TBM File Format Specification: Major Revision 0

Table of Contents

Introduction

This document is a formal specification of the file format for TrackerBoy module files, or *.tbm files. This document is for major revision 0 of the file format.

A Module file is a serialized form of the Module object.

The libtrackerboy library provides a reference implementation for serializing and deserialing module files with this format. See the io module for more documentation.

NOTE	All multi-byte fields in this specification are stored in little-endian byte order.

Definitions

Module

A container of songs, instruments and waveforms. Instruments and waveforms are shared between all songs. A module can store up to 256 songs, 64 instruments and 64 waveforms.

Order

Song order data, a list of OrderRows that determine each pattern in the song.

OrderRow

A track id assignment for each channel. Determines the tracks which make a pattern.

Pattern

A collection of Tracks, one for each channel.

Track

Song data for a single channel. A Track is a list of TrackRows.

File extensions

It is recommended that the following file extensions are used for files adhering to this specification:

***.tbm**

TrackerBoy Module file

Basic Types

Below are basic data types used throughout the format

Type name	Size (bytes)	Description
Uint8	1	Unsigned 8-bit integer (0-255)
BiasedUint8	1	Unsigned 8-bit integer, biased form (1-256)
Char	1	8-bit Character type
Bool	1	Boolean as an uint8 (0 for false, nonzero for true)
Uint16	2	Unsigned 16-bit integer, in little endian (0-65536)
Uint32	4	Unsigned 32-bit integer, in little endian
LString	1 + *length*	Length-prefixed UTF-8 string, string data prefixed by a Uint8 *length*

Structure

A TrackerBoy module consists of a Header and a Payload.

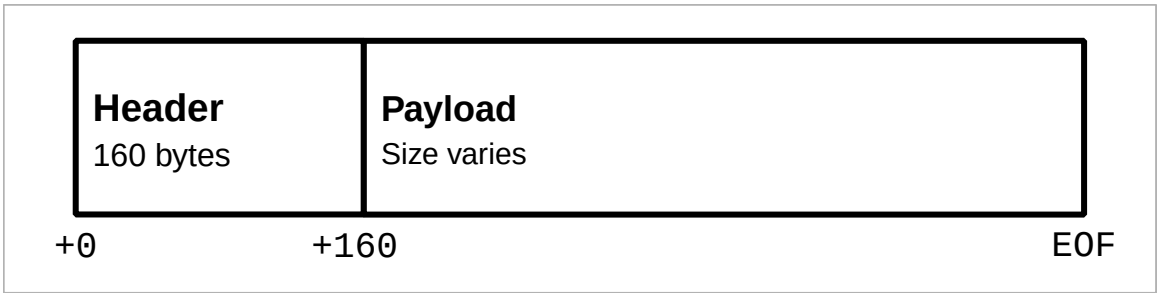

Figure 1. TBM File structure

Header

The header structure provides valuable information about the data stored in the payload, as well as versioning information.

The table below defines the layout of the Header structure. Reminder that all multi-byte fields are stored in little-endian. The size of the header is a fixed 160 bytes, with any unused space marked as reserved. Reserved fields can be utilized for future revisions of the format.

The layout of the header depends on the header revision, located in offset 24. The following table is for the current revision of the specification.

Table 1. TBM Header Structure

Offset	Size	Type	Field name
+0	12	Char[12]	signature
+12	4	Uint32	versionMajor
+16	4	Uint32	versionMinor
+20	4	Uint32	versionPatch
+24	1	Uint8	rev
+25	1	Uint8	system
+26	2	Uint16	customFramerate
+28	32	Char[32]	title
+60	32	Char[32]	artist
+92	32	Char[32]	copyright
+124	2	Uint16	numberOfInstruments
+126	2	Uint16	numberOfWaveforms
+128	32	Char[32]	reserved

Signature

Every TrackerBoy file begins with this signature:

Table 2. TBM Signature format

+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11
`'\0'`	`'T'`	`'R'`	`'A'`	`'C'`	`'K'`	`'E'`	`'R'`	`'B'`	`'O'`	`'Y'`	`'\0'`

in order to identify the file as a TrackerBoy file.

Version (major, minor, patch)

Version information is stored as three 4-byte words. This information determines which version of trackerboy that created the file. Versioning is maintained by keeping a major and minor version, followed by a patch number. For example, if the trackerboy version is v1.0.2, then the header's version fields will contain `0x1` `0x0` and `0x2` for major, minor and patch, respectively.

