Interpreting a Standard MIDI File

by Dave Sebald

In the infant days of MIDI (the early ‘80s) every sequencer on the market saved files in its own proprietary format. The reason for this was probably just marketing. Each company could claim that the unique benefits of its sequencer-- its superior resolution, its ability to save lyrics, etc-- made it a better choice than competing products.

It wasn’t long, however, before these companies began to realize that file transportability itself was a benefit. Musicians who used computers were beginning to demand features like an easy means for moving music created on a sequencer to a notation program for print-out or for moving algorithmically composed files to a sequencer. Also In their search for improved interfaces many users were beginning to jump from one sequencer to another or even from one platform to another only to find that none of their previous work could be brought into the new environment.

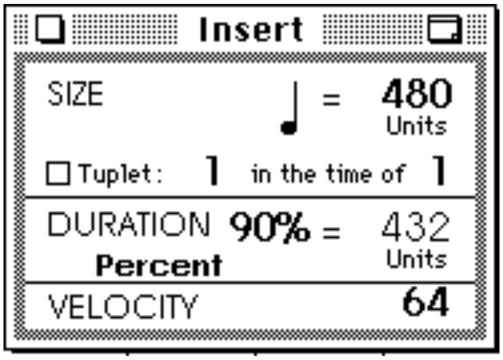
Smelling potential profits from answering this need, software companies were faced with several options. They could: 1) create a full suite of programs for every environment, 2) create translators for every competing products’ formats, or 3) add some means for exporting files in a single common format that could then be imported by any other MIDI software. The third option was obviously the cheapest so in July, 1988, members of the MIDI Manufacturers Association agreed to adopt an addition to the MIDI 1.0 specification called the **Standard MIDI File** format. Although it took a few years to catch on, today almost every sequencer, notation program, algorithmic composer, or other arcane piece of MIDI software allows the user to save and load in SMF format in addition to its own “still-the-best” proprietary format. The recent explosive growth of multimedia and world wide information exchange through the Internet has confirmed the wisdom of that decision.

At its most basic level, the function of any proprietary sequencer file is to store MIDI performance commands in the right order and in the right rhythm. Simply put, standard MIDI files represent a lowest-common-denominator method of doing that for all sequencers. Well, o.k., they’re not quite that simplistic; in fact it’s surprising for many people to see the amount of information that they actually can convey. Along with basic time stamped MIDI messages, SMFs can save what are called “meta-events.” These include data which show tempo, time signature, key signature, mode (major/minor), track names, lyrics, copyright information, structural markers, event cues, and even manufacturer-specific details.

What if your sequencer didn’t include one of these fetures? Because of its universal nature, the SMF format had to be designed for maximum flexibility . This means that the specification permits MIDI software that can’t deal with one or more aspects of the above information to simply ignore it.

It is important to emphasize that MIDI software products do not use SMFs directly. That means they do not deal with standard MIDI files in real time but rather translate them into their own proprietary formats before use. In part this is because SMFs use compression algorithms whose real-time expansion could slow the internal workings of the application and create timing errors.

The following analysis of a very simple standard MIDI file can be interesting not only because it describes the exact meaning of every part of a typical file, but because it gives some insight into how computers and humans differ in processing information in general. To create the file I used Opcode’s Musicshop in step-entry mode. For clarity’s sake I wanted to enter some repetitive data that would be easy to spot in the exported file. I set up the step-entry mode to enter quarter notes in 4/4 meter with each note being exactly the same length, 432/480’s of a beat. (Musicshop divides every beat into 480 parts.)

