70	00	49	00	6E	00	t . o . p . A . p . p . l . n .
017	73	00	74	00	61	00	6C	00	6C	00	65	00	72	00	5F	00	s . t . a . l . l . e . r . _ .
018	38	00	77	00	65	00	6B	00	79	00	62	00	33	00	64	00	8 . w . e . k . y . b . 3 . d .
019	38	00	62	00	62	00	77	00	65	00	00	00	4D	00	69	00	8 . b . b . w . e . . . M . i .
01A	63	00	72	00	6F	00	73	00	6F	00	66	00	74	00	2E	00	c . r . o . s . o . f . t . . .

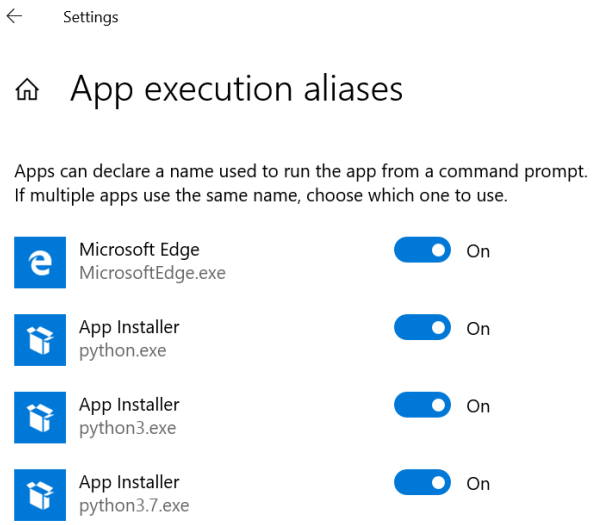
[58] https://www.tiraniddo.dev/2019/09/overview-of-windows-execution-aliases.html

If the ‘Application Type Flag’ is 0x01, the executable is a UWP[59] application. Otherwise, it is a Desktop bridge application[60].

Such Reparse Points are the executables found in the ‘\Users\UserName\AppData\Local\Microsoft\WindowsApps’ folder:



The executables with such Reparse Points can also be seen from the Windows 10/11 ‘Settings’, ‘Apps & features’, ‘App execution aliases’ page:



Example Reparse Points entries of two of the apps above:

```
[0x1C0] Reparse Point Tag Value: 0x8000001B
[0x1C0] Reparse Point Filter: IO_REPARSE_TAG_APPEXECLINK
[0x1C0] Reparse Point usage: Universal Windows Platform (UWP)
[0x1C4] Size of Reparse Data: 272
[0x1C8] AppExecLink Version: 3
[0x1CC] Package ID: Microsoft.MicrosoftEdge_8wekyb3d8bbwe
[0x218] Entry Point: Microsoft.MicrosoftEdge_8wekyb3d8bbwe!MicrosoftEdge
[0x280] Executable: C:\Windows\system32\SystemUWPLauncher.exe
[0x2D4] Application Type: UWP application
[0x1C8] Reparse Data
```

[59] https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide
[60] https://techcommunity.microsoft.com/t5/windows-dev-appconsult/desktop-bridge-8211-the-bridge-between-desktop-apps-and-the/ba-p/316488

```
[0x1B8] Reparse Point Tag Value: 0x8000001B
  [0x1B8] Reparse Point Filter: IO_REPARSE_TAG_APPEXECLINK
    bit [0] - Microsoft Tag: True
    bit [1] - High Latency: False
    bit [2] - Name surrogate: False
    bit [3] - Directory can have children: False
  [0x1B8] Reparse Point usage: Universal Windows Platform (UWP)
  [0x1BC] Size of Reparse Data: 464
  [0x1C0] AppExecLink Version: 3
  [0x1C4] Package ID: Microsoft.DesktopAppInstaller_8wekyb3d8bbwe
  [0x21C] Entry Point: Microsoft.DesktopAppInstaller_8wekyb3d8bbwe!PythonRedirector
  [0x296] Executable: C:\Program Files\WindowsApps\Microsoft.DesktopAppInstaller_1.0.30251.0_x64__8wekyb3d8bbwe\AppInstallerPythonRedirector.exe
  [0x38C] Application Type: Desktop bridge application
  [0x1C0] Reparse Data
```

0x80000017 (Windows Overlay filter or WOF)

These Reparse points are used for Windows Image File Boot (WimBoot)[61] or single-file compression[62]. This type of compression is not the NTFS native file system compression.

Offset (D)	Offset (H)	Description	Length (bytes)
Resident Content Offset		Reparse Tag 0x**80000017**	4
4	4	Reparse Data Length	2
6	6	Reserved	2
8	2	WOF current version (major)[63]	1
9	9	WOF current version (minor)[63]	1
12	C	WOF provider	4
16	10	WOF provider version(major)	1
18	12	WOF provider version(minor)	1
20	14	WOF compression format[64]	4

The ‘WOF provider’ can be any of:

Hex	Description
0x01	WIM (*WOF_PROVIDER_WIM*). Indicates that the data for the file resides in a separate WIM file. On access, data is transparently extracted, decompressed and provided to applications. If the file contents are modified, data is transparently decompressed, and the file is restored to a regular file.
0x02	FILE (*WOF_PROVIDER_FILE*). Indicates that the data for the file should be compressed and stored with the file itself. On access, data is transparently decompressed and provided to applications. If the file contents are modified, data is transparently decompressed, and the file is restored to a regular file. This provider requires Win 10.
0x03	CLOUD (*WOF_PROVIDER_CLOUD*)[65].

The ‘WOF compression format’ can take any of these values:

[61] https://docs.microsoft.com/en-us/windows/win32/w8cookbook/windows-image-file-boot--wimboot-

[62] https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/compact-os

[63] https://docs.microsoft.com/el-gr/windows-hardware/drivers/ddi/ntifs/ns-ntifs-_wof_version_info

[64] https://docs.microsoft.com/en-us/windows/win32/api/wofapi/ns-wofapi-wof_file_compression_info_v1

[65] https://github.com/tpn/winsdk-10/blob/9b69fd26ac0c7d0b83d378dba01080e93349c2ed/Include/10.0.14393.0/km/ntifs.h#L11571

Hex	Compression Algorithm	Description
0x00	XPRESS4K	4kb chunks (fastest and default value)
0x01	LZX	32kb chunks - LZX algorithm (most compact)
0x02	XPRESS8K	8kb chunks - XPress algorithm
0x03	XPRESS16K	16kb chunks - XPress algorithm

WOF provider WIM has been available since Windows 8.1 and provider FILE since Windows 10[66].

