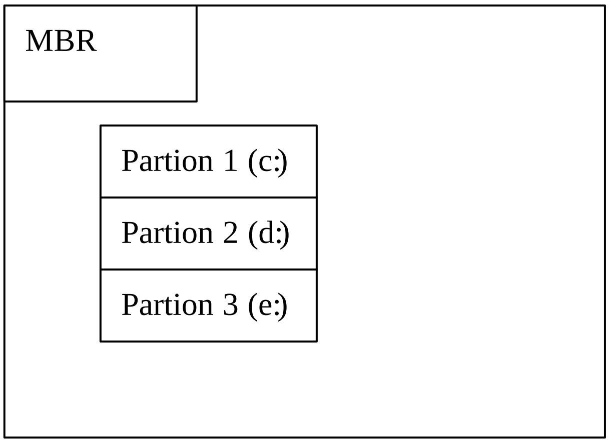
FAT File system (By kenny Hsieh 2005/1/24 version 1.0)

(1) HDD 的 Partition

於 HDD 我們通常會分為幾個磁碟(Logical driver), 於 windows 下我們可看到 C:, D:, E:…碟就是 partition 的結果. 如何去記錄這些磁碟機的所在位置, 就有所謂的 Partition Table. 注意 partion 與 partion Table 的不同



(2) MBR (Master Boot Record)

MBR 位置就固定在硬碟的第一個 Sector(Cylinder 0, Head 0 , Sector 1) . 於 MBR 內有 4 個 16 bytes 的 Partition Table, 那是不是就只能有 4 個 logical driver 呢? 當然不是, 因為還有 extend Partition 可做為延伸, 而 extend partition 的結構就類似像是 MBR , 其中只有 32 bytes 可使用.

MBR
Boot code area 446 bytes
Partition Table(1) 16 bytes
Partition Table(2) 16 bytes
Partition Table(3) 16 bytes
Partition Table(4) 16 bytes
End code 2 bytes(0x55AA)

extend Partition
No used 446 bytes
Partition Table(1) 16 bytes
Partition Table(2) 16 bytes
No used 16 bytes
No used 16 bytes
End code 2 bytes(0x55AA)

(3)Partition Ta ble

表(一)

Offset	Bytes	Name	註解
0x00 (0)	1	Boot /non boot type	00h 為 non boot, 08h 為 boot
0x01 (1)	1	Start header	
0x02 (2)	2	Start sector and cylinder	
0x04 (4)	1	Partition Type	見下表(二)
0x05 (5)	1	End header	
0x06 (6)	2	End sector and cylinder	
0x08 (8)	4	Partion relative position (sectors)	紀錄從 MBR 到此 partition 之間的 sector 數
0x0C (12)	4		

(4) Partition Type

Partition Type 是顯示此 partition 是屬於何種作業系統，或是以何種方式規劃。其中雖然有 FAT12,FAT16,FAT32 的紀錄，但是依據 microsoft white paper 來說，並不能以此為判定是否是 FAT12,FAT16 或是 FAT32，而是需要以 partition 內的 cluster 數目來作為判定，FAT12 (最大 4096 cluster), FAT16(最大 65536 cluster), FAT32 (最大 268,435,456 cluster). (後面會以 code 說明)

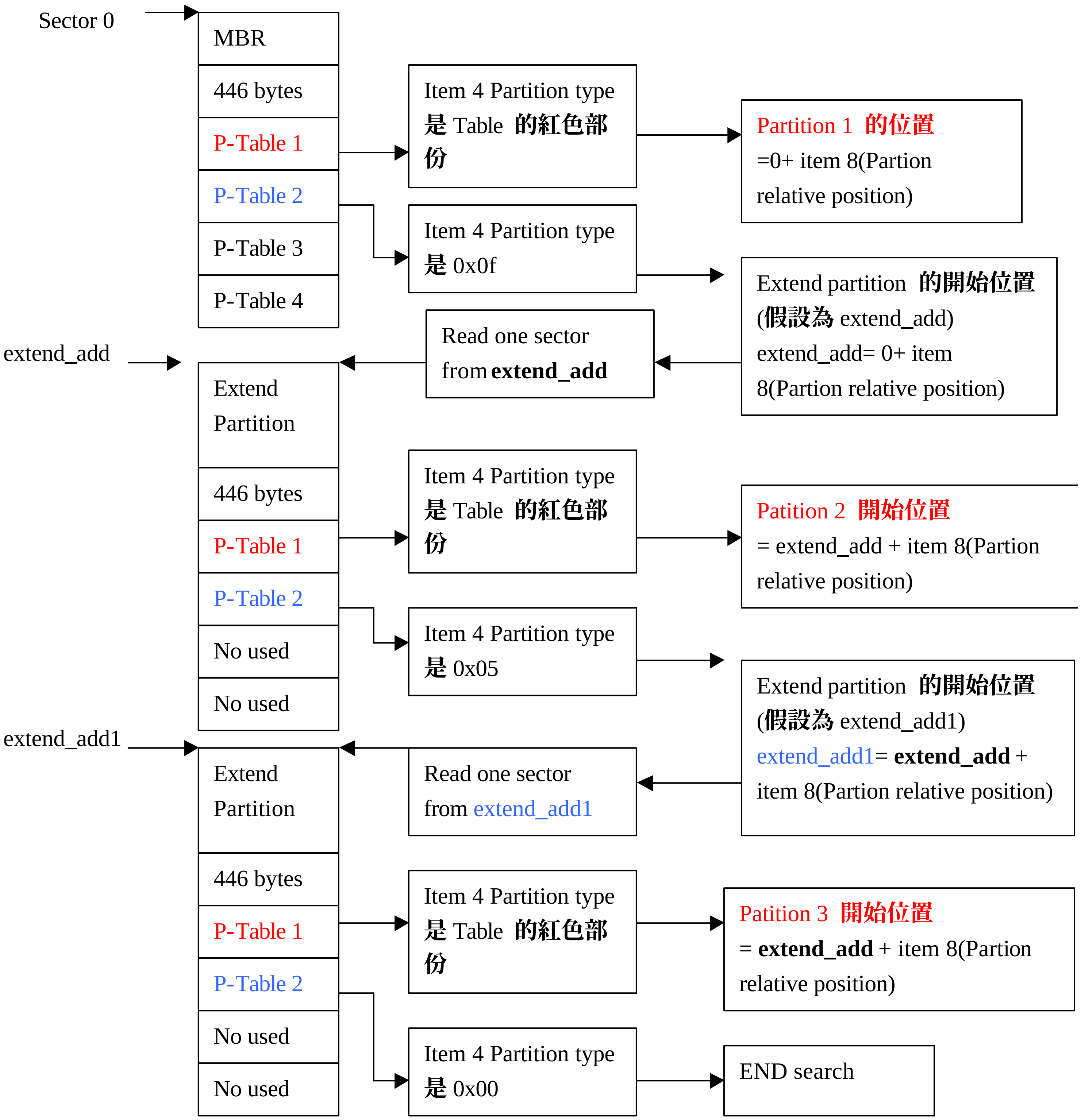
Table (二)

Value	說明
0x00	未知系統或未使用
0x01	FAT12
0x04	FAT16(partition 小於 32 MB 時)
0x05	MS-DOS extend partition
0x06	FAT16(partition 大於 32 MB 時)
0x07	NTFS file system
0x0B	FAT32(最大容量可達 2048 GB)
0x0C	使用 LBA 模式(Int 13h Extension) 的 0Bh (FAT 32)
0x0D	使用 LBA 模式(Int 13h Extension) 的 06h (FAT 16)
0x0E	使用 LBA 模式(Int 13h Extension) 的 04h (FAT 16)
0x0F	使用 LBA 模式(Int 13h Extension) 的 05h (extend partition)

