Standard MIDI File Format

Dustin Caldwell

The standard MIDI file format is a very strange beast. When viewed as a whole, it can be quite overwhelming. Of course, no matter how you look at it, describing a piece of music in enough detail to be able to reproduce it accurately is no small task. So, while complicated, the structure of the midi file format is fairly intuitive when understood.

I must insert a disclaimer here that I am by no means an expert with midi nor midi files. I recently obtained a Gravis UltraSound board for my PC, and upon hearing a few midi files (.MID) thought, "Gee, I'd like to be able to make my own .MID files." Well, many aggravating hours later, I discovered that this was no trivial task. But, I couldn't let a stupid file format stop me. (besides, I once told my wife that computers aren't really that hard to use, and I'd hate to be a hypocrite) So if any errors are found in this information, please let me know and I will fix it. Also, this document's scope does not extend to EVERY type of midi command and EVERY possible file configuration. It is a basic guide that should enable the reader (with a moderate investment in time) to generate a quality midi file.

1. Overview

A midi (.MID) file contains basically 2 things, Header chunks and Track chunks. Section 2 explains the header chunks, and Section 3 explains the track chunks. A midi file contains ONE header chunk describing the file format, etc., and any number of track chunks. A track may be thought of in the same way as a track on a multi-track tape deck. You may assign one track to each voice, each staff, each instrument or whatever you want.

2. Header Chunk

The header chunk appears at the beginning of the file, and describes the file in three ways. The header chunk always looks like:

4D 54 68 64 00 00 00 06 ff ff nn nn dd dd

The ascii equivalent of the first 4 bytes is MThd. After MThd comes the 4-byte size of the header. This will always be 00 00 00 06, because the actual header information will always be 6 bytes.

ff ff is the file format. There are 3 formats:

0 - single-track
1 - multiple tracks, synchronous
2 - multiple tracks, asynchronous

Single track is fairly self-explanatory - one track only. Synchronous multiple tracks means that the tracks will all be vertically synchronous, or in other

words, they all start at the same time, and so can represent different parts in one song. Asynchronous multiple tracks do not necessarily start at the same time, and can be completely asynchronous.
nn nn is the number of tracks in the midi file.
dd dd is the number of delta-time ticks per quarter note. (More about this later)

3. Track Chunks

The remainder of the file after the header chunk consists of track chunks. Each track has one header and may contain as many midi commands as you like. The header for a track is very similar to the one for the file:

4D 54 72 6B xx xx xx xx

As with the header, the first 4 bytes has an ascii equivalent. This one is MTrk. The 4 bytes after MTrk give the length of the track (not including the track header) in bytes.

Following the header are midi events. These events are identical to the actual data sent and received by MIDI ports on a synth with one addition. A midi event is preceded by a delta-time. A delta time is the number of ticks after which the midi event is to be executed. The number of ticks per quarter note was defined previously in the file header chunk. This delta-time is a variable-length encoded value. This format, while confusing, allows large numbers to use as many bytes as they need, without requiring small numbers to waste bytes by filling with zeros. The number is converted into 7-bit bytes, and the most-significant bit of each byte is 1 except for the last byte of the number, which has a msb of 0. This allows the number to be read one byte at a time, and when you see a msb of 0, you know that it was the last (least significant) byte of the number. According to the MIDI spec, the entire delta-time should be at most 4 bytes long.

Following the delta-time is a midi event. Each midi event (except a running midi event) has a command byte which will always have a msb of 1 (the value will be >= 128). A list of most of these commands is in appendix A. Each command has different parameters and lengths, but the data that follows the command will have a msb of 0 (less than 128). The exception to this is a meta-event, which may contain data with a msb of 1. However, meta-events require a length parameter which alleviates confusion.

One subtlety which can cause confusion is running mode. This is where the actual midi command is omitted, and the last midi command issued is assumed. This means that the midi event will consist of a delta-time and the parameters that would go to the command if it were included.