Example of a WOF Reparse Point:

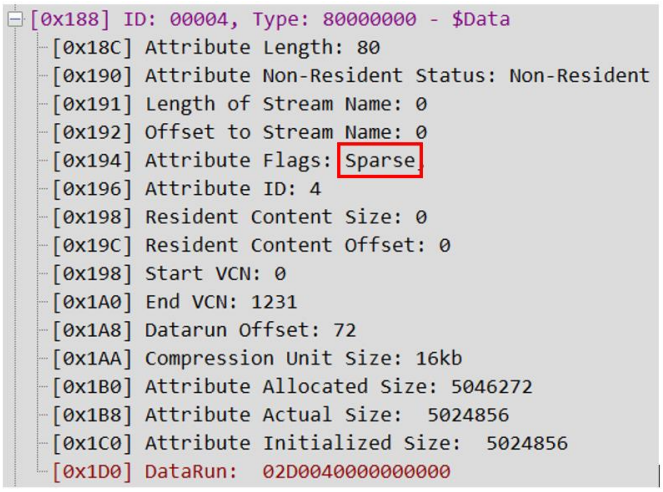
```
[0x2C0] Reparse Point Tag Value: 0x80000017
  [0x2C0] Reparse Point Filter: IO_REPARSE_TAG_WOF
  [0x2C0] Reparse Point usage: Windows Overlay Filter (WOF)
  [0x2C4] Size of Reparse Data: 16
  [0x2C8] WOF Version: 1.0
  [0x2CC] WOF Provider: File
  [0x2D0] WOF Provider Version: 1.0
  [0x2D4] WOF Compression Format: XPRESS8K
```

Such Reparse points usually follow a $Data Attribute with Stream Name ‘WofCompressedData’, which contains the Datarun to the compressed data:

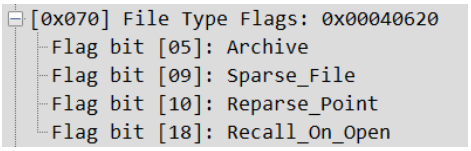
```
[0x1D8] ID: 00007, Type: 80000000 - $Data
  [0x1DC] Attribute Length: 112
  [0x1E0] Attribute Non-Resident Status: Non-Resident
  [0x1E1] Length of Stream Name: 17
  [0x1E2] Offset to Stream Name: 64
  [0x1E6] Attribute ID: 7
  [0x1E8] Resident Content Size: 0
  [0x1EC] Resident Content Offset: 0
  [0x218] Stream Name: WofCompressedData
  [0x1E8] Start VCN: 0
  [0x1F0] End VCN: 659
  [0x1F8] Datarun Offset: 104
  [0x1FA] Compression Unit Size: 0
  [0x200] Attribute Allocated Size: 2703360
  [0x208] Attribute Actual Size:  2699318
  [0x210] Attribute Initialized Size:  2699318
  [0x240] DataRun:  329402BB4A720000
```

The actual file content ($Data Attribute) is Sparse:

[66] https://docs.microsoft.com/el-gr/windows-hardware/drivers/ddi/ntifs/ns-ntifs-_wof_external_info



as indicated by the $Standard_Information Attribute's flags:



Further reading on WOF Compressed Data:
https://devblogs.microsoft.com/oldnewthing/20190618-00/?p=102597
https://github.com/wbenny/woftool/blob/master/README.md

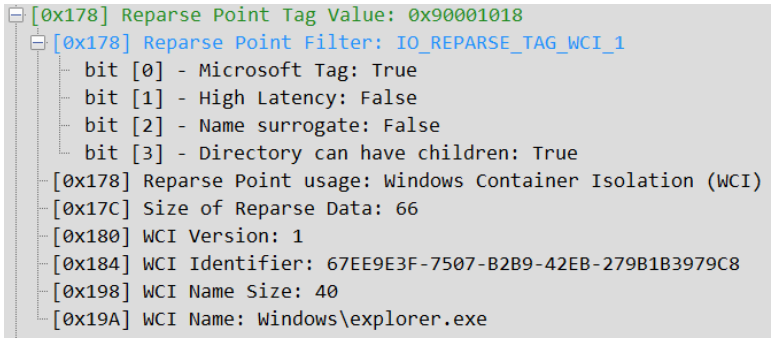
Windows Container Isolation (WCI)

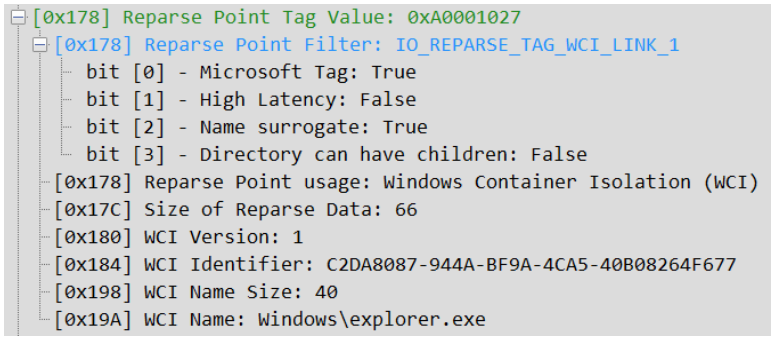
Reparse Tags: 0x80000018, 0x80000020, 0x90001018, 0xA0000027, 0xA0001027, 0xA000001F

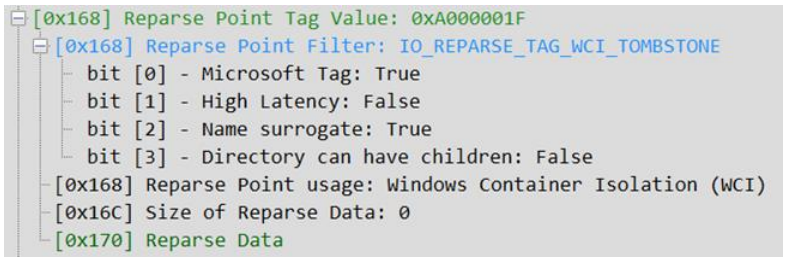
The Windows Container Isolation Reparse points are mainly found in Windows Sandbox, and share a common header structure:

Offset (D)	Offset (H)	Description	Length (bytes)
Resident Content Offset		Reparse Tag (**0x80000018**)	4
4	4	ReparseDataLength	2
6	6	Reserved	2
8	8	WCI Version	4
12	C	*Reserved*	4
16	10	WCI Identifier	16
32	20	WCI Name Size	2
34	22	WCI Name *(In Unicode)*	WCI Name Size

