READ MODE · ANSWER FIRST

How MIDI Files Work

An interactive, visual explanation of How MIDI Files Work, from setup through the complete system.

CHEAT SHEET · 5 ESSENTIAL IDEAS

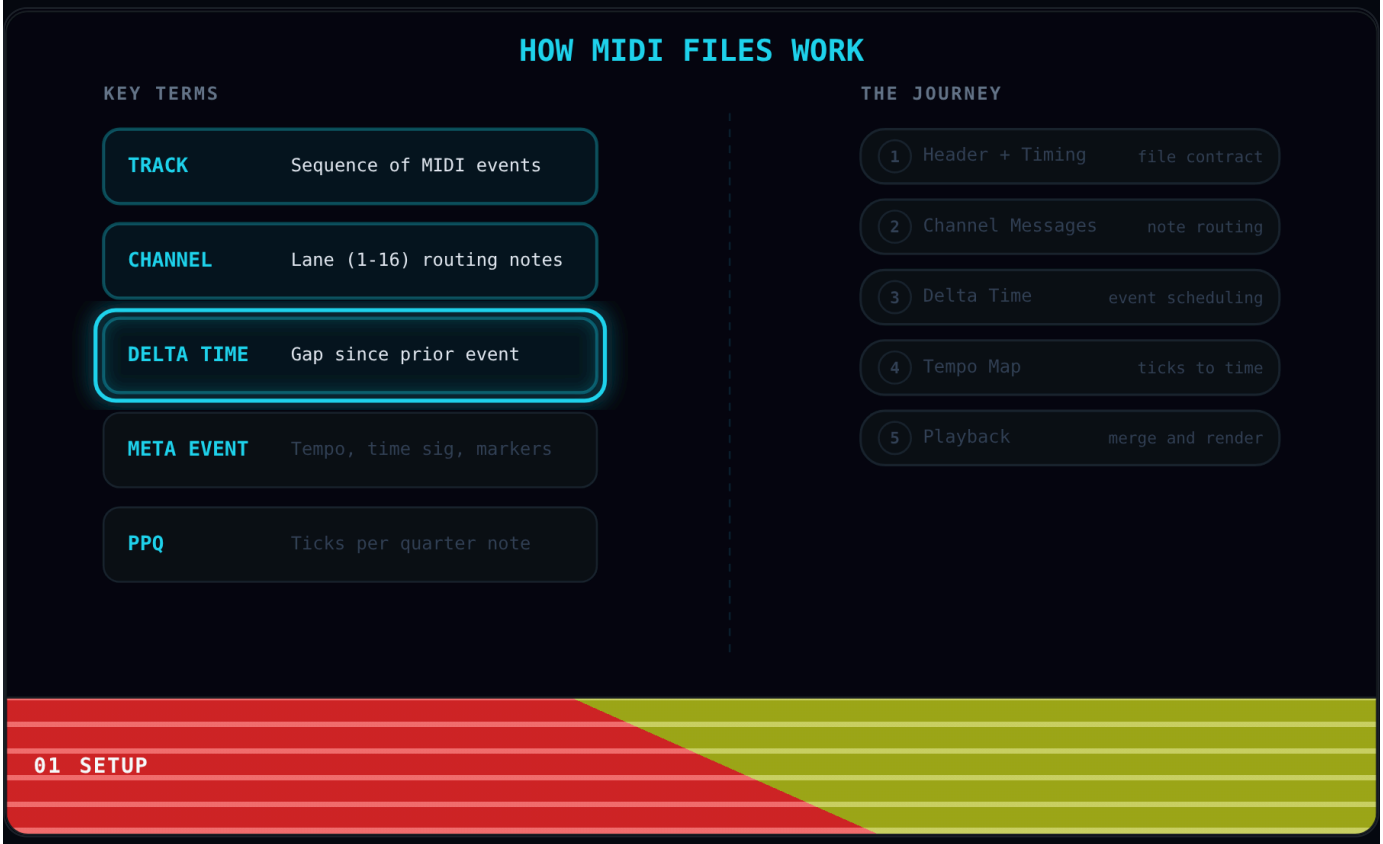
The whole story in 5 lines

How MIDI Files Work becomes easier to reason about when every stage is connected as one system.

1. The MThd header is the contract. Format, track count, and PPQ define how every later byte should be interpreted.
2. MIDI is not notation. It is an event stream that targets channels with pitches, velocities, and controller values.
3. Timing is relative in the file and only becomes a global schedule after accumulation.
4. The same MIDI ticks can feel different under another tempo map. Meta events are file-level instructions, not note data.
5. MIDI is portable performance data. The final sound, notation, or visual depends entirely on the interpreter.

STAGE 1 OF 7

Setup



Welcome. MIDI stands for Musical Instrument Digital Interface. It is one of the most widely used formats in music production, but it does not contain any sound. Think of it like sheet music for computers: it tells instruments what to play, not what to sound like.

A Track is a container for a sequence of events. A MIDI file can hold one track or many. Each track typically represents one instrument or one role in a performance, like the melody line or the drum part.

A Channel is a numbered lane (1 through 16) that routes events to a specific sound. Multiple notes on the same channel share the same instrument patch. Channel 10 is traditionally reserved for drums.

Delta Time is the spacing between events. Instead of storing absolute timestamps, MIDI stores the gap since the previous event. The parser adds these gaps up to recover the actual schedule.

A Meta Event is a file-level instruction that is not sent to any instrument. Tempo changes, time signatures, lyrics, and end-of-track markers are all meta events. They shape interpretation without making sound.

Over the next five stages we will build up the complete picture: how the file header declares structure, how channel messages carry performance data, how delta times build a schedule, how the tempo map shapes time, and how a sequencer merges everything into playback. Let us start with the container itself.

STAGE 2 OF 7

Header + Timing

Here is a raw MIDI file. It arrives as a binary container with no sound yet. Everything inside is structure and instructions. The parser needs a map before it can read anything.

The parser reads the MThd header first. These six bytes declare the file format, how many tracks follow, and the timing resolution. Without this contract, the rest of the file is meaningless.

Track chunks are allocated from the header metadata. Switch the Format control in the sidebar to Fmt 0. Notice how all events collapse into a single merged track. Switch back to Fmt 1 to see separate tracks.

PPQ is loaded into the timing grid. This number says how many ticks fit inside one quarter note. A higher PPQ gives finer timing resolution. Try adjusting the PPQ stepper in the sidebar and watch the grid change.

Each track can now carry its own event stream while sharing the same tick grid. The header made this possible by defining the rules up front.

The container is ready. Structure first, performance data second. That principle runs through everything MIDI does. Next, we will look at the events that actually carry musical performance.

STAGE 3 OF 7

Channel Messages

Now that we know how the file header sets the rules, let us look at what actually travels inside those tracks. A track event stream waits with channel lanes ready but idle.

A status byte and data bytes enter the router from the track. The status byte encodes both the message type (note-on, note-off, control change) and the target channel number.

The router selects the target channel and decodes pitch plus velocity. Switch the Phrase control in the sidebar to Drums and watch how the same routing logic carries drum hits instead of pitched notes.

Channel messages fan out so different musical parts can coexist. Try adjusting the Velocity slider and notice how the fill bars on each channel respond. Velocity is how hard the note is struck.

Note-off or release messages stop the sounding notes later in the stream. Every note-on must eventually be matched by a note-off, or the note rings forever.

The channel state now represents a live performance, not a printed score. But we still have not talked about when these events happen. That is next: delta time and scheduling.

Delta Time

We just saw how channel messages carry pitch and velocity. But how does the file know when to play them? The answer is delta time. The track exposes raw event bytes, and the schedule is still implicit.

The parser reads the next delta time from the byte stream. This number says "wait this many ticks since the previous event." It is a relative offset, not an absolute timestamp.

