Music 208 Winter 2014
John Ellinger, Carleton College

MIDI Basics

The labs and midterm project for the first segment of MUSC 108 deal with MIDI messages and Standard MIDI files. MIDI is an acronym for Musical Instrument Digital Interface. MIDI was developed in 1983 as a hardware and software specification that enables computers and synthesizers to communicate with each other. MIDI supports up to 16 channels of simultaneous sound. Each channel can have its own melody, rhythm, and instrument. The MIDI 1.0 Specification is the definitive source on the MIDI language.

Conventions

The symbol * is multiplication. The symbol ^ indicates exponentiation, 10^3 is 10^3.

Number Systems

Computers process numbers. Whether it's Microsoft word, Photoshop, MIDI or digital audio, the computer is processing streams of numbers. We're all familiar with the decimal, or base 10, number system that uses the digits 0-9. In the decimal system (base 10) each digit of a number represents a power of 10. In the binary system (base 2) each digit of a number represents a power of 2. In the hexadecimal system (base 16) each digit of a number represents a power of 16. As an example here's the number 10101 in each of the three systems. Computers process information in the binary number system, or base 2, where only two digits are used: 0 and 1. Computer programmers like the hexadecimal (hex), or base 16, number system to because its easy to convert into binary.

Decimal Number 10101

10^4	10^3	10^2	10^1	10^0
1	0	1	0	1
10,000	+0	+100	+0	+1

Binary Number 10101 is 21 decimal

2^4	2^3	2^2	2^1	2^0
1	0	1	0	1
16	+0	+4	+0	+1

Hexadecimal Number 10101 is 65,793 decimal

16^4	16^3	16^2	16^1	16^0
1	0	1	0	1
65536	+0	+256	+0	+1

The Binary Number System, Base 2

This chart counts from 0-15 in binary. A single one or zero is called a bit.

Binary	Decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

If you've got good coordination and patience you could count from 0-1023 (2^{10}) on your fingers.

The Hexadecimal Number System, Base 16

The hexadecimal system uses 16 digits: 0-9 plus A B C D E F where A = 10, B = 11, C = 12, D = 13, E = 14, and F = 15. You can also use lower case a b c d e f. Hex numbers are usually prefixed or suffixed to distinguish them from decimal numbers. A common convention is to prefix hex numbers with a dollar sign $. Another common convention is to prefix hex numbers with zero x like 0xFAD. Still another convention is to suffix the number with h or (hex); e.g.a1a2h or 7B(hex). This section will use $ to prefix hex numbers.

The MIDI standard defines and explains many of its messages and data formats in hex format. Understanding hex bytes are essential to understanding MIDI. For example the MIDI code to turn a note on is a two hex digit code. The first digit is always 9 and the second digit can range from 0-F that represents one of the 16 MIDI channels. All MIDI codes are two hex digits. Look at how much easier it is to to recognize the MIDI codes in the hex column than in the decimal column.

MIDI Note On Command	
Hex	Decimal
90	144
91	145
92	146
93	147
94	148
95	149
96	150
97	151
98	152
99	153
9A	154
9B	155
9C	156
9D	157
9E	158
9F	159

Hex Binary Conversion

Because a single hex digit ranges from 0 to 15 (0-F hex), every four bit binary number can be represented by a single hex digit.

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9

10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Bits and Bytes

A **bit** is a single binary digit, a one or a zero. A **byte** is eight bits or two hex digits. Modern computers process data most efficiently in one, two, four byte, and eight byte units. You may have heard the terms 32 bit operating system or a 64 bit operating system. That tells you the optimal data processing size of the system. A 32 bit system processes data in four byte chunks and a 64 bit system processes data in eight byte chunks.

Binary to Hex Conversions

Binary to hex conversions are easy. Given the binary number 1110100110, start by forming four bit groups from right to left: 11 1010 0110. Then convert each group of four bits to its corresponding hex digit: 11 1010 0110 = $3A6. Hex to binary conversions are just as easy. Convert each hex digit to its binary equivalent. The hex number $94 is 1001 0100 in binary.

Decimal to Hex conversions

Conversions between decimal and hex are more difficult. The Calculator applications on both Mac and Windows will do that for you.

Maximum Values

These are the maximum values of a one byte, two byte, and four byte number in all three number systems.