0x82	Linux swap
0x83	Linux native
0x85	Linux Extened

(5)FAT system Partition 的 search 方式

依據表(**一**) **中** item 0x04, 0x08 **及 表**(**二**) **中紅色及藍色** Item **我們可** search **到有幾個** partition **及 其** type , **相對位置**. **注意** Partition 2 **後的相對位置是對於** extend_add, **如** Partion3 **相對位置是** extend_add **而不是** extend_add1. Partition 4,5,…**也是如此**


Sector 0
MBR
446 bytes
P-Table 1
P-Table 2
P-Table 3
P-Table 4
Item 4 Partition type 是 Table 的紅色部份
Partition 1 的位置 =0+ item 8(Partion relative position)
Item 4 Partition type 是 0x0f
Extend partition 的開始位置 (假設為 extend_add) extend_add= 0+ item 8(Partion relative position)
Read one sector from extend_add
extend_add
Extend Partition
446 bytes
P-Table 1
P-Table 2
No used
No used
Item 4 Partition type 是 Table 的紅色部份
Patition 2 開始位置 = extend_add + item 8(Partion relative position)
Item 4 Partition type 是 0x05
Extend partition 的開始位置 (假設為 extend_add1) extend_add1= extend_add + item 8(Partion relative position)
Read one sector from extend_add1
extend_add1
Extend Partition
446 bytes
P-Table 1
P-Table 2
No used
No used
Item 4 Partition type 是 Table 的紅色部份
Patition 3 開始位置 = extend_add + item 8(Partion relative position)
Item 4 Partition type 是 0x00
END search

(6)Partition search code

於 int FSReadMBR(void) 的 function, 擷取 search 的 code

```
 int FSReadMBR(void)
{
   int  i = 0 , j = 0 ;
  for ( i = 0 ; i < 4 ; i++ )   //get the main partition info, MBR
   {
        cardP.typeFAT  =  *(pwb+446+4+i*16);
      if ( (cardP.typeFAT == 0x04 ) || (cardP.typeFAT == 0x06 ) || (cardP.typeFAT == 0x0b ) ||
         (cardP.typeFAT == 0x0c ) || (cardP.typeFAT == 0x0d ) || (cardP.typeFAT == 0x0e ) )
         {
                 ide[j].typeFAT  =  cardP.typeFAT;
                 ide[j].sAddBootSec  =  getBiUINT32(pwb+446+8+i*16);
                  j++;
          }
         else  if(cardP.typeFAT  ==  0x0f)//extend partition
        extend_add=cardP.sAddBootSec=getBiUINT32(pwb  +  446  +  8  +  i*16  );
    }

       //judge whether have next exterded partition
  if(extend_add)
  {
       FSReadSector ( extend_add , 1 , pwb ); //kenny 2005/1/19
       while(1)
       {
        for(i=0;i<2;i++)
       {
            cardP.typeFAT=*(pwb + 446 + 4 +i*16);
           if ( (cardP.typeFAT == 0x04 ) ||  (cardP.typeFAT == 0x06 ) || (cardP.typeFAT ==
           0x0b ) || (cardP.typeFAT == 0x0c ) ||(cardP.typeFAT == 0x0d ) ||(cardP.typeFAT ==
              0x0e ))
           {   // record each Partion Address
            ide[j].typeFAT = *(pwb + 446 + 4 );
            ide[j].sAddBootSec  =  cardP.sAddBootSec+getBiUINT32(pwb  +  446  +  8  );
           j++;
           }
         else if(cardP.typeFAT == 0x05)// next extend partition
           {
```

```
          //extend partition address
           cardP.sAddBootSec =extend_add+ getBiUINT32(pwb + 446 + 8 16 );
          extend_partition_exit=1;
        }

    }//for( i=0, i< 2, i++)

       if(extend_partition_exit)
       {
       FSReadSector(cardP.sAddBootSec , 1 , pwb );// Read next extend partition
       extend_partition_exit=0;
       }
       else
       break;

    }//while (1)
 }//if(extend_add)
     max_part  =  j    ; // record max partition

}// int FSReadMBR(void)
```

(7) FAT File system

FAT(File Allocation Table) 稱為”檔案配置表”, FAT file sytem 的最大特色是以 cluster 作為存放檔案的配置單元(Allocation unit), 常見的 FAT 有 FAT12 , FAT16, 及 FAT32. 下表是 FAT 的比較

FAT name	Cluster 數	Max Size 限制	說明
FAT12	2^16 =4,096	(4096)(cluster) * (8 sector/cluster)* 512 bytes =16 MB	
FAT16	2^16 =65,536	(65536)(cluster) * 32 KB (bytes/cluster) =2GB	
		(65536)(cluster) * 64 KB (bytes/cluster) =4 GB	Window xP/2000/NT
FAT 32	2^28(4 bit 保留) =268,435,456	(268435456)(cluster) * 32 KB (bytes/cluster) =8 TB(1 TB=1024 GB)	FAT32 可支援長檔名

(8) partition 的 FAT file sytem format

每個 Partion 大致由 BOOT Sector + FAT region +ROOT directory region+ File and directory DATA region 所組成. 如下圖 1,2. 前面已提過如何 search partition 的位置, 這位置就是 Boot sector 的 start Address 我們假設為 sAddBootSec, 由 Boot sector 內的一些 information, 我們將可 search 到 FAT 的起始位置 sAddFAT 及 目錄區的起始位置 sAddFDB 及 File 的 Data 起始位置. Data 區的起始位置是從 cluster number 2 開始, 這會是當作 cluster 轉換到 LBA 的一個依據, 稍後介紹完 Boot sector 後會以現有的程式作一說明.

圖 1. FAT12/FAT16

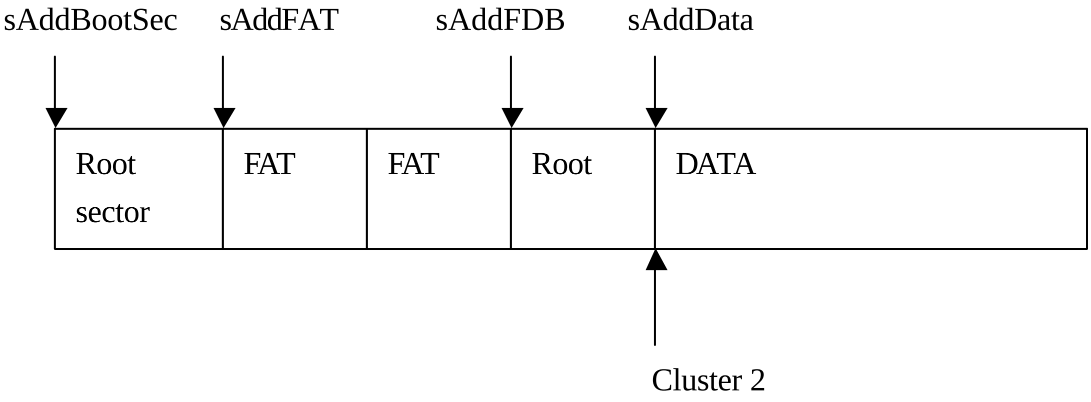
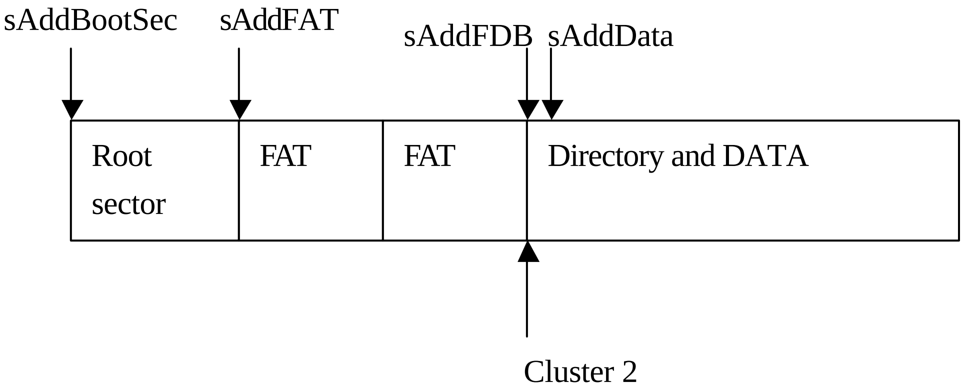


