

The T88 Tape Format

Variable-Length Records on a Fixed-Block Tape Drive

Large Scale Systems Museum

September 2026

Contents

1	Introduction	2
2	Terminology	2
3	The .tap container	3
4	Storing a T88 tape in a .tap file	3
5	Block groups	4
6	The label block	4
6.1	Layout	4
6.2	Entry values	5
6.3	Checksum	5
7	Encoding records	6
7.1	A single record	6
7.2	Several records in one group	6
7.3	Multi-block records and file marks	6
8	Group size and the T88 firmware	6
8.1	The fixed read size	6
8.2	Records that span groups	8
9	Reading a T88 tape	8
9.1	Finding the label	8

9.2	Decoding the entries	9
9.3	Errors	9
10	Limitations	10

1 Introduction

The Systems Integration Corporation/MODCOMP T88xx (“T88”) mass storage controller contains a tape drive that supports only fixed-length blocks. Its host, however, expects a tape drive that reads and writes records of any length. The T88 bridges the gap by encoding the host’s variable-length records into the fixed-length blocks that its drive can store, and decoding them again on the way back.

This document describes that encoding: the bytes as they exist on the fixed-block medium. It also describes how the encoding is stored in a .tap tape image file, which is how `tapt88` and tape drive emulators exchange it. Figure 1 shows where each piece fits.

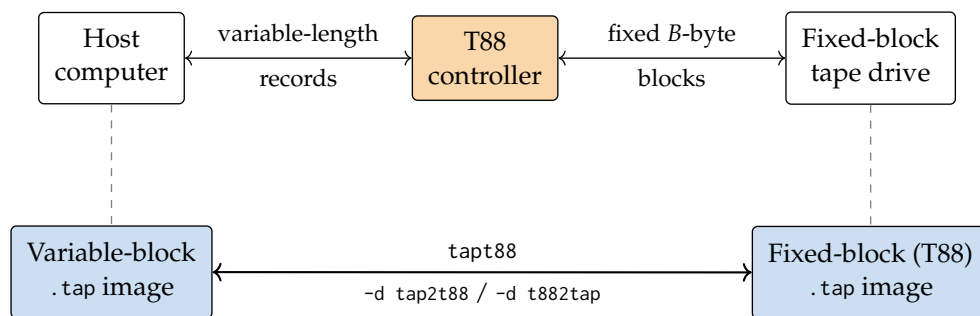


Figure 1: The T88 presents a variable-block tape drive to its host, using a fixed-block drive internally. `tapt88` converts between image files of what the host sees and what the drive stores.

2 Terminology

Block size (*B*)

The fixed size, in bytes, of every physical block on the T88’s tape drive. It is a configuration constant, typically 512 or 1024, and is not recorded anywhere on the tape. A tape must be read with the same *B* that was used to write it.

Record

A variable-length unit of data, as written or read by the host. Preserving records exactly is the reason the format exists.

File mark

A logical end-of-file marker (also called a tape mark) written by the host between files.

Data block

A physical block holding part or all of one record’s data, or standing in for a file mark.

Label block

A physical block that describes the data blocks preceding it.

Block group

A run of one or more data blocks followed by exactly one label block. A T88 tape is a sequence of block groups.

3 The .tap container

Both kinds of image that `tapt88` handles are `.tap` files: a sequence of length-framed records. Each record is stored as a 32-bit length, the data, a pad byte if the length is odd, and the length again, as shown in Figure 2. The lengths are *little-endian*; the pad byte is not counted in the length.

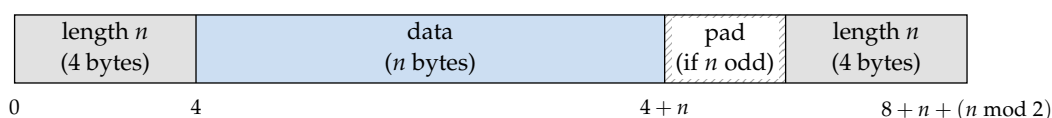


Figure 2: A record in a `.tap` file. Offsets are in bytes.

