```
*** D64 (Electronic form of a physical 1541 disk)
*** Document revision: 1.11
*** Last updated: Nov 7, 2008
*** Compiler/Editor: Peter Schepers
*** Contributors/sources: Immers/Neufeld "Inside Commodore  DOS"
                          Wolfgang Moser
```

(Note: This document will try to explain the layout of the 1541 disks, as
well as some of the files they contain. However, I do not explain GEOS   or
REL files here. See the file GEOS.TXT or REL.TXT for more   information   on
those file types or disk structure). Special thanks to Wolfgang Moser   for
his work on identifying the differences between the various 'Speeder' DOS's
that exist. This information is included in a table later in this document.

   First and foremost we have D64, which is basically a sector-for-sector
copy of a 1540/1541 disk. There are several versions of these which I will
cover shortly. The standard D64 is a 174848 byte file comprised of 256 byte
sectors arranged in 35 tracks with a varying number of sectors per track
for a total of 683 sectors. Track counting starts at 1, not 0, and goes up
to 35. Sector counting starts at 0, not 1, for the first sector, therefore
a track with 21 sectors will go from 0 to 20.

   The original media (a 5.25" disk) has the tracks laid out in circles,
with track 1 on the very outside of the disk (closest   to   the   sides) to
track 35 being on the inside of the disk (closest to the inner hub ring).
Commodore, in their infinite wisdom, varied the number of sectors per track
and data densities across the disk to optimize available storage, resulting
in the chart below. It shows the sectors/track for a standard D64. Since
the outside diameter of a circle is the largest (versus closer to the
center), the outside tracks have the largest amount of storage.

        Track   Sectors/track   # Sectors   Storage in Bytes
        -----   -------------   ---------   ----------------
         1-17   21   357   7820
        18-24   19   133   7170
        25-30   18   108   6300
        31-40(*)   17   85   6020
                                     ---
                                     683 (for a 35 track image)

  Track #Sect #SectorsIn D64 Offset   Track #Sect #SectorsIn D64 Offset
  ----- ----- ---------- ----------   ----- ----- ---------- ----------
     1   21   0 $00000   21   19   414 $19E00
     2   21   21 $01500   22   19   433 $1B100
     3   21   42 $02A00   23   19   452 $1C400
     4   21   63 $03F00   24   19   471 $1D700
     5   21   84 $05400   25   18   490 $1EA00
     6   21   105 $06900   26   18   508 $1FC00
     7   21   126 $07E00   27   18   526 $20E00
     8   21   147 $09300   28   18   544 $22000
     9   21   168 $0A800   29   18   562 $23200
    10   21   189 $0BD00   30   18   580 $24400
    11   21   210 $0D200   31   17   598 $25600
    12   21   231 $0E700   32   17   615 $26700
    13   21   252 $0FC00   33   17   632 $27800
    14   21   273 $11100   34   17   649 $28900
    15   21   294 $12600   35   17   666 $29A00
    16   21   315 $13B00   36(*) 17   683 $2AB00
    17   21   336 $15000   37(*) 17   700 $2BC00
    18   19   357 $16500   38(*) 17   717 $2CD00
    19   19   376 $17800   39(*) 17   734 $2DE00
    20   19   395 $18B00   40(*) 17   751 $2EF00
```

     (*)Tracks 36-40 apply to 40-track images only

   The directory track should be contained totally on track 18. Sectors 1-18
contain the entries and sector 0 contains the BAM (Block Availability  Map)
and disk name/ID. Since the directory is only 18 sectors large (19 less one
for the BAM), and each sector can contain only  8  entries  (32  bytes  per
entry), the maximum number of directory entries is 18 * 8 = 144. The  first
directory sector is always 18/1, even though the t/s pointer at 18/0 (first
two bytes) might point somewhere else.  It  then  follows  the  same  chain
structure as a normal file, using a sector interleave of 3. This makes  the
chain links go 18/1, 18/4, 18/7 etc.

   Note that you can extend the directory off of track  18,  but  only  when
reading the disk or image. Attempting to write to a directory sector not on
track 18 will cause directory corruption.  See  the  section  below  called
Non-Standard & Long Directories.

Each directory sector has the following layout (18/1 partial dump):

00: 12 04 81 11 00 4E 41 4D 45 53 20 26 20 50 4F 53 <- notice the T/S link
10: 49 54 A0 A0 A0 00 00 00 00 00 00 00 00 00 15 00 <- to 18/4 ($12/$04)
20: 00 00 84 11 02 41 44 44 49 54 49 4F 4E 41 4C 20 <- and how its not here
30: 49 4E 46 4F A0 11 0C FE 00 00 00 00 00 00 61 01 <- ($00/$00)

   The first two bytes of the sector ($12/$04) indicate the location of  the
next track/sector of the directory (18/4). If the track is set to $00, then
it is the last sector of the directory. It is possible,  however  unlikely,
that the directory may *not* be competely on track 18 (some disks do  exist
like this). Just follow the chain anyhow.

   When the directory is done, the track value will be $00. The sector  link
should contain a value of $FF, meaning the whole sector is  allocated,  but
the actual value doesn't matter. The drive will return  all  the  available
entries anyways. This is a breakdown of a  standard  directory  sector  and
entry:

   Bytes: $00-1F: First directory entry
          00-01: Track/Sector location of next directory sector ($00 $00 if
                 not the first entry in the sector)
             02: File type.
                 Typical values for this location are:
                   $00 - Scratched (deleted file entry)
                    80 - DEL
                    81 - SEQ
                    82 - PRG
                    83 - USR
                    84 - REL
                 Bit 0-3: The actual filetype
                          000 (0) - DEL
                          001 (1) - SEQ
                          010 (2) - PRG
                          011 (3) - USR
                          100 (4) - REL
                          Values 5-15 are illegal, but if used will produce
                          very strange results. The 1541 is inconsistent in
                          how it treats these bits. Some routines use all 4
                          bits, others ignore bit 3,  resulting  in  values
                          from 0-7.
                 Bit   4: Not used
                 Bit   5: Used only during SAVE-@ replacement
                 Bit   6: Locked flag (Set produces ">" locked files)
                 Bit   7: Closed flag (Not  set  produces  "*", or "splat"
                          files)
          03-04: Track/sector location of first sector of file
          05-14: 16 character filename (in PETASCII, padded with $A0)

```
                  15-16: Track/Sector location of first side-sector block (REL file
                         only)
                     17: REL file record length (REL file only, max. value 254)
                  18-1D: Unused (except with GEOS disks)
                  1E-1F: File size in sectors, low/high byte  order  ($1E+$1F*256).
                         The approx. filesize in bytes is <= #sectors * 254
                  20-3F: Second dir entry. From now on the first two bytes of  each
                         entry in this sector  should  be  $00  $00,  as  they  are
                         unused.
                  40-5F: Third dir entry
                  60-7F: Fourth dir entry
                  80-9F: Fifth dir entry
                  A0-BF: Sixth dir entry
                  C0-DF: Seventh dir entry
                  E0-FF: Eighth dir entry
```

Files, on a standard 1541, are stored using an interleave of 10. Assuming a
starting track/sector of 17/0, the chain would run 17/0, 17/10, 17/20,
17/8, 17/18, etc.

Note: No GEOS entries are listed in the above description. See the GEOS.TXT
      file for GEOS info.

*** Non-Standard & Long Directories

  Most Commdore floppy disk drives use a single dedicated directory track
where all filenames are stored. This limits the number of files stored on a
disk based on the number of sectors on the directory track. There are some
disk images that contain more files than would normally be allowed. This
requires extending the directory off the default directory track by
changing the last directory sector pointer to a new track, allocating the
new sectors in the BAM, and manually placing (or moving existing) file
entries there. The directory of an extended disk can be read and the files
that reside there can be loaded without problems on a real drive. However,
this is still a very dangerous practice as writing to the extended portion
of the directory will cause directory corruption in the non-extended part.
Many of the floppy drives core ROM routines ignore the track value that the
directory is on and assume the default directory track for operations.

  To explain: assume that the directory has been extended from track 18 to
track 19/6 and that the directory is full except for a few slots on 19/6.
When saving a new file, the drive DOS will find an empty file slot at 19/6
offset $40 and correctly write the filename and a few other things into
this slot. When the file is done being saved the final file information
will be written to 18/6 offset $40 instead of 19/6 causing some directory
corruption to the entry at 18/6. Also, the BAM entries for the sectors
occupied by the new file will not be saved and the new file will be left as
a SPLAT (*) file.

  Attempts to validate the disk will result in those files residing off the
directory track to not be allocated in the BAM, and could also send the
drive into an endless loop. The default directory track is assumed for all
sector reads when validating so if the directory goes to 19/6, then the
validate code will read 18/6 instead. If 18/6 is part of the normal
directory chain then the validate routine will loop endlessly.

*** BAM layout

The layout of the BAM area (sector 18/0) is a bit more complicated...

```
     00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
     -----------------------------------------------
00: 12 01 41 00 12 FF F9 17 15 FF FF 1F 15 FF FF 1F
```

```
10: 15 FF FF 1F 12 FF F9 17 00 00 00 00 00 00 00 00
20: 00 00 00 00 0E FF 74 03 15 FF FF 1F 15 FF FF 1F
30: 0E 3F FC 11 07 E1 80 01 15 FF FF 1F 15 FF FF 1F
40: 15 FF FF 1F 15 FF FF 1F 0D C0 FF 07 13 FF FF 07
50: 13 FF FF 07 11 FF CF 07 13 FF FF 07 12 7F FF 07
60: 13 FF FF 07 0A 75 55 01 00 00 00 00 00 00 00 00
70: 00 00 00 00 00 00 00 00 01 08 00 00 03 02 48 00
80: 11 FF FF 01 11 FF FF 01 11 FF FF 01 11 FF FF 01
90: 53 48 41 52 45 57 41 52 45 20 31 20 20 A0 A0 A0
A0: A0 A0 56 54 A0 32 41 A0 A0 A0 A0 00 00 00 00 00
B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
    Bytes:$00-01: Track/Sector location of the first directory sector (should
                  be set to 18/1 but it doesn't matter, and don't trust  what
                  is there, always go to 18/1 for first directory entry)
             02: Disk DOS version type (see note below)
                   $41 ("A")
             03: Unused
          04-8F: BAM entries for each track, in groups  of  four  bytes  per
                  track, starting on track 1 (see below for more details)
          90-9F: Disk Name (padded with $A0)
          A0-A1: Filled with $A0
          A2-A3: Disk ID
             A4: Usually $A0
          A5-A6: DOS type, usually "2A"
          A7-AA: Filled with $A0
          AB-FF: Normally unused ($00), except for 40 track extended format,
                  see the following two entries:
          AC-BF: DOLPHIN DOS track 36-40 BAM entries (only for 40 track)
          C0-D3: SPEED DOS track 36-40 BAM entries (only for 40 track)
```

Note: The BAM entries for SPEED, DOLPHIN and ProLogic DOS use the same
      layout as standard BAM entries.

   One of the interesting things from the BAM sector is the byte at offset
$02, the DOS version byte. If it is set to anything other than $41 or $00,
then we have what is called "soft write protection". Any attempt to write
to the disk will return the "DOS Version" error code 73 ,"CBM DOS V 2.6
1541". The 1541 is simply telling you that it thinks the disk format
version is incorrect. This message will normally come up when you first
turn on the 1541 and read the error channel. If you write a $00 or a $41
into 1541 memory location $00FF (for device 0), then you can circumvent
this type of write-protection, and change the DOS version back to what it
should be.

   The BAM entries require a bit (no pun intended) more of a breakdown. Take
the first entry at bytes $04-$07 ($12 $FF $F9 $17). The first byte ($12) is
the number of free sectors on that track. Since we are looking at the track
1 entry, this means it has 18 (decimal) free sectors. The next three bytes
represent the bitmap of which sectors are used/free. Since it is 3 bytes (8
bits/byte) we have 24 bits of storage. Remember that at most, each track
only has 21 sectors, so there are a few unused bits.

```
    Bytes: 04-07: 12 FF F9 17    Track 1 BAM
           08-0B: 15 FF FF FF    Track 2 BAM
           0C-0F: 15 FF FF 1F    Track 3 BAM
           ...
           8C-8F: 11 FF FF 01    Track 35 BAM
```

   These entries must be viewed in binary to make any sense. We will use the
first entry (track 1) at bytes 04-07:

```
      FF=11111111, F9=11111001, 17=00010111
```

In order to make any sense from the binary notation, flip the bits around.

```
               111111 11112222
      01234567 89012345 67890123
      -------------------------
      11111111 10011111 11101000
      ^                     ^
   sector 0              sector 20
```

   Since we are on the first track, we have 21 sectors, and only use up to
the bit 20 position. If a bit is on (1), the sector is free. Therefore,
track 1 has sectors 9,10 and 19 used, all the rest are free. Any leftover
bits that refer to sectors that don't exist, like bits 21-23 in the above
example, are set to allocated.

   Each filetype has its own unique properties, but most follow one simple
structure. The first file sector is pointed to by the directory and follows
a t/s chain, until the track value reaches $00. When this happens, the
value in the sector link location indicates how much of the sector is used.
For example, the following chain indicates a file 6 sectors long, and ends
when we encounter the $00/$34 chain. At this point the last sector occupies
from bytes $02-$34.

```
     1        2       3       4       5       6
   ----    -----   -----   -----   -----   -----
   17/0    17/10   17/20   17/1    17/11    0/52
  (11/00) (11/0A) (11/14) (11/01) (11/0B) (0/34)
```

*** Variations on the D64 layout

These are some variations of the D64 layout:

   1. Standard 35 track layout but with 683 error bytes added on to the end
      of the file. Each byte of the error info corresponds to a single
      sector stored in the D64, indicating if the sector on the original
      disk contained an error. The first byte is for track 1/0, and the last
      byte is for track 35/16.

   2. A 40 track layout, following the same layout as a 35 track disk, but
      with 5 extra tracks. These contain 17 sectors each, like tracks 31-35.
      Some of the PC utilities do allow you to create and work with these
      files. This can also have error bytes attached like variant #1.

   3. The Commodore 128 allowed for "auto-boot" disks. With this, t/s 1/0
      holds a specific byte sequence which the computer recognizes as boot
      code. See the document C128BOOT.TXT for more info.

   Below is a small chart detailing the standard file sizes of D64 images,
35 or 40 tracks, with or without error bytes.

```
      Disk type                 Size
      ---------                 ------
      35 track, no errors       174848
      35 track, 683 error bytes 175531
      40 track, no errors       196608
      40 track, 768 error bytes 197376
```

   The following table (provided by Wolfgang Moser) outlines the differences
between the standard 1541 DOS and the various "speeder" DOS's that exist.
The 'header 7/8' category is the 'fill bytes' as the end of the sector
header of a real 1541 disk See the document G64.TXT for a better
explanation of these bytes.

```
            Disk format            |Tracks|Header 7/8|Dos type|DiskDos
                                   |      |allsechdrs|        |vs.type
            ========================================================
            Original CBM DOS v2.6  |  35  | $0f  $0f |  "2A"  |$41/'A'
            --------------------------------------------------------
            *SpeedDOS+             |  40  | $0f  $0f |  "2A"  |$41/'A'
            --------------------------------------------------------
            ProfessionalDOS Initial|  35  | $0f  $0f |  "2A"  |$41/'A'
               (Version 1/Prototype)|  40  | $0f  $0f |  "2A"  |$41/'A'
            --------------------------------------------------------
            ProfDOS Release        |  40  | $0f  $0f |  "4A"  |$41/'A'
            --------------------------------------------------------
            Dolphin-DOS 2.0/3.0    |  35  | $0f  $0f |  "2A'  |$41/'A'
            Dolphin-DOS 2.0/3.0    |  40  | $0d  $0f |  "2A'  |$41/'A'
            --------------------------------------------------------
            PrologicDOS 1541       |  35  | $0f  $0f |  "2A'  |$41/'A'
            PrologicDOS 1541       |  40  | $0f  $0f |  "2P'  |$50/'P'
            ProSpeed 1571 2.0      |  35  | $0f  $0f |  "2A'  |$41/'A'
            ProSpeed 1571 2.0      |  40  | $0f  $0f |  "2P'  |$50/'P'
            --------------------------------------------------------
```

*Note: There are also clones of SpeedDOS that exist, such as RoloDOS and
       DigiDOS. Both are just a change of the DOS startup string.

   The location of the extra BAM information in sector 18/0, for 40 track
images, will be different depending on what standard the disks have been
formatted with. SPEED DOS stores them from $C0 to $D3, DOLPHIN DOS stores
them from $AC to $BF and PrologicDOS stored them right after the existing
BAM entries from $90-A3. PrologicDOS also moves the disk label and ID
forward from the standard location of $90 to $A4. 64COPY and Star Commander
let you select from several different types of extended disk formats you
want to create/work with.

   All three of the speeder DOS's mentioned above don't alter the standard
sector interleave of 10 for files and 3 for directories. The reason is that
they use a memory cache installed in the drive which reads the entire track
in one pass. This alleviates the need for custom interleave values. They do
seem to alter the algorithm that finds the next available free sector so
that the interleave value can deviate from 10 under certain circumstances,
but I don't know why they would bother.

   Below is a HEX dump of a Speed DOS BAM sector. Note the location of the
extra BAM info from $C0-D3.

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F      ASCII
      ------------------------------------------------ ----------------
0070: 12 FF FF 03 12 FF FF 03 12 FF FF 03 11 FF FF 01  nnnnnnnnnnnnnnnn
0080: 11 FF FF 01 11 FF FF 01 11 FF FF 01 11 FF FF 01  nnnnnnnnnnnnnnnn
0090: A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0
00A0: A0 A0 30 30 A0 32 41 A0 A0 A0 A0 00 00 00 00 00    00 2A     nnnnn
00B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  nnnnnnnnnnnnnnnn
00C0: 11 FF FF 01 11 FF FF 01 11 FF FF 01 11 FF FF 01  nnnnnnnnnnnnnnnn
00D0: 11 FF FF 01 00 00 00 00 00 00 00 00 00 00 00 00  nnnnnnnnnnnnnnnn
```


   Below is a HEX dump of a Dolphin DOS BAM sector. Note the location of the

```
     extra BAM info from $AC-BF.

          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F      ASCII
          ------------------------------------------------ ----------------
     0070: 12 FF FF 03 12 FF FF 03 12 FF FF 03 11 FF FF 01  nnnnnnnnnnnnnnnn
     0080: 11 FF FF 01 11 FF FF 01 11 FF FF 01 11 FF FF 01  nnnnnnnnnnnnnnnn
     0090: A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0
     00A0: A0 A0 30 30 A0 32 41 A0 A0 A0 A0 00 11 FF FF 01    00 2A    nnnnn
     00B0: 11 FF FF 01 11 FF FF 01 11 FF FF 01 11 FF FF 01  nnnnnnnnnnnnnnnn
     00C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  nnnnnnnnnnnnnnnn
     00D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  nnnnnnnnnnnnnnnn
```

   Below is a HEX dump of a PrologicDOS BAM sector. Note that the disk name
and ID are now located at $A4 instead of starting at $90.

```
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F      ASCII
          ------------------------------------------------ ----------------
     0070: 12 FF FF 03 12 FF FF 03 12 FF FF 03 11 FF FF 01  nnnnnnnnnnnnnnnn
     0080: 11 FF FF 01 11 FF FF 01 11 FF FF 01 11 FF FF 01  nnnnnnnnnnnnnnnn
     0090: 11 FF FF 01 11 FF FF 01 11 FF FF 01 11 FF FF 01  nnnnnnnnnnnnnnnn
     00A0: 11 FF FF 01 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0 A0  nnnn
     00B0: A0 A0 A0 A0 A0 A0 30 30 A0 32 50 A0 A0 A0 A0 00        00 2P    n
     00C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  nnnnnnnnnnnnnnnn
     00ED: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  nnnnnnnnnnnnnnnn
```

*** Error codes

   Here is the meaning of the error bytes added onto the end of any extended
D64. The CODE is the same as that generated by the 1541 drive controller...
it reports these numbers, not the error code we usually see when an error
occurs.

   Some of what comes below is taken from Immers/Neufeld book "Inside
Commodore DOS". Note the descriptions are not completely accurate as to
what the drive DOS is actually doing to seek/read/decode/write sectors, but
serve as simple examples only. The "type" field is where the error usually
occurs, whether it's searching for any SYNC mark, any header ID, any valid
header, or reading a sector.

   These first errors are "seek" errors, where the disk controller is simply
reading headers and looking at descriptor bytes, checksums, format ID's and
reporting what errors it sees. These errors do *not* necessarily apply to
the exact sector being looked for. This fact makes duplication of these
errors very unreliable.

```
     Code  Error  Type  1541 error description
     ----  -----  ----  -----------------------------
      03     21    Seek  No SYNC sequence found.

                         Each  sector  data  block  and header block  are
                         preceeded by SYNC marks. If *no* sync  sequence  is
                         found within 20 milliseconds (only ~1/10 of a  disk
                         rotation!) then this error is generated. This error
                         used to mean the entire track is bad, but  it  does
                         not have to be the case. Only a small area  of  the
                         track needs to be without a  SYNC  mark  and  this
                         error will be generated.

                         Converting this error to a D64 is very  problematic
                         because it depends on where the physical head is on
                         the disk when a read attempt is made. If it  is  on
```

```
                           valid header/sectors then it  won't  occur.  If  it
                           happens over an area without SYNC  marks,  it  will
                           happen.


     02    20   Seek   Header descriptor byte not found (HEX $08, GCR $52)

                           Each sector is preceeded by an  8-byte  GCR  header
                           block, which starts with the value  $52  (GCR).  If
                           this value is not found  after  90  attempts,  this
                           error is generated.

                           Basically, what a track  has  is  SYNC  marks,  and
                           possibly valid data blocks,  but  no  valid  header
                           descriptors.


     09    27   Seek   Checksum error in header block

                           The  header  block  contains  a   checksum   value,
                           calculated by XOR'ing the TRACK,  SECTOR,  ID1  and
                           ID2 values. If this checksum is wrong,  this  error
                           is generated.


     0B    29   Seek   Disk sector ID mismatch

                           The ID's from the header  block  of  the  currently
                           read sector are compared against the ones from  the
                           low-level header of 18/0. If there is  a  mismatch,
                           this error is generated.


     02    20   Seek   Header block not found

                           This error can be reported again when searching for
                           the correct header block. An image of the header is
                           built and searched for, but not found after 90 read
                           attempts. Note  the  difference  from  the   first
                           occurance. The first one only searches for a  valid
                           ID, not the whole header.

   Note that error 20 occurs twice during this phase. The first time is when
a header ID is being searched for, the second is  when  the  proper  header
pattern for the sector being searched for is not found.


   From this point on, all the errors apply to the specific sector  you  are
looking for. If a read passed all the previous checks, then we are  at  the
sector being searched for.

   Note that the entire sector is read before  these  errors  are  detected.
Therefore the data, checksum and off bytes are available.

     Code  Error  Type   1541 error description
     ----  -----  ----   ------------------------------
     04    22    Read   Data descriptor byte not found (HEX $07, GCR $55)

                           Each sector data block is preceeded  by  the  value
                           $07, the "data block" descriptor. If this value  is
                           not there, this error is  generated.  Each  encoded
                           sector  has  actually  260  bytes.  First  is   the
                           descriptor byte, then  follows  the  256  bytes  of
                           data, a checksum, and two "off" bytes.
```

```
     05    23    Read   Checksum error in data block

                        The checksum of  the  data  read  of  the  disk  is
                        calculated, and compared against the one stored  at
                        the end of the sector. If  there's  a  discrepancy,
                        this error is generated.


     0F    74    Read   Drive Not Ready (no disk in drive or no device 1)
```

  These errors only apply when writing to a disk. I don't see the
usefulness of having these as they cannot be present when only *reading* a
disk.

```
  Code  Error  Type   1541 error description
  ----  -----  ----   -----------------------------
   06    24    Write  Write verify (on format)


   07    25    Write  Write verify error

                      Once the GCR-encoded sector  is  written  out,  the
                      drive waits for the sector to come around again and
                      verifies the whole 325-byte GCR block.  Any  errors
                      encountered will generate this error.


   08    26    Write  Write protect on

                      Self explanatory. Remove the write-protect tab, and
                      try again.


   0A    28    Write  Write error

                      In actual fact, this error never occurs, but it  is
                      included for completeness.
```

  This is not an error at all, but when gets reported when the read of a
sector is ok.

```
  Code  Error  Type   1541 error description
  ----  -----  ----   -----------------------------
   01    00    N/A    No error.

                      Self explanatory. No errors were  detected  in  the
                      reading and decoding of the sector.
```

  The advantage with using the 35 track D64 format, regardless of error
bytes, is that it can be converted directly back to a 1541 disk by either
using the proper cable and software on the PC, or send it down to the C64
and writing it back to a 1541. It is the best documented format since it is
also native to the C64, with many books explaining the disk layout and the
internals of the 1541.

*** Supporting D64 files

  The D64 layout is reasonably robust, being that it is an electronic

representation of a physical 1541 disk. It shares *most* of the 1541
attributes and it supports all file formats, since all C64 files came from
here. The only file I have found that can't be copied to a D64 is a T64 FRZ
(FRoZen files), since you lose the extra file type attribute.

   Since the D64 layout seems to be an exact byte copy of a 1541 floppy, it
would appear to be the perfect format for *any* emulator. However, it does
not contain certain vital bits of information that, as a user, you normally
don't have access to.

   Preceeding each sector on a real 1541 disk is a header block which
contains the sector ID bytes and checksum. From the information contained
in the header, the drive determines if there's an error on that header
(27-checksum error, 29-disk ID mismatch). The sector itself also contains
info (data block signature, checksum) that result in error detection (23
checksum, 22 data block not present, etc). The error bytes had to be added
on to the D64 image, "extending" the format to take into account the
missing info.

   The disk ID is important in the copy protection of some programs. Some
programs fail to work properly since the D64 doesn't contain these ID's.
These bytes would be an addition to the format which has never been done
and would be difficult to do. (As an aside, the 4-pack ZipCode files do
contain the original master disk ID, but these are lost in the conversion
of a ZipCode to a D64. Only storing *one* of the ID's is not enough, all
the sector ID's should be kept.)

   The extended track 1541 disks also presented a problem, as there are
several different formats (and how/where to store the extra BAM entries in
a sector that was not designed for them, yet still remain compatible).
Because of the additions to the format (error bytes and 40 tracks) there
exists 4 different types of D64's, all recognizeable by their size.

   It is also the only format that uses the sector count for the file size
rather than actual bytes used. This can present some problems when
converting/copying the to another format because you may have to know the
size before you begin (see LBR format).

   It also contains no consistent signature, useful for recognizing if D64
is really what it claims to be. In order to determine if a file is a D64,
you must check the file size.

*** Overall Good/Bad of D64 Files:

   Good
   ----
   * D64 files are the most widely supported and well-defined format, as it
     is simply an electronic version of a 1541 disk

   * Supports *all* filenames, even those with $00's in them

   * Filenames are padded with the standard $A0 character

   * Supports REL and all GEOS files

   * Allows complete directory customization

   * Because it is a random-access device, it supports fast-loaders and
     random sector access

   * PC Cluster slack-space loss is minimized since the file is a larger
     fixed size

* Has a label (description) field

* Format extentible to allow for 40-track disks

* With the inclusion of error bytes, you have support for basic
  copy-protection

* Files on a disk can easily be re-written, as long as there is free
  blocks

Bad
---
* The format doesn't contain *all* the info from the 1541 disk (no sector
  header info like ID bytes, checksums). This renders some of the
  original special-loaders and copy-protection useless.

* You don't *really* know the file size of the contained C64 files in
  bytes, only blocks

* It can't store C64s FRZ files due to FRZ files needing a special flag
  that a D64 can't store. This is by no means a big problem.

* It is not an expandable filetype, like LNX or T64

* Unless most of the space on a D64 disk is used, you do end up with
  wasted space

* Directory limited to 144 files maximum

* Cannot have loadable files with the same names

* Has no recognizeable file signature (unlike most other formats). The
  only reliable way to know if a file is a D64 is by its size

* It is too easy for people to muck up the standard layout

* It is much more difficult to support fully, as you really need to
  emulate the 1541 DOS (sector interleave, REL files, GEOS VLIR files)