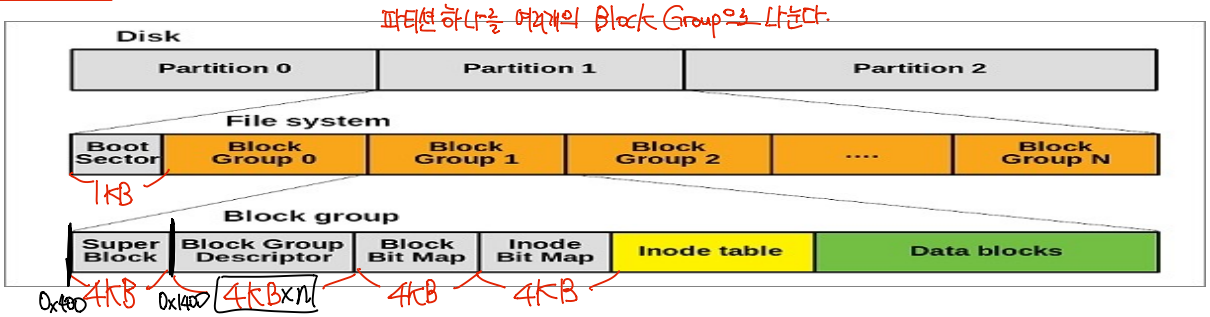
ext2 file system Walkthrough

- Block Group 개념이 도입된 이유
  - ⇒ Seek time을 줄이기 위해
  - ⇒ 연관 데이터를 Grouping함으로 seek time을 줄일 수 있다.
- BGD에는 각 Block Group의 정보를 담는다.
- Block bitmap은 Data Block의 사용여부 bitmap

ext2 Background

- **Linux용 파일 시스템**
  - 1993 by Rémy Card
  - Block group
    - block 을 여러 개의 block group으로 나눔
      - 헤드의 seek time 최소화

파티션 하나를 여러개의 Block Group으로 나눈다.



4KB = 0x1000

- Boot Block = 1KB
- Super Block = 1 Block, Block Descriptor = n Block, Block Bit Map 1 Block, inode Bit Map = 1 Block

4KB

BGD

- 각 block 그룹에 대한 정보는 super block 바로 뒤에 있는 block 에 저장된 디스크립터 테이블에 보관
- 각 그룹의 앞에 있는 2 blocks : 사용 중인 block과 inode를 표시하는 Block Bit Map / inode Bit Map
- 각 비트맵은 하나의 block에 들어가야 하기 때문에 block group 의 최대 block의 갯수= block 크기의 8배
  - (if one block is 4KB) 4KB * 8 = 32,768 (32KB) ⇒ 하나의 Block에 들어가는 bit 개수 = 32768 bit = 32KB ⇒ 즉 하나의 bitmap에서 관리하는 Data block 또는 Inode block의 개수는 32768개
  - 하나의 block 그룹이 가질 수 있는 최대 block 의 갯수 : 32KB *4KB = 128MB
  - 10GB 파티션의 최대 block 그룹 수 : 10GB(10240MB) / 128 MB = 80

각 bitmap이 여러 block의 정보를 관리한다면 데이터를 읽을 때 여러번 디스크 접근해야 함 따라서 Inode/Data block의 size가 4KB = 1 block인것.

따라서 32768개의 block × block size 4KB = 128MB

⇒ 하나의 block group이 관리하는 최대 Inode/Data block의 크기

ext2 Background

- **Super block**
  - struct ext2_super_block in include/linux/ext2_fs.h
  - offset 1024 에 위치
    - lseek(fd, 1024, SEEK_SET); /* position head above super-block */
  - Magic number
    - if (super.s_magic != EXT2_SUPER_MAGIC) exit(1); /* bad file system */

```
struct ext2_super_block {
        __u32   s_inodes_count;         /* Inodes count */
        __u32   s_blocks_count;         /* Blocks count */
        ...
        __u32   s_free_blocks_count;  /* Free blocks count */
        __u32   s_free_inodes_count; /* Free inodes count */
        __u32   s_first_data_block;     /* First Data Block */
        __u32   s_log_block_size;       /* Block size */
        ...
        __u32   s_blocks_per_group;  /* # Blocks per group */
        ...
        __u16   s_inode_size;           /* inode size */
        __u16   s_magic;                /* Magic signature, EXT2_SUPER_MAGIC */
        ...
```

ext2 Background

Block Group Descriptor

- **Group Descriptors**
  - Struct ext2_group_descr in include/linux/ext2_fs.h

```
struct ext2_group_desc
{
        __u32   bg_block_bitmap;   /* Blocks bitmap block */
        __u32   bg_inode_bitmap;   /* Inodes bitmap block */
        __u32   bg_inode_table;            /* Inodes table block */
        __u16   bg_free_blocks_count;      /* Free blocks count */
        __u16   bg_free_inodes_count;      /* Free inodes count */
        __u16   bg_used_dirs_count;        /* Directories count */
        __u16   bg_pad;
        __u32   bg_reserved[3];
};
```

  - Super block의 s_blocks_count & s_blocks_per_group을 사용하여 그룹 설명자 목록의 크기 계산

```
/* calculate number of block groups on the disk */
unsigned int group_count = 1 + (super.s_blocks_count-1) / super.s_blocks_per_group;

/* calculate size of the group descriptor list in bytes */
unsigned int descr_list_size = group_count * sizeof(struct ext2_group_descr);
```

ext2 Background

- **Group Descriptors**
  - group descriptor block 읽기

```
struct ext2_group_descr group_descr;
...
lseek(fd, 1024 + block_size, SEEK_SET);  /* position head above the group descriptor block */
read(fd, &group_descr, sizeof(group_descr));
```

ext2group.c ext2group.c reads the first (and only) group descriptor of a floppy disk and prints it to screen.

