

# NTFS - MFT

<input checked="" type="checkbox"/> 태그	<input checked="" type="checkbox"/>
⌚ 최종 편집 일자	@2024년 2월 17일 오후 9:06

## MFT 분석

[MFT \(Master File Table\)](#)

[MFT 파일](#)

[MFT 영역](#)

[MFT 구조 분석 \(\\$MFT\)](#)

[MFT Entry Structure Layout](#)

[Base & Non-base MFT Entry](#)

[File Reference Address](#)

[Fixup Array](#)

[MFT - Attributes](#)

[Common Attribute Header](#)

[Resident Attribute Header](#)

[Non-Resident Attribute Header](#)

[Cluster Runs](#)

[MFT - Attribute Types](#)

[Attribute - \\$STANDARD\\_INFOMATION \(0x10\)](#)

[Attribute - \\$ATTRIBUTE\\_LIST \(0x20\)](#)

[Attribute - \\$FILE\\_NAME \(0x30\)](#)

[Attribute - \\$OBJECT\\_ID \(0x40\)](#)

[Attribute - \\$SECURITY\\_DESCRIPTOR \(0x50\)](#)

[Attribute - \\$VOLUME\\_NAME \(0x60\)](#)

[Attribute - \\$VOLUME\\_INFORMATION \(0x70\)](#)

[Attribute - \\$DATA \(0x80\)](#)

[Attribute - \\$INDEX\\_ROOT \(0x90\)](#)

[Attribute - \\$INDEX\\_ALLOCATION \(0xA0\)](#)

[Attribute - \\$BITMAP \(0xB0\)](#)

[Attribute - \\$REPARSE\\_POINT \(0xC0\)](#)

[Attribute - \\$EA\\_INFOMATION \(0xD0\) & \\$EA\(0xE0\)](#)

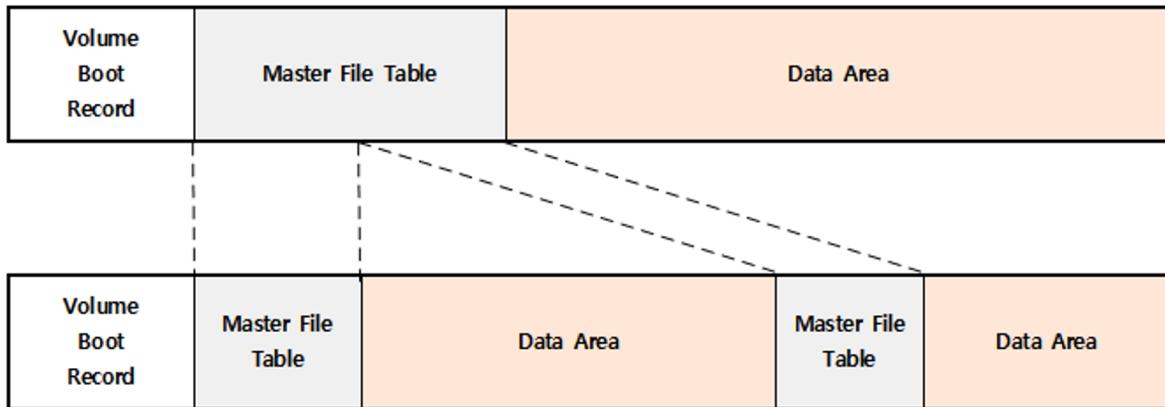
[MFT 레퍼런스](#)

## MFT 분석

### MFT (Master File Table)

#### MFT 파일

- NTFS 파일시스템에서 파일, 디렉터리, 메타데이터를 모두 파일 형태로 관리하는 파일
- FAT 파일시스템 기준 / 디렉터리 엔트리의 상위 개념



## MFT 영역

- 파일 시스템 상의 파일 수에 따라 동적으로 할당
- MFT Entry 0 ~ 15번 → 파일 시스템의 예약된 영역으로 사용됨.

Entry Number	Entry Name	Description
0	\$MFT	NTFS에서의 모든 파일들의 MFT Entry 정보를 담고 있다.
1	\$MFTMirr	\$MFT 파일의 일부 백업 데이터를 담고 있다.
2	\$LogFile	Meta Data의 트랜잭션 저널 정보를 담고 있다.
3	\$Volume	볼륨의 레이블, 식별자, 버전 등의 여러 정보를 담고 있다.
4	\$AttrDef	속성의 식별자, 이름, 크기, 등의 정보를 담고 있다.
5	.	볼륨의 루트 디렉터리 정보를 담고 있다.
6	\$Bitmap	볼륨의 클러스터 할당 정보를 담고 있다.
7	\$Boot	볼륨이 부팅 가능할 경우 부트 섹터 정보를 담고 있다.
8	\$BadClus	배드 섹터를 가지는 클러스터의 정보를 담고 있다.
9	\$Secure	파일의 보안, 접근 제어와 관련된 정보를 담고 있다.
10	\$Upcase	모든 유니코드 문자의 대문자
11	\$Extend	\$ObjID, \$Quota, \$Reparse points, \$UsnJml 등의 추가적인 파일의 정보를 기록하기 위해서 사용하는 엔트리
12~15		미래를 위해서 예약된 영역으로 남겨둠
16~		포맷 후 생성되는 파일의 정보를 위해서 사용
-	\$ObjID	파일 고유의 ID 정보를 담고 있다.
-	\$Quota	사용량의 정보를 담고 있다.
-	\$Reparse	Reparse Point에 대한 정보를 담고 있다.
-	\$UsnJnl	파일, 디렉터리의 변경 정보를 담고 있다.

- MFT Entry 0 ~ 15번
  - 0번 - \$MFT : NTFS의 모든 파일들의 MFT Entry 정보를 담고 있음.
  - 1번 - \$MFTMirr : \$MFT 파일의 일부 백업 데이터를 담고 있음.
  - 2번 - \$LogFile : Meta Data의 트랜잭션 저널 정보를 담고 있음.
  - 3번 - \$volume : 볼륨의 레이블, 식별자, 버전 등의 여러 정보를 담고 있음.
  - 4번 - \$AttrDef : 속성의 식별자, 이름, 크기, 등의 정보를 담고 있음.
  - 5번 - . : 볼륨의 루트 디렉토리 정보를 담고 있음.
  - 6번 - \$Bitmap : 볼륨의 클러스터 할당 정보를 담고 있음.
  - 7번 - \$Boot : 볼륨이 부팅 가능할 경우 부트 섹터 정보를 담고 있음.
  - 8번 - \$BadClus : 베드 섹터를 가지는 클러스터의 정보를 담고 있음.
  - 9번 - \$Secure : 파일의 보안, 접근 제어와 관련된 정보를 담고 있음.
  - 10번 - \$Upcase : 모든 유니코드 문자의 대문자
  - 11번 - \$Extend : \$ObjID, \$Quota, \$Reparse points, \$UsnJrnl 등의 추가적인 파일의 정보를 기록하기 위해서 사용하는 엔트리
  - 12 ~ 15 : 미래를 위해서 예약된 영역으로 남겨둠
  - 16 ~ : 포맷 후 생성되는 파일의 정보를 위해서 사용
    - \$ObjID : 파일의 고유 ID 정보를 담고 있음.
    - \$Quota : 사용량의 정보를 담고 있음.
    - \$Reparse : Reparse Point에 대한 정보를 담고 있음.
    - \$UsnJrnl : 파일, 디렉토리의 변경 정보를 담고 있음.

