MIDI Files Specification

Somascape

22/11/2017

Source:

- http://www.somascape.org/midi/tech/mfile.html

Conventions

- A byte is an 8-bit quantity.
- Bit 0 is the least significant bit (lsb), bit 7 is the most significant bit (msb) of a byte. These are also referred to as the bottom and top bits, respectively.
- I have used the C (programming language) notation of representing hexadecimal numbers by prefacing them with "0x", e.g. 0x7F (=127 decimal). Numbers shown in brown are also hex (without the prefix), e.g. `FF 2F 00`.

What is a MIDI file ?

MIDI files are *the* standard means of transferring MIDI data amongst users – it is a common format across all computing platforms. MIDI files contain the standard channel based MIDI messages, along with sequencer-related data (e.g. tempo, time and key signature, track names, etc.) and System Exclusive messages. Each message (also referred to as an **event**) is time stamped.

Any decent MIDI sequencer should allow MIDI files to be loaded and saved, in addition to the use of any proprietary file format.

MIDI files differ from most other types of music files in that they do not contain encoded sound (e.g. as in a WAV file). Consequently, compared with WAV or even MP3 files, MIDI files are extremely compact.

MIDI file structure

The content of a MIDI file is structured as a series of blocks of data referred to as **chunks**, of which two types have been defined :

- **MThd** : Header chunk, containing a few global properties of the file as a whole.
- **MTrk** : Track chunk, containing MIDI and/or sequencer-specific data.

A valid MIDI file will contain a single Header chunk followed by one or more Track chunks.

Each chunk has an 8 byte header that identifies which type it is, and gives the size of its associated data :

- bytes 0 - 3 : **identifier** (either "**MThd**" or "**MTrk**")
- bytes 4 - 7 : **chunklen** (the number of bytes comprising the following data)

followed by "chunklen" bytes of data. I.e. a chunk is (8 + chunklen) bytes long.

To allow for future expansion, a MIDI file reader should skip over (ie ignore) any chunk types that it does not know about, which it can easily do by reading the offending chunk's chunklen. Any MIDI file reader should definately support these two standard chunk types, though.

Note that multi-byte values (eg **chunklen**, above) are stored in big-endian format, i.e. with their MSB first.

Header chunks

The **Header** chunk (identifier = **MThd**) has a fixed length and contains a few global properties pertaining to the file as a whole. It should be the first chunk in the file, and this should be the only occurrence.

Although the Header chunk currently always contains 6 bytes of data (ie chunklen = 6), this should not be assumed. I.e. the chunklen value should always be read and acted upon, to allow for possible future extension to the standard.

Currently these 6 bytes of data represent 3 x 16-bit parameters (in big-endian format) : **format**, **ntracks** and **tickdiv**.

- **format** : describes the chunk structure of a MIDI file. It determines how the following **MTrk** chunks relate to one another. The term **type** is often used synonymously with **format**. This parameter can take one of 3 values, i.e. there are 3 types of MIDI files :

- **format type = 0**

  The MIDI file contains just a single MTrk chunk, that can potentially contain multi-channel MIDI data.

- **format type = 1**

  The file contains two or more MTrk chunks (as specified by the following parameter, **ntracks**) that are to be played simultaneously, i.e. analogous to a mulitrack tape recorder. The first track is a **tempo** track that should only contain tempo related Meta events (i.e. no actual MIDI data) – this is clarified later. This is the most commonly used format, as the various instrumental parts within a composition can be stored in separate tracks, allowing for easier editing. It is possible to store multi-channel data in a track, though it is more usual to keep data relevant to a single MIDI channel in each track.

- **format type = 2**

  The file contains one or more MTrk chunks (as specified by the following parameter, **ntracks**) that are intended to be played independently, i.e. analogous to a drum machine's pattern memory. A format 2 file can be likened to multiple format 0 files all wrapped up in a single file.

- **ntracks** : the number of MTrk chunks following this MThd chunk.

  For a format 0 MIDI file, ntracks can only be "1".

  For format 1 or 2 files, ntracks can be any value. There is no limitation as far as the file format is concerned, though sequencer software will generally impose a limit. If you were to attempt to load a MIDI file containing more tracks than your sequencer can handle, it should load as many as it is able to, ignoring the remainder.