  - bg_block_bitmap, bg_inode_bitmap 및 bg_inode_table 필드를 통해 block/inode bitmap & inode table 위치 파악

```
#define BASE_OFFSET 1024  /* location of the super-block in the first group */
#define BLOCK_OFFSET(block) (BASE_OFFSET + (block-1)*block_size)
```

  - Block은 1부터 번호 매김
    - Block 0 : null block, does not correspond to any disk offset
    - Block 1 : 첫번째 그룹의 수퍼 블록
    - Block 2 : Group Descriptor 블럭

inode

- **모든 파일마다 하나의 inode**
- **해당 inode가 어느 그룹에 속하는지 확인**
  - N번째 inode가 속한 block group = (n-1 / INODES_PER_GROUP)
  - Block pointer (i_block)
    - Disk Data block을 가르키는 포인터 배열
    - 15개 (4byte * 15= 60 byte)
    - inode structue의 크기
      - Super block 의 inode structure size, s_inode_size
      - 128byte, 256byte

요새 사용되는 size

Inode Table의 Inode 구조(128Bytes)

☐ 구조

EXT2 파일시스템의 Inode의 구조

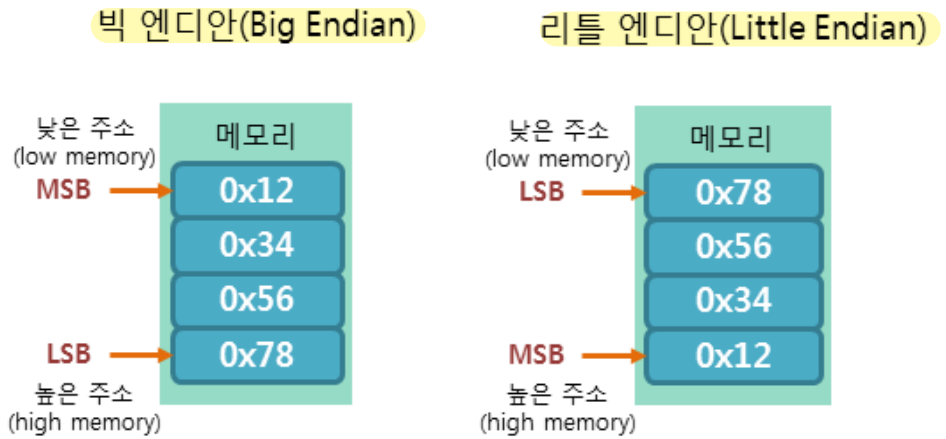
	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0x0000	File Mode		UID		Size				Last Access Time				Created Time			
0x0010	Last Modified Time				Deleted Time				GID		Hard Link Count		Block Count			
0x0020	Flags				OS Description 1				Direct Pointer 0				Direct Pointer 1			
0x0030	Direct Pointer 2				Direct Pointer 3				Direct Pointer 4				Direct Pointer 5			
0x0040	Direct Pointer 6				Direct Pointer 7				Direct Pointer 8				Direct Pointer 9			
0x0050	Direct Pointer 10				Direct Pointer 11				Single Indirect Pointer				Double Indirect Pointer			
0x0060	Triple Indirect Pointer				Generation Number				File ACL				Directory ACL			
0x0070	Block Address of Fragmentation				OS Description 2											

Basic Commands for making ext2

- **fdisk**
  - 디스크 파티션 생성 및 확인
  - % fdisk -l
- **mkfs**
  - 파티션의 파일 시스템 생성
  - % mkfs –t ext2 /dev/sdb1 → 첫번째 파티션
    ↓ 두번째 디스크
- **mount**
  - 파일 시스템의 연결
  - % mount /dev/sdb1 /home/usb %unmount /dev/sdb1
  - automount /media/
- **fsck**

Byte order

- 데이터를 메모리에 저장할 때 바이트(byte) 단위로 나눠서 저장
  - 32bit (4 Byte) / 64bit (8 Byte)
- 바이트 저장 순서(byte order)
  - 바이트가 저장되는 순서
    - big endian
      - 낮은 주소에 데이터의 높은 바이트(MSB, Most Significant Bit)부터 저장
      - SPARC 등 RISC CPU 계열
    - little endian
      - 낮은 주소에 데이터의 낮은 바이트(LSB, Least Significant Bit)부터 저장
      - 인텔 CPU 계열