Examples:







related blog posts:
1. https://research.checkpoint.com/2021/playing-in-the-windows-sandbox/
2. https://dfir.ru/2020/08/15/containerized-registry-hives-in-windows/
3. https://osdfir.blogspot.com/2021/07/windows-container-forensics.html
4. https://googleprojectzero.blogspot.com/2021/04/who-contains-containers.html

Windows Subsystem for Linux (WSL)

".. WSL uses a new type of reparse point to represent symbolic links. As a result, these links will work only inside WSL and cannot be resolved by other Windows components such as File Explorer or cmd.exe ..."[67]

Such Reparse Tags are:

[67] https://docs.microsoft.com/en-us/archive/blogs/wsl/wsl-file-system-support#symbolic-links

0x80000023 (representation of a UNIX domain socket)[68]

```
[0x138] Reparse Point Tag Value: 0x80000023
  [0x138] Reparse Point Filter: IO_REPARSE_TAG_AF_UNIX
    bit [0] - Microsoft Tag: True
    bit [1] - High Latency: False
    bit [2] - Name surrogate: False
    bit [3] - Directory can have children: False
  [0x138] Reparse Point usage: WSL to represent a UNIX domain socket
  [0x13C] Size of Reparse Data: 0
  [0x140] Reparse Data
```

0x80000025 (representation of a UNIX character special file)

```
[0x130] Reparse Point Tag Value: 0x80000025
  [0x130] Reparse Point Filter: IO_REPARSE_TAG_LX_CHR
    bit [0] - Microsoft Tag: True
    bit [1] - High Latency: False
    bit [2] - Name surrogate: False
    bit [3] - Directory can have children: False
  [0x130] Reparse Point usage: WSL to represent a UNIX character special file
  [0x134] Size of Reparse Data: 0
```

0x80000026 (representation of a UNIX block special file)

0xA000001D (representation of a UNIX symbolic link)[69]

```
[0x128] Reparse Point Tag Value: 0xA000001D
  [0x128] Reparse Point Filter: IO_REPARSE_TAG_LX_SYMLINK
    bit [0] - Microsoft Tag: True
    bit [1] - High Latency: False
    bit [2] - Name surrogate: True
    bit [3] - Directory can have children: False
  [0x128] Reparse Point usage: WSL to represent a UNIX symbolic link
  [0x12C] Size of Reparse Data: 11
  [0x130] Flags: 0x02000000
  [0x134] SymLink Target: usr/bin
```

Cloud Files Filter Tags (9000101A - 9000F01A)

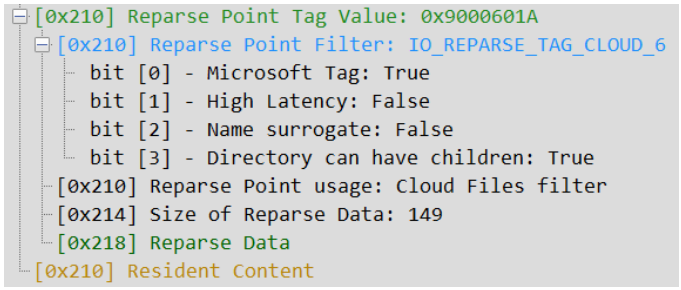
Used by Cloud Files filters, and usually found in OneDrive files and folders. Their Reparse Data structure is yet unknown. Examples can be seen below:

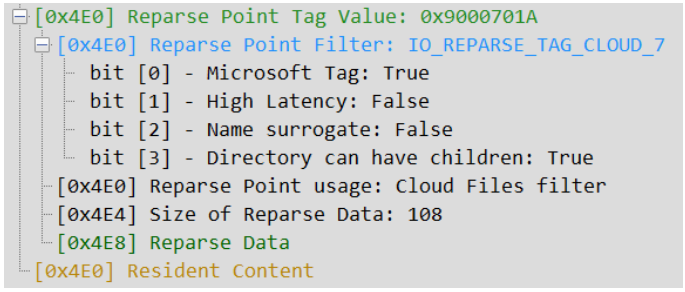
```
[0x190] Reparse Point Tag Value: 0x9000401A
  [0x190] Reparse Point Filter: IO_REPARSE_TAG_CLOUD_4
    bit [0] - Microsoft Tag: True
    bit [1] - High Latency: False
    bit [2] - Name surrogate: False
    bit [3] - Directory can have children: True
  [0x190] Reparse Point usage: Cloud Files filter
  [0x194] Size of Reparse Data: 232
  [0x198] Reparse Data
[0x190] Resident Content
```

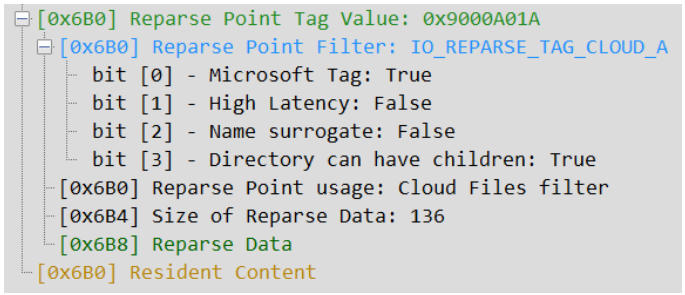
---

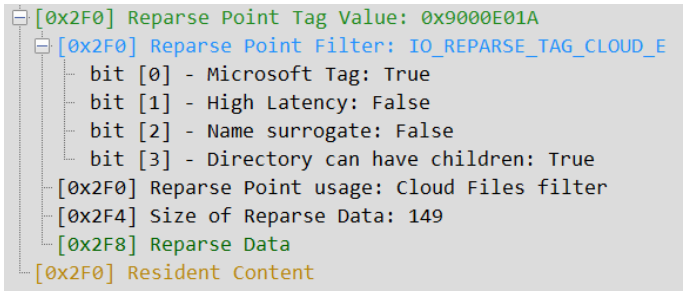
[68] https://devblogs.microsoft.com/commandline/windowswsl-interop-with-af_unix/

[69] https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc976871(v=technet.10)#symbolic-links









0x80000014 (NFS Reparse Data Buffer)

These Reparse points are created by the Network File System client and contain symbolic link or device information. Their buffer structure[70] is:

Offset (D)	Offset (H)	Description	Length (bytes)
Resident Content Offset		Reparse Tag (0x80000014)	4
4	4	Reparse Data Length	2
6	6	Reserved	2

[70] https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-fscc/ff4df658-7f27-476a-8025-4074c0121eec