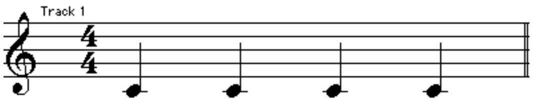


SIZE ♩ = 480 Units

☐ Tuplet: 1 in the time of 1

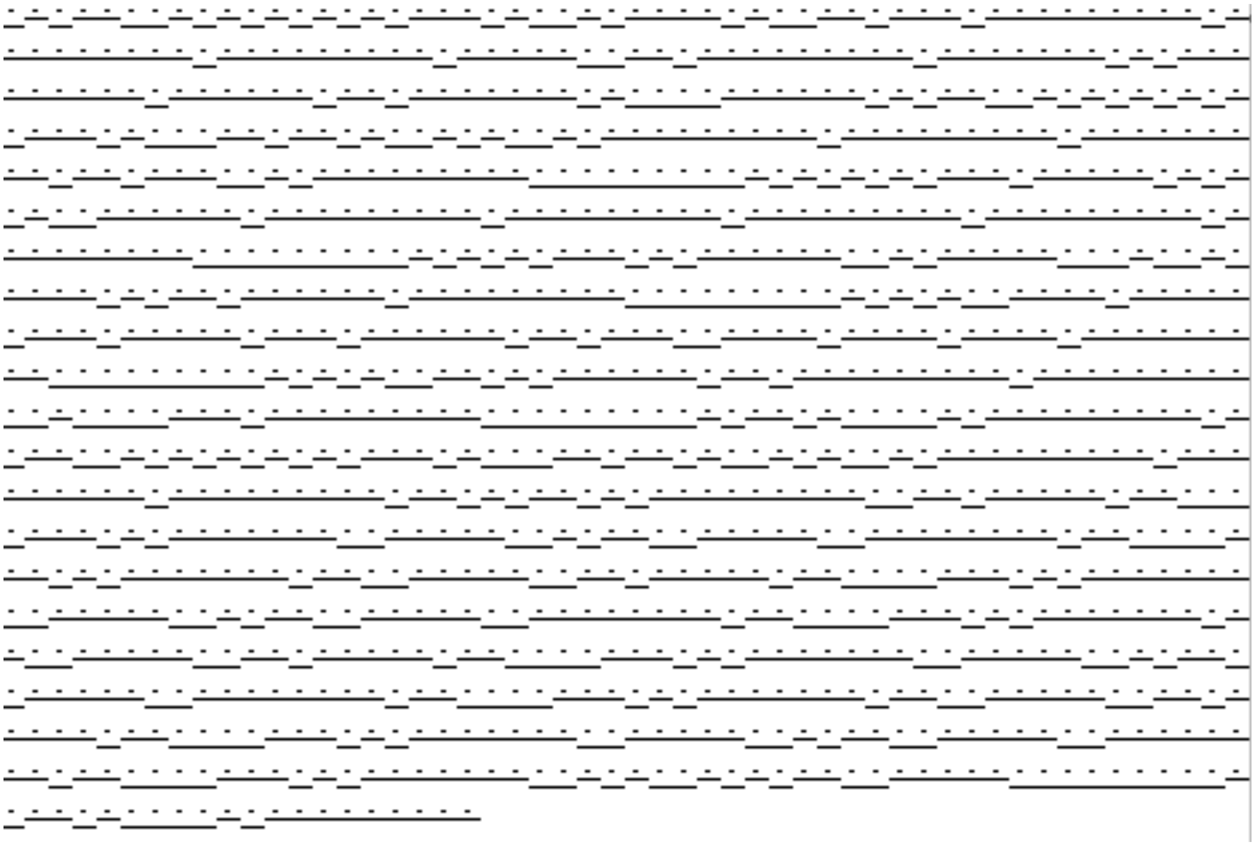
DURATION 90% = 432 Units
Percent

VELOCITY 64

I then entered this sequence of notes...

...saved it in standard MIDI file format...

...and opened it again in a program that displays files in their natural numeric format.