圖 2. FAT32



(9) Boot sector

在 Boot sector 中，FAT12/FAT16 及 FAT32 的前 36 個 bytes 是相同的，後面 36 ~512 bytes 是不同的，將目前 8202 程式中用到的部分用紅色標示

Boot Sector and BPB Structure(FAT12/FAT16/FAT32)

Name	Offset (byte)	Size (bytes)	Description
BS_jmpBoot	0	3	跳到啟動程式開始位址
BS_OEMName	3	8	Oem 名稱和版本
BPB_BytsPerSec	11	2	每個 sector 有多少 bytes
BPB_SecPerClus	13	1	每個 cluster 有多少 sector, This value must be a power of 2 that is greater than 0. The legal values are 1, 2, 4, 8, 16, 32, 64, and 128.
BPB_RsvdSecCnt	14	2	Boot sector 佔用的 sector 數
BPB_NumFATs	16	1	FAT 有幾份. This field should always contain the value 2 for any FAT volume of any type.
BPB_RootEntCnt	17	2	Root 區有多少 FDB. For FAT12 and FAT16 volumes, this field contains the count of 32-byte directory entries in the root directory. For FAT32 volumes, this field must be set to 0. For FAT12 and FAT16 volumes, this value should always specify a count that when multiplied by 32 results in an even multiple of BPB_BytsPerSec. For maximum compatibility, FAT16 volumes should use the value 512.
BPB_TotSec16	19	2	Partition 的 sector 總數目(第一欄). This field is the old 16-bit total count of sectors on the volume. This count includes the count of all sectors in all four regions of the volume. This field can be 0; if it is 0, then BPB_TotSec32 must be non-zero. For FAT32 volumes, this field must be 0. For FAT12 and FAT16 volumes, this field contains the sector count, and BPB_TotSec32 is 0 if the total sector count "fits" (is less than 0x10000).
BPB_Media	21	1	0xF8 is the standard value for "fixed" (non-removable) media. For removable media, 0xF0 is frequently used.
BPB_FATSz16	22	2	FAT12/FAT16 中每個 FAT 使用的 sector 數目. This field is the FAT12/FAT16 16-bit count of sectors occupied by ONE FAT. On FAT32 volumes this field must be 0, and BPB_FATSz32 contains the FAT size count.
BPB_SecPerTrk	24	2	每 track 有幾個 sector
BPB_NumHeads	26	2	Number of heads for interrupt 0x13.
BPB_HiddSec	28	4	隱藏的 sector 數目. (MBR)

BPB_TotSec32	32	4	Partition 的 sector 總數目(第二欄) .This field is the new 32-bit total count of sectors on the volume. This count includes the count of all sectors in all four regions of the volume. This field can be 0; if it is 0, then BPB_TotSec16 must be non-zero. For FAT32 volumes, this field must be non-zero. For FAT12/FAT16 volumes, this field contains the sector count if BPB_TotSec16 is 0 (count is greater than or equal to 0x10000).

Fat12 and Fat16 Structure Starting at Offset 36

Name	**Offset (byte)**	**Size (bytes)**	**Description**
BS_DrvNum	36	1	磁碟編號(0x00 for floppy disks, 0x80 for hard disks).
BS_Reserved1	37	1	Reserved (used by Windows NT).
BS_BootSig	38	1	Extended boot signature (0x29). This is a signature byte that indicates that the following three fields in the boot sector are present.
BS_VolID	39	4	磁碟機序號(Volume serial Number)
BS_VolLab	43	11	磁碟機標籤(Volume label)
BS_FilSysType	54	8	One of the strings “**FAT12** ”, “**FAT16** ”, or “**FAT** ”. **NOTE:** Many people think that the string in this field has something to do with the determination of what type of FAT—FAT12, FAT16, or FAT32—that the volume has. This is not true. You will note from its name that this field is not actually part of the BPB. This string is informational only and is not used by Microsoft file system drivers to determine FAT typ,e because it is frequently not set correctly or is not present. See the FAT Type Determination section of this document. This string should be set based on the FAT type though, because some non-Microsoft FAT file system drivers do look at it.
Bootstrap code	62	448	Bootstrap 啟動程式
Ending code	510	2	Boot sector 結尾(0x55AA)

FAT32 Structure Starting at Offset 36

Name	Offset (byte)	Size (bytes)	Description
BPB_FATSz32	36	4	FAT32 中每個 FAT 使用的 sector 數目. This field is only defined for FAT32 media and does not exist on FAT12 and FAT16 media.
BPB_ExtFlags	40	2	延伸旗標(only for FAT 32)
BPB_FSVer	42	2	檔案系統版本(only for FAT 32)
BPB_RootClus	44	4	存放根目錄的第一個 cluster number. (only for FAT32). This is set to the cluster number of the first cluster of the root directory, usually 2 but not required to be 2.
BPB_FSInfo	48	2	存放檔案系統資訊的磁區編號.(only for FAT32) **NOTE:** There will be a copy of the FSINFO structure in BackupBoot, but only the copy pointed to by this field will be kept up to date (i.e., both the primary and backup boot record will point to the same FSINFO sector).
BPB_BkBootSec	50	2	備份啟動磁區的編號 Usually 6. No value other than 6 is recommended.
BPB_Reserved	52	12	保留
BS_DrvNum	64	1	磁碟編號(only for FAT 32, 0x80 for hard disks)
BS_Reserved1	65	1	保留
BS_BootSig	66	1	Extended boot signature (0x29), only for FAT 32
BS_VolID	67	4	磁碟機序號(Volume serial Number) , only for FAT 32
BS_VolLab	71	11	磁碟機標籤(Volume label) , only for FAT 32
BS_FilSysType	82	8	Always set to the string ”**FAT32 **”. Please see the note for this field in the FAT12/FAT16 section earlier. This field has nothing to do with FAT type determination.

(10)FAT , ROOT(FDB), 及 DATA 區 的 address

從上面的 Boot sector 表我們就可算出每個區域的 LBA. 在 8202 code 中的 FSReadBoot() function 就是在完成這些動作.

a. 圖 1 FAT12/FAT16 的各區域位置如下(其中 FDB 的數目是固定的, FDBSIZE=32 bytes)

sAddFAT = sAddBootSec + BPB_RsvdSecCnt

sAddFDB = sAddFAT + BPB_FATSz16* BPB_NumFATs

RootBlockSize = (BPB_RootEntCnt *FDBSIZE + BPB_BytsPerSec - 1)/ BPB_BytsPerSec

sAddData = sAddFDB + RootBlockSize

b. 圖 2 FAT32 的各區域位置如下(其中 FDB 的數目是不固定的, FDBSIZE=32 bytes)

sAddFAT = sAddBootSec + BPB_RsvdSecCnt

sAddData = sAddFAT + BPB_FATSz32* BPB_NumFATs;

sAddFDB= sAddData + (BPB_RootClus - 2) * BPB_SecPerClus

c. 由圖 1, 2 可知 data 是從cluster number 2 開始, 因此我們若是給一個 file DATA cluster number N 我們就可以下列公式找到真正的 LBA

