Common File Formats for Emulated TRS-80 Floppy Disks

Tim Mann
http://tim-mann.org
Last revised 4 Jul 2008

Most current TRS-80 Model I/III/4 emulators use one of three common file formats for emulated floppy disk media. The extension .DSK is usually used for all three formats, and some emulators transparently support two or three of them, while others support only one. The two most common formats both originated with emulators written by Jeff Vavasour, while the third originated with an emulator written by David Keil, so I'll take the liberty of giving the formats names that use their initials.

- The JV1 format originated in Jeff's Model I emulator for MS-DOS. It's a very simple, limited format that can represent only single-density media with 256 byte sectors and a Model I directory on track 17.
- The JV3 format originated in Jeff's Model III/4 emulator for MS-DOS. It is much more flexible than JV1, but still cannot represent everything the real hardware can do, so some copy-protected disks cannot be represented. The format has been extended to support 8-inch floppies and sector sizes of other than 256 bytes, but not all emulators support these extensions. I indicate below which features are extensions and which emulators are known to support which extensions.
- The DMK format originated in David's Model III/4 emulator for Windows. Unlike JV1 and JV3, DMK is able to represent essentially everything that a real TRS-80 floppy disk controller could do, including copy protected TRS-80 disks.

This document describes the JV1 and JV3 formats in complete detail and indicates (where known) which emulators support which. DMK is quite different from JV1 and JV3 and is not described in this document, but there is a description on David Keil's Web page. You probably don't need any of this information unless you are writing an emulator or working with unusual disks.

The JV1 Format

The JV1 format is simply an array of 256-byte sectors stored in a file. Byte 0 of the file is byte 0 of track 0, sector 0; byte 256 is byte 0 of track 0, sector 1, and so forth. There are 10 sectors per track (i.e., single density), numbered 0 through 9, and only one side. Tracks are numbered starting at 0. All sectors on track 17 are formatted with the nonstandard data address mark 0xFA, indicating a TRSDOS 2.3 directory. All other sectors are formatted with the standard data address mark 0xFB.

Note: In emulations of the WD1791/3 floppy disk controller used in the Model III and 4, it is best to present track 17 as being formatted with the standard deleted data address mark 0xF8. The WD1791/3 cannot write 0xFA, and upon reading, returns it as 0xFB. So Model III/4 operating systems use 0xF8 on the directory track instead.

The JV3 Format

The JV3 format consists of a fixed-size array of *sector headers*, followed by an area containing the data for each sector. As an extension to allow for more sectors, a second block of sector headers and data can follow. Here is the format in pseudo-C notation. The length of each data area depends on the content of the headers, as described below.

```
typedef struct {
  SectorHeader headers1[2901];
  unsigned char writeprot;
  unsigned char data1[];
```

```
  SectorHeader headers2[2901];
  unsigned char padding;
  unsigned char data2[];
} JV3;
```

Write Protect and Padding

The field *writeprot* should normally contain 0xFF. As an extension, a value of 0x00 in his field indicates that the disk is write-protected (i.e., it is not writable).

The field *padding* is currently unused. It should contain 0xFF.

The Sector Header

Each sector in a JV3 file is described by a three-byte header. The first byte gives the sector's track number, the second gives its sector number on the track, and the third is a set of flags.

```
typedef struct {
  unsigned char track;
  unsigned char sector;
  unsigned char flags;
} SectorHeader;

#define JV3_DENSITY     0x80  /* 1=dden, 0=sden */
#define JV3_DAM         0x60  /* data address mark code; see below */
#define JV3_SIDE        0x10  /* 0=side 0, 1=side 1 */
#define JV3_ERROR       0x08  /* 0=ok, 1=CRC error */
#define JV3_NONIBM      0x04  /* 0=normal, 1=short */
#define JV3_SIZE        0x03  /* in used sectors: 0=256,1=128,2=1024,3=512
                                 in free sectors: 0=512,1=1024,2=128,3=256 */

#define JV3_FREE        0xFF  /* in track and sector fields of free sectors */
#define JV3_FREEF       0xFC  /* in flags field, or'd with size code */
```

The *track* field gives both the physical track number on which the sector lies and the logical track number that is formatted in the sector's ID. Numbering starts from 0. Thus it is not possible to represent a copy-protected disk where sectors were deliberately formatted with incorrect track numbers. The format allows for 255 tracks (numbered 0 through 0xFE), but emulators may impose lower limits. You can expect at least 80 tracks to be supported; xtrs currently allows up to 96.

The *sector* field gives the logical sector number that is formatted into the sector's ID field. The physical order of sectors on the track is not explicitly represented, but xtrs (and perhaps other emulators) present them in the order they are recorded in the JV3 file; thus when a TRS-80 program formats a track with interleave and reads back the sector ids, they come back in the same interleaved order.

The *flags* field packs in a lot of information:

- If JV3_DENSITY is set, the sector was formatted in double density (MFM); if not, it was formatted in single density (FM).
- The 2-bit JV3_DAM field encodes the sector's data address mark. Here are the basic meanings of the codes:

JV3_DAM value	Single density	Double density
0x00	0xFB (Normal)	0xFB (Normal)
0x20	0xFA (User-defined)	0xF8 (Deleted)
0x40	0xF9 (User-defined)	Invalid; unused
0x60	0xF8 (Deleted)	Invalid; unused

The treatment of single density data address marks by the Western Digital 1771 and 179x controllers used in original TRS-80 hardware is rather inconvenient. The WD1771 (used in the TRS-80 Model I) is capable of writing any of the four DAMs shown above, and can distinguish amongst all four on reading. The WD179x (used in the Model III and 4), however, can write only 0xFB or 0xF8, and on reading, it cannot distinguish between 0xFB and 0xFA, or between 0xF8 and 0xF9.

TRS-80 operating systems differentiate directory sectors from ordinary data sectors by giving them different data address marks. Unfortunately, Model I TRSDOS uses 0xFA on directory sectors, which a WD179x cannot write and cannot distinguish from the 0xFB used on data sectors. Other Model I operating systems (such as LDOS) typically write 0xF8 on directory sectors but accept either 0xFA or 0xF8 when reading them, allowing them to read TRSDOS disks. Compatibility problems remain when attempting to use Model I TRSDOS disks on a Model III or 4, and when attempting to use LDOS disks with Model I TRSDOS.

An emulator can easily paper over these compatibility problems by behaving differently from the real hardware. All currently known emulators do so, making (at least) the following changes to strictly correct behavior: (1) If TRS-80 software attempts to write or format a 0xF8 DAM in single density, it is instead recorded in the JV3 header as 0xFA. (2) If TRS-80 software reads a single density sector that has the 0xFA DAM using an emulated WD179x, it is returned as 0xF8 instead. These changes hide all the DOS compatibility problems described in the previous paragraph.

It might be worthwhile to make precisely correct behavior available as a run-time emulator option, as this might allow more types of "protected" self-booting TRS-80 disks to work correctly. This option is available in xtrs version 3.2 and later, but not in any other known emulators.

The following tables summarize the DAM behavior of both the WD1771 and the WD179x. On the WD1771, read status bits 6,5 are used to report the DAM of the sector just read, while write command bits 1,0 select the DAM to write; on the WD179x, only read status bit 5 and write command bit 0 are significant: read status bit 6 is always 0, while write command bit 1 is used for a function unrelated to DAM selection. The behavior of these bits on the actual hardware is shown first in each cell; the modifications typically made by emulators follow in parentheses. It should be emphasized that WD1771 behavior shown here correctly describes the actual hardware, even though it contradicts the Western Digital data sheet; the data sheet erroneously transposes the column headings for status bits 6 and 5. As an aside, it seems likely that the engineers who designed the WD179x were misled by this error, as the value that the WD179x returns in bit 5 is the value that the WD1771 was documented as returning in bit 5, but that it actually returns in bit 6.

Read status bits 6,5:

DAM	WD1771	WD179x single	WD179x double
0xFB	0,0	0,0	0,0
0xFA	0,1	0,0 (0,1)	impossible
0xF9	1,0	0,1	impossible
0xF8	1,1	0,1	0,1

Write command bits 1,0:

Bits	WD1771	WD179x single	WD179x double
0,0	0xFB	0xFB	0xFB
0,1	0xFA	0xF8 (0xFA)	0xF8
1,0	0xF9	0xFB	0xFB
1,1	0xF8 (0xFA)	0xF8 (0xFA)	0xF8

- JV3_SIDE reflects both the physical side on which the sector is recorded and the side number formatted in the sector's ID. Thus it is not possible to represent a copy-protected disk where sectors were formatted with incorrect side numbers or with values other than 0 or 1 in the sector ID's 8-bit side field.

- JV3_ERROR, if set, indicates that the sector should show a data CRC error when read. The actual value of the two CRC bytes is not represented.

- JV3_NONIBM is an xtrs-specific extension that partially supports the WD1771's feature of "non-IBM" sector sizes. Only a small subset of the functionality is supported, with some special kludges specifically to make the VTOS 3.0 copy protection scheme work. It is probably best to treat this bit as a "must be zero" field in other emulators; see the xtrs source code if you are really sure you want to know how it works, and keep your barf bag handy.

- JV3_SIZE is an extension to support sector sizes of other than 256 bytes. This field is encoded differently depending on whether the sector is in use or free (not in use):

Size	IBM size code in sector ID	JV3_SIZE field value if in use	JV3_SIZE field value if free
128	00	1	2
256	01	0	3
512	02	3	0
1024	03	2	1

  Thus, if a sector is in use, xor'ing its JV3_SIZE field with 1 gives the IBM size code that appears in its sector ID size field. If a sector is free, xor'ing its JV3_SIZE field with 2 gives its IBM size code. This representation was chosen because the original JV3 format supports only 256-byte sectors, always places zeros in the JV3_SIZE field for sectors that are in use, and always places ones in the field for sectors that are free.

If a sector header is not in use (free), its track and sector fields must contain 0xFF. Its flags field must contain 0xFC plus the appropriate free JV3_SIZE field value given above.

The Data Blocks

For each sector header, there is one data block. The blocks are placed in the same order as the headers that describe them, and are tightly packed. Thus to find the data block for the *n*th header, you need to know the total size of data blocks 0 to *n* - 1. A sector header marked as free *does* have a corresponding data block, of the size indicated in its size field. However, the file can (and in fact should; see below) end immediately after the last in-use data block.

The Second Header Block

The second header block is present if and only if the file is long enough to contain it. It starts immediately after the last data block described by the first header block. Thus to find the second header block, you need to know the total size of all data blocks described by the first header block.

It's obvious, by the way, how to extend the format to more than two header blocks, but no known emulators support more than two. Having more than two is not useful to describe any floppy format that was supported by any real TRS-80. One block is sufficient for 5.25-inch DD drives, and two are sufficient for 8-inch drives, 5.25-inch HD drives, or 3.5-inch HD drives.

Allocating and Freeing Sectors

Allocating and freeing sectors is a little tricky (if you support the extension to sizes other than 256 bytes). When a new emulated disk is first created, it should have one header block, filled with free 256-byte sectors. When a new sector is formatted, if a free sector of the correct size exists, it can be reused; otherwise, the next sector after the highest in-use sector can be chosen and changed to the correct size. In the latter case, since there is no data after the sector that is being resized, resizing it does not require anything to move. When a sector is erased (because the track it is on is being reformatted), it should normally be freed without changing its size, so that the data of later sectors need not be moved.

If the highest in-use sector is freed, the emulator should search backward for the new highest in-use sector, and should truncate the file to eliminate any sector data for trailing free sectors, as well as the second header block if it is no longer needed. Why do this? If the user reformats some sectors to be shorter or longer than they used to be, the nominal start of the second header block (as defined above) will shift. If you have garbage at the end of the file, and the block shifts to start somewhere in the garbage, then when you read the file back in again later, you'll have a second header block full of garbage that will confuse you.

Support

Here is a summary of which emulators support which formats and which features. The data is believed current as of this document's revision date. Future versions of some of these emulators may support more (or fewer!) features. If an emulator does not support a feature, it will still work with disks created by emulators that do support the feature, as long as those disks didn't require that feature when they were created. For example, an emulator that supports the second header block will not create a second header block unless you format more that 2901 sectors on one disk; note that an 80 track, double sided, double density 5.25-inch disk contains less than 2901 sectors.

	JV1	JV3 Basic features	JV3 Write protect	JV3 Second header block	JV3 Sizes other than 256	JV3 Non-IBM kludge	DMK
Jeff Vavasour Model I v3.02u	Yes	No	No	No	No	No	No
Jeff Vavasour Model III/4 v2.3	No	Yes	No (future)	No	No	No	No
Matthew Reed Model I/III v1.10	Yes	Yes	No (future)	No (future)	No	No	No
Matthew Reed Model 4 v1.01	Yes	Yes	Yes	Yes	No	No	No
Matthew Reed TRS32 1.0	Yes	Yes	Yes	Yes	Yes	Yes	No
xtrs Model I/III/4 v3.7	Yes	Yes	Yes	Yes	Yes	Yes	Yes
WinTRS80 Model I/III/4 v1.02	Yes	Yes	Respected but not generated	Yes	No	No	No
Yves Lempereur Model I	Yes	No	No	No	No	No	No
David Keil Model III/4	Yes	Yes, with some limitations	Respected but not generated	No	Yes	No	Yes

Additional information or corrections for this table are welcome.

Back to the TRS-80 Page | My home page