## MFT 구조 분석 (\$MFT)

### MFT Entry Structure Layout

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0x00	Signature			Offset to Fixup array		Entries in Fixup array		\$LogFile Sequence Number (LSN)								
0x10	Sequence Number		Hard Link count		Offset to File Attribute		Flags		Real Size of MFT Entry			Allocated Size of MFT Entry				
0x20	File Reference to Base Entry					Next Attribute ID		Align to 4B Boundary		Number of this MFT Entry						

→ MFT Entry Header의 레이아웃

Address Range	Size	Field Name	Description
0x00~0x03	4 Byte	Signature	Signature : "FILE"
0x04~0x05	2 Byte	Offset to Fixup Array	Fixup 배열의 시작 위치
0x06~0x07	2 Byte	Entries in Fixup Array	Fixup 배열이 포함하는 항목 수
0x08~0x0F	8 Byte	\$LogFile Sequence Number	\$LogFile에 존재하는 해당 파일의 트랜잭션 위치
0x10~0x11	2 Byte	Sequence Number	순서 번호로 MFT Entry 생성 후 할당/해제 시 1씩 증가
0x12~0x13	2 Byte	Hard Link count	해당 MFT Entry에 연결된 하드 링크
0x14~0x15	2 Byte	Offset to File Attribute	해당 Entry의 첫 번째 파일 속성의 위치
0x16~0x17	2 Byte	Flags	0x01 : 사용 중, 0x02 : 디렉터리
0x18~0x1B	4 Byte	Real Size of MFT Entry	실제 사용 중인 크기
0x1C~0x1F	4 Byte	Allocated Size of MFT Entry	MFT Entry에 할당된 크기 (1,024 Byte)
0x20~0x27	8 Byte	File Reference to Base Entry	해당 Entry가 non-base일 때 base Entry 주소
0x28~0x29	2 Byte	Next Attribute ID	다음 속성 ID
0x2A~0x2B	2 Byte	Align to 4B Boundary	XP에만 존재
0x2C~0x2F	4 Byte	Number of this MFT Entry	XP에만 존재
MFT Entry Header 0x00 ~ 0x2F ( Size : 48 Byte )			

→ MFT Entry Header에 포함되어 있는 정보

- 위의 정보 중에서 중요한 필드
  - Offset to Fixup Array, \$LogFile Sequence Number
  - Offset to File Attribute, File Reference to Base Entry

MFT x																
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h	46	49	4C	45	30	00	03	00	A3	0C	24	0D	00	00	00	FILE0...£.\$...
0010h	01	00	01	00	38	00	01	00	A8	01	00	00	00	04	00	....8....
0020h	00	00	00	00	00	00	00	06	00	00	00	00	00	00	00	.....
0030h	30	00	2D	8D	00	00	00	00	10	00	00	00	60	00	00	00
0040h	00	00	18	00	00	00	00	48	00	00	00	18	00	00	00	.....H.....
0050h	5E	A0	B0	95	E6	19	D2	01	5E	A0	B0	95	E6	19	D2	01
0060h	5E	A0	B0	95	E6	19	D2	01	5E	A0	B0	95	E6	19	D2	01
0070h	06	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0080h	00	00	00	00	00	01	00	00	00	00	00	00	00	00	00	.....
0090h	00	00	00	00	00	00	00	30	00	00	00	68	00	00	00	.....0.h...
00A0h	00	00	18	00	00	00	03	00	4A	00	00	00	18	00	01	00

## Base & Non-base MFT Entry

- MFT Entry Header (0x20 ~ 0x27)까지의 데이터
  - Base Entry or Non-base Entry 주소를 담고 있음
  - 파일 속성 내용이 큰 경우, 하나 이상의 MFT Entry 사용
- Base MFT Entry → 해당 파일의 첫 MFT Entry를 사용
- Non-base MFT Entry → Base MFT Entry를 제외하고 나머지 부분을 담고 있는 Entry

- File Reference to Base MFT Entry
  - 00 00 00 00 00 00 00 00 → Base MFT Entry 구조
  - 00 00 00 00 00 00 00 00 이 아닌 경우 → Non-Base MFT Entry 구조

## File Reference Address

Sequence Value	MFT Entry Address
16bit ( 2 Byte )	48bit ( 6 Byte )

- MFT Entry Address / 48 비트의 고유한 주소 보유
  - File Reference Address = Sequence Number + MFT Entry Address
- Sequence Value → MFT Entry Header의 0x10 ~ 0x11에 존재
- MFT Entry Address
  - \$MFT 파일을 시작으로 0이라는 값을 가지며 순차적으로 1씩 증가

## Fixup Array

- Fixup Array 영역 → MFT Entry의 데이터 무결성을 판단하는 영역
  - MFT Entry → 1024바이트
  - 하나의 MFT Entry는 2개의 섹터를 사용
- Fixup Array
  - 2섹터 활용 → 해당 MFT Entry가 무결성을 가지고 있는지 판단하기 위해서
  - 각 섹터 맨 마지막 2바이트를 이용, Fixup Array를 구현

위 방식은 아래와 같은 구조에서도 활용됨.

- FILE Records (MFT Entries) in the \$MFT
- INDX Records in directories and other indexes
- RCRD Records in the \$LogFile
- RSTR Records in the \$LogFile

MFT	x	0 1 2 3 4 5 6 7 8 9 A B C D E F	0123456789ABCDEF
0000h	46 49 4C 45	30 00 03 00 A3 0C 24 0D 00 00 00 00 F	FILED...£,\$.....
0010h	01 00 01 00	38 00 01 00 A8 01 00 00 00 04 00 00	....8.....
0020h	00 00 00 00	00 00 00 06 00 00 00 00 00 00 00 00 00 00	.....
0030h	30 00 2D 8D	00 00 00 10 00 00 00 60 00 00 00 00 00 00 00	0.....H.....
0040h	00 00 18 00	00 00 00 48 00 00 18 00 00 00 00 00 00 00 00	.....H.....
0050h	5E A0 B0 95	E6 19 D2 01 5E A0 B0 95 E6 19 D2 01	^ °•æ.Ø. ^ °•æ.Ø.
0060h	5E A0 B0 95	E6 19 D2 01 5E A0 B0 95 E6 19 D2 01	^ °•æ.Ø. ^ °•æ.Ø.
0070h	06 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0080h	00 00 00 00	00 01 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0090h	00 00 00 00	00 00 00 30 00 00 00 68 00 00 00 00 00 00 00	.....0..h..
00A0h	00 00 18 00	00 00 03 00 4A 00 00 18 00 01 00	.....J.....
00B0h	05 00 00 00	00 05 05 5E A0 B0 95 E6 19 D2 01	.....^ °•æ.Ø.
00C0h	5E A0 B0 95	E6 19 D2 01 5E A0 B0 95 E6 19 D2 01	^ °•æ.Ø. ^ °•æ.Ø.
00D0h	5E A0 B0 95	E6 19 D2 01 00 40 00 00 00 00 00 00 00 00 00	^ °•æ.Ø. @.....
00E0h	00 40 00 00	00 00 00 06 00 00 00 00 00 00 00 00 00 00 00	@.....
00F0h	04 03 24 00	4D 00 46 00 54 00 00 00 00 00 00 00 00 00 00	..\$.M.F.T.....
0100h	80 00 00 00	50 00 00 01 00 40 00 00 00 01 00	€.....P.....@.....
0110h	00 00 00 00	00 00 00 FF 3E 00 00 00 00 00 00 00 00 00	.....ý>.....
0120h	40 00 00 00	00 00 00 00 F0 03 00 00 00 00 00 00 00 00	@.....ð.....ð.....
0130h	00 00 F0 03	00 00 00 00 00 F0 03 00 00 00 00 00 00 00	..ð.....ð.....
0140h	32 80 3C 00	00 0C 32 80 02 28 38 1A 00 FA FF FF	2€<..2€(8..úý
0150h	80 80 80 80	00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....^.....
			0.