FirstSectorofCluster= sAddData + (N - 2) * BPB_SecPerClus

以上的公式就在 8202 code 裡的 FSClus2LBA() 中

```
 UINT32 FSClus2LBA(UINT32 clus)
{
 UINT32    lba;
 lba = cardP.sAddData + (clus - 2) * cardP.secPerClus;
   return lba;
   }
```

(11) FAT12/FAT16 **及** FAT32 **實際的辨別方式**

實際的辨別方式應當以看 data **區的** cluster **數目來決定**

```
  if(BPB_FATSz16 != 0)
  FATSz = BPB_FATSz16;
  else
 FATSz  =  BPB_FATSz32;

  if(BPB_TotSec16 != 0)
  totSec = BPB_TotSec16;
  else
  totSec = BPB_TotSec32;

  dataSec  =  totSec  - (BPB_RsvdSecCnt + (FATSz * BPB_NumFATs)   +
  RootBlockSize);
dataSec  =  dataSec/ BPB_SecPerClus;
 if (dataSec<4085)
      /*Partition is FAT12;*/
else  if  (dataSec<65525)
      /*Partition is FAT16;*/
 else
      /*Partition is FAT32;*/
```

(12) FSReadBoot() function

FSReadBoot function 就是完成以上 partition 各個區域的起始位置

```
int FSReadBoot()
{
     UINT16      nReserved;
     BYTE         nFAT;
    if ( FSReadSector(cardP.sAddBootSec, 1, pwb) ) return 1;  //read BOOT sector
    cardP.bytePerSec = getBiUINT16(pwb+11);// BPB_BytsPerSec
     cardP.secPerClus = pwb[13];
     nFAT = *(pwb+16);// BPB_NumFATs
     cardP.nFDBinRoot = getBiUINT16(pwb+17);// BPB_RootEntCnt
     nReserved = getBiUINT16(pwb+14);// BPB_RsvdSecCnt
    cardP.sAddFAT = cardP.sAddBootSec + nReserved; //reserved sectors
     if (cardP.typeFAT<=CARD_FAT16) { //only for FAT12, FAT16
           UINT16   RootBlockSize;
           UINT32   totSec32;
           int        dataSec;
          cardP.secPerFAT = getBiUINT16(pwb+22)&0x0000ffff;// BPB_FATSz16
         RootBlockSize = (cardP.nFDBinRoot*FDBSIZE + cardP.bytePerSec - 1)/cardP.bytePerSec;
          cardP.sAddFDB = cardP.sAddFAT + cardP.secPerFAT*2;
          cardP.sAddData = cardP.sAddFDB + RootBlockSize;
           totSec32 = getBiUINT16(pwb+19);// BPB_TotSec16
          if (totSec32==0) totSec32=getBiUINT32(pwb+32);// BPB_TotSec32
          dataSec = totSec32 - (nReserved + (nFAT*cardP.secPerFAT)  + RootBlockSize);
          dataSec = dataSec/cardP.secPerClus;
           //correct FAT determination,
          if (dataSec<4085) {
                cardP.typeFAT = CARD_FAT12;
                 cardP.nItemSizeFAT = 3; //1.5 Bytes for each item in FAT
          } else if (dataSec<65525) {
                cardP.typeFAT = CARD_FAT16;
             cardP.nItemSizeFAT = 2*2; //2 Bytes for each item in FAT
          } else {
                cardP.typeFAT = CARD_FAT32;
               cardP.nItemSizeFAT = 2*4; //4 Bytes for each item in FAT
           }

     }
    if (cardP.typeFAT > CARD_FAT16)
     {
          BYTE       nFAT;
          UINT32   iFATSize,  iRoot;
         cardP.secPerFAT  =  getBiUINT32(pwb+36);// BPB_FATSz32
         nFAT  =  *(pwb+16);// BPB_NumFATs
         iFATSize  =  getBiUINT32(pwb+36);// BPB_FATSz32
         iRoot  =  getBiUINT32(pwb+44);// BPB_RootClus
        cardP.sAddData = cardP.sAddFAT + iFATSize * nFAT;
         cardP.sAddFDB  =  FSClus2LBA(iRoot);
     }
     return  0;
}
```

(13)Root Directory

目錄是存放檔案,子目錄的基本資訊, 每個磁碟機都有一個最原始的目錄(always 存在), 於 format 後就已產生, 這個目錄就叫做 Root Directory. 在 FAT12/FAT16 中 Root Directory 的位置是固定的(sAddFDB),且 FDB 的總數也是固定(BPB_RootEntCnt), 因此, 在 Root Directory 的檔案及子目錄將會受限制. 而 FAT32 的 Root Directory 的位置是由 BPB_RootClus 所決定, 且 size 是可變的, 就像其他目錄或檔案一樣是以 cluster chain 的形式來完成.
Root directory 與其他 directory 不同的是, 它沒有 file name(而是以"\"表示), 也沒有 date and time stamp, 也沒有包含兩個 special file "." And ".." .
而且它有其他目錄所沒有的屬性 ATTR_VOLUME_ID.

(14)目錄結構-FDB

FDB(File Description Block) 是一個 32 bytes 的資料區塊,內容有檔案(目錄)的起頭, 檔案名稱, 建立時間等資訊, 但是不含資料本身(由 FAT 表來 search). 所以每個 FDB 就代表每一個 file 及 directory 的資訊. 對於子目錄來說前兩個特殊的 FDB ,其名稱是 "." 及 "..".

FAT 16 Byte Directory Entry Structure

Name	Offset (byte)	Size (bytes)	Description
主檔名 DIR_Name[0~7]	0	8	DIR_Name[0] 0x00: 表示未使用, 且從此後就無 FDB 0x05:表示為字元"0xE5" 0xE5: 表示檔案被刪除 0x2E: 目錄名稱 "."
副檔名 DIR_Name[8~11]	8	11	ASCII code

DIR_Attr	11	1	File attributes: ATTR_READ_ONLY 0x01 ATTR_HIDDEN 0x02 ATTR_SYSTEM 0x04 ATTR_VOLUME_ID 0x08 =>有此屬性，則 DIR_Name 就表示 Volume label ATTR_DIRECTORY 0x10 =>表示此 FDB 是為目錄 ATTR_ARCHIVE 0x20 =>當檔案被 create, written, rename 時被設定 The upper two bits of the attribute byte are reserved and should always be set to 0 when a file is created and never modified or looked at after that.
Reserved	12	6	Reserved
DIR_LstAccDate	18	2	Last access date. Note that there is no last access time, only a date. This is the date of last read or write. In the case of a write, this should be set to the same date as DIR_WrtDate.
Reserved	20	2	值為 0
DIR_WrtTime	22	2	Time of last write. Note that file creation is considered a write.
DIR_WrtDate	24	2	Date of last write. Note that file creation is considered a write.
DIR_FstClus	26	2	word of this entry’ s first cluster number.
DIR_FileSize	28	4	32-bit DWORD holding this file’ s size in bytes.