  Naturally the actual number of MTrk chunks following this MThd chunk should be the same as the value specified here for ntracks.

- **tickdiv** : specifies the timing interval to be used, and whether **timecode** (Hrs.Mins.Secs.Frames) or **metrical** (Bar.Beat) timing is to be used. With metrical timing, the timing interval is tempo related, whereas with timecode the timing interval is in absolute time, and hence not related to tempo.

  Bit 15 (the top bit of the first byte) is a flag indicating the timing scheme in use :

  - **Bit 15 = 0 : metrical timing**

    Bits 0 - 14 are a 15-bit number indicating the number of sub-divisions of a quarter note (aka pulses per quarter note, ppqn). A common value is 96, which would be represented in hex as `00 60`. You will notice that 96 is a nice number for dividing by 2 or 3 (with further

repeated halving), so using this value for tickdiv allows triplets and dotted notes right down to hemi-demi-semiquavers to be represented.

– **Bit 15 = 1 : timecode**

Bits 8 - 15 (i.e. the first byte) specifies the number of frames per second (fps), and will be one of the four SMPTE standards - 24, 25, 29 or 30, though expressed as a negative value (using 2's complement notation), as follows :

```
fps Representation (hex)
24  E8
25  E7
29  E3
30  E2
```

Bits 0 - 7 (the second byte) specifies the sub-frame resolution, i.e. the number of sub-divisions of a frame. Typical values are 4 (corresponding to MIDI Time Code), 8, 10, 80 (corresponding to SMPTE bit resolution), or 100.

A timing resolution of 1 ms can be achieved by specifying 25 fps and 40 sub-frames, which would be encoded in hex as `E7 28`.

A complete MThd chunk thus contains 14 bytes (including the 8 byte header).

Example

```
Data (hex)    Interpretation
4D 54 68 64   identifier, the ascii chars 'MThd'
00 00 00 06   chunklen, 6 bytes of data follow . . .
00 01         format = 1
00 11         ntracks = 17
00 60         tickdiv = 96 ppqn, metrical time
```

Track chunks

Track chunks (identifier = **MTrk**) contain a sequence of time-ordered **events** (MIDI and/or sequencer-specific data), each of which has a **delta time** value associated with it - i.e. the amount of time (specified in **tickdiv** units) since the previous event.

It is possible to mix messages having different MIDI channels within the same track chunk, i.e. multi-channel tracks are possible.

The role of the various Track chunks (particularly the first), and the types of events that they can contain, depends on the MIDI file's **format** (as defined in the MThd chunk).

- For **format 0** MIDI files, there should only be one MTrk chunk, and this can therefore contain any valid event - i.e. all timing related and note (potentially multi MIDI channel) data.
- For **format 1** files, the first MTrk chunk is a *global tempo track* and should contain all timing related events and no note data. The second and subsequent MTrk chunks contain the actual note data, and should not contain any timing related events. As all tracks are played together, they all follow the tempo map described in the global tempo track (the first MTrk chunk).
- For **format 2** files, each track is a separate entity (like drum patterns within a drum machine), and can each contain any type of event. There is no global tempo track - each track may have its own tempo map. Any timing related events are specific to the track in which they occur.

Note In the above description, *note data* refers to all MIDI events (Channel MIDI messages), whereas *timing related* events refers to the following Meta events : Marker, Cue Point, Tempo, SMPTE Offset, Time Signature, and Key Signature. Key Signature events are not strictly timing related, though they fall into this group. These Meta events are all detailed later.

The actual data within an MTrk chunk takes the following form:

- delta-time: a variable length quantity, (1 to 4 bytes) denoting the time since the previous event.
- event: 2 or more bytes describing the event, of which there are three main types : MIDI, SysEx and Meta.

repeated for "**chunklen**" bytes. Track chunks thus contain a sequence of "**delta-time / event**" pairs. The three event types can be intermixed within a Track chunk.

In all cases the last event in each MTrk chunk should be an **End of Track** Meta event.

Delta times

The **delta-time** specifies the number of **tickdiv** intervals since the previous event (or from the nominal start of the track if this is the first event). If an event is to occur at the very start of a track, or simultaneously with the previous event, then it will have a delta-time of 0. The delta-time is a *variable length quantity* in that it is specified using 1, 2, 3 or 4 bytes, as necessary.