- Fixup Array
  - MFT Entry / 섹터 2개 사용
  - 두 번째 섹터 맨 마지막의 값 확인시 → 값 동일한 것을 확인 가능
  - 값이 하나라도 변조된 경우, 무결성 훼손

## MFT - Attributes

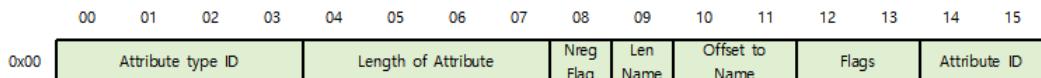
- Attributes 영역 / MFT 영역에서 가장 큰 비중을 차지하고 있으며, 가장 중요한 영역
  - 가장 방대한 속성 값이 존재하며, 디렉터리 및 파일의 모든 내용을 담고 있는 영역
  - 각 파일의 메타 데이터는 속성이라는 구조를 통해서 표현
- Attributes 영역 / 여러개의 속성 구조로 관리가 됨
  - 하나의 속성 구조 → 속성 헤더와 속성 내용을 가짐
  - MFT Attribute / Attribute 크기에 따라서 Resident 속성과 Non-Resident 속성으로 나뉨
  - Resident Attribute → 속성 헤더 바로 뒤에 바로 속성 내용 저장 (MFT Entry 내부)
  - Non-Resident Attribute → 속성 내용의 크기가 너무 커서 별도의 클러스터에 저장 (MFT Entry 외부)
- \$DATA 속성의 경우, 파일의 크기에 따라서 속성이 달라짐.
  - 파일의 크기 < 700 byte : Resident 속성
  - 파일의 크기 > 700 byte : Non-Resident 속성

Attribute 종류 → 아래와 같음.

Attribute Value	Attribute Name	Description
0x10	\$STANDARD_INFORMATION	파일의 생성, 접근, 수정 시간, 소유자 등의 일반적인 정보
0x20	\$ATTRIBUTE_LIST	추가적인 속성들의 리스트
0x30	\$FILE_NAME	파일 이름(유니코드), 파일의 생성, 접근, 수정 시간
0x40	\$VOLUME_VERSION	볼륨 정보 (Windows NT 1.2 버전에만 존재)
0x40	\$OBJECT_ID	16바이트의 파일, 디렉터리의 고유값
0x50	\$SECURITY_DESCRIPTOR	파일의 접근 제어와 보안 속성
0x60	\$VOLUME_NAME	볼륨 이름
0x70	\$VOLUME_INFORMATION	파일 시스템의 버전과 다양한 플래그
0x80	\$DATA	파일 내용
0x90	\$INDEX_ROOT	인덱스 트리의 루트 노드
0xA0	\$INDEX_ALLOCATION	인덱스 트리의 루트와 연결된 노드
0xB0	\$BITMAP	\$MFT와 인덱스의 할당 정보 관리
0xC0	\$SYMBOLIC_LINK	심볼릭 링크 정보 (Windows 2000+)
0xC0	\$REPARSE_POINT	심볼릭 링크에서 사용하는 reparse point 정보 (Windows 2000+)
0xD0	\$EA_INFORMATION	OS/2 응용 프로그램과 호환성을 위해 사용 (HPFS)
0xE0	\$EA	OS/2 응용 프로그램과 호환성을 위해 사용 (HPFS)
0x100	\$LOGGED.Utility_STREAM	암호화된 속성의 정보와 키값 (Windows 2000+)

- Attribute는 총 17개로 위와 같은 값이 포함됨.

### Common Attribute Header



- Common Attribute Header / 이름 해석 그대로 Attribute Header의 가장 기본적인 포맷
  - Resident Attribute Header & Non-Resident Attribute Header에 필수적으로 들어가는 구조

Address Range	Size	Field Name	Description
0x00~0x03	4 Byte	Attribute Type ID	속성 타입 식별값
0x04~0x07	4 Byte	Length of Attribute	속성 헤더를 포함한 속성 전체 길이
0x08~0x08	1 Byte	Non-resident flag	Non-resident 속성인지 확인 (0x01 : Non-resident 속성)
0x09~0x09	1 Byte	Length of Name	자신의 속성 이름 길이
0x0A~0x0B	2 Byte	Offset to Name	속성 이름이 저장된 곳의 시작 위치
0x0C~0x0D	2 Byte	Flags	속성의 상태 표현
0x0E~0x0F	2 Byte	Attribute identifier	속성의 고유한 식별자로 MFT Entry에 같은 속성이 여러개 일 경우 구별하기 위해서 사용
<b>Common Attribute Header 0x00 ~ 0x0F ( Size : 16 Byte )</b>			

- Non-Resident Flag
  - 0x00 : Resident 속성
  - 0x01 : Non-Resident 속성
- Flags
  - 0x0001 : 압축된 속성
  - 0x4000 : 암호화된 속성
  - 0x8000 : Sparse 속성

## Resident Attribute Header

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0x00	Attribute type ID			Length of Attribute				Nreg Flag	Len Name	Offset to Name		Flags		Attribute ID		
0x10	Length of the Attribute			Offset to Attribute data		Idx Flag	Un used	Attribute Name(if exist,)								

- Resident Attribute Header
  - Common Attribute Header 0x00 ~ 0x0F 같은 값을 가진다.
  - Layout Structure → 확인시 위와 같음