Here is what a sequencer actually “sees” in this .mid file-- not really numbers, just pulses of on or off current at precisely timed intervals. (Dashes = 5 volts (high) or 0 volts (low). Dots = bits or time units:



But it’s easier for humans to think of the above “ons” and “offs” as numbers. Here it is in binary. Using reverse logic 1 = off, 0 = on:

```
1010011010101010100010110100001011001000100000000010
0000000010000000001000001100100000000010000000101000
0000001000000100100000001011110000001010011010101010
1000101110010010110101101000000000100000000010000000
0010010001101000000000011111111010101010001000001010
1011000000100000000010000000001000000000100000000010
0000000011111111101010100010100000011010000011101101
0000101001000000100000000011111111101010110000100000
1000100000100010000001001000110000100001000010000000
0011111111101010110010100000010010000000001000000000
1101111000100000000011111111101001011110100000000010
1001101010101010001011100100101101011010000000001000
0000001000000000100101001010000000001100100000100111
1000101000000011000001101001100000110000000010011110
0010100000001001100000110010000010011110001010000000
1100000110100110000011000000001001111000101000000010
0110000011001000001001111000101000000011000001101001
1000001100000000100111100010100000001001100000110010
0000100111100010100000001100000110100110000011000000
0010011110001010000000110101101010011000001111111110
10010111101000000000
```

It’s also easier for humans to group the binary number into bytes (10 bits including start bit and stop bit):

```
1010011010  1010101000  1011010000  1011001000  1000000000  1000000000
1000000000  1000001100  1000000000  1000000010  1000000000  1000000100
1000000010  1111000000  1010011010  1010101000  1011100100  1011010110
1000000000  1000000000  1000000000  1001000110  1000000000  1111111110
1010101000  1000001010  1011000000  1000000000  1000000000  1000000000
1000000000  1000000000  1111111110  1010100010  1000000110  1000001110
1101000010  1001000000  1000000000  1111111110  1010110000  1000001000
1000001000  1000000100  1000110000  1000010000  1000000000  1111111110
1010110010  1000000100  1000000000  1000000000  1101111000  1000000000
1111111110  1001011110  1000000000  1010011010  1010101000  1011100100
1011010110  1000000000  1000000000  1000000000  1001010010  1000000000
1100100000  1001111000  1010000000  1100000110  1001100000  1100000000
1001111000  1010000000  1001100000  1100100000  1001111000  1010000000
1100000110  1001100000  1100000000  1001111000  1010000000  1001100000
1100100000  1001111000  1010000000  1100000110  1001100000  1100000000
1001111000  1010000000  1001100000  1100100000  1001111000  1010000000
1100000110  1001100000  1100000000  1001111000  1010000000  1101011010
1001100000   1111111110   1001011110   1000000000
```

And it’s easier to interpret these bytes if we strip off the start bit and stop bit from each one:

```
 01001101    01010100    01101000    01100100    00000000    00000000
 00000000    00000110    00000000    00000001    00000000    00000010
 00000001    11100000    01001101    01010100    01110010    01101011
 00000000    00000000    00000000    00100011    00000000    11111111
 01010100    00000101    01100000    00000000    00000000    00000000
 00000000    00000000    11111111    01010001    00000011    00000111
 10100001    00100000    00000000    11111111    01011000    00000100
 00000100    00000010    00011000    00001000    00000000    11111111
 01011001    00000010    00000000    00000000    10111100    00000000
 11111111    00101111    00000000    01001101    01010100    01110010
 01101011    00000000    00000000    00000000    00101001    00000000
 10010000    00111100    01000000    10000011    00110000    10000000
 00111100    01000000    00110000    10010000    00111100    01000000
 10000011    00110000    10000000    00111100    01000000    00110000
 10010000    00111100    01000000    10000011    00110000    10000000
 00111100    01000000    00110000    10010000    00111100    01000000
 10000011    00110000    10000000    00111100    01000000    10101101
 00110000     11111111     00101111     00000000
```

Real computer geeks often find it easier to represent binary numeration in hexidecimal, each digit representing a nibble.

```
  4D          54          68          64          00          00
  00          06          00          01          00          02
  01          E0          4D          54          72          6B
  00          00          00          23          00          FF
  54          05          60          00          00          00
  00          00          FF          51          03          07
  A1          20          00          FF          58          04
  04          02          18          08          00          FF
  59          02          00          00          BC          00
  FF          2F          00          4D          54          72
  6B          00          00          00          29          00
  90          3C          40          83          30          80
  3C          40          30          90          3C          40
  83          30          80          3C          40          30
  90          3C          40          83          30          80
  3C          40          30          90          3C          40
  83          30          80          3C          40          AD
  30          FF          2F          00
```

Returning to binary, we lay out the numbers vertically to permit clearly visible grouping (with lines) and allow room for interpreting these groups according to the MIDI specification.