Superblock Information

```
Filesystem volume name:   ssuos_fs
Last mounted on:          /media/su/ssuos_fs
Filesystem UUID:          5c93ecd7-e708-4e47-a6bd-e1363cf077e2
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      ext_attr resize_inode dir_index filetype sparse_super large_file
Filesystem flags:         signed_directory_hash
Default mount options:    user_xattr acl
Filesystem state:         not clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              243840
Block count:              975264
Reserved block count:     48763
Free blocks:              958038
Free inodes:              243829
First block:              0
Block size:               4096
Fragment size:            4096
Reserved GDT blocks:      238
Blocks per group:         32768
Fragments per group:      32768
Inodes per group:         8128
Inode blocks per group:   508
Filesystem created:       Tue Nov 15 22:57:48 2022
Last mount time:          Tue Nov 15 23:01:39 2022
Last write time:          Tue Nov 15 23:01:39 2022
Mount count:              1
Maximum mount count:      -1
Last checked:             Tue Nov 15 22:57:48 2022
Check interval:           0 (<none>)
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)
First inode:              11
Inode size:               256
Required extra isize:     32
Desired extra isize:      32
Default directory hash:   half_md4
Directory Hash Seed:      876563a7-f575-4216-9eff-aca668a3af5c


Group 0: (Blocks 0-32767)
  Primary superblock at 0, Group descriptors at 1-1
  Reserved GDT blocks at 2-239
  Block bitmap at 240 (+240)
  Inode bitmap at 241 (+241)
  Inode table at 242-749 (+242)
  32010 free blocks, 8115 free inodes, 2 directories
  Free blocks: 756-1026, 1028-1535, 1537-32767
  Free inodes: 12-13, 15-16, 18-8128
```

- ***$fdisk –l***
- ***$dumpe2fs /dev/sdb1***
  - 슈퍼블럭, 그룹 정보 등 표시
  - *dumpfs (BSD)*
- **975,264 blocks * 4,096 bytes per block = 약 3.72 GB (4,096 = 0x1000)**
- **8,128 inodes per group (0x1fc0)**
- **256 bytes per inode (0x100)**
- **Groups form an intermediate structure within an ext2 volume**

```
su@su-virtual-machine:~$ sudo mkfs.ext2 -L ssuos_fs /dev/sdb1
mke2fs 1.45.5 (07-Jan-2020)
/dev/sdb1 contains a ext2 file system
        last mounted on /media/su/0776e6b2-0be3-405b-ab62-1f4785230ad6 on Tue Nov 15 22:42:39 2022
Proceed anyway? (y,N) y
Creating filesystem with 975264 4k blocks and 243840 inodes
Filesystem UUID: 5c93ecd7-e708-4e47-a6bd-e1363cf077e2
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
```

USB내 디렉토리 및 파일 구조

- 현재 디렉토리(루트 디렉토리)에 존재하는 디렉토리 및 파일 구조

```
root@su-virtual-machine:/media/su/ssuos_fs# ls -al
total 60
drwxr-xr-x  4 root root  4096 11월 30 15:44 .
drwxr-x---+ 4 root root  4096 11월 30 15:46 ..
-rw-r--r--  1 root root    26 11월 15 23:02 fs.txt
-rwxr-xr-x  1 root root 16704 11월 30 15:44 helloworld
-rw-r--r--  1 root root    80 11월 30 15:44 helloworld.c
-rw-r--r--  1 root root  3283 11월 15 23:03 linux.txt
drwx------  2 root root 16384 11월 15 22:58 lost+found
drwxr-xr-x  2 root root  4096 11월 18 16:06 ssuos
```

리눅스에서도 참조되지 않는 데이터가 유지하는 디렉토리

- fs.txt 파일 내용

```
root@su-virtual-machine:/media/su/ssuos_fs# cat fs.txt
SSUOS File System : EXT2

root@su-virtual-machine:/media/su/ssuos_fs#
```

```
root@su-virtual-machine:/media/su/ssuos_fs# tree
.
├── fs.txt
├── helloworld
├── helloworld.c
├── linux.txt
├── lost+found
└── ssuos
    └── a.txt

2 directories, 5 files
```

0x400 = 1024

0x100 = 256
200 = 512
400 = 1024
⋮

0x0003b880 = 243840(10)