Three reserved values may appear in place of a length. They stand alone, with no data or trailing length:

Value	Meaning
0x00000000	tape mark (file mark)
0xfffffffffe	erase gap; carries no information
0xffffffffff	end of medium

Some `.tap` files are several tape images concatenated back to back, each ending in its own end-of-medium marker. `tapt88` reads through embedded end-of-medium markers to the physical end of the file, and writes a single end-of-medium marker at the end of its output.

In a *variable-block* image, each `.tap` record is one host record, whatever its length, and each tape mark is one file mark.

4 Storing a T88 tape in a .tap file

A *fixed-block* (T88) image uses the same container, but each `.tap` record is one physical block from the T88's drive, so every record's length is exactly B . Data blocks and label blocks look the same at this level; only the label's contents tell them apart (Section 9.1). Figure 3 shows the layering.

A `.tap` tape mark is never used inside a T88 image; file marks are recorded in label blocks instead (Section 6.2). `tapt88` treats a tape mark in a T88 image as a sign of a corrupt or foreign file. Erase gaps and end-of-medium markers are skipped, as in a variable-block image.

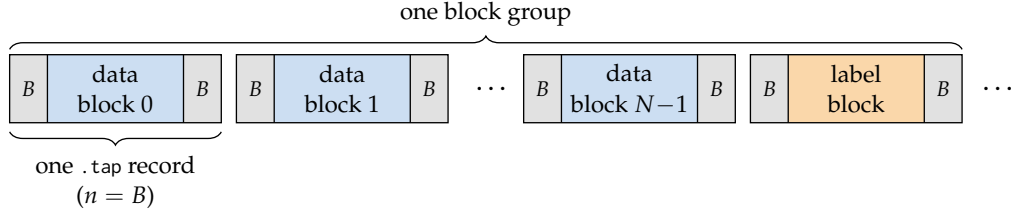


Figure 3: A T88 image: every physical block, data or label, is its own .tap record of length B .

5 Block groups

The content of a T88 tape is a sequence of block groups. Each group is one or more data blocks followed by exactly one label block, all B bytes long (Figure 4). The label describes every record and file mark in the group's data blocks; there is no length or marker anywhere else on the tape.

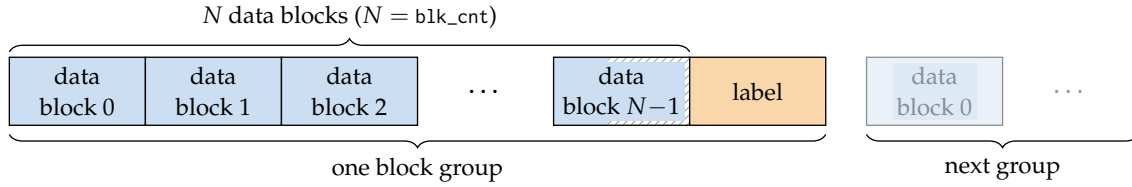


Figure 4: A block group. Shaded areas hold record data; hatched areas are padding.

A group may hold one record, several records, parts of records, file marks, or a mix. These rules always hold:

- Every record starts at the beginning of a data block. A data block never holds bytes from two different records.
- Every data block of a record except the last is completely full. The last block holds between 1 and B bytes of the record, and the rest of it is padding.
- A file mark takes up one data block of its own, which holds no data.
- A group has at most 254 data blocks, the number of entries in the label.

6 The label block

6.1 Layout

The label has 512 bytes of defined content, whatever the block size. When $B > 512$, the rest of the block is padding, written as zeros. All fields are 16-bit *big-endian* integers, unlike the little-endian lengths of the .tap container. Figure 5 shows the layout.

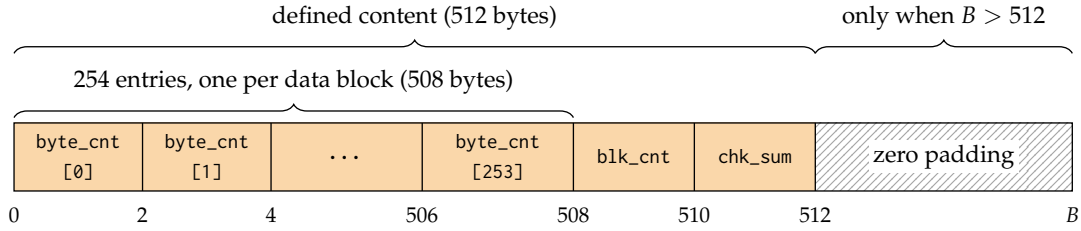


Figure 5: The label block. Offsets are in bytes.