FAT 32 Byte Directory Entry Structure

Name	**Offset (byte)**	**Size (bytes)**	**Description**
主檔名 DIR_Name[0~7]	0	8	DIR_Name[0] 0x00: 表示未使用，且從此後就無 FDB 0x05:表示為字元”0xE5” 0xE5: 表示檔案被刪除 0x2E: 目錄名稱 “.”
副檔名 DIR_Name[8~11]	8	11	ASCII code

DIR_Attr	11	1	File attributes: ATTR_READ_ONLY 0x01 ATTR_HIDDEN 0x02 ATTR_SYSTEM 0x04 ATTR_VOLUME_ID 0x08 =>有此屬性，則 DIR_Name 就表示 Volume label ATTR_DIRECTORY 0x10 =>表示此 FDB 是為目錄 ATTR_ARCHIVE 0x20 =>當檔案被 create, written, rename 時被設定 ATTR_LONG_NAME ATTR_READ_ONLY | ATTR_HIDDEN | ATTR_SYSTEM | ATTR_VOLUME_ID The upper two bits of the attribute byte are reserved and should always be set to 0 when a file is created and never modified or looked at after that.
DIR_NTRes	12	1	Reserved for use by Windows NT. Set value to 0 when a file is created and never modify or look at it after that.
DIR_CrtTimeTenth	13	1	Millisecond stamp at file creation time. This field actually contains a count of tenths of a second. The granularity of the seconds part of DIR_CrtTime is 2 seconds so this field is a count of tenths of a second and its valid value range is 0-199 inclusive.
DIR_CrtTime	14	2	Time file was created.
DIR_CrtDate	16	2	Date file was created.
DIR_LstAccDate	18	2	Last access date. Note that there is no last access time, only a date. This is the date of last read or write. In the case of a write, this should be set to the same date as DIR_WrtDate.
DIR_FstClusHI	20	2	High word of this entry’ s first cluster number (always 0 for a FAT12 or FAT16 volume).
DIR_WrtTime	22	2	Time of last write. Note that file creation is considered a write.
DIR_WrtDate	24	2	Date of last write. Note that file creation is considered a write.
DIR_FstClusLO	26	2	Low word of this entry’ s first cluster number.
DIR_FileSize	28	4	32-bit DWORD holding this file’ s size in bytes.

(14-1)下列檔名的字元是不合規範

(a)小於 0x20 的字元(除了 DIR_Name[0] 的 0x05)

(b) 0x22, 0x2A, 0x2B, 0x2C, 0x2E, 0x2F, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F, 0x5B, 0x5C, 0x5D, and 0x7C

(14-2)User 輸入的檔名皆轉為大寫如下

“foo.bar” -> “FOO BAR”“FOO.BAR” -> “FOO BAR”
“Foo.Bar” -> “FOO BAR”
“foo” -> “FOO “
“foo.” -> “FOO “
“PICKLE.A” -> “PICKLE A “
“prettybg.big” -> “PRETTYBGBIG”
“.big” -> illegal, DIR_Name[0] cannot be 0x20

(14-3)由屬性 ATTR_DIRECTORY 我們可知此 FDB 是目錄或是檔案, 於 FAT32 中由 ATTR_LONG_NAME(0x0f), 可知道是否有長檔名存在.(後面說明長檔名形式)

(14-4)檔案或是目錄的起頭是由 DIR_FstClus(FAT16) 或是 (DIR_FstClusHI<<16)+ DIR_FstClusLO(FAT32) 來找出, 由此 cluster number (由 FSClus2LBA() 轉為 LBA), 再配合 FAT 表(後面說明)及 DIR_FileSize 就可將檔案讀出資料.

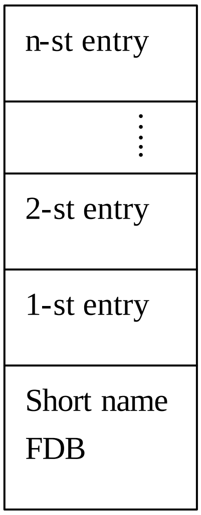
(15) 長檔名

長檔名的 support 可讓檔名最多有 255 英文字元(127 中文字), 為了與原先的短檔名相容, 長檔名只是在原先的 FDB 上增加多組 32 bytes(Long Directory Entry Structure) 的 information. 下圖 a 是一個長檔名切割為 n 個 32 bytes 的順序, 圖 B 是真正長檔名存放的格式.

圖 A

1-st entry
2-st entry
……….
n-st entry

圖 B



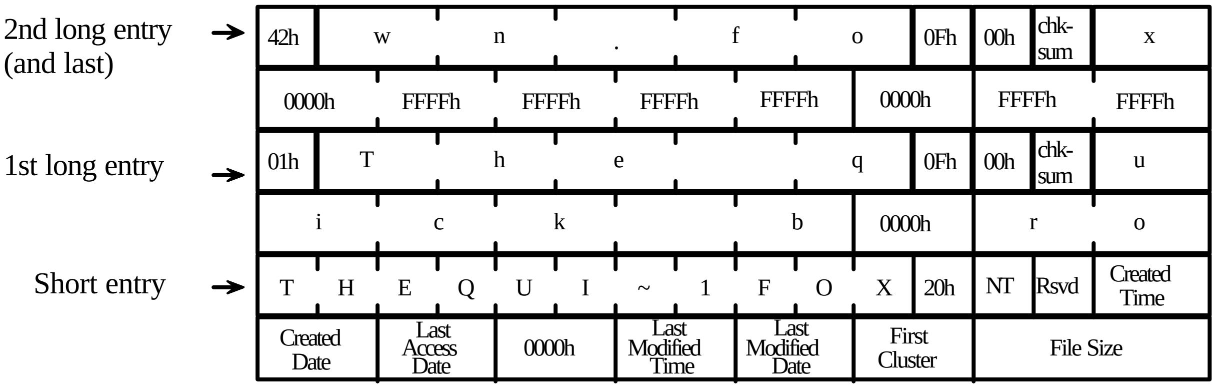
(15-1) FAT Long Directory Entry Structure

Name	**Offset (byte)**	**Size (bytes)**	**Description**
LDIR_Ord	0	1	N 個 Entry 的順序(1….(0x40 | N), 最後一個 entry 是 以 0x40 | N 表示.
LDIR_Name1	1	10	Unicode, 於此 entry, 長檔名 1-5 字
LDIR_Attr	11	1	一定為 ATTR_LONG_NAME(0x0f), 所以由此 Byte 可知 FDB 是否為有長檔名
LDIR_Type	12	1	If zero, indicates a directory entry that is a sub-component of a long name. NOTE: Other values reserved for future extensions. Non-zero implies other dirent types.
LDIR_Chksum	13	1	8.3 短檔名的 Checksum .
LDIR_Name2	14	12	Unicode, 於此 entry, 長檔名 6-11 字
LDIR_FstClusLO	26	2	一定為 0x00
LDIR_Name3	28	4	Unicode, 於此 entry, 長檔名 12-13 字.

由上表可知, 開始對 FDB 作處理時, 先以 第 11 byte 的屬性(不管是 short name or long name entry, 此 byte 都是為屬性), 只要為 0x0f 就是表示此 32 bytes 的資料為 long name entry 而不是 short name FDB. 再來以 LDIR_Ord 只要有 0x40 的 mask 就表示為最後的 entry，且可知道有幾個 entry 在 short name FDB 之前，例如,0x43=0x40 | 0x03 就表示共有 3 個 entry.

(15-2) Long name example

在此我們舉一個 long name **的例子如下，若是** file name = "The quick brown.fox".