Revision

The `rev` field marks the file format revision. Starts at 0 and is incremented whenever there is a change to the format. TrackerBoy is backwards-compatible with files of this specification. In other words, TrackerBoy can understand any module whose `rev` field is the same or less than the current revision.

System

The system field determines which Game Boy model this module is for. Since the driver is typically updated every vblank, the system field determines the framerate, tick rate or vblank interval for the driver. The available choices are listed in the following table:

Table 3. Valid system values

Identifier	Value	System name	Tick rate
systemDmg	0	DMG	59.7 Hz
systemSgb	1	SGB	61.1 Hz
systemCustom	2	N/A	varies

If the system is `systemCustom`, then a custom tick rate is used instead of the system's vblank. The custom tick rate is stored in the `customFramerate` field of the header. This custom tick rate must be a positive number and nonzero. The implementation should default to 30 fps when the custom tick rate does not meet this criteria.

If the system does not match any of these values, then the implementation should treat the system as the default, `systemDmg`.

Author information (title, artist, copyright)

These fields in the header are fixed 32 byte strings. Assume ASCII encoding. Any unused characters in the string should be set to 0 or `\0`. Since these strings are fixed, null-termination is not needed.

NOTE	The size and naming of these strings are identical to the ones in *.gbs file format. This is intentional, as exporting to gbs is a planned feature.

Instrument and Waveform counts

The `numberOfInstruments` and `numberOfWaveforms` fields determine how many instruments and waveforms are stored in the payload, respectively. Both of these fields should be a number in range of 0 to 64.

Reserved

The remainder of the header is the `reserved` field(s). This is an unused section that may be utilized in future versions of the format if needed.

This field **should** be zeroed but the specification does not require it. The field can be safely ignored when processing module files.

Payload

The payload is located right after the header (offset 160). It contains a fixed number of "blocks" or tagged data with a size.

Blocks

A block in the payload contains three parts: the id, the length and the data. The format of the block is shown below:

Table 4. Format of a payload block.

Offset	Size	Description
0	4	Identifier
4	4	Length
8	*Length*	Data

Block types

Each block has an identifier, which determines the type of data present in the block. The table below lists all recognized identifiers in the payload.

Table 5. Block types used in TBM files

Identifier	Value (Uint32)	Description
"INDX"	0x58444E49	Index block, contains ids and names for instruments and waveforms.
"COMM"	0x4D4D4F43	User set comment data for a module
"SONG"	0x474E4F53	Container for the module’s song.
"INST"	0x54534E49	Container for all instruments in the module
"WAVE"	0x45564157	Container for all waveforms in the module

Thus there are 5 blocks total in the payload, in the following order: INDX, COMM, SONG, INST, WAVE.

INDX block format

The INDX block contains the ID and name data for all instruments and waveforms in the module. This block is composed as an array of `Index` records, which is a pair of a `Uint8` ID and an `LString` name. The total number of records in this block is the sum of the `numberOfInstruments` and `numberOfWaveforms` fields from the Header.

Instrument index records are stored first, followed by waveform index records.

COMM block format

The COMM block just contains a UTF-8 string that is the user's comment data. The string is not null-terminated since the length of the string is the length of the block. If the user has no comment set, then this block is empty (length = 0).

SONG block format

The SONG block contains the data for module's song.

Song data is composed of the following, in this order:

1. The name, `LString`
2. A `SongFormat` record
3. The song order, as an array of `OrderRow` or `Char[4][patternCount]`
4. The track data, as a sequence of `TrackFormat` and `RowFormat` records

Song name

The first part of a SONG block is the song's name, as an `LString`.

SongFormat

Following the name is a `SongFormat` record:

Table 6. The SongFormat record

Offset	Size	Type	Field name
+0	1	BiasedUint8	rowsPerBeat
+1	1	BiasedUint8	rowsPerMeasure
+2	1	Uint8	speed
+3	1	BiasedUint8	patternCount
+4	1	BiasedUint8	rowsPerTrack
+5	2	Uint16	numberOfTracks

rowsPerBeat

number of rows that make up a beat, used by the front end for highlighting and tempo calculation.

rowsPerMeasure

number of rows that make up a measure, used by the front end for highlighting.

speed

Initial speed setting for the song in Q4.4 format

patternCount

number of patterns for the song

rowsPerTrack

the size, in rows, of a track (all tracks have the same size).

numberOfTracks

number of tracks stored in this song block.