Only the bottom 7 bits of each of these bytes contributes towards the delta-time, the top bit being used to indicate (when it is set) that another byte follows, i.e. bit 7 of each byte is used to indicate continuation or end of the delta-time data. Consequently the last byte of any delta-time value will have its top bit clear.

Here are some examples of numbers represented as variable-length quantities :

```
Value (hex)   Representation (hex)
00000000      00
00000040      40
0000007F      7F
00000080      81 00
00002000      C0 00
00003FFF      FF 7F
00004000      81 80 00
00100000      C0 80 00
001FFFFF      FF FF 7F
00200000      81 80 80 00
08000000      C0 80 80 00
0FFFFFFF      FF FF FF 7F
```

Most of the time, musical events are sufficiently close together that the delta-time can be expressed in a single byte (i.e. for delta-times of 0 - 127). When a larger delta-time needs to be specified then additional bytes are used. This provides an efficient way of storing delta-time values.

Note that this concept of variable length quantities is not exclusive to delta-times, and is used elsewhere in MIDI files (e.g. the *length parameter* for SysEx and Meta events).

Events

There are three main types of event that can occur within a MTrk chunk - MIDI, SysEx and Meta. Events are not all the same size - they use as many bytes as are necessary to describe each specific event. The first byte of the event data (often referred to as the event's status byte, and identifiable by their having bit 7 set) identifies the event type :

- **Midi events (status bytes 0x8n - 0xEn)**

  Corresponding to the standard Channel MIDI messages, i.e. where "n" is the MIDI channel (0 - 15). This status byte will be followed by 1 or 2 data bytes, as is usual for the particular MIDI message. Any valid Channel MIDI message can be included in a MIDI file.

If the first (status) byte is less than 128 (hex 80), this implies that **running status** is in effect, and that this byte is actually the first data byte (the status carrying over from the previous MIDI event). This can only be the case if the immediately previous event was also a MIDI event, i.e. SysEx and Meta events interrupt (clear) running status.

- SysEx events (status bytes 0xF0 and 0xF7)

  There are a couple of ways in which system exclusive messages can be encoded - as a single message (using the 0xF0 status), or split into packets (using the 0xF7 status). The 0xF7 status is also used for sending **escape sequences**.

- Meta events (status byte 0xFF)

  These contain additional information which would not be in the MIDI data stream itself. E.g. TimeSig, KeySig, Tempo, TrackName, Text, Marker, Special, EOT (End of Track) events being some of the most common.

Note that the status bytes associated with **System Common** messages (0xF1 to 0xF6 inclusive) and **System Real Time** messages (0xF8 to 0xFE inclusive) are not valid within a MIDI file. Generally none of these messages are relevant within a MIDI file, though for the rare occasion when you do need to include one, it should be embedded within a **SysEx escape sequence**.

The main differences between the contents of a MIDI file and a live stream of MIDI data (i.e. flowing along a MIDI cable) are :

- In MIDI files, all events have an associated delta time value. In a MIDI data stream, events happen when they are received.
- System Common messages (0xF1 to 0xF6 inclusive) and System Real Time messages (0xF8 to 0xFE inclusive) cannot normally occur within a MIDI file (though see escape sequences).
- As mentioned above, within a MIDI file the System Reset status byte (0xFF) is used to signify a Meta event, hence Meta events are only found in MIDI files.

MIDI events

MIDI events include all of the Channel MIDI messages. In all cases the status byte is nibblised, with the top nibble (i.e. the top 4 bits) indicating the message type (0x8 to 0xE), and the lower nibble indicating the MIDI channel (referred to as “n” below).

The first three of these messages specify a MIDI note number : 0 - 127 (C-1 - G9), with middle C (C4) being note number 60 (0x3C). The standard 5 octave

synthesizer keyboard range is 36 - 96, whereas the 88-note piano keyboard range is 21 - 108. Refer to this ready reckoner table of MIDI note numbers for the complete mapping.

Note Off

3 bytes : `8n note velocity`

Stop sounding the specified **note**, on MIDI channel **n**.

note is the key number (0 - 127).

velocity is the release velocity (0 - 127). A default value of 64 is used in the absence of velocity sensors.