	Binary	Decimal	Hexadecimal
1 byte	2^8 (11111111)	255	FF
2 bytes	2^{16} (1111111111111111)	65,535	FFFF
4 bytes	2^{32} (11111111111111111111111111111111)	4,294,967,295	FFFFFFFF

MIDI Messages

The MIDI standard spells out the rules and grammar for sending MIDI messages. MIDI

messages function the same way across all operating systems (Mac, Windows, Linux). Thanks to Apple, Microsoft, and the Linux community, most of the low level details are built into the operating system and are well documented. These low level details provide MIDI programmers with routines to read the clock time, send a MIDI message, and listen for incoming MIDI without having to understand the internals of the hardware.

Data Bytes and Status Bytes

All MIDI messages consist of one or more bytes. The first byte in a MIDI message is called the status byte. The remaining bytes are called data bytes. Data bytes have different interpretations depending on its status byte. The distinguishing feature between data bytes and status bytes is the most significant bit (MSB) or the left most bit of the binary eight bit number. The MSB of all data bytes is zero and the MSB of all status bytes is one. The MSB determines the range of values for each type of byte. You'll always use two hex digits for a status byte and usually use decimal numbers for the data bytes.

	Binary	Hex	Decimal
Data byte	**0**0000000 -**0**1111111	0-7F	0-127
Status byte	**1**0000000 -**1**1111111	80-FF	128-255

Message Length

The majority of MIDI messages are three bytes long: one status byte followed by two data bytes.
< *status* > + < *data 1* > + < *data 2* >

A few MIDI messages are two bytes long: one status byte followed by one data byte.
< *status* > + < *data 1* >

Some MIDI messages vary in size: one status byte followed by zero or more data bytes.
< *status* > + < *data* >...

NON and NOF messages

The two MIDI messages you'll use most often are those that turn a note on (NON) and a note off (NOF). Both NON and NOF messages are three bytes long, a status byte followed by two data bytes. Each of the bytes has a specific meaning.

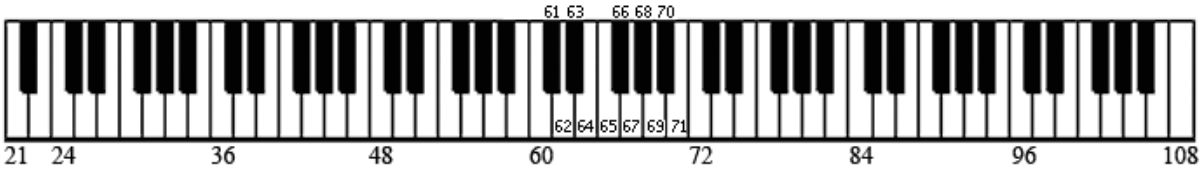
Status Byte

If the left hex digit of the status byte is an 8, it's a NOF message. If the left hex digit of the status byte is a 9, it's a NON message. The right hex digit of the status byte indicates which of the 16 MIDI channels the note is to be played on. According to users, MIDI

channels are numbered from 1-16. According to the MIDI specification, MIDI channels are numbered from from 0-$F. You'll be using the zero based system. To send a NON message to channel 1 you'd use $90 as the status byte. To send a NOF message on channel 12 you'd use $8b for the status byte.

Data 1 Byte

Data 1 byte is mapped a note on the piano keyboard. The full range of data bytes is from 0-127. The piano only has 88 keys. The lowest key on the piano is MIDI note number 21. The highest key is MIDI note number 108. Middle C is MIDI note number 60. The black and white keys are numbered consecutively. Because there are 12 half steps in an octave, when you add or subtract 12 from any MIDI note number you'll get that note one octave higher or lower. This picture shows the MIDI note numbers for all C's on the piano: 24, 36, 48, 60, 72, 84, 96, 108.



Data 2 Byte

The data 2 byte is called velocity which indicates how soft or loud the the note is played. The MIDI velocity range is 0-127 with zero being silence and 127 being the loudest. Musical dynamics cover nine levels: silence (rest), ppp, pp, p, mp, mf, f, ff, and fff. It's up to you to determine how the velocity numbers are mapped to the nine dynamic levels.

NOF

There are two ways to turn a note off: use NOF status byte, or use the NON status byte and a velocity of zero. You should use the NOF status byte in this class.

Status	Data 1	Data 2	Comments
8n	60	100	NOF status, velocity is ignored
9n	60	0	NON status, velocity = 0

Every note you turn on has to be turned off. Once a note is turned on it will keep sounding until it receives a matching NOF message. You may experience "stuck notes" when you're working with MIDI. That happens when a NON message is not paired with a corresponding NOF message. Usually the software has some kind of "Panic" button or option that will send a NOF message to every note on every channel.