-le32
little endian
32bit

Detailed Superblock Information

```
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400  80 b8 03 00 a0 e1 0e 00  7b be 00 00 56 9e 0e 00  |........{...V...|
00000410  75 b8 03 00 00 00 00 00  02 00 00 00 02 00 00 00  |u...............|
00000420  00 80 00 00 00 80 00 00  c0 1f 00 00 c3 9b 73 63  |..............sc|
00000430  c3 9b 73 63 01 00 ff ff  53 ef 00 00 01 00 00 00  |..sc....S.......|
00000440  dc 9a 73 63 00 00 00 00  00 00 00 00 01 00 00 00  |..sc............|
00000450  00 00 00 00 0b 00 00 00  00 01 00 00 38 00 00 00  |............8...|
00000460  02 00 00 00 03 00 00 00  5c 93 ec d7 e7 08 4e 47  |........\.....NG|
00000470  a6 bd e1 36 3c f0 77 e2  73 73 75 6f 73 5f 66 73  |...6<.w.ssuos_fs|
00000480  00 00 00 00 00 00 00 00  2f 6d 65 64 69 61 2f 73  |......../media/s|
00000490  75 2f 73 73 75 6f 73 5f  66 73 00 00 00 00 00 00  |u/ssuos_fs......|
000004a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000004c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 ee 00  |................|
000004d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000004e0  00 00 00 00 00 00 00 00  00 00 00 00 87 65 63 a7  |.............ec.|
000004f0  f5 75 42 16 9e ff ac a6  68 a3 af 5c 01 00 00 00  |.uB.....h..\....|
00000500  0c 00 00 00 00 00 00 00  dc 9a 73 63 00 00 00 00  |..........sc....|
00000510  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000550  00 00 00 00 00 00 00 00  00 00 00 00 20 00 20 00  |............ . .|
00000560  01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000570  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001000  f0 00 00 00 f1 00 00 00  f2 00 00 00 0a 7d b3 1f  |.............}..|
```

```
struct ext2_super_block {
    __le32  s_inodes_count;     /* Inodes count */
    __le32  s_blocks_count;     /* Blocks count */
    __le32  s_r_blocks_count;   /* Reserved blocks count */
    __le32  s_free_blocks_count;    /* Free blocks count */
    __le32  s_free_inodes_count;    /* Free inodes count */
    __le32  s_first_data_block; /* First Data Block */
    __le32  s_log_block_size;   /* Block size */
    __le32  s_log_frag_size;    /* Fragment size */
    __le32  s_blocks_per_group; /* # Blocks per group */
    __le32  s_frags_per_group;  /* # Fragments per group */
    __le32  s_inodes_per_group; /* # Inodes per group */
    __le32  s_mtime;        /* Mount time */
    __le32  s_wtime;        /* Write time */
    __le16  s_mnt_count;        /* Mount count */
    __le16  s_max_mnt_count;    /* Maximal mount count */
    __le16  s_magic;        /* Magic signature */
    __le16  s_state;        /* File system state */
    __le16  s_errors;       /* Behaviour when detecting errors */
    __le16  s_minor_rev_level;  /* minor revision level */
    __le32  s_lastcheck;        /* time of last check */
    __le32  s_checkinterval;    /* max. time between checks */
    __le32  s_creator_os;       /* OS */
    __le32  s_rev_level;        /* Revision level */
    __le16  s_def_resuid;       /* Default uid for reserved blocks */
    __le16  s_def_resgid;       /* Default gid for reserved blocks */
    __le32  s_first_ino;        /* First non-reserved inode */
    __le16   s_inode_size;      /* size of inode structure */
    __le16  s_block_group_nr;   /* block group # of this superblock */
    __le32  s_feature_compat;   /* compatible feature set */
    __le32  s_feature_incompat;     /* incompatible feature set */
    __le32  s_feature_ro_compat;    /* readonly-compatible feature set */
    __u8    s_uuid[16];     /* 128-bit uuid for volume */
    char    s_volume_name[16];  /* volume name */
    char    s_last_mounted[64];     /* directory where last mounted */
    __le32  s_algorithm_usage_bitmap; /* For compression */
    __u8    s_journal_uuid[16]; /* uuid of journal superblock */
    __u32   s_journal_inum;     /* inode number of journal file */
    __u32   s_journal_dev;      /* device number of journal file */
    __u32   s_last_orphan;      /* start of list of inodes to delete */
    __u32   s_hash_seed[4];     /* HTREE hash seed */
    __u8    s_def_hash_version; /* Default hash version to use */
    __u8    s_reserved_char_pad;
    __u16   s_reserved_word_pad;
    __le32  s_default_mount_opts;
    __le32  s_first_meta_bg;    /* First metablock block group */
    __u32   s_reserved[190];    /* Padding to the end of the block */
};
```

$ hexdump /dev/sdb1

- file contents를 hexadecimal, decimal, octal, ascii로 표시
- 00 bytes 연속은 “*” 로 생략
- 오른쪽에 아스키값 표시 가능 -C (“canonical”) 옵션(One-byte character display)
- 00000400 = 1024

