Formatted Disk Image (FDI) file format version 2.0 description (1st revision)

Words, 3-byte values and Double-words are always big-endian, that is most-significant byte stored at the lowest address.

1. File Header
2. Track Description
3. Types 0xDn and 0xFn: raw FM/GCR/MFM data track
4. Type 0xCn: FM/GCR raw decoded-data track
5. Type 0xEn: MFM raw decoded-data track
6. Type 0x8n-0xBn: pulses-index streams track

1. File Header

The header is at least 512 bytes in size.

Offset	Name	Size in bytes	Value	Description
+ 0	signature	27	"Formatted Disk", " Image file", 0xD,0xA	File signature
+ 27	creator	30	0x20 by default	Creator's signature. For example: "ApH's Disk2FDI PC version 0.97"
+ 57	CR	2	0xD,0xA	
+ 59	comment	80	0x1A by default	Comment describing the disk image contents
+ 139	EOF	1	0x1A	End Of File (for DOS "type" command)
+ 140	version	2	2,0	Image version number
+ 142	ltrack	2	?	Last track in image (number of tracks - 1)
+ 144	lhead	1	?	Last head in image (number of heads - 1)
+ 145	type	1	?	0=8", 1=5.25", 2=3.5", 3=3"
+ 146	rotspeed	1	?	Base rotation speed - 128 (in rotations/min)
+ 147	flags	1	0000 00??	Flags =1 => Disk is write protected =1 => Image is index-synchronized Reserved for future use
+ 148	tpi	1	?	TPI: 0=48, 1=67, 2=96, 3=100, 4=135, 5=192
+ 149	headwidth	1	?	In TPI equivalent: 0=48, 1=67, ..., 5=192
+ 150	reserved	2	0	Reserved for future use
+ 152 -XXX	tracks	2T	See section 2	Tracks description (90 double-sided cylinders reserved in first 512 bytes) cylinder-ordered, then head-ordered for each cylinder

If there are more than 180 tracks, then one or more 512-byte blocks are allocated for the extra tracks.

2. Track Description

For each word in the "tracks" field of the header:

Offset	Name	Size in bytes	Value	Description
+ 0	type	1	?	Track type:

The FDI 2.0 specification features 3 levels of data representation.
Lower levels offer a richer representation, especially for non-standard or protected tracks.

- High-level types:
  - 0x0=blank track (size field should be 0)
  - 0x1=standard Amiga double-density track
  - 0x2=standard Amiga high-density track
  - 0x3=standard ST double-density 9-sector track
  - 0x4="standard" ST double-density 10-sector track
  - 0x5=standard PC double-density 8-sector track
  - 0x6=standard PC double-density 9-sector track
  - 0x7=standard PC high-density 15-sector track
  - 0x8=standard IBM high-density 18-sector track
  - 0x9=standard IBM extra-high-density 36-sector track
  - 0xA=standard C= 1541 track
  - 0xB=standard DOS 3.2.x- disk][track
  - 0xC=standard DOS 3.3+ disk][track
  - 0xD=standard Apple GCR 3.5" track
  - 0xE=standard IBM single-density 10-sector track
  - 0xF-0x7F=reserved for future use

- Mid-level types:
  - 0xCn=FM and/or GCR raw decoded-data track
  - 0xDn=raw FM and/or GCR data track
    - For types 0xCn and 0xDn, n=bit rate:
      - 0=125Kbit/s
      - 1=150Kbit/s
      - 2=250Kbit/s
      - 3=300Kbit/s
      - 4=500Kbit/s
      - 5=Apple 3.5" GCR, tracks 48-63 (281.25Kbit/s)
      - 6=Apple 3.5" GCR, tracks 32-47 (312.5Kbit/s)
      - 7=Apple 3.5" GCR, tracks 16-31 (343.75Kbit/s)
      - 8=Apple 3.5" GCR, tracks 0-15 (375Kbit/s)
      - 9=Commodore 1541 speed zone 1 (tracks 25-30)
      - 10=Commodore 1541 speed zone 2 (tracks 18-24)
      - 11=Commodore 1541 speed zone 3 (tracks 1-17)
      - 12-14=reserved for future use
      - 15=implied bit rate