8	8	NFS Reparse Data Buffer Type	8
16	10	Buffer	Variable

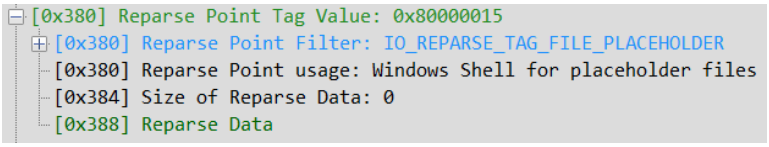
The ‘NFS Reparse Data Buffer Type’ can be:

Hex	Use	Description
00000000014B4E4C	NFS_SPECFILE_LNK	Unicode string containing the symbolic link data
0000000000524843	NFS_SPECFILE_CHR	major and minor device numbers for the character NFS special device
00000000004B4C42	NFS_SPECFILE_BLK	major and minor device numbers for the block NFS special device
000000004F464946	NFS_SPECFILE_FIFO	named pipe device - DataBuffer field is empty
000000004B434F53	NFS_SPECFILE_SOCK	socket device - DataBuffer field is empty

The buffer content for NFS types CHR & BLK is 2 32-bit integers signifying Major & Minor Device numbers. The buffer content of the LNK types is a Unicode string, the length of which, is obtained by: ‘Reparse Data Length’ - 8

0x80000015 (FILE_PLACEHOLDER)

These Reparse Tags were mainly observed in Win 8.1 SkyDrive (*currently OneDrive*), and the $Data content in files with these Reparse Points, is sparse. A second $Data attribute with Stream name “ms-properties” has the availability status & information of the file.



REPARSE_GUID_DATA_BUFFER (Non-Microsoft reparse tags)

According to Microsoft’s specification[71], these Reparse points have the following Dataa buffer structure:

Offset (D)	Offset (H)	Description	Length (bytes)
Resident Content Offset		Reparse Tag	4
4	4	Reparse Data Length	2
6	6	Reserved	2
8	8	Reparse GUID	16
24	18	DataBuffer	Variable

The GUID ‘s purpose is to identify the owner of the reparse point.

[71] https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-fscc/a4d08374-0e92-43e2-8f88-88b94112f070

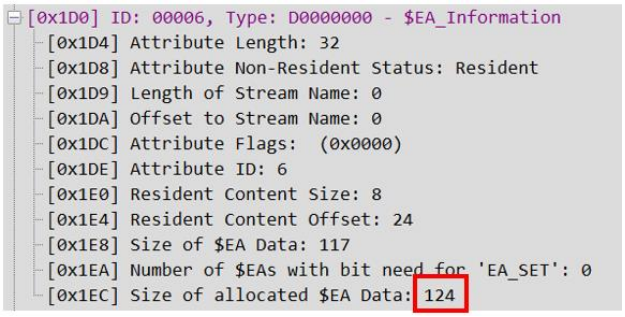
$EA_INFORMATION

This Attribute is always Resident.
Its resident content size is 8 bytes long, and its structure is as follows:

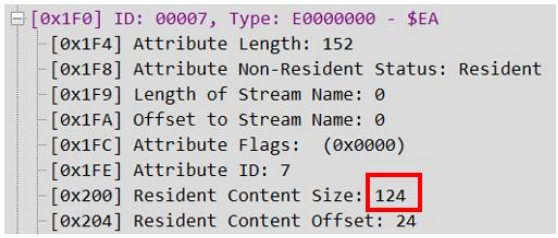
Offset (D)	Offset (H)	Description	Length (bytes)
0	0	Size of the extended attribute ($EA) data *(packed)*	2
2	2	Number of extended attributes which have NEED_EA bit set*****	2
4	4	Size of the extended attribute ($EA) data *(unpacked)*	4

***** "*If set, the file to which the EA belongs cannot be interpreted without understanding the associated extended attributes*"[72]

Example of such and Attribute:



Note that the Size of allocated $EA data is the same as the Resident content size of the following $EA Attribute:



[72] https://opensource.apple.com/source/ntfs/ntfs-94/kext/ntfs_layout.h.auto.html

$EA

The Extended Attribute ($EA) is usually Resident (*not specified in the current $AttrDef Metadata file*). There can be more than one $EA entries, with the maximum number specified by the $EA content size (*up to 65536 bytes long*). The structure[73] of each entry is the following:

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	Next Entry Offset (Size of the EA entry)	4
4	4	EA Buffer Flag	1
5	5	EA Buffer Name Length *(Nr of characters)*	1
6	6	EA Buffer Value Size	1
8	8	EA Buffer Name	NameLength
9	*9*	*Reserved*	*1*
9 + (Name Length)		EA Buffer Data	EA Buffer Value Size

Following $EA entry offsets are calculated as follows:

'$EA Attribute's Resident content offset' = 1st $EA entry offset
'1st $EA entry offset '+' Size of 1st $EA entry' = 2nd $EA entry offset
'2nd $EA entry offset +' Size of 2nd $EA entry' = 3rd $EA entry offset ..

The entry content (*EA Buffer Data*) depends on the 'EA Buffer Name'. Some have known structures as listed below:

EA Name: $CI.CATALOGHINT[74]

This extended attribute holds a 'hint' for the catalog[76] file used to verify the embedded digital signature of a file.

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	EA Priority	2
2	2	Hint or Name Size	2
4	4	Hint or Name (*in ASCII*)	Hint Size

The EA Priority values[75] can be:

Value	Priority	Usage
0	IoPriorityVeryLow	The system uses this value for background I/O operations
1	IoPriorityLow	Low-priority hint level.
2	IoPriorityNormal	Specifies a normal-priority hint level *(default setting)*

[73] https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/ns-wdm-_file_full_ea_information
[74] https://posts.specterops.io/host-based-threat-modeling-indicator-design-a9dbbb53d5ea?gi=c050fce2c763
[75] https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/ne-wdm-_io_priority_hint
&
https://www.nirsoft.net/kernel_struct/vista/IO_PRIORITY_HINT.html

3	IoPriorityHigh	Specifies a high-priority hint level *(system reserved)*
4	IoPriorityCritical	The highest-priority hint level *(system reserved*)
5	MaxIoPriorityTypes	Marks the limit for priority hints

Example of such an entry:

```
[0x2B0] $EA Buffer offset: 0
  [0x2B0] EA Buffer Length: 124
  [0x2B4] EA Buffer Flag: 00
  [0x2B5] EA Buffer Name length: 15
  [0x2B6] EA Buffer Value size: 99
  [0x2B8] EA Name: $CI.CATALOGHINT
    [0x2C8] EA Priority: IoPriorityLow
    [0x2CA] EA Name/Hint length: 95
    [0x2CC] EA Name/Hint: MicrosoftWindowsClientDesktopRequiredPackage04~31bf3856ad364e35~amd64~~100224491000cat
  [0x2C8] EA Value Data
```

The EA name hint of these entries is a .cat file (*digitally signed catalog file*)[76].