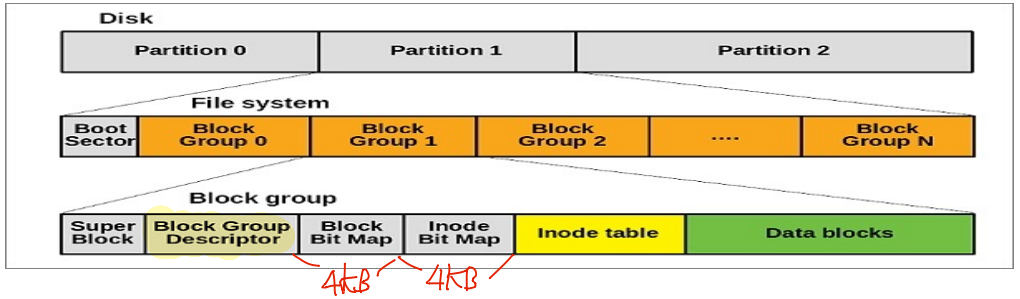
EXT2 파일시스템의 SuperBlock의 구조

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0x0000	Inode count				Block count				Reserved Block count				Free Block count			
0x0010	Free Inode count				First Data Block				Log Block Size				Log Fragmentation Size			
0x0020	Block Per Group				Fragmentation Per Group				Inode Per Group				Modified Time			
0x0030	Write Time				Mount count		Maximum Mount count		Magic Signature		Filesystem State		Errors		Minor Version	
0x0040	Last Consistency Check Time				Check Interval				Creator OS				Major Version			
0x0050	UID		GID		First Inode				Inode Size		Block Group Number		Compatible Feature Flags			
0x0060	Incompatible Feature Flags				Read-Only Feature Flags				UUID(File System ID)							
0x0070	UUID(File System ID)								Volume Name							
0x0080	Volume Name								Last Mounted							
0x0090	Last Mounted															
0x00A0	Last Mounted															
0x00B0	Last Mounted															
0x00C0	Last Mounted								Compression Algorithm				PB	PDB	Padding	
0x00D0	Journal UUID															
0x00E0	Journal Inode Number				Journal Device				Last Orphan				Hash Seed			
0x00F0	Hash Seed												Hash Ver,		Padding	
	Default Mount Option															

Group descriptor, Block bitmap, inode bitmap, inode table

```
00001000  f0 00 00 00 f1 00 00 00  f2 00 00 00 0a 7d b3 1f  |.............}..|
00001010  02 00 04 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00001020  f0 80 00 00 f1 80 00 00  f2 80 00 00 12 7d c0 1f  |.............}..|
00001030  00 00 04 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

```
struct ext2_group_desc
{
    __le32  bg_block_bitmap;        /* Blocks bitmap block */
    __le32  bg_inode_bitmap;        /* Inodes bitmap block */
    __le32  bg_inode_table;         /* Inodes table block */
    __le16  bg_free_blocks_count;   /* Free blocks count */
    __le16  bg_free_inodes_count;   /* Free inodes count */
    __le16  bg_used_dirs_count; /* Directories count */
    __le16  bg_pad;
    __le32  bg_reserved[3];
};
```




```
000f0000  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
000f0050  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff 0f 00  |................|
000f0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000f0080  08 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000f0090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

```
000f1000  ff 27 01 00 00 00 00 00  00 00 00 00 00 00 00 00  |.'..............|
000f1010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000f13f0  00 00 00 00 00 00 00 00  ff ff ff ff ff ff ff ff  |................|
000f1400  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
```

- inode table은 0xf2에서 시작
  - 한개의 block은 4,096 bytes 이기 때문에 해당 inode는 0xf2 * 0x1000 = 0xf2000번지에 있음

0x1000 = 4096 = 4K (한개의 block size)

Inode table

- **inode structure 당 256바이트를 고려하여 한 inode에서 다른 inode로 쉽게 이동**
- **256 = 0x100이므로 F2000, F2100, F2200, F2300 ….**
  - 앞의 몇개 inode는 시스템적으로 이미 예약

```
/*
 * Special inode numbers
 */
#define EXT2_BAD_INO         1  /* Bad blocks inode */
#define EXT2_ROOT_INO        2  /* Root inode */
#define EXT2_BOOT_LOADER_INO     5  /* Boot loader inode */
#define EXT2_UNDEL_DIR_INO   6  /* Undelete directory inode */

/* First non-reserved inode for old ext2 filesystems */
#define EXT2_GOOD_OLD_FIRST_INO 11
```

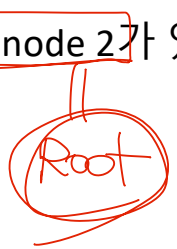
```
000f2000  00 00 00 00 00 00 00 00  17 9b 73 63 17 9b 73 63  |..........sc..sc|
000f2010  17 9b 73 63 00 00 00 00  00 00 00 00 00 00 00 00  |..sc............|
000f2020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000f2100  ed 41 00 00 00 10 00 00  ca 9c 73 63 ae 9c 73 63  |.A........sc..sc|
000f2110  ae 9c 73 63 00 00 00 00  00 00 04 00 08 00 00 00  |..sc............|
000f2120  00 00 00 00 11 00 00 00  ee 02 00 00 00 00 00 00  |................|
000f2130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000f2180  20 00 00 00 38 92 85 db  38 92 85 db cc c0 b9 c2  | ...8...8.......|
000f2190  17 9b 73 63 00 00 00 00  00 00 00 00 00 00 00 00  |..sc............|
000f21a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000f2600  80 81 00 00 00 c0 40 00  17 9b 73 63 17 9b 73 63  |......@...sc..sc|
000f2610  17 9b 73 63 00 00 00 00  00 00 01 00 88 3b 00 00  |..sc.........;..|
000f2620  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000f2650  00 00 00 00 00 00 00 00  00 00 00 00 f3 02 00 00  |................|
000f2660  00 00 00 00 00 00 00 00  00 00 00 00 01 00 00 00  |................|
000f2670  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000f2680  20 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  | ...............|
000f2690  17 9b 73 63 00 00 00 00  00 00 00 00 00 00 00 00  |..sc............|
000f26a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

```
000f2a00  c0 41 00 00 00 40 00 00  17 9b 73 63 17 9b 73 63  |.A...@....sc..sc|
000f2a10  17 9b 73 63 00 00 00 00  00 00 02 00 20 00 00 00  |..sc........ ...|
000f2a20  00 00 00 00 00 00 00 00  ef 02 00 00 f0 02 00 00  |................|
000f2a30  f1 02 00 00 f2 02 00 00  00 00 00 00 00 00 00 00  |................|
000f2a40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000f2a80  20 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  | ...............|
000f2a90  17 9b 73 63 00 00 00 00  00 00 00 00 00 00 00 00  |..sc............|
000f2aa0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