Note On

3 bytes : `9n note velocity`

Start sounding the specified `note`, on MIDI channel `n`.

note is the key number (0 - 127).

velocity is the attack velocity (0 - 127). A default value of 64 is used in the absence of velocity sensors. A value of 0 has a special meaning and is interpreted as a Note Off (thus allowing the use of running status for a sequence of Note On and Off commands).

Polyphonic Pressure

3 bytes : `An note pressure`

Apply aftertouch pressure to the specified note, on MIDI channel n.

note is the key number (0 - 127).

pressure is the amount of aftertouch (0 - 127).

Controller

3 bytes : `Bn controller value`

Set the specified **controller**, on MIDI channel **n**, to **value**.

controller is the controller number (0 - 127). Controllers 120 - 127 are the Channel Mode messages.

value is the value to which the controller is to be set (0 - 127).

Refer to my **MIDI Software Specification** document for details regarding controller number assignment and Channel Mode messages.

Program Change

2 bytes : `Cn program`

Select the specified **program** (i.e. voice, or instrument), on MIDI channel **n**.

program is the voice number to be selected (0 - 127).

Channel Pressure

2 bytes : **Dn pressure**

Apply aftertouch pressure to all notes currently sounding on MIDI channel **n**.

pressure is the amount of aftertouch (0 - 127).

Pitch Bend

3 bytes : `En lsb msb`

Apply pitch bend to all notes currently sounding on MIDI channel `n`.

lsb (0 - 127) and **msb** (0 - 127) together form a 14-bit number, allowing fine adjustment to pitch. Using hex, `00 40` is the central (no bend) setting. `00 00` gives the maximum downwards bend, and `7F 7F` the maximum upwards bend.

The amount of pitch bend produced by these minimum and maximum settings is determined by the receiving device's Pitch Bend Sensitivity, which can be set using RPN 00 00.

SysEx events

There are a couple of ways of encoding System Exclusive messages. The normal method is to encode them as a single event, though it is also possible to split messages into separate packets (**continuation events**). A third form (an **escape sequence**) is used to wrap up arbitrary bytes that could not otherwise be included in a MIDI file.

Single (complete) SysEx messages

`F0 length message`

length is a `variable length quantity` (as used to represent delta-times) which specifies the number of bytes in the following **message**.

message is the remainder of the system exclusive message, minus the initial 0xF0 status byte.

Thus, it is just like a normal system exclusive message, though with the additional length parameter.

Note that although the terminal 0xF7 is redundant (strictly speaking, due to the use of a **length** parameter) it *must* be included.

Example

The system exclusive message :

```
F0 7E 00 09 01 F7
```

would be encoded (without the preceding delta-time) as :

```
F0 05 7E 00 09 01 F7
```

(In case you're wondering, this is a **General MIDI Enable** message.)

SysEx messages sent as packets - Continuation events

Some older MIDI devices, with slow onboard processors, cannot cope with receiving a large amount of data en-masse, and require large system exclusive messages to be broken into smaller packets, interspersed with a pause to allow the receiving device to process a packet and be ready for the next one.

This approach can of course be used with the method described above, i.e. with each packet being a self-contained system exclusive message (i.e. each starting with 0xF0 and ending with 0xF7).

Unfortunately, some manufacturers (notably Casio) have chosen to bend the standard, and rather than sending the packets as self-contained system exclusive messages, they act as though running status applied to system exclusive messages (which it doesn't - or at least it shouldn't).

What Casio do is this : the first packet has an initial 0xF0 byte but doesn't have a terminal 0xF7. The last packet doesn't have an initial 0xF0 but does have a terminal 0xF7. All intermediary packets have neither. No unrelated events should occur between these packets. The idea is that all the packets can be stitched together at the receiving device to create a single system exclusive message.

Putting this into a MIDI file, the first packet uses the 0xF0 status, whereas the second and subsequent packets use the 0xF7 status. This use of the 0xF7 status is referred to as a **continuation event**.