4. Conclusion

If this explanation has only served to confuse the issue more, the appendices contain examples which may help clarify the issue. Also, 2 utilities and a graphic file should have been included with this document:
DEC.EXE - This utility converts a binary file (like .MID) to a tab-delimited text file containing the decimal equivalents of each byte.
REC.EXE - This utility converts a tab-delimited text file of decimal values into a binary file in which each byte corresponds to one of the decimal values.
MIDINOTE.PS - This is the postscript form of a page showing note numbers with a keyboard and with the standard grand staff.

1. MIDI Event Commands

Each command byte has 2 parts. The left nybble (4 bits) contains the actual command, and the right nybble contains the midi channel number on which the command will be executed. There are 16 midi channels, and 8 midi commands (the command nybble must have a msb of 1).

In the following table, x indicates the midi channel number. Note that all data bytes will be <128 (msb set to 0).

Hex	Binary	Data	Description
8x	1000xxxx	nn vv	Note off (key is released) nn=note number vv=velocity
9x	1001xxxx	nn vv	Note on (key is pressed) nn=note number vv=velocity
Ax	1010xxxx	nn vv	Key after-touch nn=note number vv=velocity
Bx	1011xxxx	cc vv	Control Change cc=controller number vv=new value
Cx	1100xxxx	pp	Program (patch) change pp=new program number
Dx	1101xxxx	cc	Channel after-touch cc=channel number
Ex	1110xxxx	bb tt	Pitch wheel change (2000H is normal or no change) bb=bottom (least sig) 7 bits of value tt=top (most sig) 7 bits of value

are of the format:
FF xx nn dd
All meta-events start with FF followed by the command (xx), the length, or number of bytes that will contain data (nn), and the actual data (dd).

Hex	Binary	Data	Description
00	00000000	nn ssss	Sets the track's sequence number. nn=02 (length of 2-byte sequence number) ssss=sequence number
01	00000001	nn tt ..	Text event- any text you want. nn=length in bytes of text tt=text characters
02	00000010	nn tt ..	Same as text event, but used for copyright info. nn tt=same as text event
03	00000011	nn tt ..	Sequence or Track name nn tt=same as text event
04	00000100	nn tt ..	Track instrument name nn tt=same as text event
05	00000101	nn tt ..	Lyric nn tt=same as text event
06	00000110	nn tt ..	Marker nn tt=same as text event
07	00000111	nn tt ..	Cue point nn tt=same as text event
2F	00101111	00	This event must come at the end of each track
51	01010001	03 tttttt	Set tempo tttttt=microseconds/quarter note
58	01011000	04 nn dd ccbb	Time Signature nn=numerator of time sig. dd=denominator of time sig. 2=quarter 3=eighth, etc. cc=number of ticks in metronome click bb=number of 32nd notes to the quarter note
59	01011001	02 sf mi	Key signature sf=sharps/flats (-7=7 flats, 0=key of C, 7=7 sharps) mi=major/minor (0=major, 1=minor)

7F	01111111	xx dd ..	Sequencer specific information xx=number of bytes to be sent

The following table lists system messages which control the entire system. These have no midi channel number. (these will generally only apply to controlling a midi keyboard, etc.)

Hex	Binary	Data	Description
F8	11111000		Timing clock used when synchronization is required.
FA	11111010		Start current sequence
FB	11111011		Continue a stopped sequence where left off
FC	11111100		Stop a sequence

The following table lists the numbers corresponding to notes for use in note on and note off commands.

Octave #	Note Numbers											
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0	0	1	2	3	4	5	6	7	8	9	10	11
1	12	13	14	15	16	17	18	19	20	21	22	23
2	24	25	26	27	28	29	30	31	32	33	34	35
3	36	37	38	39	40	41	42	43	44	45	46	47
4	48	49	50	51	52	53	54	55	56	57	58	59
5	60	61	62	63	64	65	66	67	68	69	70	71
6	72	73	74	75	76	77	78	79	80	81	82	83
7	84	85	86	87	88	89	90	91	92	93	94	95
8	96	97	98	99	100	101	102	103	104	105	106	107
9	108	109	110	111	112	113	114	115	116	117	118	119
10	120	121	122	123	124	125	126	127				

BIBLIOGRAPHY

"MIDI Systems and Control" Francis Rumsey 1990 Focal Press

"MIDI and Sound Book for the Atari ST" Bernd Enders and Wolfgang Klemme 1989 M&T Publishing, Inc.