Address Range	Size	Field Name	Description
0x00~0x03	4 Byte	Attribute Type ID	속성 타입 식별값
0x04~0x07	4 Byte	Length of Attribute	속성 헤더를 포함한 속성 전체 길이
0x08~0x08	1 Byte	Non-resident flag	Non-resident 속성인지 확인 (0x01 : Non-resident 속성)
0x09~0x09	1 Byte	Length of Name	자신의 속성 이름 길이
0x0A~0x0B	2 Byte	Offset to Name	속성 이름이 저장된 곳의 시작 위치
0x0C~0x0D	2 Byte	Flags	속성의 상태 표현
0x0E~0x0F	2 Byte	Attribute identifier	속성의 고유한 식별자로 MFT Entry에 같은 속성이 여러개 일 경우 구별하기 위해서 사용
0x10~0x13	4 Byte	Length of the Attribute	헤더 뒤에 오는 속성 내용의 크기
0x14~0x15	2 Byte	Offset to Attribute Data	속성 내용이 시작하는 위치
0x16~0x16	1 Byte	Indexed flag	1이라는 값을 가지면 인덱스 된 속성
0x17~0x17	1 Byte	Unused	Unused
0x18~0x1F	8 Byte	Attribute Name	속성 이름이 있는 경우 속성 이름, 없으면 바로 속성 내용
<b>Resident Attribute Header 0x00 ~ 0x1F ( Size : 32 Byte )</b>			

- Offset to Attribute Data
  - 속성 내용이 시작하는 위치
- Index Flag
  - 1이라는 값을 가진 경우, 인덱스 된 속성 의미
  - \$FILE\_NAME 속성 / 1로 설정
- Attribute Name
  - 속성 명이 들어가는 자리. / 위 예제에서는 속성명이 없음

### Non-Resident Attribute Header

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15										
0x00	Attribute type ID			Length of Attribute				Nreg Flag	Len Name	Offset to Name		Flags		Attribute ID												
0x10	Start Virtual Cluster Number of the runlist							End Virtual Cluster Number of the runlist																		
0x20	Offset to runlist	Compression unit size	Unused			Allocated size of attribute content																				
0x30	Real size of attribute content							Initialized size of attribute content																		
0x40	Attribute Name(if exist)																									

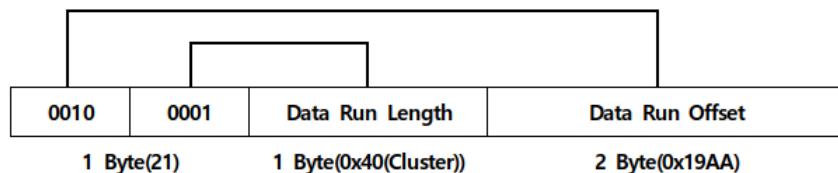
- Non-Resident Attribute Header 도 Common Attribute Header를 가지고 있음

Address Range	Size	Field Name	Description
0x00~0x03	4 Byte	Attribute Type ID	속성 타입 식별값
0x04~0x07	4 Byte	Length of Attribute	속성 헤더를 포함한 속성 전체 길이
0x08~0x08	1 Byte	Non-resident flag	Non-resident 속성인지 확인 (0x01 : Non-resident 속성)
0x09~0x09	1 Byte	Length of Name	자신의 속성 이름 길이
0x0A~0x0B	2 Byte	Offset to Name	속성 이름이 저장된 곳의 시작 위치
0x0C~0x0D	2 Byte	Flags	속성의 상태 표현
0x0E~0x0F	2 Byte	Attribute identifier	속성의 고유한 식별자로 MFT Entry에 같은 속성이 여러개 일 경우 구별하기 위해서 사용
0x10~0x17	8 Byte	Start VCN of the runlist	속성 내용이 담긴 런리스트의 시작 VCN
0x18~0x1F	8 Byte	End VCN of the runlist	속성 내용이 담긴 런리스트의 끝 VCN
0x20~0x21	2 Byte	Offset to runlist	속성 내부의 런리스트 시작 위치
0x22~0x23	2 Byte	Compression unit size	압축 속성일 경우에 압축 단위
0x24~0x27	4 Byte	Unused	Unused
0x28~0x2F	8 Byte	Allocated size of attribute content	속성 내용에 할당된 클러스터의 크기
0x30~0x37	8 Byte	Real size of attribute content	속성 내용의 실제 크기
0x38~0x3F	8 Byte	Initialized size of attribute content	속성 내용의 초기화된 크기
0x40~0x47	8 Byte	Attribute Name	속성 이름이 있는 경우 속성 이름, 없으면 바로 속성 내용
<b>Non-Resident Attribute Header 0x00 ~ 0x47 ( Size : 72 Byte )</b>			

- offset to Runlist : 런 리스트 시작 위치
  - 런 리스트 시작 위치 = 해당 속성 헤더 시작 주소 + Offset to Runlist Value

## Cluster Runs

- Non-Resident 속성
  - 할당 받는 클러스터 → 파일에 따라서 수천 수만개가 됨.
  - 비연속적으로 할당된 클러스터를 효과적으로 관리하기 위해서
- NTFS \$MFT → 클러스터 런(Cluster Runs) 활용
  - 클러스터 런 → 런리스트 (RunList) 형태로 관리



- 위의 구조 기반
  - Data Run의 첫 바이트 기준 / Data Run Length, Data Run Offset의 길이가 정해짐.

- Logical Cluster Number (LCN) : 볼륨의 첫 번째 클러스터부터 순차적인 번호
  - Data Run Offset
- Virtual Cluster Number (VCN) : 파일의 첫 번째 클러스터부터 순차적인 번호
  - Data Run Length
- Data Run Length (위 사진 기준)
  - 0x40 → 64 클러스터
  - 1클러스터 당 4KB → 64 클러스터는  $64 * 4 = 256\text{KB}$  크기를 가지게 됨.

## MFT - Attribute Types

- 일반적인 파일은 총 3가지의 속성 보유
  - \$STANDARD\_INFOMATION : 파일의 생성, 접근, 수정 시간, 소유자 등의 정보가 담김
  - \$FILE\_NAME : 파일이름 (유니코드), 파일의 생성, 접근, 수정 시간 등의 정보가 담김
  - \$DATA : 파일 내용

\$ 이후에 대문자 문자열이 오는 경우 → 속성 이름

\$ 이후 첫글자만 대문자인 경우, 메타데이터 파일을 의미

### Attribute - \$STANDARD\_INFOMATION (0x10)

- \$STANDARD\_INFOMATION → 속성 식별 값 0x10을 가지는 속성

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15				
Attribute Header (Resident or Non-resident)																			
0x00                      Created Time								Modified Time											
0x10                      MFT Modified Time								Last Accessed Time											
0x20                      Flags		Maximum number of version				Version		Class ID											
0x30                      Owner ID		Security ID				Quota Charged													
0x40                      Update Sequence Number (UCN)																			