계속해서 볼륨을 읽기 위해 루트 디렉토리의 inode인 inode 2를 확인

inode 1이 F2000에 있으므로 F2100에 inode 2가 있음

↳ 만약 inode 1이 0xF2000이 아니라면

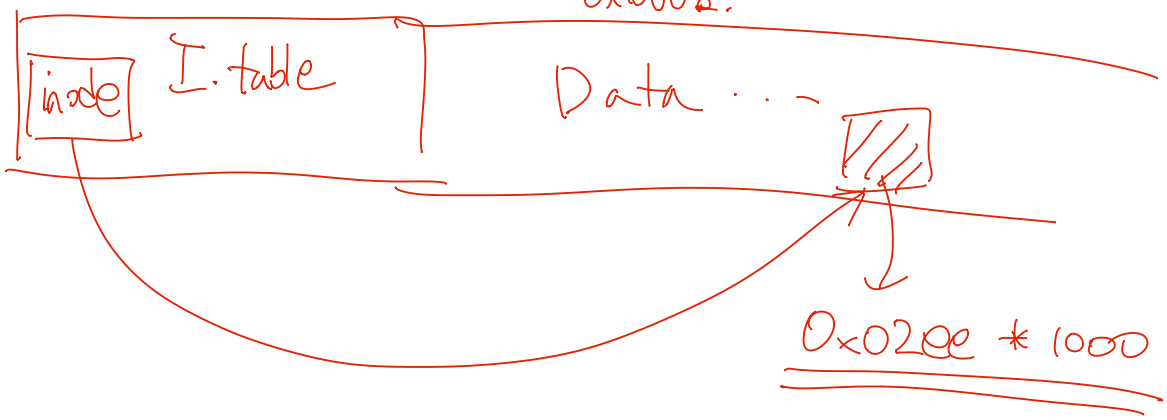


Inode structure

- 루트 디렉토리의 i_mode는 0x41ed (0100 0001 1110 1101), directory file, S_IFDIR, rwxr_xr_x

```
000f2100  ed 41 00 00 00 10 00 00  ca 9c 73 63 ae 9c 73 63  |.A........sc..sc|
000f2110  ae 9c 73 63 00 00 00 00  00 00 04 00 08 00 00 00  |..sc............|
000f2120  00 00 00 00 11 00 00 00  ee 02 00 00 00 00 00 00  |................|
000f2130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

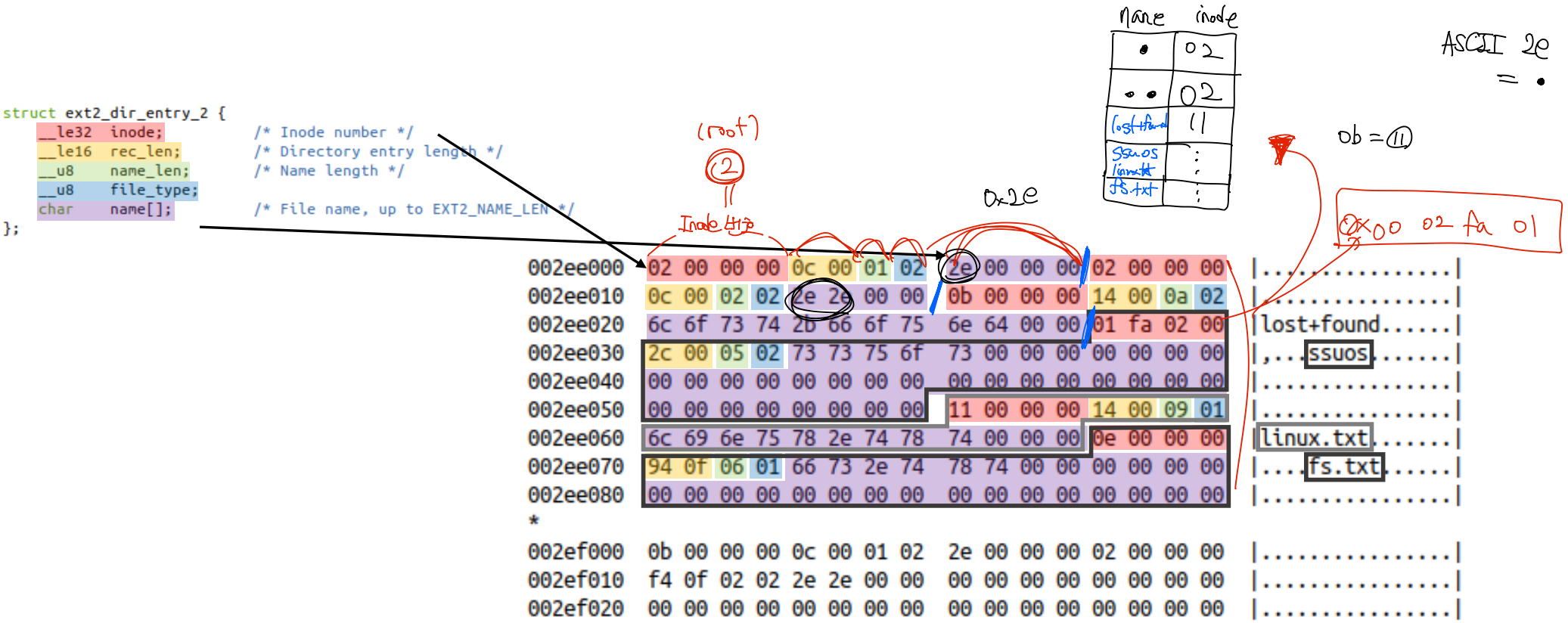
- 첫번째 Data block : 0x02ee * 1000