01001101	(M)	ASCII numbers for "MThd" identify this file as a SMF. Header information tells the sequencer how to interpret the following tracks.	**CHUNK 1**
01010100	(T)		
01101000	(h)		
01100100	(d)		
00000000			
00000000		Length of this header chunk in 4 bytes	
00000000			
00000110			
00000000		Format type (Type 1 = multitrack)	
00000001			
00000000		Number of tracks in this file.	
00000010			
00000001		Relative timing shown by MSN (0000) 480 ticks/beat (0001 11100000)	
11100000			
01001101	(M)	ASCII numbers for "MTrk" identify the first track. This track is for Meta-commands, not MIDI data.	**CHUNK 2**
01010100	(T)		
01110010	(r)		
01101011	(k)		
00000000			
00000000		Length of the first track chunk in 4 bytes	
00000000			
00100011			
00000000		Separator?	
11111111		SMPTE offset command (first 3 bytes) in	
01010100			
00000101			
01100000		hours,	
00000000		minutes,	
00000000		seconds,	
00000000		frames,	
00000000		and subframes.	
00000000		Separator?	
11111111		Tempo setting command (first 3 bytes) in microseconds (last 3 bytes)	
01010001			
00000011			
00000111			
10100001			
00100000			
00000000		Separator?	
11111111		Meter signature command (first 3 bytes).	
01011000			
00000100			
00000100		Numerator of time sign	
00000010		Denominator of time sign	
00011000		Clocks per beat	
00001000		32nd notes per beat	
00000000		Separator?	
11111111		Key signature command (first 3 bytes)	
01011001			
00000010			
00000000		No b's or #'s	
00000000		Major mode	
10111100		?	

00000000		Separator?	
11111111		End of track command (3 bytes)	
00101111			
00000000			
01001101	(M)	ASCII numbers for "MTrk" identify the second track, which is actually the first track of MIDI data.	**CHUNK 3**
01010100	(T)		
01110010	(r)		
01101011	(k)		
00000000		Length of the second track chunk in 4 bytes	
00000000			
00000000			
00101001			
00000000		At the beginning (00000000)	
10010000		Turn on a note on channel 0 (144)	
00111100		The note is middle C (60)	
01000000		struck mezzo-forte (64)	
10000011		After 432 ticks (1 - 4 bytes)	
00110000			
10000000		Turn off a note on channel 0 (140)	
00111100		The note is middle C (60)	
01000000		released with mezzo-forte speed (64)	
00110000		After 48 ticks (1 - 4 bytes)	
10010000		Turn on a note on channel 0	
00111100		The note is middle C	
01000000		struck mezzo-forte	
10000011		After 432 ticks (1 - 4 bytes)	
00110000			
10000000		Turn off a note on channel 0 (140)	
00111100		The note is middle C (60)	
01000000		released with mezzo-forte speed (64)	
00110000		After 48 ticks (1 - 4 bytes)	
10010000		Turn on a note on channel 0	
00111100		The note is middle C	
01000000		struck mezzo-forte	
10000011		After 432 ticks (1 - 4 bytes)	
00110000			
10000000		Turn off a note on channel 0 (140)	
00111100		The note is middle C (60)	
01000000		released with mezzo-forte speed (64)	
00110000		After 48 ticks (1 - 4 bytes)	
10010000		Turn on a note on channel 0	
00111100		The note is middle C	
01000000		struck mezzo-forte	
10000011		After 432 ticks (1 - 4 bytes)	
00110000			
10000000		Turn off a note on channel 0 (140)	
00111100		The note is middle C (60)	
01000000		released with mezzo-forte speed (64)	
10101101		After 5808 ticks (1-4 bytes)	
00110000			
11111111		End of track command (3 Bytes)	
00101111			
00000000			