Song Order

Next is the song order, an array of `OrderRow` records with the dimension being the patternCount field from the song format record. An `OrderRow` record is a set of 4 `Uint8` track ids, with the first being the track id for channel 1 and the last being the id for channel 4.

Track Data

Finally, the rest of the block contains the pattern data for every track in the song. Each track gets its own `TrackFormat` record and an array of `RowFormat` records.

Table 7. The TrackFormat record

Offset	Size	Type	Field name
+0	1	Uint8	channel
+1	1	Uint8	trackId
+2	1	BiasedUint8	rows

channel

determines which channel the track is for. *Valid values 0-3*.

trackId

determines the track id to use for this track.

rows

the number of RowFormat records that follow this structure.

Table 8. The RowFormat record

Offset	Size	Type	Field name
+0	1	Uint8	rowno
+1	8	TrackRow	rowdata

rowno

the index in the track's row array to set

rowdata

the data to set at this index, where the `TrackRow` type is:

Table 9. The TrackRow structure

Offset	Size	Type	Field name
+0	1	Uint8	note
+1	1	Uint8	instrument

+2	6	Effect[3]	effects

with `Effect` being:

Table 10. The Effect structure

Offset	Size	Type	Field name
+0	1	Uint8	effectType
+1	1	Uint8	effectParam

and `effectType` should be any of the following:

Table 11. List of valid effect types

Value	Effect Syntax	Identifier
0	---	etNoEffect
1	Bxx	etPatternGoto
2	C00	etPatternHalt
3	Dxx	etPatternSkip
4	Fxy	etSetTempo
5	Txx	etSfx
6	Exx	etSetEnvelope
7	V0x	etSetTimbre
8	I0x	etSetPanning
9	Hxx	etSetSweep
10	Sxx	etDelayedCut
11	Gxx	etDelayedNote
12	L00	etLock
13	0xy	etArpeggio
14	1xx	etPitchUp
15	2xx	etPitchDown
16	3xx	etAutoPortamento
17	4xy	etVibrato
18	5xx	etVibratoDelay
19	Pxx	etTuning
20	Qxy	etNoteSlideUp
21	Rxy	etNoteSlideDown

The last `RowFormat` record for the last track ends the `SONG` block.

INST block format

The INST block contains instrument data for every instrument in the module. The data is stored in the same order as the instrument `Index` records in the INDX block. Thus the number of instruments contained in this block is the value of the `numberOfInstruments` field in the Header.

A single instrument's data is structured as followed:

1. An `InstrumentFormat` record
2. 4 sequences each composed of:
    a. A `SequenceFormat` record
    b. The sequence's data

InstrumentFormat

The `InstrumentFormat` record defines settings for the instrument.

Offset	Size	Type	Field name
+0	1	Uint8	channel
+1	1	Bool	envelopeEnabled
+2	1	Uint8	envelope

channel

determines which channel the instrument is for, 0 is channel 1, 3 is channel 4, etc. *Valid values 0-3*.

envelopeEnabled

set to true if the instrument has an initial envelope setting.

envelope

the initial envelope setting that is used if `envelopeEnabled` was true.

Sequence data

Following the InstrumentFormat record is the sequence data for each of the instrument's sequences. Data for a sequence is structured as a `SequenceFormat` record followed by the sequence data. There are four kinds of sequences for every instrument. The kind of sequence the data is for is determined by its order in the block:

Order	SequenceKind
0	skArp
1	skPanning
2	skPitch
3	skTimbre

The `SequenceFormat` record:

Offset	Size	Type	Field name
+0	2	Uint16	length
+2	1	Bool	loopEnabled
+3	1	Uint8	loopIndex

length

The length of the sequence data. Following this record will be this number of `Uint8` bytes that are the sequence's data. *Valid values 0-256*.

loopEnabled

Determines if this sequence has a loop index.

loopIndex

The index of the loop point.

WAVE block format

The WAVE block contains waveform data for every waveform in the module. Like the INST block, the data is stored in the same order as the waveform `Index` records in the INDX block. Thus the number of waveforms contained in this block is the value of the `numberOfWaveforms` field in the Header.

A single waveform just contains the 32-bit PCM sample array as an array of 16 `Uint8`.

EOF

The module file should be at end of file (EOF) after the WAVE block.