Address Range	Size	Field Name	Description
0x~~	~ Byte	Attribute Header	Resident OR Non-Resident Attribute Header
0x00~0x07	8 Byte	Created Time	생성시간
0x08~0x0F	8 Byte	Modified Time	수정시간
0x10~0x17	8 Byte	MFT Modified Time	MFT 수정시간
0x18~0x1F	8 Byte	Last Accessed Time	마지막 접근시간
0x20~0x23	4 Byte	Flags	Flag 값
0x24~0x27	4 Byte	Maximum number of version	파일의 버전 최댓값
0x28~0x2B	4 Byte	Version	파일의 버전
0x2C~0x2F	4 Byte	Class ID	인덱스 된 클래스 ID
0x30~0x33	4 Byte	Owner ID	파일 소유자의 ID 값으로 \$Quota 파일에서 인덱스로 사용
0x34~0x37	4 Byte	Security ID	\$Secure 파일의 인덱스로 사용되고 ACL 적용 때 사용
0x38~0x3F	8 Byte	Quota Charged	사용자 할당량 중 해당 파일의 할당 된 크기
0x40~0x47	8 Byte	Update Sequence Number	파일의 USN 값으로 \$UsnJrnI에서 인덱스로 사용
<b>\$STANDARD_INFORMATION 0x00 ~ 0x47 ( Size : 72 Byte )</b>			

### Attribute - \$ATTRIBUTE\_LIST (0x20)

- \$ATTRIBUTE\_LIST / 속성 식별 값 0x20을 가지는 속성
- 파일이나 디렉터리의 속성 크기가 커져, 하나의 MFT 엔트리에 담을 수 없는 경우 사용

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0x~~	Attribute Header (Resident or Non-resident)															
0x00	Attribute Type		Length of Entry		Len Name	Offset Name	Start VCN of attribute									
0x10	File Reference Address						Attribute ID		Attribute Name (if exist.)							

Address Range	Size	Field Name	Description
0x~~	~ Byte	Attribute Header	Resident OR Non-Resident Attribute Header
0x00~0x03	4 Byte	Attribute Type	속성 타입
0x04~0x05	2 Byte	Length of Entry	엔트리의 길이
0x06~0x06	1 Byte	Length Name	이름의 길이
0x07~0x07	1 Byte	Offset Name	이름 시작 위치
0x08~0x0F	8 Byte	Start VCN of Attribute	속성 시작 VCN 위치
0x10~0x17	8 Byte	File Reference Address	속성 위치의 파일 참조 주소
0x18~0x19	2 Byte	Attribute ID	속성 ID
0x1A~0x1F	6 Byte	Attribute Name	속성 이름이 있는 경우 속성이를, 없으면 바로 속성 내용
<b>\$ATTRIBUTE_LIST 0x00 ~ 0x1F ( Size : 32 Byte )</b>			

### Attribute - \$FILE\_NAME (0x30)

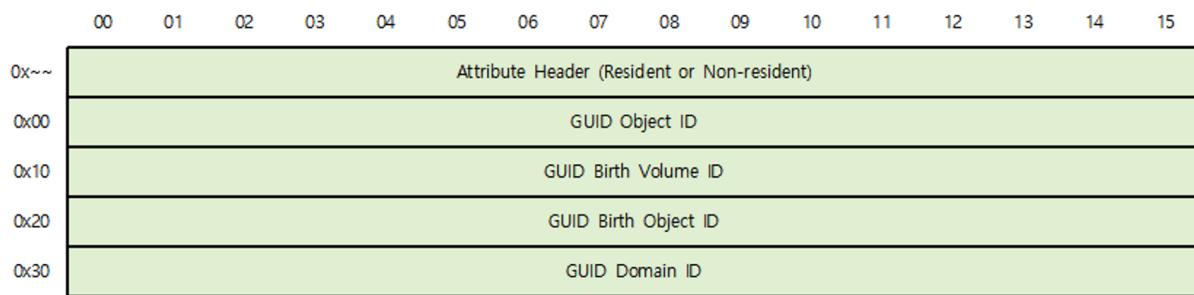
- \$FILE\_NAME / 속성 식별 값 0x30을 가지는 속성
- 파일이나 디렉토리의 이름을 담는 속성으로 해당 속성으로 파일의 이름이나 디렉터리의 이름 파악 가능.

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	
Attribute Header (Resident or Non-resident)																	
0x~~	Attribute Header (Resident or Non-resident)																
0x00	File Reference Addr of parent directory										Created Time						
0x10	Modified Time										MFT Modified Time						
0x20	Last Accessed Time										Allocated size of file						
0x30	Real Size of file										Flags		Reparse Value				
0x40	Len Name Space	Name Space	File Name (as Length)														

Address Range	Size	Field Name	Description
0x~~	~ Byte	Attribute Header	Resident OR Non-Resident Attribute Header
0x00~0x07	8 Byte	File Reference Addr of parent directory	부모 디렉터리의 파일 참조 주소
0x08~0x0F	8 Byte	Created Time	생성시간
0x10~0x17	8 Byte	Modified Time	수정시간
0x18~0x1F	8 Byte	MFT Modified Time	MFT 수정시간
0x20~0x27	8 Byte	Last Accessed Time	마지막 접근시간
0x28~0x2F	8 Byte	Allocated Size of File	파일이 할당된 크기
0x30~0x37	8 Byte	Real Size of File	파일의 실제 크기
0x38~0x3B	4 Byte	Flags	\$STANDARD_INFORMATION 속성 플래그와 동일
0x3C~0x3F	4 Byte	Reparse Value	해당 속성의 Reparse Point
0x40~0x40	1 Byte	Length Name	이름 길이
0x41~0x41	1 Byte	Name Space	이름의 표현 형식
0x42~0x4F	14 Byte	File Name	유니코드로 인코딩된 파일 이름 (유동적)
<b>\$FILE_NAME 0x00 ~ 0x4F ( Size : 80 Byte )</b>			

### Attribute - \$OBJECT\_ID (0x40)

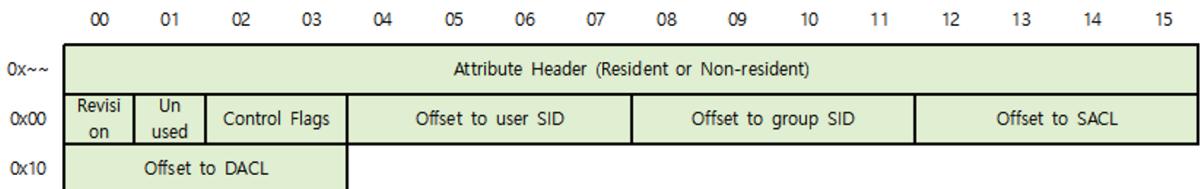
- \$OBJECT\_ID는 속성 식별 값 0x40을 가지는 속성
- 모든 MFT 레코드에는 고유한 GUID 값이 할당됨.
- 레코드에서는 아래 4개 값을 다룸
  - GUID Object ID, GUID Birth Volume ID
  - GUID Birth Object ID, GUID Domain ID



Address Range	Size	Field Name	Description
0x~~	~ Byte	Attribute Header	Resident OR Non-Resident Attribute Header
0x00~0x0F	16 Byte	GUID Object ID	각각 파일 또는 디렉터리가 가지는 고유한 128bit의 ID
0x10~0x1F	16 Byte	GUID Birth Volume ID	파일이 생성된 볼륨의 ID 값
0x20~0x2F	16 Byte	GUID Birth Object ID	파일이 처음 생성됐을 때 받은 오브젝트 ID 값
0x30~0x3F	16 Byte	GUID Domain ID	네트워크 환경에서 사용되는 도메인 ID 값
<b>\$OBJECT_ID 0x00 ~ 0x3F ( Size : 64 Byte )</b>			