Example

A 3-packet message :

```
F0 43 12 00
43 12 00 43 12 00
43 12 00 F7
```

with a 200-tick delay between the first two, and a 100-tick delay between the final two, would be encoded (without the initial delta-time, before the first packet) :

```
F0 03 43 12 00            first packet (the 4 bytes F0,43,12,00 are transmitted)
81 48                     200-tick delta-time
F7 06 43 12 00 43 12 00   second packet (the 6 bytes 43,12,00,43,12,00 are transmitted)
64                        100-tick delta-time
F7 04 43 12 00 F7         third packet (the 4 bytes 43,12,00,F7 are transmitted)
```

See the note below regarding distinguishing packets and escape sequences (which both use the 0xF7 status).

Escape sequences

```
F7 length bytes
```

length is a *variable length quantity* which specifies the number of bytes in **bytes**.

This has nothing to do with System Exclusive messages as such, though it does use the 0xF7 status. It provides a way of including bytes that could not otherwise be included within a MIDI file, e.g. System Common and System Real Time messages (Song Position Pointer, MTC, etc).

Note that Escape sequences do not have a terminal 0xF7 byte.

Example

The **Song Select** System Common message :

```
F3 01
```

would be encoded (without the preceding delta-time) as :

```
F7 02 F3 01
```

You are not restricted to single messages per escape sequence - any arbitrary collection of bytes may be included in a single sequence.

Note **Parsing the 0xF7 status byte**

When an event with an 0xF7 status byte is encountered whilst reading a MIDI file, its interpretation (SysEx packet or escape sequence) is determined as follows :

- When an event with 0xF0 status but lacking a terminal 0xF7 is encountered, then this is the first of a Casio-style multi-packet message, and a *flag* (boolean variable) should be set to indicate this.
- If an event with 0xF7 status is encountered whilst this *flag* is set, then this is a continuation event (a system exclusive packet, one of many).

  If this event has a terminal 0xF7, then it is the last packet and *flag* should be cleared.
- If an event with 0xF7 status is encountered whilst *flag* is clear, then this event is an escape sequence.

Naturally, the *flag* should be initialised clear prior to reading each track of a MIDI file.

Meta events

Meta events are used for special non-MIDI events, and use the 0xFF status that in a MIDI data stream would be used for a System Reset message (a System Reset message would not be useful within a MIDI file).

They have the general form : `FF type length data`

type specifies the type of Meta event (0 - 127).

length is a *variable length quantity* (as used to represent delta times) specifying the number of bytes that make up the following *data*. Some Meta events do not have a **data** field, whereupon **length** is 0.

The use of a *variable length quantity*, rather than a fixed single byte, for **length** means that data fields longer than 127 bytes are possible.

The **length** field should always be read, and should not be assumed, as the definition may change. A MIDI file reader/player should ignore any Meta event types that it does not know about. It should also ignore any additional data if an event's length is longer than expected (it is safe to assume that any extension to the data field will be *appended* to the current definition). For example if at some time in the future the Sequence Number Meta event is extended with a third data byte, then the first 2 will still have the same interpretation as currently.

Meta event types 0x01 to 0x0F inclusive are reserved for text events. In each case it is best to use the standard 7-bit ASCII character set to ensure reliable interchangeability when transferring files between different computing platforms, however an 8-bit character set may be used. Many text events are best located at or near the beginning of a track (e.g. Copyright, Sequence/Track name, Instrument name), whereas others (Lyric, Marker, Cue point) can occur at various places within a track – their position being an integral aspect of the event.

Although most Meta events are optional, a few are mandatory. Also some events have restrictions regarding their placement.

Sequence Number

`FF 00 02 ss ss`

ss ss is a 16-bit value specifying the sequence number (as would be used in a **MIDI Cue** message).

This event is optional, though if present should occur at time = 0, and prior to any MIDI events. It should not occur more than once in any single MTrk chunk.

For format 0 and 1 MIDI files (which contain just one sequence) this event should occur only in the first track (the only track, in the case of format 0 files). A collection of format 0/1 files, each with a unique sequence number, could be loaded by a jukebox-style file player, such that individual files (sequences) could be selected using a MIDI Cue message.

For format 2 MIDI files this event can occur in each track, such that a MIDI Cue message could be used to identify each pattern/sequence (MTrk chunk) as part of a song sequence.

A shortened version can be used in format 2 MIDI files : the 2 data bytes can be omitted (thus length must be 0), whereupon the sequence number is derived from the MTrk chunk's position within the file.