Offset	Field	Contents
0	byte_cnt[0..253]	One entry per data block in the group; byte_cnt[<i>i</i>] describes data block <i>i</i> . Entries at and beyond blk_cnt are zero.
508	blk_cnt	The number of data blocks in the group, and so the number of valid byte_cnt entries.
510	chk_sum	A checksum over the label (Section 6.3).

6.2 Entry values

Each valid byte_cnt entry says what its data block holds:

Value	Meaning
0x0000	The whole block belongs to the record in progress, which continues in the next block.
0x0001–0xffffd	The block is the last one of the record in progress, and holds this many bytes of it. The record ends here.
0xffff (EMPTY)	The record in progress ended at the end of the previous block. This block holds no data.
0xfefe (FILE_ID)	A file mark. This block holds no data. No record may be in progress.

Since a record’s last block holds at most B bytes, and B is far smaller than 0xfefe, the length values never collide with the two sentinels.

6.3 Checksum

chk_sum is a 16-bit sum, seeded with 1, of all 254 byte_cnt entries (valid or not) and blk_cnt, wrapping modulo 65536. It does not cover itself, the padding beyond byte 512, or any data block.

```

chk_sum := 1
for i in 0 .. 253:
    chk_sum := (chk_sum + byte_cnt[i]) mod 65536
chk_sum := (chk_sum + blk_cnt) mod 65536

```

This matches the T88 firmware’s own routine, ts_gchk().

7 Encoding records

7.1 A single record

Figure 6 shows the simplest case: one 160-byte record at $B = 512$. It needs one data block, whose entry is the record's length, followed by a label. Two 512-byte blocks hold 160 bytes of data.



Figure 6: One 160-byte record, $B = 512$. The `byte_cnt` entry for each data block is shown in hex beneath it.

7.2 Several records in one group

A group may hold several records. Each non-zero entry closes out one record, and the next record starts in the next block. Figure 7 shows records of 160, 156 and 168 bytes in one group: four blocks instead of the six that three single-record groups would take.

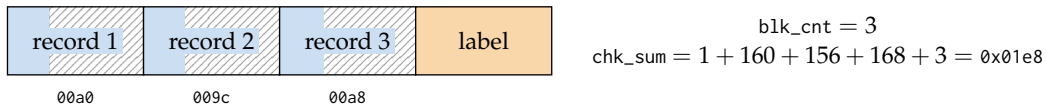


Figure 7: Three records batched into one group, $B = 512$.

7.3 Multi-block records and file marks

A record longer than one block uses `0x0000` for every block but its last. A file mark uses a data block of its own, marked `0xfefe`, and holding no data. Figure 8(a) shows a 1024-byte record, a file mark, and a 40-byte record, as `tapt88` writes them.

A record whose length is an exact multiple of B can be closed in either of two ways. Its last block can carry the entry B , as in Figure 8(a). Or that block can carry `0x0000`, and an extra block with no data carries `0xffff` (EMPTY), as in Figure 8(b). A reader must accept both. `tapt88` writes the first.

A zero-length record is a single EMPTY block with no record in progress. `tapt88` never writes one, because a `.tap` file can't hold a zero-length record (Section 10).

8 Group size and the T88 firmware

8.1 The fixed read size

The format itself lets a writer end a group anywhere, but the T88 firmware does not. It reads the tape through a FIFO buffer of `FIFO_SIZ = 65536` bytes, and every physical read it issues asks for