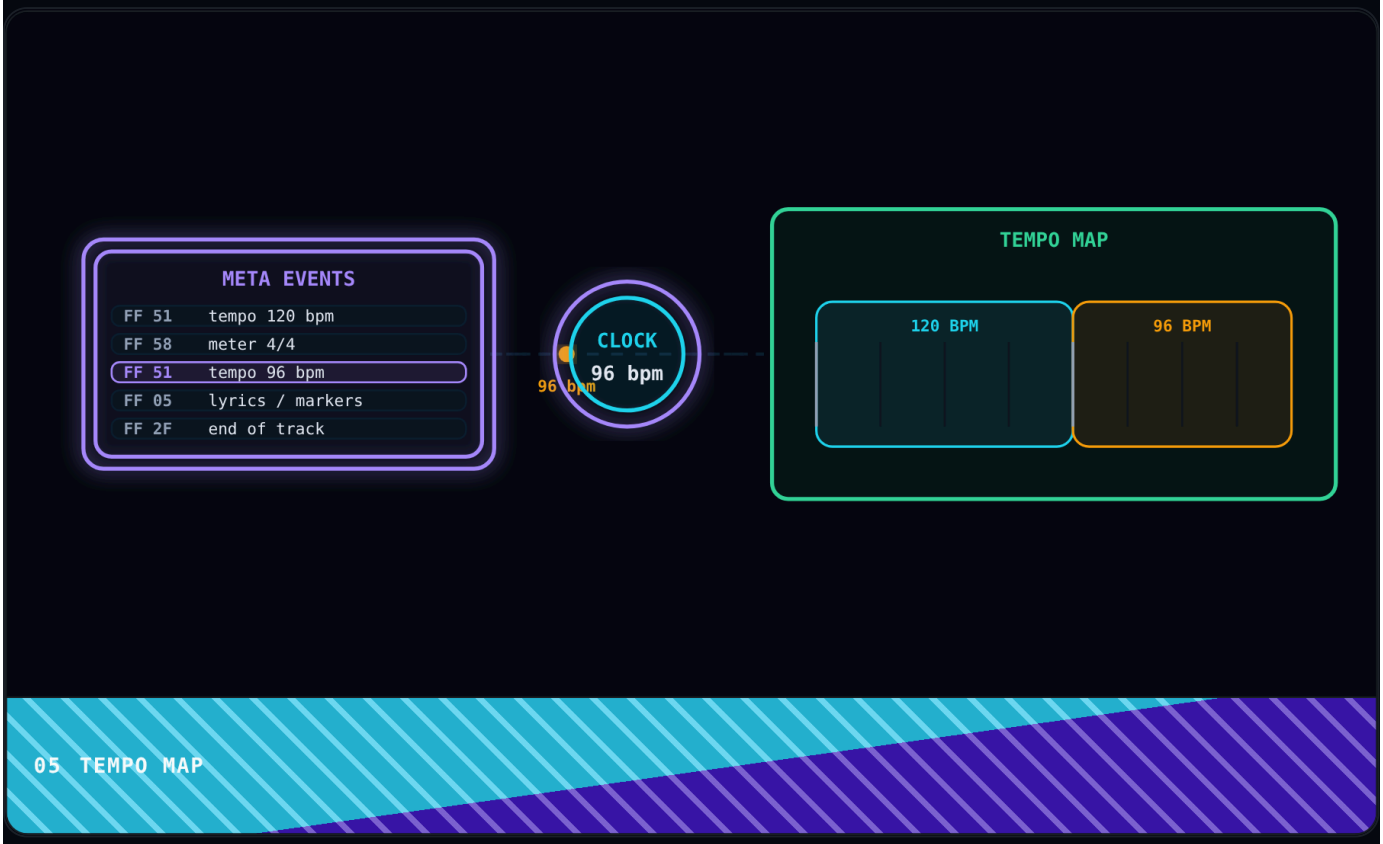
That delta is added to a running counter to form an absolute tick position. Remember PPQ from the header stage? That resolution determines how fine-grained these ticks are.

The event is placed onto the timeline at its absolute tick position. Switch the Feel control in the sidebar to Swing and watch how the gaps become uneven, creating a rhythmic feel from the same note data.

Toggle the Run stat switch in the sidebar. Running status can suppress repeated status bytes without changing the musical meaning. It is a compression trick that saves space in the file.

After accumulation, the track is a fully scheduled sequence ready to merge with other tracks. But ticks still are not real time. We need a tempo map to convert them. That is next.

Tempo Map



05 TEMPO MAP

We have ticks from the delta time stage, but ticks are not real time. This is where meta events come in. A dedicated meta lane sits beside the musical tracks, carrying instructions that no instrument will ever hear.

The first tempo event (FF 51) establishes how many microseconds one quarter note takes. This single number converts every tick in the file into wall-clock milliseconds.