Text

`FF 01 length text`

length is a `variable length quantity` specifying the number of bytes making up the following `text`.

text is any amount of ASCII text.

This event is optional, and is used to include comments and other user information. There are other text-based Meta events for specific purposes, such as Track Name, Copyright Message, Lyrics, etc. It is recommended, though not essential, that this event is placed near the start of a track.

Copyright

```
FF 02 length text
```

This event contains a copyright message comprising the characters "(C)" along with the year and owner of the copyright. This event is optional, though if present it should be the very first event (at time 0) in the first MTrk chunk. There should be no other occurrences within the MIDI file, so if there are multiple copyright messages, they should all be placed together in a single event.

Sequence / Track Name

```
FF 03 length text
```

This event is optional. It's interpretation depends on its context. If it occurs in the first track of a format 0 or 1 MIDI file, then it gives the **Sequence Name**. Otherwise it gives the **Track Name**.

Instrument Name

```
FF 04 length text
```

This optional event is used to provide a textual clue regarding the intended instrumentation for a track (e.g. "Piano" or "Flute", etc). If used, it is reccommended to place this event near the start of a track.

It may be used with the **MIDI Channel Prefix** Meta event to specify the MIDI channel that this instrument name description applies to. Alternatively, the MIDI channel could be specified textually within the **Instrument Name** Meta event.

Lyric

```
FF 05 length text
```

This optional event contains a song's lyric (the words to be sung). Generally, each syllable of a lyric will be in a separate Lyric event, placed at appropriate points throughout a track.

Marker

```
FF 06 length text
```

This optional event is used to label points within a sequence, e.g. rehearsal letters, loop points, or section names (such as "First verse").

For a format 1 MIDI file, **Marker** Meta events should only occur within the first MTrk chunk.

Cue Point

```
FF 07 length text
```

This optional event is used to describe something that happens within a film, video or stage production at that point in the musical score. E.g. "Car crashes", "Door opens", etc.

For a format 1 MIDI file, **Cue Point** Meta events should only occur within the first MTrk chunk.

Program Name

```
FF 08 length text
```

This optional event is used to embed the patch/program name that is called up by the immediately subsequent Bank Select and Program Change messages. It serves to aid the end user in making an intelligent program choice when using different hardware.

This event may appear anywhere in a track, and there may be multiple occurrences within a track.

Device Name

```
FF 09 length text
```

This optional event is used to identify the hardware device used to produce sounds for this track.

It should occur only once in a track, at the beginning before any sendable MIDI data. It should also precede any **Program Name** Meta events.

MIDI Channel Prefix

```
FF 20 01 cc
```

cc is a byte specifying the MIDI channel (0-15).

This optional event is used to associate any subsequent SysEx and Meta events with a particular MIDI channel, and will remain in effect until the next **MIDI Channel Prefix** Meta event or the next MIDI event.

It's use is particularly relevant in format 0 MIDI files, where multi-channel data is contained in the single MTrk chunk. E.g. if you want to use **Instrument Name** Meta events then you can either include the MIDI channel (textually) within these events, or you could precede them with a **MIDI Channel Prefix** Meta event, so that it is clear which MIDI channel each **Instrument Name** event refers to.

It is also useful when converting a MIDI file from format 0 to 1, and back again, as any association between non MIDI events and a particular MIDI channel can be retained. E.g. in a format 1 MIDI file, where each track contains data for a single MIDI channel (that's not a neccessity, it's just a convention) there will be various SysEx and Meta events distributed amongst the various tracks and hence associated with the same MIDI channel as the MIDI events within each track. Thus when converting to a format 0 MIDI file, the SysEx and Meta events from each track can be clustered together and preceded by an appropriate **MIDI Channel Prefix** event. When converting back to a format 1 MIDI file, these clusters of SysEx and Meta events can be placed in separate tracks along with their associated MIDI events, thus restoring the original structure.

MIDI Port

```
FF 21 01 pp
```

pp is a byte specifying the MIDI port (0-127).

Many systems provide a number of separately addressable MIDI ports in order to get around bandwidth issues and the 16 MIDI channel limit. This optional event specifies the MIDI output port on which data within this MTrk chunk will be transmitted.