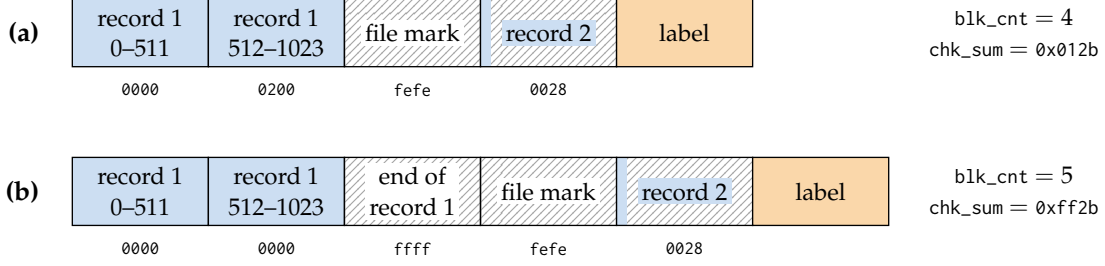


Figure 8: A 1024-byte record, a file mark and a 40-byte record, $B = 512$. (a) The last block of record 1 carries its length. (b) The same content with record 1 closed by an EMPTY entry instead.

the same number of blocks,

$$\frac{\text{FIFO_SIZ} - B}{B},$$

whatever was actually written. It expects the last block of each read to be the group's label. So on a tape that the T88 will read, every group except the last must be exactly that many blocks long, including the label (Figure 9). The last group may be shorter.

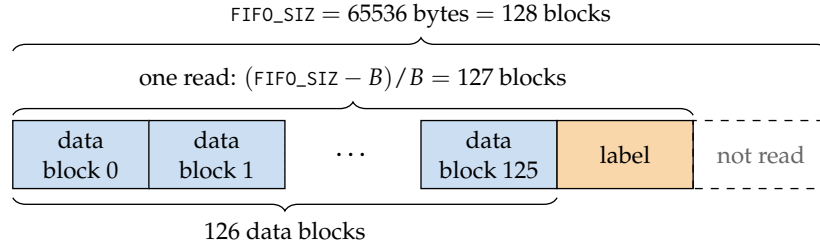


Figure 9: One firmware read at $B = 512$. A full group is 126 data blocks and its label.

The firmware's write path produces exactly this layout. When it opens a fresh write buffer, `ts_gettwr()` sets aside room for the label before any data goes in, and it flushes the group as soon as the data blocks fill the rest. `tapt88` does the same, so its output can be read by a real T88. Table 1 gives the group size for common block sizes. Because B is at least 512, a full group never reaches the 254-block limit of the label.

B	Blocks per read	Data blocks per full group	Data bytes per full group
512	127	126	64 512
1024	63	62	63 488
2048	31	30	61 440
4096	15	14	57 344
8192	7	6	49 152
16384	3	2	32 768

Table 1: Full group sizes. The largest block size that leaves room for even one data block is 21 845 bytes.

8.2 Records that span groups

Records and file marks are packed into groups back to back. When a record doesn't fit in the rest of the current group, the group is closed after as many of the record's blocks as fit, and the rest of the record continues at the start of the next group. Nothing marks the split except that the first group's entries run out while the record is still in progress: its last entries are all $0x0000$. A reader must carry that record over into the next group.

Figure 10 shows eight 10 240-byte records at $B = 512$. Each record is exactly 20 blocks, so the first six records fill 120 of the first group's 126 data blocks. Record 7 gets the remaining six blocks and continues for 14 more in the second group. These are the labels `tapt88` writes for this input.

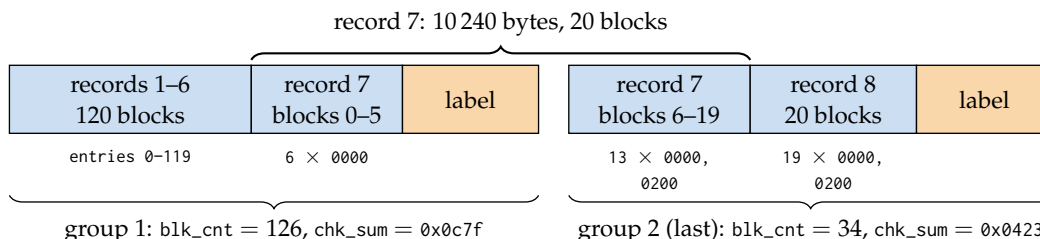


Figure 10: Record 7 spans two groups, $B = 512$. In group 1, each of records 1–6 ends with a $0x0200$ entry (at indices 19, 39, ..., 119), and record 7's six blocks are all $0x0000$.