2nd long entry (and last) →	42h	w	n	.	f	o	0Fh	00h	chk- sum	x					
	0000h	FFFFh	FFFFh	FFFFh	FFFFh	0000h	FFFFh	FFFFh							
1st long entry →	01h	T	h	e		q	0Fh	00h	chk- sum	u					
	i	c	k		b	0000h	r	o							
Short entry →	T	H	E	Q	U	I	~	1	F	O	X	20h	NT	Rsvd	Created Time
	Created Date	Last Access Date	0000h	Last Modified Time	Last Modified Date	First Cluster	File Size								

(16) gen_card_dir() 的 function 說明

gen_card_dir 是一個 iteration 的 function, 其目的就是找出所有的目錄, 將目錄的名稱及起始點作一記錄. 開始的參數就是我們之前找到的 Root 起始點 sAddFDB, 即呼叫 gen_card_dir(sAddFDB) 就可完成

```
void gen_card_dir(UINT32 lba)
{
    UINT32  iRes;
    UINT32  index;
    BYTE    pFDB[33];

    if ( FSGetStatus() == CARD_STATUS_NG ) {
        pFsJpeg->iso9660_dir_cnt = 0;
        return;
    }

    index=0;
    do {
        UINT32  myClus=0;
        UINT32  myLBA;
         iRes = FSGetFDB(lba, index++, pFDB);//Get 32 bytes FDB
        if (iRes==0) {
            break; //when iRes==0, no extra FDB(FDB =0X00)
        }
        if (iRes==1) {
            break; //read error
        }
        if (pFDB[0]==0xe5) { // 跳過被刪除的檔 FDB
            if (lba!=iRes) { //Jeff 20020226, next FDB locates in next lba, but in the same
cluster
                lba = iRes;
                index = 0;
            }
            continue; //deleted item
        }
        if (pFDB[0]==0x05) pFDB[0]=0xe5;//特殊字元 E5
if (pFDB[11]&0x10
 && !(pFDB[11]&0x02)// add for not displaying the files in recycle bin
 ) { //找出目錄的屬性 0x10
 if (pFDB[0]==0x2e) { // 跳過 . 及 ..
     if (lba!=iRes) { // next FDB locates in next lba, but in the same cluster
         lba = iRes;
         index = 0;
     }
     continue;
 }

 if (pFsJpeg->iso9660_dir_cnt >= ISO_DIR_MAX) { //max-dir limition
     break;
 }
 // 找出目錄開始的 cluster
 myClus = ( getBiUINT16(pFDB+20)<<16 ) | ( getBiUINT16(pFDB+26) );
 myLBA = FSClus2LBA(myClus);

 iso9660_dir[pFsJpeg->iso9660_dir_cnt].AD_lba = myClus; //記錄開始位置
 iso9660_dir[pFsJpeg->iso9660_dir_cnt].loc = l2msf(myLBA);
 iso9660_dir[pFsJpeg->iso9660_dir_cnt].dir = 0;
 iso9660_assign_name(pFDB, iso9660_dir[pFsJpeg->iso9660_dir_cnt].name, 8, 0);
 #if defined (FAT_FILE_MODE) || defined(FILE_MODE_WRITE)
 iso9660_dir[pFsJpeg->iso9660_dir_cnt].parent_dir=dir_stack[dir_stack_index];
 // 記錄此目錄的上層目錄 index
#endif //  #if defined (FAT_FILE_MODE)
 pFsJpeg->iso9660_dir_cnt++;

 if (lba!=myLBA) {
       #if defined (FAT_FILE_MODE) || defined(FILE_MODE_WRITE)
       dir_stack_index++;
      dir_stack[dir_stack_index]=pFsJpeg->iso9660_dir_cnt-1;
      #endif //  #if defined (FAT_FILE_MODE)
     gen_card_dir(myLBA);//再次呼叫 gen_card_dir 找下一層目錄
    #if defined (FAT_FILE_MODE) || defined(FILE_MODE_WRITE)
       dir_stack_index--;
    #endif //  #if defined (FAT_FILE_MODE)
```

```
                if ( FSGetStatus() == CARD_STATUS_NG ) {
                    pFsJpeg->iso9660_dir_cnt = 0;
                    return;
                }
            }
        }

        if (lba!=iRes) { //next FDB locates in next lba, but in the same cluster
            lba = iRes;
            index = 0;
        }
    } while (iRes);
}
```

(16) read_card_file(UINT32 lba, UINT32 iDir)的 function 說明

read_card_file 的目的是在 search 某個目錄下的所有檔案的起始點及檔名並且作一記錄. 呼叫 function 的方式就是以 gen_card_dir() 所產生的目錄起始位置作為參數，如下所示

read_card_file(msf2l(iso9660_dir[i].loc), i);

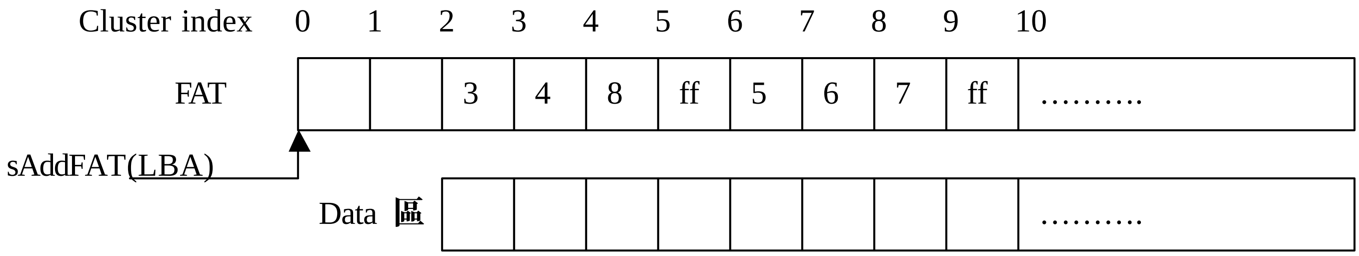
```
int read_card_file(UINT32 lba, UINT32 iDir)
{
    BYTE      pFDB[33];
    UINT32    index;
    UINT32    iRes;
    BYTE      szFile[13];
    int       bFILE_found = 0;
    UINT32    lba_longname=0;     //linrc for FAT long file name....
    UINT32    index_longname=0;

    index = 0;
    do {
        UINT16    myCmd;
        polling();
        //    省略一些 code, 請見 8202 code
        iRes = FSGetFDB(lba, index++, pFDB);// Get 32 bytes FDB
        if (iRes==0) break; //when iRes==0, no extra FDB(FDB=0x00)
        if (iRes==1) {
            return 0; //read error
        }
        if (pFDB[0]==0xe5)   continue; //deleted item
        if (pFDB[0]==0x05) pFDB[0]=0xe5; //特殊字 0xe5


        #if defined(READ_FAT_LONGNAME)&&defined(SUPPORT_FS_LONGNAME)      /
        if((pFDB[11]==0x0f)&&((pFDB[0]&0xF0)==0x40))// 有 long name entry
        {
            lba_longname    = lba;              // record the long file name FDB lba...
            index_longname = index-1;
        }
        #endif
```

```
if ( ( (pFDB[11]&0x10)==0) && //非目錄, 找出 short name FDB
       (pFDB[11]!=0x0f)    //skip long file name, long-name-n + ... + long-name-0 + old-FDB
       )
  {
       BYTE      szFileMain[10], szFileExt[4];
       UINT32   iFile;
       int        type;
      UINT32   iLBA, iClus, iSize;
       int        i;

       iso9660_assign_name(pFDB, szFileMain, 8, 0); //main file name
     iso9660_assign_name(pFDB+8, szFileExt, 3, 0); //ext file name
      strcpy(szFile, szFileMain);
      i = strlen(szFileMain);
      szFile[i] = '.';
      szFile[i+1] = '\0';
      strcat1(szFile, szFileExt);

     iSize = getBiUINT32(pFDB+28); //file size
   iClus = ( getBiUINT16(pFDB+20)<<16 ) | ( getBiUINT16(pFDB+26) ); //記錄 file 起點
      iLBA = FSClus2LBA(iClus);

      #ifdef FAT_FILE_MODE//wthsin, 2004/10/5 09:45am
    iso9660_file[ pFsJpeg->iso9660_file_cnt ].parent_dir = iDir; //記錄 file 的上一層目錄 index
    strcpy(iso9660_file[ pFsJpeg->iso9660_file_cnt ].file_ext_name,szFileExt); //記錄 ext file name
       #endif

       #if (defined(READ_FAT_LONGNAME)&&defined(SUPPORT_FS_LONGNAME))
       strcpy(iso9660_file[ pFsJpeg->iso9660_file_cnt ].file_short_name,szFileMain);//記錄 main
                                                                                      //file name
       #endif

      iFile = strlen(szFile);
      ConvertLowerCaseToUpperCase( szFile );//將 file name 改為大寫
    type = ProcessGeneralFileType( szFile, iSize );//找出可 support 的檔

      #ifdef SPHE8202_FAT_WRITE_API//SPHE8202_CARD_WRITE
      iso9660_file[ pFsJpeg->iso9660_file_cnt ].presave_time = getBiUINT16(pFDB+18);
```

```
            iso9660_file[  pFsJpeg->iso9660_file_cnt ].save_time = getBiUINT32(pFDB+22);
            iso9660_file[  pFsJpeg->iso9660_file_cnt ].ext_info = getBiUINT32(pFDB+8);
            #endif

            if( !type )
            {
              type  =  ProcessNonGeneralFileType(  szFile,  l2msf(iLBA),  iSize,  iDir  );
                if(  type  ==  -1 )
                        break;
            }

           if  (type)  {
                if  (pFsJpeg->iso9660_file_cnt >= ISO_FILE_MAX) //max-file limition
                        break;

                #ifdef  SUPPORT_FS_LONGNAME
            SetISO9660FileInfo(  l2msf(  iLBA  ),  iSize,  iDir,  szFile,  type,  0,  lba_longname,
                 index_longname );// 記錄長檔名 information
                 #else
               //  紀錄 short name information,
             //  iso9660_file[ pFsJpeg->iso9660_file_cnt  ].AD_lba       = AD_lba;
             //iso9660_file[ pFsJpeg->iso9660_file_cnt  ].loc            =  uiLoc;
               //iso9660_file[ pFsJpeg->iso9660_file_cnt  ].size          =  uiSize;
               //iso9660_file[ pFsJpeg->iso9660_file_cnt  ].dir            =  uiDir;
               //iso9660_file[ pFsJpeg->iso9660_file_cnt ].name=    szFile;
             SetISO9660FileInfo(  l2msf(  iLBA  ),  iSize,  iDir,  szFile,  type,  0  );
                 #endif

                bFILE_found  =  1;

              //Maoyong  2004.05.26,  show  file/dir  reading  cnt/percent
                 show_card_reading_osd(SHOW_READ_PERCENT);
            }

        }
    }  while  (iRes);
    return(bFILE_found);
}
```

(17)FAT (File Allocation Table)

檔案配置表是紀錄檔案實際位置的表格(以 cluster 來表示), 靠著前面所述 FDB 先找出 file 的起始 cluster idnex, 再配合此 FAT 表, 就可將 file 讀出資料. FAT 的單元的 bits 數會依據 FAT12/FAT16/FAT32 而有所不同, FAT12 來說每個單元是 12 bits, FAT16 來說每個單元是 16 bits, FAT32 來說每個單元是 32 bits,(實際只用 28 bits, 4 bits 保留). 每個單元有兩種意義: 一是單元本身的編號(index), 另一個是單元內所含的值. 如下圖



由上圖可知 data 區是以 單元 cluster index 來互相對應(從 cluster 2 開始). FAT 的前兩個單元(index 0,1)是沒有數值的. 以 FAT32 的數值來說, 32bits 中的最高 4 bits, 是不被更動, 除非是 format 後才會被更動. 以下單元的數值表示特別意義(以 FAT32 為例, 數值&0x0FFFFFFF 的結果)

(a)0x00000000: 表示 cluster is free

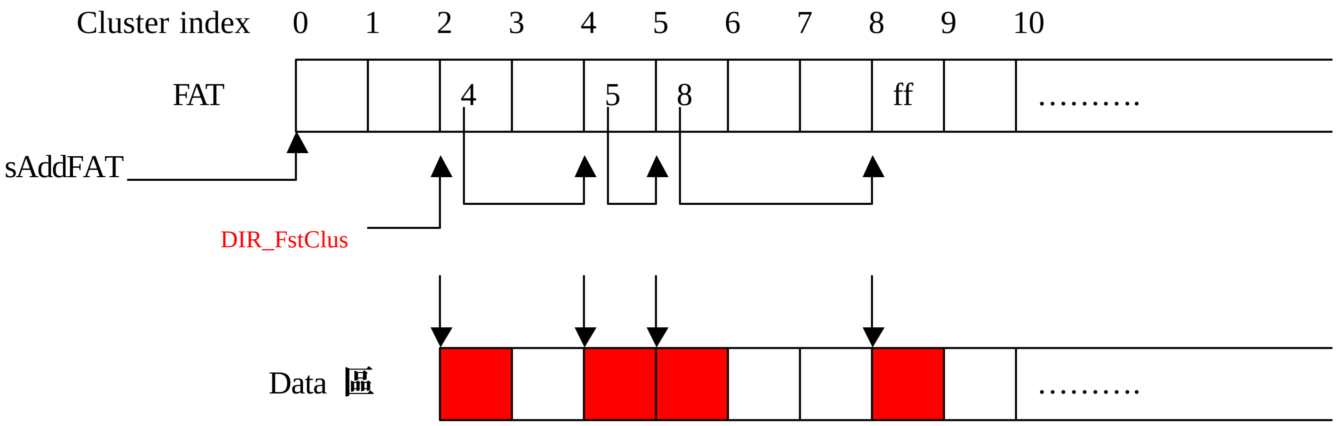
(b)0x0FFFFFF7: 表示 bad cluster

(c)0x0FFFFFFF: 所有 bit 為 1, 表示 file 的 cluster number 終了

要讀出單元內的值需改用 LBA, 如 cluster 2 位置為 sAddFAT+ (2*4 bytes/ BPB_BytsPerSec).

(18)File 的讀取方式

首先是以 FDB 先找出 file 的起始 DIR_FstClus (cluster idnex), 再以 FAT 表找到此 index 內單元的值, 再以此單元內的值找出下個 index, 如此循環直到單元的值為 0xffff(FAT16). 下圖表示此 search 方式, DATA 區的紅色部分表示被讀出的 file 內容.



(19)PMP code PlayFile(DIR_REC *FileInfo) 說明

這個 function **是可將選擇到的** file **作播放，呼叫的方式是將上述所記錄的** iso9660_file **當作參數如** PlayFile(&(iso9660_file[index]));

```
void PlayFile(DIR_REC *FileInfo)
{
     UINT32 fileloc, filelen;
     int vid = 0;
     BYTE bHasShowGUI = 0;

      pFsJpeg->gbInJpeg = 0;
      pFsJpeg->gbfsSlide = 0;
      pFsJpeg->gfsnot2NormalSpeed = 0;
     Mp3_kbitrate = 0;
      pFsJpeg->iGraphMode = 0;

     fileloc = FileInfo->loc; // file 的起始位置 msf
     filelen = FileInfo->size; // file 的大小
     cardFile.stLBA = msf2l(fileloc); // file 的起始位置 LBA
      cardFile.stClus = FSLBA2Clus(cardFile.stLBA); // file 的起始位置 cluster number
     cardFile.iFileSize = filelen; // 設定 cardFile 參數
     cardFile.curLBA = cardFile.stLBA; // 設定 cardFile 參數
     cardFile.curClus = cardFile.stClus; // 設定 cardFile 參數
      cardFile.iLeaveSize = cardFile.iFileSize; // 設定 cardFile 參數

      // 省略一些 code 主要是依據 file 的 type 來 LoadModual
      //LoadModual(MODUAL_MPEG);  or LoadModual(MODUAL_WMA); …….

      FS_MainLoop(&vid,  fileloc,  filelen);    // 開始將 file 傳入播放
      // 省略一些 code 主要是做 UI 的顯示處理

}
```

(19-1) int FS_MainLoop(int *vid, UINT32 msf, UINT32 len) 說明

FS_MainLoop **會依據** file **的** type **選擇不同的** mainloop, **例如** Mp3 **的** file **會對應到** FS_MP3MainLoop(msf, len), Jpeg file **會對應到** FS_JPEGMainLoop(vid, msf, len);

```
 int FS_MainLoop(int *vid, UINT32 msf, UINT32 len)
{
  ////省略一些 code
  if (pFsJpeg->gifsFuncBtn == FS_FUNC_MP3)
  {
   //省略一些 code
  FS_MP3MainLoop(msf, len);
  //省略一些 code
  }


}
```

(19-2) void FS_MP3MainLoop(UINT32 msf, UINT32 len) **說明**

將傳入的 file **導入** MediaMainLoop(); **內**

```
void FS_MP3MainLoop(UINT32 msf, UINT32 len)
{
//省略一些 code
 if (media_type==MEDIA_CARD) // 設定 cardFile 參數
 {
        cardFile.stLBA = msf2l(msf);
        cardFile.stClus = CardLBA2Clus( cardFile.stLBA );
        cardFile.iFileSize = len;
        cardFile.curLBA = cardFile.stLBA;
        cardFile.curClus = cardFile.stClus;
        cardFile.iLeaveSize = cardFile.iFileSize;
 }
//省略一些 code
if ((GetCurrentFileType() == CDROM) || (GetCurrentFileType() == CDROM_WMA) || (GetCurrentFileType()
                                        == CDROM_AAC) || (GetCurrentFileType() == CDROM_WAV))
PlayMP3(msf, s_len); //主要呼叫 InitializeCDPlayback() , 將 srv_kernel = srv_cardfile;
MediaMainLoop(); // 呼叫 MediaMainLoop
//省略一些 code
}
```

(19-3) void MediaMainLoop(void) 說明

```
void MediaMainLoop(void)
{
    MediaMainLoop_Init();
    do {

        srv_kernel();//srv_kernel = srv_cardfile
        Polling();
    } while (!IsAVDMediaInterrupt());

    MediaMainLoop_Exit();
}
```

(19-4) int srv_cardfile(void) 說明

在此 function 中, 會依據 file 的大小來呼叫 FSReadStream() function 來抓取 file 的內容

```
int srv_cardfile(void)
{
 //省略一些 code
 if ((GetCurrentFileType() == CDROM)||(GetCurrentFileType() == CDROM_WMA))
 {          dma_ilen        = 1024;
       if ( FSReadStream(&cardFile, 2, stream_ptr) ) { // 每次以 2 個 sector 大小來讀取
            source_end = 1; //read error
              return 0;
        }
        if (cardFile.iLeaveSize<=0) { // file leave size 於 FsreadStream() 中遞
             source_end = 1;           //減, <=0 表示結束
               return 0;
        }
        if (++cbv_y>=CBV_H) cbv_y=0;
         regs0->dvddsp_ry = cbv_y;
    }
//省略一些 code
parser(&do_system, dma_ilen);
    if (cardFile.iLeaveSize<=0)      source_end = 1;
    return 0;
}
```

(19-5) BYTE FSReadStream(CARD_FILE *cfFile, UINT32 nSec, BYTE *buf) 說明

此 function **就是讀** file **的資料**, **每此以** nSec sectors **讀入** buf **中**. **呼叫的方式如** FSReadStream(&cardFile, 2, stream_ptr), **其中** cardFile **就是前面所述的** cardFile structure.

```
BYTE FSReadStream(CARD_FILE *cfFile, UINT32 nSec, BYTE *buf)
{
    UINT32   offset=0;
    if(play_state == VCD_STATE_STOP)
        return 0;
    watchdog_renew(0xffff); // reset watchdog
   while (nSec > 0) {
         UINT32   iLBAinClus, iBat;
         ControlWithinOneFile(cfFile);
        iLBAinClus = FSLBAinClus(cfFile->curLBA);// 找出目前 LBA 相對於目前的 cluster
                                                  //尾端還剩多少個 LBA
         if (iLBAinClus==0) {
             iBat = 1;
         }
        else if (nSec >= (iLBAinClus+1)) {
             iBat = iLBAinClus + 1;
         } else {
             iBat = nSec;
         }
        if(iBat >= SCSIREAD_MAX_SECTOR) //for CardReadSector can't work when
             iBat = SCSIREAD_MAX_SECTOR;// nSec>=SCSIRead_MAX_SECTOR*2

             if ( FSReadSector(cfFile->curLBA, iBat, buf+offset) ) {// 呼叫 read sector function
                                                                   //讀取 data 區的資料
              return 1;
         }
        if (iLBAinClus == (iBat-1)){
              #ifdef USE_NAV_CACHE
             if (cardP.typeFAT==CARD_FAT12) { //Jeff 20040130
                    cfFile->curClus = FSGetNextClus(cfFile->curClus); // 利用 FAT 表讀取 file
                                                                      //的下一個 cluster
```

```
            } else {
                  cfFile->curClus = FSCacheGetNextClus(cfFile->curClus);
             }
             #else
             cfFile->curClus = FSGetNextClus(cfFile->curClus);
             #endif

             cfFile->curLBA = FSClus2LBA(cfFile->curClus);// 將 LBA 更改為 cluster
        } else {
             cfFile->curLBA += iBat;
         }

        s_msf = l2msf(cfFile->curLBA);
      offset = offset + cardP.bytePerSec * iBat;
        cfFile->iLeaveSize = cfFile->iLeaveSize - cardP.bytePerSec * iBat;//修改剩下所要讀取
                                                                            //的 file size
        nSec -= iBat;// update 所需讀的 sector

    }

    return 0;
} while (nSec > 0)
```

(19-6) UINT32 FSGetNextClus(UINT32 clus) 的說明

FSGetNextClus () function 就是依據 FAT 表來 search file 的下一個 cluster number.

```
UINT32 FSGetNextClus(UINT32 clus)
{
    UINT32   lba;
    UINT32   nOffset;
    UINT32   nextLink;
//將 cluster number 更換為 FAT 得實際位置
 lba = cardP.sAddFAT + clus * cardP.nItemSizeFAT / 2 / cardP.bytePerSec;
    nOffset = (clus * cardP.nItemSizeFAT / 2) % cardP.bytePerSec;

    if (cardP.typeFAT==CARD_FAT12) {
        UINT32   nByteOffset;
        UINT32   low, high;

      nByteOffset = clus * cardP.nItemSizeFAT % cardP.bytePerSec % 2;

      if ( FSReadSector(lba, 2, pwb) ) {// 讀出 FAT12 單元開始的 2 sectors
          return 0xffffffff; //read error
        }
        // 找出 FAT12 單元內容(12 bits)
      if (nByteOffset == 0) {
           low = *(pwb+nOffset);
          high = *(pwb+nOffset+1) & 0x0f;
         nextLink = ( ( high << 8) | low) & 0xfff;
      } else {
          low = ( *(pwb+nOffset) >> 4) & 0x0f;
          high = (*(pwb+nOffset+1) << 4) & 0xff0;
           nextLink = (high | low) & 0xfff;
        }
    } else {
        // 讀出 FAT16 or  FAT32 單元開始的 1 sector
      if ( FSReadSector(lba, 1, pwb) ) return 0xffffffff; //read error

       if (cardP.typeFAT==CARD_FAT32) {
           nextLink = getBiUINT32(pwb+nOffset); // 找出 FAT32 單元內容(32 bits)
        } else {
             //FAT16
            nextLink = getBiUINT16(pwb+nOffset); // 找出 FAT16 單元內容(16 bits)
         }

        if (cardP.typeFAT==CARD_FAT32) {
            if (nextLink >= 0xfffff8) nextLink=0;//??? 覺得應當以 0xfffffff(<2^28)
        } else {
             //FAT16
            if (nextLink >= 0xfff8) nextLink=0; //??? 覺得應當以 0xffff(<2^16)
         }
    }

    return nextLink;
}
```