EA Name: LXATTRB, LXXATTR

Windows Subsystem for Linux (LxFs) – These $EA entries are usually found in WSL1 (*Windows 10 versions before v. 1903*)[77] or as Microsoft refers to it as 'Legacy WSL'.

The same applies to systems that do not support the Virtual Machine Platform optional feature.

LXATTRB

stores basic file information. Its 56 bytes long and its structure is as follows:

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	Flags	2
2	2	Version	2
4	4	Mode	4
8	8	User ID of owner	4
12	C	Group ID of owner	4
16	10	Device ID	4
20	14	Last Access Time (nanoseconds)	4
24	18	Last Modified Time (nanoseconds)	4
28	1C	iNode Changed Time (nanoseconds)	4
32	20	Last Access Time (unix seconds)	8
40	28	Last Modified Time (unix seconds)	8
48	30	iNode Changed Time (unix seconds)	8

[76] https://docs.microsoft.com/en-us/windows-hardware/drivers/install/catalog-files
[77] Builds lower than 18362 do not support WSL 2
https://docs.microsoft.com/en-us/windows/wsl/install-manual

The timestamps are Unix seconds + nanoseconds. They be converted to Date & Time in Powershell with the following command:

```
PS C:\> $timestamp = [System.DateTimeOffset]::FromUnixTimeSeconds(1632827065).Addticks(244160000/100)
PS C:\> $timestamp.ToString("dd/MM/yyyy HH:mm:ss.fffffff")
28/09/2021 11:04:25.2441600
```

Example entry:

```
[0x274] EA Name: LXATTRB
    [0x27C] Flags: 0
    [0x27E] Version: 1.0
    [0x280] Mode: 100666 (octal)
    [0x284] User ID of owner: 1000
    [0x288] Group ID of owner: 1000
    [0x28C] Device ID (Major): 0
    [0x28C] Device ID (Minor): 0
    [0x290] Last Access Time (ns): 244160000
    [0x294] Modified Time (ns): 244160000
    [0x298] Inode Changed Time (ns): 510846100
    [0x29C] Last Access Time (s): 1632827065
    [0x2A4] Modified Time (s): 1632827065
    [0x2AC] Inode Changed Time (s): 1632827354
    [-----] Last Accessed TimeStamp: 28/09/2021 11:04:25.2441600
    [-----] Last Modified TimeStamp: 28/09/2021 11:04:25.2441600
    [-----] Inode Changed TimeStamp: 28/09/2021 11:09:14.5108461
[0x27C] EA Value Data
```

The User IDs & User home path can be found in the ‘passwd’ file at: “C:\Users\UserName\AppData\Local\lxss\rootfs\etc” which holds information such as:

root:x:0:0:root:/root:/bin/bash or
testuser:x:1000:1000:"",,,:/home/testuser:/bin/bash

Where each line holds has the following seven comma delimited fields[78]:

- Username.
- Encrypted password (x means that the password is stored in the /etc/shadow file).
- User ID number (UID).
- User’s group ID number (GID).
- Full name of the user (GECOS).
- User home directory.
- Login shell (defaults to /bin/bash).

https://github.com/viruscamp/lxssattr/wiki/Output-Samples-LxFs

[78] https://linuxize.com/post/how-to-list-users-in-linux/

LXXATTR and LX.

Both are used for storing of Linux extended file attributes in name-value pairs. The structure of LXXATTR is as follows:

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	Version	4
Nodes *(repeats for the length of the EA buffer size)*			
0*	0	Entry Length*	4
4	4	Entry Value Length	2
6	6	Entry Name Length	1
7	7	Entry Name *(ASCII)*	Entry Name Length
7 +'Entry Name Length'		Entry Value *(UTF8)*	Entry Value Length
7 +'Entry Name Length' +'Entry Value Length'		Unknown	1

***** *The initial offset is 4. After the first node, the previous 'Entry Length' is added to the offset, and this repeats for the length of the '$EA buffer Value size'.*

Example of an LXXATTR extended attribute:

```
Extended Attribute Entries
  [0x268] $EA Buffer offset: 0
    [0x268] EA Buffer Length: 292
    [0x26C] EA Buffer Flag: 00
    [0x26D] EA Buffer Name length: 7
    [0x26E] EA Buffer Value size: 273
    [0x270] EA Name: LXXATTR
      [0x278] LXXATTR Version: 0x00000100
      [0x27C] Entry: 1
        [0x27C] Entry Length: 43
        [0x280] Entry Value Length: 20
        [0x282] Entry Name Length: 15
        [0x283] Entry Name: user.Comment2me
        [0x292] Entry Value: Remember to add more
        [0x2A6] Unknown: 0x65
      [0x2A7] Entry: 2
        [0x2A7] Entry Length: 103
        [0x2AB] Entry Value Length: 86
        [0x2AD] Entry Name Length: 9
        [0x2AE] Entry Name: user.Info
        [0x2B7] Entry Value: All my passwords are backed up in linux extended attributes. Good luck finding them :)
        [0x30D] Unknown: 0xC2
      [0x30E] Entry: 3
        [0x30E] Entry Length: 66
        [0x312] Entry Value Length: 45
        [0x314] Entry Name Length: 13
        [0x315] Entry Name: user.Password
        [0x322] Entry Value: Ubuntu VM - username: rooter pass: ip-address
        [0x34F] Unknown: 0x00
      [0x350] Entry: 4
        [0x350] Entry Length: 0
        [0x354] Entry Value Length: 35
        [0x356] Entry Name Length: 14
        [0x357] Entry Name: user.Password3
        [0x365] Entry Value: Kali VM - username: root pass: toor
        [0x388] Unknown: 0x00
    [0x278] EA Value Data
```

The **LX.**# stucture is similar to the LXXATTR, but instead of the nodes being in one $EA, each node is a separate $EA entry with the node entry Name appended to the $EA “LX.” Name. The structure is as follows:

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	Type* *(ASCII)*	4
4	4	Value *(UTF8)*	$EA BufferValue Size