Group 1's checksum is $1 + 6 \times 512 + 126 = 3199 = 0x0c7f$, and group 2's is $1 + 2 \times 512 + 34 = 1059 = 0x0423$.

9 Reading a T88 tape

9.1 Finding the label

A reader can't tell a label from a data block by where it is, since groups vary in length, or by its contents alone, since a data block can hold anything. Instead it reads blocks one at a time, and tests each one as a possible label for the blocks read so far (Figure 11). A block is the label when both of these are true:

1. its `blk_cnt` equals the number of data blocks read so far in this group; and
2. its `chk_sum` matches the checksum computed from its own `byte_cnt` entries and `blk_cnt`.

Neither test is reliable alone, but a data block that passes both by accident is very unlikely. If 254 data blocks go by with no label, the tape is corrupt or isn't in this format.

If the file ends between groups, the tape is complete. If it ends partway through a group, before a label is found, the tape is truncated.

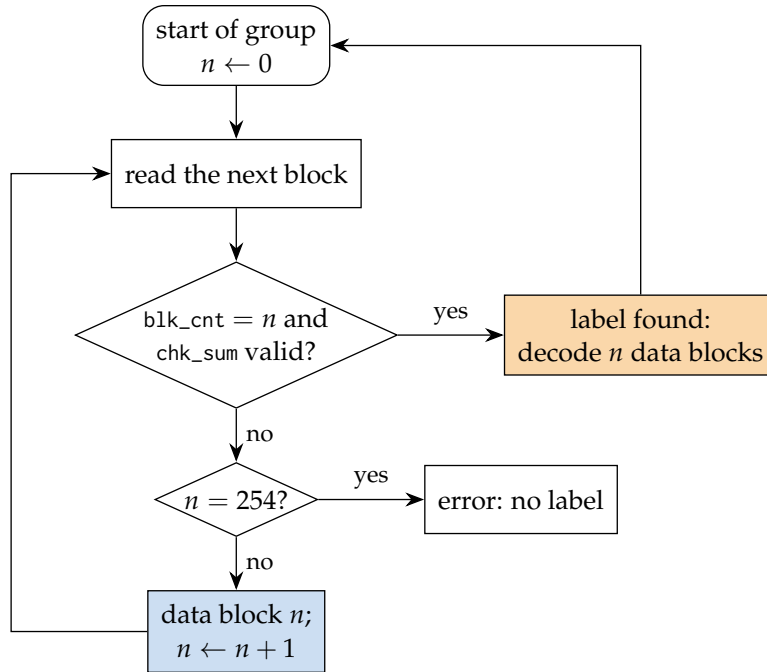


Figure 11: Finding the end of a block group.

9.2 Decoding the entries

With the label found, the reader walks `byte_cnt[0]` through `byte_cnt[blk_cnt−1]` in order, keeping one record in progress, which starts out empty:

- `0x0000`: append all B bytes of the data block to the record in progress.
- `0x0001–0xffffd`: append that many bytes from the start of the data block, and emit the record.
- `0xffff`: emit the record in progress as it stands, which may be zero-length. Ignore the data block.
- `0xfefe`: emit a file mark. If a record is in progress, the tape is corrupt. Ignore the data block.

When the entries run out with a record still in progress, the record carries over into the next group. A group whose label has `blk_cnt = 0`, with no record carried over, holds a single zero-length record.

9.3 Errors

The format’s only protection is the label checksum, and a reader must not try to work around a failure. An undetected wrong boundary would silently corrupt every record after it. So `tapt88` stops at the first group it can’t decode, keeping the records it has already written, and exits with status 2. The same happens if the tape ends while a record is still in progress, except that `tapt88` writes out the incomplete record first, with a warning.

10 Limitations

- Only the label is checksummed. Corruption within a data block isn't detected.
- The block size isn't recorded on the tape. Reading a tape with the wrong B fails, usually because the .tap record lengths don't match B .
- In a .tap file, a zero-length record and a tape mark are both stored as a 32-bit zero, so they can't be told apart. A zero-length record read from a T88 tape becomes, in effect, a second file mark in a variable-block image.