### Attribute - \$SECURITY\_DESCRIPTOR (0x50)

- \$SECURITY\_DESCRIPTOR / 속성 식별 값 0x50을 가지는 속성
- \$SECURITY\_DESCRIPTOR
  - 파일이 옮겨지거나 이름이 바뀌어도, 유지되는 파일이나 디렉토리 고유의 숫자를 가짐



Address Range	Size	Field Name	Description
0x~~	~ Byte	Attribute Header	Resident OR Non-Resident Attribute Header
0x00~0x00	1 Byte	Revision	대부분 1의 값을 가짐
0x01~0x01	1 Byte	Unused	Unused
0x02~0x03	2 Byte	Control Flags	Flag 값
0x04~0x07	4 Byte	Offset to user SID	유저 SID 값이 있는 주소
0x08~0x0B	4 Byte	Offset to group SID	그룹 SID 값이 있는 주소
0x0C~0x0F	4 Byte	Offset to SACL	SACL(System Access control list)의 주소
0x10~0x13	4 Byte	Offset to DACL	DACL(Directory Access Control List)의 주소
<b>\$SECURITY_DESCRIPTOR 0x00~0x13 ( Size : 20 Byte )</b>			

### Attribute - \$VOLUME\_NAME (0x60)

- \$VOLUME\_NAME / 속성 식별 값 0x60을 가지는 속성
  - 속성 → 단순히 볼륨 명을 다루는 속성

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0x~~	Attribute Header (Resident or Non-resident)															
0x00	Volume Name															

Address Range	Size	Field Name	Description
0x~~	~ Byte	Attribute Header	Resident OR Non-Resident Attribute Header
0x00~0x0F	16 Byte	Volume Name	볼륨 이름
\$VOLUME_NAME 0x00~0x0F ( Size : 16 Byte )			

### Attribute - \$VOLUME\_INFORMATION (0x70)

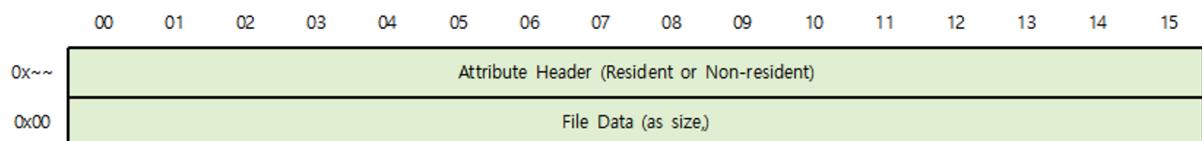
- \$VOLUME\_INFORMATION / 속성 식별 값 0x70을 가지는 속성
  - 볼륨의 버전 및 상태를 다루는 속성

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0x~~	Attribute Header (Resident or Non-resident)															
0x00	0x00000000								Major Num	Minor Num	Flags	0x00000000				

Address Range	Size	Field Name	Description
0x~~	~ Byte	Attribute Header	Resident OR Non-Resident Attribute Header
0x00~0x07	8 Byte	Reserved Area	0x0000000000000000
0x08~0x08	1 Byte	Major Number	주된 버전 번호
0x09~0x09	1 Byte	Minor Number	보조 버전 번호
0x0A~0x0B	2 Byte	Flags	Flag 값
0x0C~0x0F	4 Byte	Reserved Area	0x00000000
\$VOLUME_INFORMATION 0x00~0x0F ( Size : 16 Byte )			

### Attribute - \$DATA (0x80)

- \$DATA / 속성 식별값 0x80을 가지는 속성
- 해당 속성 데이터 크기가 700 byte 초과시
  - Resident → Non-Resident 형식으로 속성이 저장



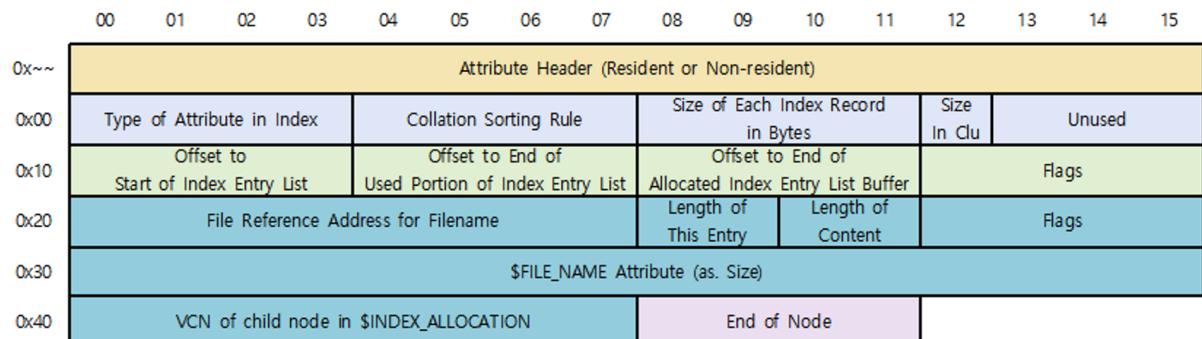
Address Range	Size	Field Name	Description
0x~~	~ Byte	Attribute Header	Resident OR Non-Resident Attribute Header
0x00~0x0F	16 Byte	File Data	파일 데이터 (유동적)
<b>\$DATA 0x00~0x0F ( Size : 16 Byte )</b>			

### Attribute - \$INDEX\_ROOT (0x90)

- \$INDEX\_ROOT / 속성 식별 값 0x90을 가지는 속성
  - 다른 속성들과 달리 좀 더 복잡한 구조를 가짐

Previous Attributes	Attribute Header	Index Root Header	Index Node Header	Index Entry 1	Index Entry 2	End of Node	Unused Space	Next Attributes
---------------------	------------------	-------------------	-------------------	---------------	---------------	-------------	--------------	-----------------