  - 0xEn=MFM raw decoded-data track, upward compatible with FDI version 1.0:
    - tracks imaged into type 0xEn version 2.0 will work with a software designed for type 0xEn version 1.0.
  - 0xFn=raw MFM data track
    - For types 0xEn and 0xFn, n=bit rate:
      - 0=125Kbit/s
      - 1=150Kbit/s
      - 2=250Kbit/s
      - 3=300Kbit/s
      - 4=500Kbit/s
      - 5=1Mbit/s
      - 6-14=reserved for future use
      - 15=implied bit rate

- Low-level type:
  0x8n-0xBn=pulses-index streams track

Offset	Name	Size in bytes	Value	Description
+ 1	size	1	(tracksize/256)	Size of track data.

Size of track data in image in multiple of 256 bytes. If the real size is not a multiple of 256 bytes, then it is increased to the next 256-byte boundary, and the added bytes are set to 0.
Exceptions:
- in the case of an Amiga double-density track (type 1):
  high 4 bits=first sector in track
  low 4 bits=size of track data in image in multiple of 512 bytes
- for types 0x8n-0xBn, the size (multiple of 256 bytes) is a 14-bit value, which 6 higher bits are coded in the 6 lower bits of the "type" byte, and the 8 lower bits are coded in the "size" byte.

3. Types 0xDn and 0xFn: raw FM/GCR/MFM data track

Offset	Name	Size in bytes	Description
+ 0	size	4	Exact size (in bits) of track until it loops
+ 4	indexpos	4	Offset in bits to the index signal from the beginning of the data
+ 8	rawdata	size/8(+1)	Raw data, FM and/or GCR encoded for type 0xDn, or MFM-encoded for type 0xFn

4. Type 0Cxh: FM/GCR raw decoded-data track

Offset	Name	Size in bytes	Description
+ 0	encoding	1	Type of encoding: 0=standard FM 1=Commodore GCR 2-0xFF=reserved for future use
+ 1	indexpos	3	Offset in bits to the index signal from the beginning of the data, re-encoded to FM/GCR
+ 4	cookedata	n	Data descriptors, see below:

Note that no Apple GCR was defined, because the self-synchronization process can lose some '0' bits, so an Apple GCR track should be imaged as a type 0xDn.

List of standard FM (type 0) descriptors:

data type description	data type byte	packed data extension: size in bytes-description of the field	unpacked data DD stands for data byte
one "0" bit	0x00	-	0
one "1" bit	0x01	-	1
deleted data FM sync	0x02	-	1*0xF56A
data FM sync	0x03	-	1*0xF56B
data FM sync	0x04	-	1*0xF56E
standard data FM sync	0x05	-	1*0xF56F
standard FM index sync	0x06	-	1*0xF77A
standard FM header sync	0x07	-	1*0xF57E
In this section, FM-decoded data is re-encoded by inserting a '1' bit before every data bit. For RLE data, a size of 0 means 256 bytes.			
RLE FM/GCR-encoded data	0x08	1-size in bytes, 1-data byte	X*DD
RLE FM-decoded data	0x09	1-size in bytes, 1-data byte	X*DD
FM/GCR-encoded data	0x0A	2-size in bits, ceil(size in bits/8)-data	X*DD
FM/GCR-encoded data	0x0B	2-(size in bits-65536), ceil(size in bits/8)-data	X*DD
FM-decoded data	0x0C	2-size in bits, ceil(size in bits/8)-data	X*DD
FM-decoded data	0x0D	2-(size in bits-65536), ceil(size in bits/8)-data	X*DD
reserved	0x0E ... 0xFE	- - -	
end of buffer	0xFF	-	

List of Commodore GCR (type 1) descriptors:

data type description	data type byte	packed data extension: size in bytes-description of the field	unpacked data DD stands for data byte
reserved	0x00	-	
reserved	0x01	-	
CBM GCR sync mark	0x02	2-size in bits	X '1' bits
reserved	0x03 ... 0x07	- - -	
In this section, every GCR-decoded nibble is re-encoded by using the standard Commodore GCR encoding table. For RLE data, a size of 0 means 256 bytes.			
RLE FM/GCR-encoded data	0x08	1-size in bytes, 1-data byte	X*DD
RLE C= GCR-decoded data	0x09	1-size in bytes, 1-data byte	X*DD
FM/GCR-encoded data	0x0A	2-size in bits, ceil(size in bits/8)-data	X*DD
FM/GCR-encoded data	0x0B	2-(size in bits-65536), ceil(size in bits/8)-data	X*DD
CBM GCR-decoded data	0x0C	2-size in nibbles, ceil(size in nibbles/2)-data	X*DD
reserved	0x0D ... 0xFE	- - -	
end of buffer	0xFF	-	

5. Type 0Exh: MFM raw decoded-data track

Offset	Name	Size in bytes	Description
+ 0	encoding	1	Type of encoding: 0=standard MFM 1-0xFF=reserved for future use
+ 1	indexpos	3	Offset in bits to the index signal from the beginning of the data, re-encoded to MFM
+ 4	cookedata	n	Data descriptors, see below:

Please note that the following descriptors have been significantly simplified over FDI version 1.0: Types 0x00 to 0x0D, and 0xFF are unchanged, while all other types have been removed.

List of standard MFM (type 0) descriptors:

data type description	data type byte	packed data extension: size in bytes-description of the field	unpacked data DD stands for data byte
In this section, MFM re-encoded data for types 0x02 and 0x03 include the first MFM synchronization bit. They also include the next MFM synchronization bit if they are immediately followed by MFM-decoded data of type 0x0C or 0x0D.			
one "0" bit	0x00	-	0
one "1" bit	0x01	-	1
standard MFM sync	0x02	-	1*0x4489
standard Index sync	0x03	-	1*0x5224
one MFM sync bit	0x04	-	1 if both neighboring bits are 0, 0 otherwise.
reserved	0x05 ... 0x07	- - -	
In this section, MFM re-encoded data do NOT include the 2 outter MFM sync bits. For RLE data, a size of 0 means 256 bytes.			
RLE MFM-encoded data	0x08	1-size in bytes, 1-data byte	X*DD
RLE MFM-decoded data	0x09	1-size in bytes, 1-data byte	X*DD
MFM-encoded data	0x0A	2-size in bits, ceil(size in bits/8)-data	X*DD
MFM-decoded data	0x0B	2-(size in bits-65536), ceil(size in bits/8)-data	X*DD
MFM-decoded data	0x0C	2-size in bits, ceil(size in bits/8)-data	X*DD
MFM-decoded data	0x0D	2-(size in bits-65536), ceil(size in bits/8)-data	X*DD
reserved	0x0E ... 0xFE	- - -	
end of buffer	0xFF	-	

6. Type 0x8n-0xBn: pulses-index streams track

Offset	Name	Size in bytes	Description
+ 0	numpulses	4	Number of pulses recorded for the track
+ 4	averagesz	3	Size in bytes of the "average" stream in the file + Compression type of this stream
+ 7	minsize	3	Size in bytes of the "minimum" stream in the file + Compression type of this stream
+ 10	maxsize	3	Size in bytes of the "maximum" stream in the file + Compression type of this stream
+ 13	indexsize	3	Size in bytes of the "index" stream in the file + Compression type of this stream
+ 16	averagedt	averagesz	"average" stream data
+16+averagesz	mindata	minsize	"minimum" stream data (optional)
+16+averagesz	maxdata	maxsize	"maximum" stream data (optional)
+minsize			
+16+averagesz	indexdata	indexsize	"index" stream data
+minsize			
+maxsize			

Each of the size fields (averagesz, minsize, maxsize and indexsize) contains the size of the corresponding data in the lower 22 bits, and the compression type in the high 2 bits.

Description of the 4 streams of data:

The track data is represented by the pulses recorded directly from the floppy drive Read Data line. Each stream is a list of sequential pulses, the last pulse directly preceding the first (data loops). Each pulse is described at the same position in each stream.

- Average stream:

When uncompressed, each pulse in this stream is a 4-byte data representing the average time elapsed since the previous strong pulse. A pulse is detected as strong or weak using the "index" stream.
Additionally, the whole track "time" can be calculated by adding all data from each strong pulse in this stream. This whole track time is guaranteed to fit in 32 bits. The real time of each pulse can then be calculated by dividing this whole time by the real track time (0.2 sec for a standard PC 3.5" disk drive or low-density 5.25" disk drive, 1/6 sec for a standard PC high-density disk drive).

- Minimum stream:

This stream is optional. When uncompressed, each pulse in this stream is a 4-byte data representing the minimum time elapsed since the previous strong pulse. It is in fact encoded as (average - real minimum), so the real value must be calculated from the stream minimum as (average - stream minimum).
When using this minimum value, keep in mind that changing the average value towards this minimum value must be done according to the minimum and maximum values from the other neighboring pulses, so that the whole track time is maintained and no minimum or maximum value is exceeded.

- Maximum stream:

This stream is optional. Still, it cannot exist without the minimum stream.
When uncompressed, each pulse in this stream is a 4-byte data representing the maximum time elapsed since the previous strong pulse. It is in fact encoded as ((average - real minimum) - (real maximum - average)), so the real value must be calculated from the stream minimum and maximum values as (average + stream minimum - stream maximum).
When using this maximum value, keep in mind that changing the average value towards this maximum value must be done according to the minimum and maximum values from the other neighboring pulses, so that the whole track time is maintained and no minimum or maximum value is exceeded.

- Index stream:

When uncompressed, each pulse in this stream is a 2-byte data representing the average status of the index signal quickly after the pulse occured.
The most significant byte is a count for the '1' state of the index signal.
The least significant byte is a count for the '0' state of the index signal.
Additionally, the corresponding pulse can be detected as strong or weak by the following algorithm:

- search in the stream for the maximum of the sum of these 2 bytes
- if the current sum of these 2 bytes = maximum sum => pulse is strong
- if the current sum of these 2 bytes < maximum sum => pulse is weak

Possible values (compression types) for the high 2 bits of averagesz, minsize, maxsize and indexsize:

0=no compression
1=Huffman-type compression
2=reserved for future use
3=reserved for future use

For type 1, a stream is encoded into one or more sub-streams as follows:

A sub-stream corresponds to a portion of the stream, from a determined low bit number to a determined high bit number of each value (for example from bit number 8 to bit number 15 of each value in the stream). If more than one sub-stream encodes the stream, the sub-streams are ordered from the higher orders first to the lower orders last. The low bit number of the last sub-stream is always 0.

Each sub-stream is encoded as follows:

First byte:
bits 0-6: low-order bit number (included) of the stream values that is encoded into the sub-stream.
bit 7:
0=decoded values must be zero-extended.
1=decoded values must be sign-extended, from bit 7 or bit 15.

Second byte:
bits 0-6: high-order bit number (included) of the stream values that is encoded into the sub-stream.
bit 7:
0=8-bit values in the tree.
1=16-bit values in the tree.

Following bytes: Huffman tree, encoded as bits. Bit 7 to bit 0 of the first byte, then bit 7 to bit 0 of the next byte, etc.
For each bit:
0=node has 2 branches.
1=node is a leaf.
The bit following a 0 bit corresponds to the left branch.
The bit following a 1 bit corresponds to the deepest available right branch.
When no right branch is available after a 1 bit, then the tree is complete, and the tree leaf values begin at the byte following immediately the byte containing the last 1 bit of the tree. The tree values are 8 or 16-bit values (depending on bit 7 of the second byte of the sub-stream) that correspond to a path in the Huffman tree. The values are stored in the same order as the 1 bits are found in the encoded Huffman tree.

The packed data immediately follow the tree leaf values, a 0 bit meaning a left branch in the tree, and a 1 bit meaning a right branch. The bits are stored in bits 7 to 0 of the first byte, then bits 7 to 0 of the second byte, etc. When a leaf is reached, then the corresponding tree leaf value must be zero or sign-extended, then output to the appropriate bits of the decoded stream. The number of values to decode corresponds to the number of pulses recorded for the track.

---- End of Formatted Disk Image (FDI) file format version 2.0 description ----

Vincent "ApH" Joguin.