```
struct ext2_inode {
    __le16  i_mode;     /* File mode */
    __le16  i_uid;      /* Low 16 bits of Owner Uid */
    __le32  i_size;     /* Size in bytes */
    __le32  i_atime;    /* Access time */
    __le32  i_ctime;    /* Creation time */
    __le32  i_mtime;    /* Modification time */
    __le32  i_dtime;    /* Deletion Time */
    __le16  i_gid;      /* Low 16 bits of Group Id */
    __le16  i_links_count;  /* Links count */
    __le32  i_blocks;   /* Blocks count */
    __le32  i_flags;    /* File flags */
    union {
        struct {
            __le32  l_i_reserved1;
        } linux1;
        struct {
            __le32  h_i_translator;
        } hurd1;
        struct {
            __le32  m_i_reserved1;
        } masix1;
    } osd1;             /* OS dependent 1 */
    __le32  i_block[EXT2_N_BLOCKS];/* Pointers to blocks */
    __le32  i_generation;   /* File version (for NFS) */
    __le32  i_file_acl; /* File ACL */
    __le32  i_dir_acl;  /* Directory ACL */
    __le32  i_faddr;    /* Fragment address */
    union {
        struct {
            __u8    l_i_frag;   /* Fragment number */
            __u8    l_i_fsize;  /* Fragment size */
            __u16   i_pad1;
            __le16  l_i_uid_high;   /* these 2 fields    */
            __le16  l_i_gid_high;   /* were reserved2[0] */
            __u32   l_i_reserved2;
        } linux2;
        struct {
            __u8    h_i_frag;   /* Fragment number */
            __u8    h_i_fsize;  /* Fragment size */
            __le16  h_i_mode_high;
            __le16  h_i_uid_high;
            __le16  h_i_gid_high;
            __le32  h_i_author;
        } hurd2;
        struct {
            __u8    m_i_frag;   /* Fragment number */
            __u8    m_i_fsize;  /* Fragment size */
            __u16   m_pad1;
            __u32   m_i_reserved2[2];
        } masix2;
    } osd2;             /* OS dependent 2 */
};
```

Directory

- **디렉토리의 데이터 블록은 디렉토리 항목의 배열**
  - 데이터 블록 0x02ee (또는 오프셋 0x002ee000에 있는 루트 디렉토리용)
  - 현재 디렉토리(".")와 상위 디렉토리("..")도 명시적 디렉토리 항목으로 저장



fs.txt와 linux.txt의 inode structure

```
root@su-virtual-machine:/media/su/ssuos_fs# ls -i
    14 fs.txt              17 linux.txt
```

- **fs.txt**
  - 파일 크기 : 26 bytes
  - Data block의 시작점 : 0x0600 * 1000

```
000f2d00  a4 81 00 00 1a 00 00 00  e1 ea 86 63 ae 9c 73 63  |...........c..sc|
000f2d10  05 9c 73 63 00 00 00 00  00 00 01 00 08 00 00 00  |..sc............|
000f2d20  00 00 00 00 01 00 00 00  00 06 00 00 00 00 00 00  |................|
000f2d30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|

00600000  53 53 55 4f 53 20 46 69  6c 65 20 53 79 73 74 65  |SSUOS File Syste|
00600010  6d 20 3a 20 45 58 54 32  0a 0a 00 00 00 00 00 00  |m : EXT2........|
```

- **linux.txt**
  - 파일 크기 : 3,283 bytes
  - Data block의 시작점 : 0x0403 * 1000

```
000f3000  a4 81 00 00 d3 0c 00 00  a6 fd 86 63 e6 2e 77 63  |...........c..wc|
000f3010  29 9c 73 63 00 00 00 00  00 00 01 00 08 00 00 00  |).sc............|
000f3020  00 00 00 00 01 00 00 00  03 04 00 00 00 00 00 00  |................|
000f3030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|

00403000  4c 69 6e 75 78 20 28 2f  cb 88 6c 69 cb 90 6e ca  |Linux (/..li..n.|
00403010  8a 6b 73 2f 20 28 6c 69  73 74 65 6e 29 20 4c 45  |.ks/ (listen) LE|
00403020  45 2d 6e 75 75 6b 73 20  6f 72 20 2f cb 88 6c c9  |E-nuuks or /..l.|
00403030  aa 6e ca 8a 6b 73 2f 20  4c 49 4e 2d 75 75 6b 73  |.n..ks/ LIN-uuks|
    ...
00403cc0  74 20 70 72 6f 67 72 61  6d 2e 5b 34 30 5d 5b 32  |t program.[40][2|
00403cd0  31 5d 0a 00 00 00 00 00  00 00 00 00 00 00 00 00  |1]..............|
00403ce0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