MIDI file specs and general MIDI specs were also obtained by sending e-mail to LISTSERV@AUVM.AMERICAN.EDU with the phrase GET MIDISPEC PACKAGE in the message.

DEC.CPP

```
/*  file  dec.cpp
by  Dustin Caldwell    (dustin@gse.utah.edu)
*/
#include <dos.h>
```

```
#include <stdio.h>
#include <stdlib.h>
void helpdoc();
main()
{
        FILE *fp;
        unsigned char ch, c;
        if((fp=fopen(_argv[1], "rb"))==NULL)            /* open file to read */
        {
                printf("cannot open file %s\n",_argv[1]);
                helpdoc();
                exit(-1);
        }
        c=0;
        ch=fgetc(fp);
        while(!feof(fp))                        /* loop for whole file */
        {
                printf("%u\t", ch);             /* print every byte's decimal equiv. */
                c++;
                if(c>8)                                 /* print 8 numbers to a line */
                {
                        c=0;
                        printf("\n");
                }
                ch=fgetc(fp);
        }
        fclose(fp);                     /* close up */
}
void helpdoc()                  /* print help message */
{
        printf("\n   Binary File Decoder\n\n");
        printf("\n Syntax:  dec binary_file_name\n\n");
        printf("by Dustin Caldwell  (dustin@gse.utah.edu)\n\n");
        printf("This is a filter program that reads a binary file\n");
        printf("and prints the decimal equivalent of each byte\n");
        printf("tab-separated. This is mostly useful when piped \n");
        printf("into another file to be edited manually.  eg:\n\n");
        printf("c:\>dec sonata3.mid > son3.txt\n\n");
        printf("This will create a file called son3.txt which can\n");
```

```
        printf("be edited with any ascii editor. \n\n");
        printf("(rec.exe may also be useful, as it reencodes the \n");
        printf("ascii text file).\n\n");
        printf("Have Fun!!\n");
}
---------------------------- REC.CPP ---------------------------------
/*  File  rec.cpp
        by Dustin Caldwell   (dustin@gse.utah.edu)
*/
#include <dos.h>
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
void helpdoc();
main()
{
        FILE *rfp, *wfp;
        unsigned char ch, c;
        char s[20];
        if((rfp=fopen(_argv[1], "r"))==NULL)                    /* open the read file */
        {
                printf("cannot open file %s \n",_argv[1]);
                helpdoc();
                exit(-1);
        }
        if((wfp=fopen(_argv[2], "wb"))==NULL)                   /* open the write file */
        {
                printf("cannot open file %s \n",_argv[1]);
                helpdoc();
                exit(-1);
        }
        c=0;
        ch=fgetc(rfp);
        while(!feof(rfp))                       /* loop for whole file */
        {
                if(isalnum(ch))                 /* only 'see' valid ascii chars */
                {
                        c=0;
                        while(isdigit(ch))      /* only use decimal digits (0-9) */
```

```
                        {
                                s[c]=ch;        /* build a string containing the number */
                                c++;
                                ch=fgetc(rfp);
                        }
                        s[c]=NULL;                      /* must have NULL terminator */
                        fputc(atoi(s), wfp);/* write the binary equivalent to file */
                }
                ch=fgetc(rfp);                  /* loop until next number starts */
        }
        fclose(rfp);                    /* close up */
        fclose(wfp);
}
void helpdoc()          /* print help message */
{
        printf("\n   Text File Encoder\n\n");
        printf("\n Syntax:  rec text_file_name binary_file_name\n\n");
        printf("by Dustin Caldwell  (dustin@gse.utah.edu)\n\n");
        printf("This is a program that reads an ascii tab-\n");
        printf("delimited file and builds a binary file where\n");
        printf("each byte of the binary file is one of the decimal\n");
        printf("digits in the text file.\n");
        printf(" eg:\n\n");
        printf("c:\>rec son3.txt son3.mid\n\n");
        printf("(This will create a file called son3.mid which is\n");
        printf("a valid binary file)\n\n");
        printf("(dec.exe may also be useful, as it decodes binary files)\n\n");
        printf("Have Fun!!\n");
}
```