**The from observation, it appears that the Type is ‘lxea’.*

Example of such entries:

```
[0x1AC] $EA Buffer offset: 60
  [0x1AC] EA Buffer Length: 116
  [0x1B0] EA Buffer Flag: 00
  [0x1B1] EA Buffer Name length: 15
  [0x1B2] EA Buffer Value size: 90
  [0x1B4] EA Name: LX.USER.COMMENT
    [0x1C4] EA LX Type: lxea
    [0x1C8] EA LX Value: All my passwords are backed up in linux extended attributes. Good luck finding them :)
  [0x1C4] EA Value Data
[0x220] $EA Buffer offset: 176
  [0x220] EA Buffer Length: 76
  [0x224] EA Buffer Flag: 00
  [0x225] EA Buffer Name length: 16
  [0x226] EA Buffer Value size: 49
  [0x228] EA Name: LX.USER.PASSWORD
    [0x239] EA LX Type: lxea
    [0x23D] EA LX Value: Ubuntu VM - username: rooter pass: ip-address
  [0x239] EA Value Data
```

Note: It is possible to compress the LX Value with the Zlib[79] . For example, the raw Value:

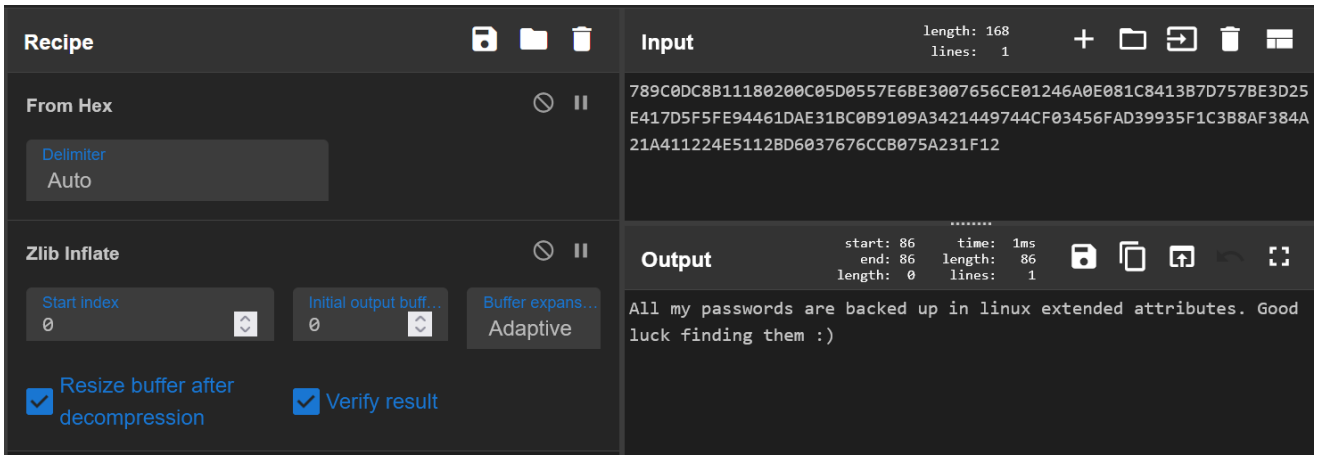
789C0DC8B11180200C05D0557E6BE3007656CE01246A0E081C8413B7D757BE3D25E417D5F5FE94461DAE31BC0B
9109A3421449744CF03456FAD39935F1C3B8AF384A21A411224E5112BD6037676CCB075A231F12

Would appear as:

```
[0x279] Entry: 2
  [0x279] Entry Length: 0 (Last Entry)
  [0x27D] Entry Value Length: 84
  [0x27F] Entry Name Length: 15
  [0x280] Entry Name: user.compressed
  [0x28F] Entry Value: x▯ð▯▯ ▲▯▯U~k▯
  [0x2E3] Unknown: 0x00
```

In such cases, using a tool like CyberChef[80] can decode the content:

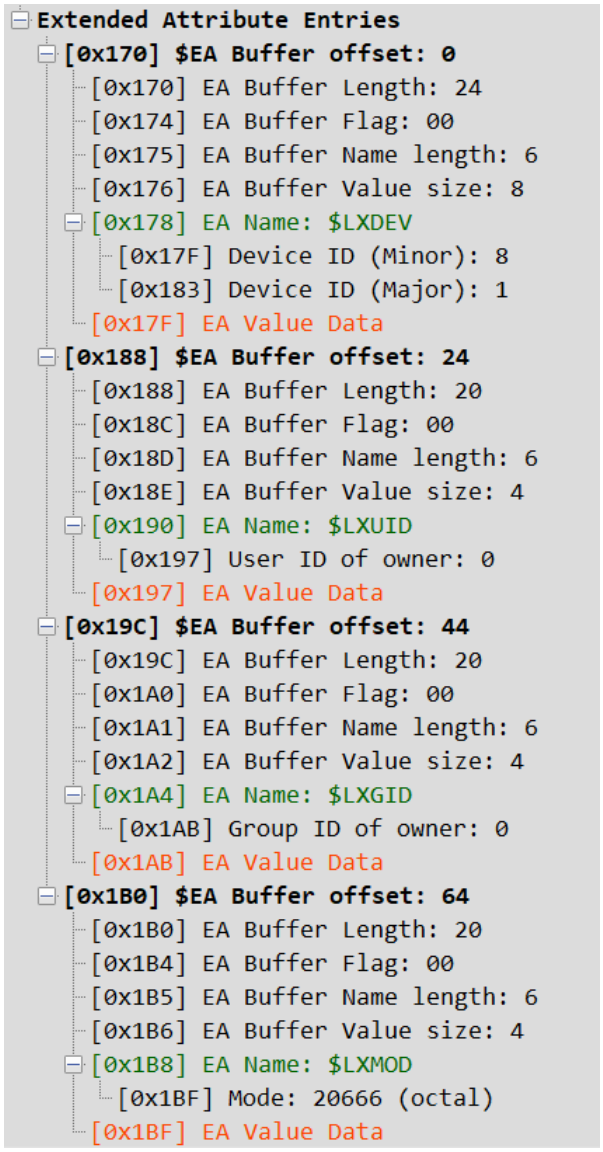
[79] https://zlib.net/
[80] https://gchq.github.io/CyberChef/



EA Names: $LXUID, $LXGID, $LXMOD, $LXDEV

Windows Subsystem for Linux (WslFs). In newer versions of WSL, the LXATTRB $EA has been replaced with these 4 separate $EA entries[81] *($LXDEV is not always present*). Their content is always 4 byte long except $LXDEV which is 8 bytes long, with the first 4 bytes being the minor Device ID and the last 4 bytes the major Device ID:

[81] https://docs.microsoft.com/en-us/windows/wsl/file-permissions



https://github.com/0xbadfca11/lxsstat/wiki/WSL-filesystem

https://github.com/viruscamp/lxssattr/wiki/Output-Samples-WslFs

$KERNEL Extended Attributes

Extended Attributes with a name startiung with “$Kernel” are Kernel Extended attributes[82] , used mostly to enhance file signature validation performance.