- \$INDEX\_ROOT의 Layout Structure



Address Range	Size	Field Name	Description
0x~~	~ Byte	Attribute Header	Resident OR Non-Resident Attribute Header
0x00~0x03	4 Byte	Type of Attribute in Index	인덱스 엔트리가 담고 있는 속성 식별 값 ( 디렉터리인 경우 0x30 )
0x04~0x07	4 Byte	Collation Sorting Rule	인덱스 엔트리가 담고 있는 형식(형식에 맞게 정렬됨)
0x08~0x0B	4 Byte	Size of Each Index Record in Bytes	\$INDEX_ALLOCATION 속성이 가지는 인덱스 레코드의 바이트 크기
0x0C~0x0C	1 Byte	Size of Each Index Record in Cluster	\$INDEX_ALLOCATION 속성이 가지는 인덱스 레코드의 클러스터 크기
0x0D~0x0F	3 Byte	Unused	Unused
0x10~0x13	4 Byte	Offset to Start of Index Entry List	인덱스 엔트리 목록의 시작 위치
0x14~0x17	4 Byte	Offset to End of Used Portion of Index Entry List	인덱스 엔트리의 실제 크기(인덱스 노드 헤더 포함)
0x18~0x1B	4 Byte	Offset to End of Allocated Index Entry List Buffer	인덱스 엔트리의 할당 크기(인덱스 노드 헤더 포함)
0x1C~0x1F	4 Byte	Flags	Flag 값
0x20~0x27	8 Byte	File Reference Address for Filename	해당 파일 및 디렉터리의 파일 참조 주소
0x28~0x29	2 Byte	Length of This Entry	해당 인덱스 엔트리의 총 크기
0x2A~0x2B	2 Byte	Length of Content	해당 인덱스 엔트리가 담고 있는 \$FILE_NAME 속성의 크기
0x2C~0x2F	4 Byte	Flags	Flag 값
0x30~0x3F	16 Byte	\$FILE_NAME Attribute	\$FILE_NAME의 속성이 들어가는 자리로 크기가 유동적으로 다다르기 때문에 정확한 값을 크기를 찾을 수 없다.
0x40~0x47	8 Byte	VCN of child node in \$INDEX_ALLOCATION	해당 인덱스 엔트리가 자식 노드를 가지는 경우 \$INDEX_ALLOCATION 속성에 위치한 자식 인덱스 노드의 위치
0x48~0x4B	4 Byte	End of Node	0xFFFFFFFF 으로 파일의 끝을 의미
<b>\$INDEX_ROOT 0x00~0x4B ( Size : 76 Byte )</b>			

### Attribute - \$INDEX\_ALLOCATION (0xA0)

- \$INDEX\_ALLOCATION 속성 식별 값 0xA0을 가지는 속성
- \$INDEX\_ALLOCATION 속성 추가시, \$BITMAP 속성 또한 같이 따라오게 됨.

Previous Attributes	Attribute Header	Index Record Header	Index Node Header	Index Entry 1	Index Entry 2	End of Node	Unused Space	Next Attributes

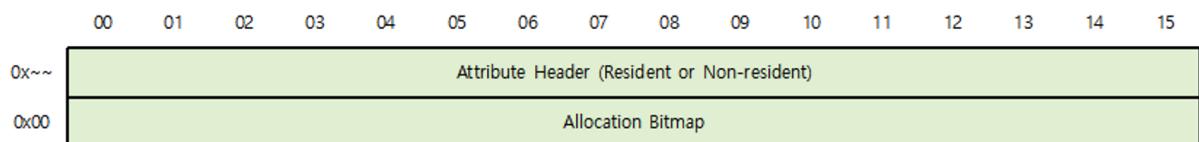
	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15						
0x~~	Attribute Header (Resident or Non-resident)																					
0x00	Signature("INDX")		Offset to Fixup Array		Num of Entries in FA		\$LogFile Sequence Number (LSN)															
0x10	The VCN of this record in the full index stream					Offset to Start of Index Entry List			Offset to End of Used Portion of Index Entry List													
0x20	Offset to End of Allocated Index Entry List Buffer			Flags			File Reference Address for Filename															
0x30	Length of This Entry	Length of Content	Flags			\$FILE_NAME Attribute (as. Size)																
0x40	\$FILE_NAME Attribute (as. Size)																					
0x50	VCN of child node in \$INDEX_ALLOCATION					End of Node																

Address Range	Size	Field Name	Description
0x~~	~ Byte	Attribute Header	Resident OR Non-Resident Attribute Header
0x00~0x03	4 Byte	Signature	시그니처 "INDX"
0x04~0x05	2 Byte	Offset to Fixup Array	Fixup Array의 위치
0x06~0x07	2 Byte	Num of Entries in Fixup Array	Fixup Array에 저장된 항목의 수
0x08~0x0F	8 Byte	\$LogFile Sequence Number (LSN)	\$LogFile에 존재하는 해당 파일의 트랜잭션 위치
0x10~0x17	8 Byte	The VCN of this record in the full index stream	\$INDEX_ALLOCATION 속성에서 해당 인덱스 레코드가 저장된 위치
0x18~0x1B	4 Byte	Offset to Start of Index Entry List	인덱스 엔트리 목록의 시작 위치
0x1C~0x1F	4 Byte	Offset to End of Used Portion of Index Entry List	인덱스 엔트리의 실제 크기(인덱스 노드 헤더 포함)
0x20~0x23	4 Byte	Offset to End of Allocated Index Entry List Buffer	인덱스 엔트리의 할당 크기(인덱스 노드 헤더 포함)
0x24~0x27	4 Byte	Flags	Flag 값
0x28~0x2F	8 Byte	File Reference Address for Filename	해당 파일 및 디렉터리의 파일 참조 주소
0x30~0x31	2 Byte	Length of This Entry	해당 인덱스 엔트리의 총 크기
0x32~0x33	2 Byte	Length of Content	해당 인덱스 엔트리가 담고 있는 \$FILE_NAME 속성의 크기
0x34~0x37	4 Byte	Flags	Flag 값
0x38~0x4F	24 Byte	\$FILE_NAME Attribute	\$FILE_NAME의 속성이 들어가는 자리로 크기가 유동적으로 다 다르기 때문에 정확한 값을 크기를 찾을 수 없다.
0x40~0x47	8 Byte	VCN of child node in \$INDEX_ALLOCATION	해당 인덱스 엔트리가 자식 노드를 가지는 경우 \$INDEX_ALLOCATION 속성에 위치한 자식 인덱스 노드의 위치
0x48~0x4B	4 Byte	End of Node	0xFFFFFFFF 으로 파일의 끝을 의미
<b>\$INDEX_ALLOCATION 0x00~0x4B ( Size : 76 Byte )</b>			

INDEX NAME	Index Data	Location
\$I30	\$FILE_NAME 속성	각각 디렉터리의 MFT
\$SDH	Security Descriptors	\$Secure 메타 데이터 파일
\$SII	Security IDs	\$Secure 메타 데이터 파일
\$O	Object IDs	\$ObjId 메타 데이터 파일
\$O	Owner IDs	\$Quota 메타 데이터 파일
\$Q	Quotas	\$Quota 메타 데이터 파일

### Attribute - \$BITMAP (0xB0)

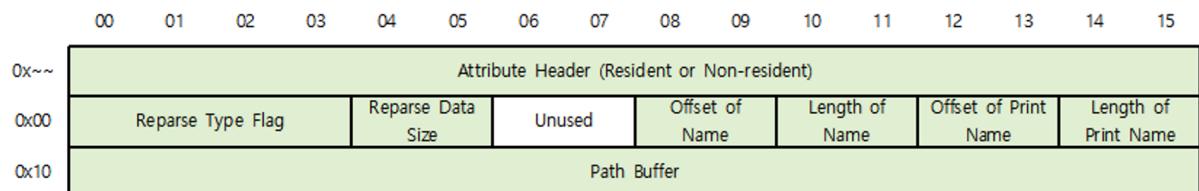
- \$BITMAP / 속성 식별 값 0xB0을 가지는 속성
  - NTFS에 할당 정보를 관리 해야하는 데이터들이 많은데
  - 이중 MFT와 index의 할당 정보를 관리하는데 사용하는 속성



Address Range	Size	Field Name	Description
0x~~	~ Byte	Attribute Header	Resident OR Non-Resident Attribute Header
0x00~0x0F	16 Byte	Allocation Bitmap	BITMAP 할당 정보
\$BITMAP 0x00~0x0F ( Size : 16 Byte )			

### Attribute - \$REPARSE\_POINT (0xC0)

- \$REPARSE\_POINT / 속성 식별 값 0xC0을 가지는 속성
  - NTFS 5.0 버전 이후로 생성된 속성
  - 이전 버전 → \$SYMBOLIC\_LINK 속성 존재
  - 마운트, 잭션, 심볼릭 링크 등에 대한 정보를 담고 있는 속성



Address Range	Size	Field Name	Description
0x~~	~ Byte	Attribute Header	Resident OR Non-Resident Attribute Header
0x00~0x03	4 Byte	Reparse Type Flag	Reparse Type Flag
0x04~0x05	2 Byte	Reparse Data Size	Reparse 데이터 크기
0x06~0x07	2 Byte	Unused	Unused (사용 안함)
0x08~0x09	2 Byte	Offset of Name	대상 이름의 오프셋
0x0A~0x0B	2 Byte	Length of Name	대상 이름의 길이
0x0C~0x0D	2 Byte	Offset of Print Name	대상의 출력 이름의 오프셋
0x0E~0x0F	2 Byte	Length of Print Name	대상의 출력 이름의 길이
0x10~0x1F	- Byte	Path Buffer	Path Buffer (유동적)
<b>\$REPARSE_POINT 0x00~0x1F ( Size : 32 Byte )</b>			

### Attribute - \$EA\_INFOMATION (0xD0) & \$EA(0xE0)

- OS/2 & WinNT Server에서 확장된 HPFS 속성을 구현시 사용된 속성

## MFT 레퍼런스

- 해당 글을 중점으로 정리를 진행하였습니다.

[MFT File System Structure Analysis](#)

- 그 외로 참고한 문서들입니다.

[https://github.com/proneer/Slides/blob/master/Filesystem/\(FP\)\\_NTFS.pdf](https://github.com/proneer/Slides/blob/master/Filesystem/(FP)_NTFS.pdf)

 <http://forensic-proof.com/archives/470>

 <http://forensic-proof.com/archives/584>

**MFT Entry Attribute Concepts | NTFS Concepts**

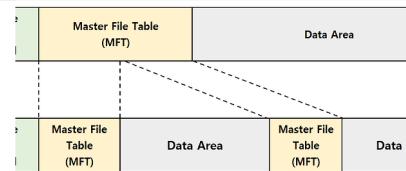
MFT Entry Attribute Concepts / NTFS Concepts from File System Forensic Analysis

 [https://flylib.com/books/en/2.48.1/mft\\_entry\\_attribute\\_concepts.html](https://flylib.com/books/en/2.48.1/mft_entry_attribute_concepts.html)

### MFT(Master File Table) 구조

지난 시간에 NTFS 파일 시스템을 배웠습니다. 이번에는 NTFS 파일 시스템에서 가장 중요한 MFT 파일의 구조를 배워 보겠습니다. MFT 파일이란? MFT 파일은 NTFS 파일 시스템에서 파일, 디렉터리, 메타데이터를 모두 "파일" 형태로 관리하는 파일입니다. 예전에 배웠던 FAT 파일 시스템에서 본 디렉터리 엔트리의 상위 개념이라

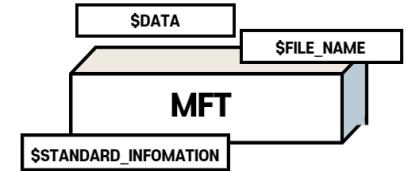
👉 <https://lemonpoo22.tistory.com/217>



### MFT File System Structure Analysis

POST URL : <https://ws1004-4n6.notion.site/MFT-File-System-Structure-Analysis-24a7386dae0246758173188eaab5bd2f>  
MFT File System Structure Analysis MFT(Master File Table) ws1004-4n6.notion.site ★읽어 보시면서 이상한 부분이나 잘못된 개념, 오탈자가 있다면 댓글로 알

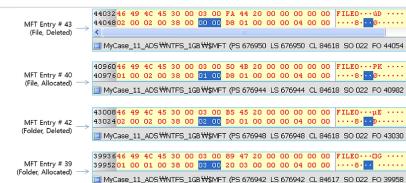
👉 <https://blog.forensicresearch.kr/32>



### [File System] NTFS (19) - MFT 분석시 고려 사항

안녕하세요, 도깨비 포렌식입니다. 오늘은 NTFS에서 MFT 분석시 고려사항에 대해 알아보겠습니다. 모두 화이팅하세요!!! MFT 분석 고려 사항 일부 파일/디렉토리는 2개의 \$DATA 속성을 갖고 있을 수도 있습니다. 추가적인 \$DATA 속성은 윈도우에서 보이지 않는 영역이므로, 데이터를 숨기는데 이용될 수도 있습니다. MFT 엔

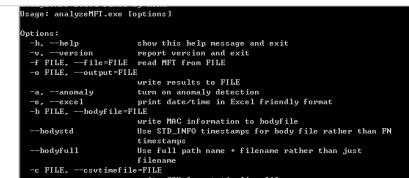
👉 <https://goblinforensics.tistory.com/519>



### 윈도우 침해사고 MFT 및 이벤트로그 분석

MFT 분석 MFT는 Master File Table의 약자로 NTFS 파일 시스템에 존재하는 모든 폴더, 파일의 메타 데이터 정보를 저장합니다. 파일로는 %ROOT%/\$MFT 형태로 존재하지만, 탐색기 상에서는 확인할 수 없어서 아래 두 가지 방법을 통해 추출합니다. 1. forecopy를 통한 \$MFT 추출 2. Disk Image를 통한 \$MFT 추출 MFT 정보를

👉 <https://nampill.tistory.com/entry/윈도우-침해사고-MFT-및-이벤트로그-분석>



### velog

개발자들을 위한 블로그 서비스. 어디서 글 쓸지 고민하지 말고 블로그에서 시작하세요.

velog

👉 <https://velog.io/@rinm/NTFS-structure>

### [운영체제] MTF 분석을 위한 NTFS 파일 시스템 이론

File System - 파일시스템은 보조 기억 장치에 데이터의 저장과 검색 방식을 제어하는데 사용 → 체계적인 저장 방식과 검색 방식을 사용하여 데이터 저장, 검색 등을 최적화 NTFS(New Technology File System) - 마이크로소프트에 의해 개발 - 1993년 Windows NT 3.1과 함께 발표 - Bitmap, MFT(Master File Table)

👉 <https://dyoerr9030.tistory.com/entry/악성코드-MTF-분석을-위한-NTFS-파일-시스템-이론>