Files

- **실제 파일 내용 확인**
  - 텍스트 파일은 찾기 쉽고 추가 디렉토리는 루트 디렉토리와 동일한 방식으로 찾으면 됨
  - *fs.txt*
    - 오프셋 0xf2d00으로 변환되는 inode e에 있으며, 여기서 파일의 첫 번째 데이터 블록은 전체 데이터 블록 0x0600있음
    - Inode #'s offset = F2000 + (100 * (# - 1))

```
000f2d00  a4 81 00 00 1a 00 00 00  05 9c 73 63 ae 9c 73 63  |..........sc..sc|
000f2d10  05 9c 73 63 00 00 00 00  00 00 01 00 08 00 00 00  |..sc............|
000f2d20  00 00 00 00 01 00 00 00  00 06 00 00 00 00 00 00  |................|
000f2d30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

    - 실제 데이터는 오프셋 0x1000 * 0x0600 = 0x600000에서 찾을 수 있음

```
00600000  53 53 55 4f 53 20 46 69  6c 65 20 53 79 73 74 65  |SSUOS File Syste|
00600010  6d 20 3a 20 45 58 54 32  0a 0a 00 00 00 00 00 00  |m : EXT2........|
00600020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

하나의 Block size가 0x1000 <4KB이므로 끝났다.

Files

- **실제 파일 내용 확인**
  - 텍스트 파일은 찾기 쉽고 추가 디렉토리는 루트 디렉토리와 동일한 방식으로 찾으면 됨
  - *linux.txt*
    - *오프셋 0xf300으로 변환되는 Inode 11에 있으며, 첫번째 데이타 블럭은 0x0403*
    - Inode #'s offset = F2000 + (100 * (# - 1))

F2000 + (100x10) = F3000

```
000f3000  a4 81 00 00 d3 0c 00 00  29 9c 73 63 95 9c 73 63  |........).sc..sc|
000f3010  29 9c 73 63 00 00 00 00  00 00 02 00 08 00 00 00  |).sc............|
000f3020  00 00 00 00 01 00 00 00  03 04 00 00 00 00 00 00  |................|
000f3030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

(0x0403) x (1000)

- 실제 데이터는 오프셋 0x1000 * 0x0403 = 0x403000에서 찾을 수 있음

```
00403000  4c 69 6e 75 78 20 28 2f  cb 88 6c 69 cb 90 6e ca  |Linux (/..li..n.|
00403010  8a 6b 73 2f 20 28 6c 69  73 74 65 6e 29 20 4c 45  |.ks/ (listen) LE|
00403020  45 2d 6e 75 75 6b 73 20  6f 72 20 2f cb 88 6c c9  |E-nuuks or /..l.|
00403030  aa 6e ca 8a 6b 73 2f 20  4c 49 4e 2d 75 75 6b 73  |.n..ks/ LIN-uuks|
00403040  29 5b 31 31 5d 20 69 73  20 61 6e 20 6f 70 65 6e  |)[11] is an open|
00403050  2d 73 6f 75 72 63 65 20  55 6e 69 78 2d 6c 69 6b  |-source Unix-lik|
                                   ...
00403cc0  74 20 70 72 6f 67 72 61  6d 2e 5b 34 30 5d 5b 32  |t program.[40][2|
00403cd0  31 5d 0a 00 00 00 00 00  00 00 00 00 00 00 00 00  |1]..............|
00403ce0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

a.out (ELF) format

Big Endian, Little endian 상관 X

- /usr/include/elf.h
- Elf magic number : 7f 45 4c 46

```
*
00802000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00802010  03 00 3e 00 01 00 00 00  60 10 00 00 00 00 00 00  |..>.....`.......|
00802020  40 00 00 00 00 00 00 00  80 39 00 00 00 00 00 00  |@........9......|
00802030  00 00 00 00 40 00 38 00  0d 00 40 00 1f 00 1e 00  |....@.8...@.....|
00802040  06 00 00 00 04 00 00 00  40 00 00 00 00 00 00 00  |........@.......|
00802050  40 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |@.......@.......|
00802060  d8 02 00 00 00 00 00 00  d8 02 00 00 00 00 00 00  |................|
00802070  08 00 00 00 00 00 00 00  03 00 00 00 04 00 00 00  |................|
00802080  18 03 00 00 00 00 00 00  18 03 00 00 00 00 00 00  |................|
00802090  18 03 00 00 00 00 00 00  1c 00 00 00 00 00 00 00  |................|
008020a0  1c 00 00 00 00 00 00 00  01 00 00 00 00 00 00 00  |................|
008020b0  01 00 00 00 04 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Magic number

- Format indicator
  - Version 7 Unix source code부터 사용
  - 많은 운영 체제의 프로그램에서 일반적임
  - 강력한 형식의 데이터를 구현하며 프로그램 실행 시간에 데이터 형식을 읽는 제어 프로그램에 대한 인밴드 신호 형식으로 바이너리 데이터를 식별하는 상수
  - Compiled Java class files (bytecode) / Mach-O binaries 0xCAFEBABE
  - GIF GIF89a 0x 47 49 46 38 39 61
  - PDF %PDF 0x 25 50 44 46
  - Boot sector 0xAA55

ASCII

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL