Applications and Implications of MIDI 2.0

by

Julian Hamelberg

S.B., Computer Science and Engineering and Music, Massachusetts Institute of Technology (2022)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2023

Authored by:	Julian Hamelberg Department of Electrical Engineering and Computer Science May 19, 2023
Certified by:	Ian Hattwick Lecturer Thesis Supervisor
Accepted by:	Katrina LaCurts Chair, Master of Engineering Thesis Committee

Applications and Implications of MIDI 2.0

by

Julian Hamelberg

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2023, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Since its introduction in 1983, Musical Instrument Digital Interface (MIDI) has been the standard for connecting electronic music instruments, computers, and other audio devices to play, edit, and record music. The MIDI Association recently announced a new specification, MIDI 2.0, to add more flexibility to the MIDI protocol while still being backwards compatible with the MIDI 1.0 specification. This thesis presents an analysis of MIDI 2.0 by comparing it to previous versions of MIDI and the limitations of those specifications including keyboard bias, 12-tone bias, limited controller value resolution, and limited per note expression. In addition, we examine the core features of the MIDI 2.0 specification including MIDI Capability Inquiry (MIDI-CI) and Universal MIDI Packets (UMPs).

To further demonstrate the capabilities of MIDI 2.0, we provide examples of MIDI-CI messages and implement a Python library for creating and sending UMPs using Apple's CoreMIDI framework to explore creative use cases of UMPs. Several Python applications are presented to demonstrate the use of new features of MIDI 2.0 such as note attributes, new pitch representations, and per-note expression. Finally, we analyze MIDI 2.0 to investigate implications of the updated specification, how it can increase musical expression, and how it can be used creatively by independent developers and musicians.

Thesis Supervisor: Ian Hattwick
Title: Lecturer

Acknowledgments

This thesis project would not be possible without all of my mentors, friends, and family.

First and foremost, I would like to thank my thesis supervisor, Ian Hattwick, for his continued support and enthusiasm for this project. After every meeting with Ian, I left more excited about the project.

The support from my close friends has been incredible. Late night chats and reviewing my writing. Without them, it would have been very difficult to finish project.

Lastly, I would like to thank my parents and brother. Their continued love and support has made this thesis possible.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

Chapter 1

Introduction

MIDI (Musical Instrument Digital Interface) is a technical specification that describes how devices communicate musical information [11]. Since 1983, the standard has been widely adopted by the music technology industry, musicians, and developers for its universality and expandability [14]. It has enabled musical devices to communicate with each other and has allowed for creative possibilities such as instrument control, sequencing, sampling, beatmaking, composition, and more [15].

In this paper, we will discuss our work exploring the implications and creative possibilities of the largest update to MIDI since its inception, MIDI 2.0. In Chapter 2, we cover the original MIDI specification (MIDI 1.0), additions to the specification prior to MIDI 2.0 (including MPE), and its limitations. In Chapter 3, we cover the MIDI 2.0 specification, changes from the original MIDI specification, and new features including MIDI-CI and Universal MIDI Packets. In Chapter 4, we describe an implementation of MIDI 2.0 in Python and creative applications of the new features in MIDI 2.0. In Chapter 5, we analyze MIDI 2.0 and answer the research questions:

- How does MIDI 2.0 increase musical expressiveness when compared to previous versions of MIDI?

- Given the evolving state of the MIDI 2.0 specification, how can developers implement MIDI 2.0?

In Chapter 6, we put together the overall picture of MIDI, its future, and our future work.

1.1 Scope of the work

MIDI 2.0 is still in development with an update to MIDI-CI in November 2022 [7], announcements at NAMM 2023 [22] in April hinting at more updates for the specification, and many reserved and undefined fields in the specification documents. Considering the context of current state of MIDI 2.0, the work in this thesis mainly focuses on Universal MIDI Packets, their implementation, and how the new data format enhances musical expressivity. The thesis goes into detail on how MIDI-CI currently works; however, there is currently not enough to create a meaningful implementation myself.