$KERNEL.PURGE.ESBCACHE

The most common Kernel $EA, and its purpose is to auto-delete any $EA[83]s on a file beginning with this $EA Name, if any of the following reasons is written on the USN Jounal ($J):

- USN_REASON_DATA_OVERWRITE
- USN_REASON_DATA_EXTEND
- USN_REASON_DATA_TRUNCATION
- USN_REASON_REPARSE_POINT_CHANGE

There are possibly 3 different ‘Cached Signing Level’ structures[84], Identified by the Signing Version (Major):

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	Content Size	4
4	4	Signing Version (Major)	2

Current Win10 versions appear to use Signing version 3.2 (*Major 3, Minor 2*), whose structure is as follows:

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	Content Size	4
4	4	Signing Version (Major)	2
6	6	Signing Version (Minor)	1
7	7	Signing Level	1
8	8	Kernel Update sequence number (USN) timestamp*	8
16	10	Buffer Last BlackList timestamp*	8
24	18	Cached Signing Level Flags	4
28	1C	Extra Data Size	2
30	1E	Extra Data	Extra Data Size

* *Timestamps in Windows Filetime format*

[82] https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/kernel-extended-attributes
[83] https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/kernel-extended-attributes#auto-deletion-of-kernel-extended-attributes
[84] https://github.com/googleprojectzero/sandbox-attacksurface-analysis-tools/blob/7712d242b632b0c1b70827ba712f36241816ec10/NtApiDotNet/NtSecurityNative.cs#L200

The ‘Signing Level’ can be any of:

Dec	Hex	Signing Level
0	00	Unchecked
1	01	Unsigned
2	02	DeviceGuard
3	03	Custom 1
4	04	Authenticode
5	05	Custom 2
6	06	Store
7	07	Antimalware
8	08	Microsoft
9	09	Custom 4
10	0A	Custom 5
11	0B	Dynamic Code Generation
12	0C	Windows
13	0D	Windows Protected Process Light
14	0E	Windows TCB
15	0F	Custom 6

And the ‘Cached Signing Level Flags’ can be one or a combination of:

Dec	Hex	Cached Signing Level Flag
01	00000001	UntrustedSignature
02	00000002	TrustedSignature
03	00000003	Unknown 4
04	00000004	Doesn't Use USN Journal
16	00000010	Has Per App Rules
32	00000020	Set In Test Mode
64	00000040	Protected Light Verification

Starting at offset 30 (0x1E) from the start of the $EA Buffer content, the Extra Data can hold one or more entries structured as follows:

Offset (D)	Offset (H)	Description	Length (bytes)
0*	0*	Extra Data Entry Size	1
1	1	Hash Type	1
2	2	Hash Algorithm	4
6	6	Hash Size	1
7	7	Hash	Hash Size

**Subsequent entries start at at the end of the previous ones.*

The possible Hash types are:

Value	Hash Type
0	File Hash
1	Signer Hash
2	WIM Guid
3	Timestamp
4	Device Guard PolicyHash
5	AntiCheat Policy Hash

With only ‘File Hash’ and ‘Signer Hash’ having been observed at the time of writing. Finally, the Hash Algorithms are:

Value	Hash Algorithm
00008004	SHA1
0000800C	SHA256
0000800D	SHA384
0000800E	SHA512

Example of a $KERNEL.PURGE.ESBCACHE $EA attribute with 2 Extra Data entries:

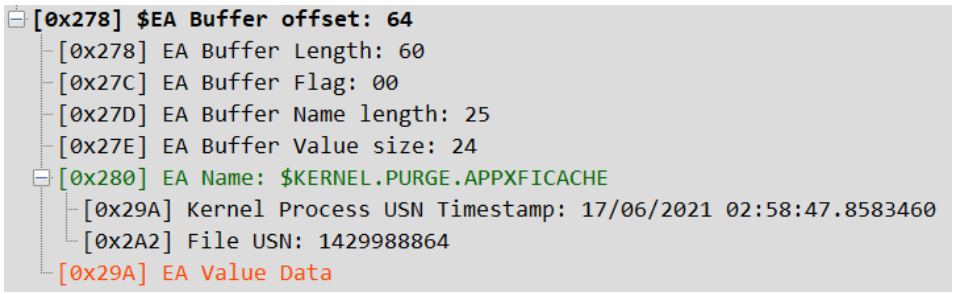
```
[0x210] $EA Buffer offset: 0
  [0x210] EA Buffer Length: 128
  [0x214] EA Buffer Flag: 00
  [0x215] EA Buffer Name length: 22
  [0x216] EA Buffer Value size: 96
  [0x218] EA Name: $KERNEL.PURGE.ESBCACHE
    [0x22F] Content Size: 96
    [0x233] Signing Version (Major): 3
    [0x235] Signing Version (Minor): 2
    [0x236] Signing Level: Windows
    [0x237] Kernel Process USN Timestamp: 06/02/2016 14:05:59.9114620
    [0x23F] Buffer Last BlackList Time: 12/12/2017 17:42:57.0000000
    [0x247] Cached Signing Level Flag: TrustedSignature
    [0x247] Cached Signing Level Flag: Protected Light Verification
    [0x24B] Extra Data size: 66
    [0x24D] Extra Data (blob)
      [0x24D] ExtraData #1
        [0x24D] ExtraData #1 Size: 39
        [0x24E] Hash #1 Type: Signer Hash
        [0x24D] Hash #1 Algorithm: SHA256
        [0x253] Hash #1 Size: 32
        [0x254] Hash #1: B9590CE5B1B3F377EAA6F455574C977919BB785F12A444BEB27CF494F0CD334D
      [0x274] ExtraData #2
        [0x274] ExtraData #2 Size: 27
        [0x275] Hash #2 Type: File Hash
        [0x274] Hash #2 Algorithm: SHA1
        [0x27A] Hash #2 Size: 20
        [0x27B] Hash #2: 2B7BA68EF916B4874C8DEC220D80B8B39F213B7F
  [0x22F] EA Value Data
```

$KERNEL.PURGE.APPXFICACHE

The size of this $EA content is fixed to 16 bytes long:

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	Kernel Process USN Timestamp	8
8	8	File Update Sequence number (USN)	8

Example entry:



Other Kernel Extended Attribute Names with currently unknown structure include:

$KERNEL.PURGE.SEC.FILEHASH

$KERNEL.PURGE.SEC.CODEINTEGRITY

$KERNEL.PURGE.SMARTLOCKER.VALID

$KERNEL.PURGE.APPID.SIGNERINFO

$KERNEL.PURGE.APPID.VERSIONINFO

$KERNEL.PURGE.APPID.HASHINFO

$KERNEL.SMARTLOCKER.ORIGINCLAIM

$KERNEL.SMARTLOCKER.HASH

$KERNEL.SEC.ENDPOINTDLP

$LOGGED_UTILITY_STREAM

This Attribute is currently used for various purposes. The content structure depends on the Attribute Stream Name:

Encrypted File Stream (*$EFS*)

According to Microsoft's documentation of the Encrypting File System Remote (EFSRPC) Protocol[85], there are 3 EFSRPC Metadata Versions. When a $Logged_Utility attribute with stream name $EFS is Resident, it most likely has EFSRPC Metadata Version 3 content is stored in it, with the following structure:

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	Signature (ASCII)	6
6	6	Version (Major)	4
10		Version (Minor)	4
14		ClearText Redirection Length	4
18		ClearText Redirection String (UTF8)	ClearText Redirection Length
18 +'CRL'		Total Header Length	4
18 +'CRL' +4		Original File Extension Offset	4
18 +'CRL' +8		Original File Extension Length	4
18 +'CRL' +12		Publishing Licence Offset	4
18 +'CRL' +16		Encrypted Data Offset	4
18 +'CRL' +20		Original File Size	8
18 +'CRL' +28		Unused/Reserved	4
18 +'CRL' +32		MetaData Offset	4
18 +'CRL' +36		MetaData Length	4
0 +Original File Extension Offset		Original File Extension Data	Original File Extension Length
0 +Publishing Licence Offset +4		Publishing Licence Data Length	4
0 +Publishing Licence Offset		Publishing Licence Data	Publishing Licence Data Length
0 +MetaData Offset		MetaData	MetaData Length
0 +Encrypted Data Offset		Encrypted Data	Original File Size - MetaData Length

[85] https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-EFSR/%5BMS-EFSR%5D.pdf (2.2.2.3 EFSRPC Metadata Version 3)

Online encryption backup (*$EfsBackup*)

TBD

Transactional NTFS Data *($TXF_DATA)*

When the Attribute has the “$TXF_DATA” Stream Name, the content is Resident and 56 bytes long:

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	$MFT Record Nr of RM root*	6
6	6	$MFT Record Sequence Nr of RM root*	2
8	8	Flags	8
16	10	TxF file ID	8
24	18	LSN for NTFS Metadata	8
32	20	LSN for User Data	8
40	28	LSN for Directory Index	8
48	30	USN index	8

*****The Default Resource Manager root is MFT Nr 5, Seq.Nr. 5 (Root directory)*

Example of such an entry from the Root directory (\):

```
[0x1E0] Stream Name: $TXF_DATA
  [0x1F8] $MFT Record ID of root: 0005000000000005
    [0x1F8] $MFT Record Nr.: 5
    [0x1FE] $MFT Record Seq.Nr.: 5
  [0x200] Flags: 0x0000000100000001
  [0x208] TxF file ID (TxID): 0x0000000000000000 (0)
  [0x210] LSN for NTFS Metadata: 0
  [0x218] LSN for User Data: 0
  [0x220] LSN for Directory Index: 12288
  [0x228] USN Index: 1048576
[0x1F8] Resident Content
```

And from the ‘C:\Users’ directory of the same Volume:

```
[0x1E8] Stream Name: $TXF_DATA
  [0x200] $MFT Record ID of root: 0005000000000005
    [0x200] $MFT Record Nr.: 5
    [0x206] $MFT Record Seq.Nr.: 5
  [0x208] Flags: 0x0000000100000001
  [0x210] TxF file ID (TxID): 0x0000000000000006 (6)
  [0x218] LSN for NTFS Metadata: 6656
  [0x220] LSN for User Data: 0
  [0x228] LSN for Directory Index: 108032
  [0x230] USN Index: 0
[0x200] Resident Content
```

Desired Storage Class (*$DSC*)

When the Attribute has the “$DSC” Stream Name, the content is Resident and 8 bytes long:

Offset (D)	Offset (H)	Description	Length (bytes)
0	0	Storage Tier Class	4
4	4	Flags	4

Where the Tier Class [86] can be one of:

Value	Tier Class
0	Unspecified
1	Capacity
2	Performance
3	Max

The default NTFS value is 2 (Performance), as seen by the fsutil command:

```
PS C:\> fsutil behavior query defaultntfstier
DefaultNtfsTier = 2 (Performance tier)
```

The possible values and meaning of the Desired Storage Class Flag as currently unknown.

Example of such an entry:

```
[0x2A0] Stream Name: $DSC
    [0x2A8] Tier Class: Performance
    [0x2AC] Flag: 0
```

‘fsutil file layout full_filepath’ command will show the $DSC Tier info of a file:

```
Stream                  : 0x100  :$DSC:$LOGGED_UTILITY_STREAM
    Attributes          : 0x00000000: *NONE*
    Flags               : 0x0000001c: Resident | No clusters allocated | Has Parsed Information
    Size                : 8
    Allocated Size      : 8
    DSC                 : Tier Class:    Performance
                        : Flags:         0x00000000: *NONE*
```

[86] https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/ntifs/ne-ntifs-_file_storage_tier_class

End of Record Signature

The end of a FILE record is marked by the following 8 bytes:

0x FF FF FF FF 00 00 00 00

or

0x FF FF FF FF 82 78 47 11

Which represent “-1” and “0” or “289896834” respectively:

```
strcpy((char *)a4, "FILE0");
v12 = (*(_DWORD *)(a2 + 360) >> 9) + 1;
*(_WORD *)(a4 + 6) = v12;
*(_WORD *)(a4 + 48) = 1;
v13 = (unsigned __int16)(2 * v12 + 55) & 0xFFF8;
*(_DWORD *)(a4 + 24) = v13;
*(_WORD *)(a4 + 20) = v13;
*(_WORD *)(a4 + 22) = a6 != 0 ? 2 : 0;
*(_DWORD *)(a4 + 28) = *(_DWORD *)(a2 + 360);
*(_WORD *)(a4 + 42) = *(_WORD *)(a3 + 4);
*(_DWORD *)(a4 + 44) = *(_DWORD *)a3;
result = (unsigned int)(v13 + 8);
*(_DWORD *)(a4 + 24) = result;
*(_DWORD *)(v13 + a4) = -1;
*(_DWORD *)(v13 + a4 + 4) = 289896834;
```

at the end of the Allocated part of the record. The first 4 bytes indicate an Attribute type of -1 or 0xFFFFFFFF and the second 4 bytes indicate an invalid record length of 0 or 289896834.