The Eight MIDI Status Messages

Eight different MIDI Status messages, $8n-$FF, are defined in the MIDI specification. The left hex digit indicates the type of message, the right hex digit indicates the MIDI channel, except for %F0-$FF. The MIDI protocol is very specific about how many data bytes follow a status byte. The most common number is two. Data bytes have different meanings depending on the status byte they're associated with. Data bytes can indicate which key was pressed down, which key was released, how loud the note is, what button was pushed, whether the sustain pedal is up or down, and the position of knobs, wheels, and sliders. We'll concentrate on the NON, NOF, Control Change, and Program Change messages in this class.

Message	Status 80-FF	Number data bytes	Data 1 (0-127)	Data 2 (0-127)
Note Off NOF	8n	2	Note Number	Velocity (ignored)
Note On NON	9n	2	Note Number	Velocity
Aftertouch(Poly)	An	2	Note Number	Pressure Value
Control Change	Bn	2	Control Number	Control Value
Program Change	Cn	1	Patch Number	Not Used
Aftertouch (Channel)	Dn	1	Note Number	Pressure Value
Pitch Wheel	En	2	Pitch Bend LSB	Pitch Bend MSB
System Exclusive	Fx	varies	Mfr. specific	varies
n = channels 0-F				
x = 0-F but is not related to channel				

Time

The MIDI specification does not say anything about scheduling when notes are turned on or off. That's left to the programmer writing the software. All computers have a really fast clock that the programmer can use to tell time. The clocks in the iMacs in the MRC lab run in the gigahertz range, over two billion cycles per second. MIDI messages are usually timed in milliseconds (thousandths of a second) or sometimes microseconds (millionths of a second). Even during the fastest most complex MIDI passages the computer is basically at idle.

MIDI Time Stamps

Say you want to record your performance on a MIDI keyboard. When you click the Record button, the software remembers the computer clock time. When an incoming MIDI message is received, the operating system attaches a time stamp to the MIDI message. Most time stamps are in millisecond resolution, although the iMacs use microseconds. In order to play back the song, the software compares the time the record button was pressed with the time stamp of the MIDI messages and is able to play back the messages the way they were recorded. If the tempo needs to be changed during playback the time stamps may need to be recalculated. There are three common strategies used for calculating time during playback: chronological, differential, and proportional.

Chronological Time

The clock is set to zero when you click the play button. The software continuously checks the clock to see if it matches the time stamp. When it matches the message is sent. Chronological time means "wait until the clock reaches this time and then send the message."

Differential Time

The clock is set to zero when you click the play button. The software keeps track of time differences between two adjacent messages, say message N and message N+1. The procedure goes something like this. Assume the MIDI time stamps are T0, T1, T2, T3 ... . T0, the first time stamp, may not be zero, it's almost impossible to play the first note at exactly the time the record button was clicked. The first time difference is equal to T0-0. The second second time difference is T1-T0. The third time difference is T2-T1, etc. These time differences tell the computer how long it needs to wait before playing the next message. Computers have built in timers that work just like an alarm clock. The programmer sets the "alarm" to go off when the calculated time difference has expired. While waiting it can other things. Difference time means "wait this amount of time and then send the message." You can calculate a running total of chronological time by adding up all the time differences.

PPQ Time

PPQ (or PPQN) stands for Parts Per Quarter Note. PPQ time is very similar to musical time where all note durations are based on proportions of a quarter note. These proportions are independent of the tempo. You can assign an arbitrary value like 480 units or parts to one quarter note. Then an eighth note would get 240 parts, a sixteenth note 120 parts, a half note 960 parts, a dotted eighth note 360 parts, etc. One PPQ is called a tick. If the PPQ value is 480 and the tempo is 60, there are 480 ticks every second. The duration of each tick would be 2.0833 ms (1000/480). At a tempo of 90,there are one and a half quarter notes per second, or 720 (480+240) ticks per

second. The duration of each tick would be 1.3888 ms (1000/720). PPQ time means "check the tempo, calculate the duration of one PPQ at that tempo, multiply that by the number of PPQs and wait that amount of time before sending the message." While it sounds more complicated, this is the method used in MIDI files. It is also the best method to use for changing tempos.