Naturally, this event should be placed prior to any MIDI events that are to be affected. Usually it would be placed at time=0 (i.e. at the start of a track), however it is possible to place more than one such event in any MTrk chunk, should you wish to output data through a different port later in the track.

End of Track

```
FF 2F 00
```

This mandatory event must be the last event in each MTrk chunk, and that should be the only occurrence per track. Note that there is no data field, hence length is 0.

Tempo

```
FF 51 03 tt tt tt
```

tt tt tt is a 24-bit value specifying the tempo as the number of microseconds per quarter note.

Specifying tempos as *time per beat*, rather than the more usual (musically) *beat per time*, ensures precise long-term synchronisation with a time-based synchronisation protocol such as SMPTE or MIDI time code.

Ideally, Tempo events should only occur where MIDI clocks would be located – a convention intended to maximise compatibility with other synchronisation devices, thus allowing easy transfer of time signature / tempo map information between devices.

There should generally be a Tempo event at the beginning of a track (at time = 0), otherwise a default tempo of 120 bpm will be assumed. Thereafter they can be used to effect an immediate tempo change at any point within a track.

For a format 1 MIDI file, **Tempo** Meta events should only occur within the first MTrk chunk (i.e. the tempo track).

SMPTE Offset

```
FF 54 05 hr mn se fr ff
```

hr is a byte specifying the hour, which is also encoded with the SMPTE format (frame rate), just as it is in MIDI Time Code, i.e. **0rrhhhhh**, where :

rr = frame rate : 00 = 24 fps, 01 = 25 fps, 10 = 30 fps (drop frame), 11 = 30 fps (non-drop frame)

hhhhh = hour (0-23)

mn se are 2 bytes specifying the minutes (0-59) and seconds (0-59), respectively.

fr is a byte specifying the number of frames (0-23/24/28/29, depending on the frame rate specified in the **hr** byte).

ff is a byte specifying the number of fractional frames, in 100ths of a frame (even in SMPTE-based tracks using a different frame subdivision, defined in the MThd chunk).

This optional event, if present, should occur at the start of a track, at time = 0, and prior to any MIDI events. It is used to specify the SMPTE time at which the track is to start.

For a format 1 MIDI file, a **SMPTE Offset** Meta event should only occur within the first MTrk chunk.

Time Signature

```
FF 58 04 nn dd cc bb
```

nn is a byte specifying the numerator of the time signature (as notated).

dd is a byte specifying the denominator of the time signature as a negative power of 2 (i.e. 2 represents a quarter-note, 3 represents an eighth-note, etc).

cc is a byte specifying the number of MIDI clocks between metronome clicks.

bb is a byte specifying the number of notated 32nd-notes in a MIDI quarter-note (24 MIDI Clocks). The usual value for this parameter is 8, though some sequencers allow the user to specify that what MIDI thinks of as a quarter note, should be notated as something else.

Examples

A time signature of 4/4, with a metronome click every 1/4 note, would be encoded :

```
FF 58 04 04 02 18 08
```

There are 24 MIDI Clocks per quarter-note, hence cc=24 (0x18).

A time signature of 6/8, with a metronome click every 3rd 1/8 note, would be encoded :

```
FF 58 04 06 03 24 08
```

Remember, a 1/4 note is 24 MIDI Clocks, therefore a bar of 6/8 is 72 MIDI Clocks. Hence 3 1/8 notes is 36 (=0x24) MIDI Clocks.

There should generally be a **Time Signature** Meta event at the beginning of a track (at time = 0), otherwise a default 4/4 time signature will be assumed. Thereafter they can be used to effect an immediate time signature change at any point within a track.

For a format 1 MIDI file, **Time Signature** Meta events should only occur within the first MTrk chunk.

Key Signature

```
FF 59 02 sf mi
```

sf is a byte specifying the number of flats (-ve) or sharps (+ve) that identifies the key signature (-7 = 7 flats, -1 = 1 flat, 0 = key of C, 1 = 1 sharp, etc).

mi is a byte specifying a major (0) or minor (1) key.

For a format 1 MIDI file, **Key Signature** Meta events should only occur within the first MTrk chunk.

Sequencer Specific Event

`FF 7F length data`

The first 1 or 3 bytes of **data** is a manufacturer's ID code (same format as for System Exclusive messages). This optional event can be used to store sequencer-specific information.