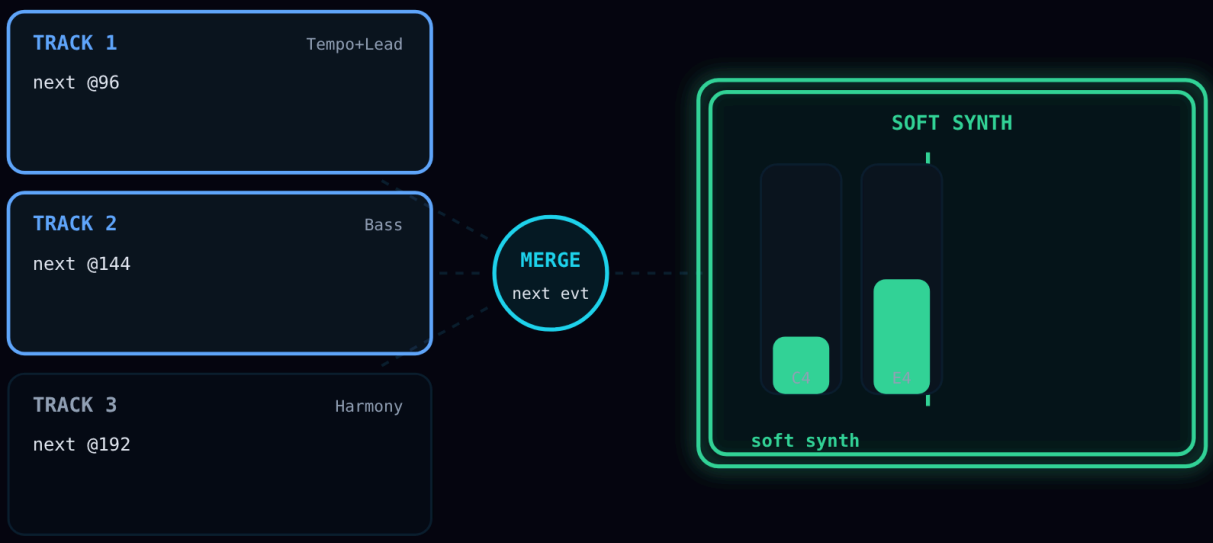
Time signature events (FF 58) group ticks into visible bars and beats. Switch the Meter control in the sidebar to 3/4 and watch the beat grid reorganize.

Later tempo changes reshape the speed of everything that follows. Adjust the Changes stepper to add more tempo segments and see how the same tick range covers different durations.

Text and lyric markers (FF 05) attach to positions on the timeline without becoming notes. Toggle the Lyrics switch to see them appear and disappear. They are metadata, not performance.

End-of-track (FF 2F) marks the boundary and closes the chunk. Now every tick has a real-time meaning. That sets us up for the final stage: merging all tracks into a single playback stream.

Playback



06 PLAYBACK

We have arrived at the finish line. Every track now carries absolute-tick events shaped by the header, channel routing, delta accumulation, and tempo map we built in the previous stages.

The sequencer selects the earliest event across all tracks. This is a merge queue: track boundaries dissolve and only timing order matters.

That event is dispatched toward the active output interpreter. Switch the View control in the sidebar between Synth, Roll, and Score. The same MIDI data produces completely different visual representations.

The renderer turns the same instructions into sound, bars, or notation. Toggle Quantize to snap expressive timing into cleaner grid positions. Notice how the score view tidies up while the synth stays the same.

Note-offs and later events keep flowing as the playhead advances. The sequencer is a clock-driven loop that never stops until the last end-of-track marker fires.

That is the core lesson: MIDI is portable performance data, not baked-in audio and not a finished score. The same file can become a piano concerto, a drum pattern visualizer, or sheet music, depending on who reads it. Now let us step back and see the whole picture together.

Recap

07 RECAP

We started with the header. The MThd block declares the format, track count, and PPQ resolution. Without this contract, nothing else in the file has meaning.

Then we learned how channel messages carry the actual performance. Status bytes route notes, velocities, and controller values to numbered channel lanes. This is the musical content.

Next came delta time. Events store only the gap since the previous event. The parser accumulates these deltas into absolute tick positions, turning a stream of offsets into a global schedule.

The tempo map gave those ticks real-time meaning. Meta events set the tempo, time signature, and markers. The same ticks can feel fast or slow depending on the tempo curve.

Finally, playback merged all tracks by absolute tick and dispatched them to an interpreter. Synth, piano roll, or notation: the renderer decides what MIDI becomes.

That is the complete picture. MIDI is a layered format: structure on the outside, performance data on the inside, timing in between, and interpretation at the end. Every layer depends on the one before it, and together they turn a compact binary file into music.