1.2 Contributions of this thesis

The contributions of this thesis are as follows:

- An analysis of MIDI 1.0 and its limitations
- An analysis and explanation of MIDI 2.0 in its current state
- A Python library that implements MIDI 2.0 Universal MIDI Packets
- Examples of creative applications using Universal MIDI Packets
- An analysis of applications of MIDI 2.0 for creative uses in digital music instrument design.

The contributions described here will help facilitate future research in music technology and digital instrument music design.

Chapter 2

An Introduction to MIDI 1.0

The MIDI 1.0 specification describes a command set of MIDI messages that convey musical instructions such as note pitch or instrument selection. The MIDI Manufacturers Association (MMA) is the authority for the specification worldwide except Japan, and the Association for Musical Electronics Industry (AMEI) is the authority for the specification in Japan. Originally designed to connect different synthesizers together, MIDI is now a protocol available on all operating systems that allows digital music instruments to send musical data to your computer. It's a serial protocol where all the information is transmitted over a single cable as a stream of bits [15] [18]. Every cable can send data to 16 different channels. Each message encodes a musical event that is either sent to all channels or a specific channel and uses one byte to represent a command (or status) and (if necessary) one or two bytes to represent data for the command.

2.1 MIDI 1.0 Messages

MIDI 1.0 specifies a number of standard messages, including Note-on and off, continuous controller, polyphonic and channel pressure, etc. All of these standard messages share a similar format consisting of 2-3 bytes. The first byte is called the "status" byte, and is differentiated by having its MSB (most significant bit) set to 1. Subsequent bytes are called "data" bytes, and are indicated by having their MSB set to 0.

Due to the first bit of each byte being reserved for differentiating status bytes, the resulting precision of each byte is 7-bits. This is one of most prominent drawbacks of the MIDI 1.0 standard.

MIDI Note-on Message

A Note-on message instructs a synthesizer to begin playing a note. The status byte contains the command and the channel the command is sent to. The Note On command starts with 9 in hex, the last four bits represent the channel, so a note on message to channel 1 would be `0x90`. The next byte of a note on message is the note number. Middle C is 60 in decimal, so the byte would be `0x3C`. The last byte is the velocity of the note (or how loud it will be played). To make the note as loud as possible we can send 127 in decimal or `0x7F`. All together, sending `0x90, 0x3C, 0x7F` will play Middle C as loud as possible on channel 1.

To stop playing a note you would send a Note-off message. The Note-off message type contained in the MIDI 1.0 spec is 8 in hex, so `0x80` would be the Note-off message corresponding to the Note-on message described above.

Other MIDI Messages

Other standard MIDI messages include:

- **Continuous Control (CC) (`0xB0`)** contain channel-wide expression information. A CC message contains bytes for status, CC number, and (7-bit) CC value.

- **Registered Parameter Number and Non Registered Parameter Number (RPN/NRPN)** are a compound sequence of CC messages that can affect pitch bend sensitivity, the tuning of the entire MIDI device, or anything else as defined by the manufacturer. RPN messages are explained in more detail in Section 2.2

- **Polyphonic Pressure (`0xA0`)** sends a separate 7-bit pressure value per key when a pressure is applied to a key after a note is begun, called aftertouch.

- **Channel Pressure (`0xD0`)** sends a single aftertouch value per channel, and only has one data byte representing how hard the key is pressed after the note was struck.
- **Pitch Bend (`0xE0`)** is unique in that it contains a 14-bit value which combines the values of the two data bytes after the status byte.

System Exclusive Messages

The specification also allows devices to receives messages that are longer than three bytes as System Exclusive (Sys-Ex) messages. These Sys-Ex messages can be anything from device firmware updates to instrument preset information.

2.2 Limitations of MIDI

While incredibly useful, the simplicity of the MIDI specification limits the expressivity of MIDI instruments. Many musicians believe that music generated exclusively from MIDI data may sound bland, robotic, and ultimately not very expressive due to it being an event based specification unable to encode all of the details of a human performance [19] [14]. This has encouraged developers and in some cases the MMA to create their own extensions to the specification to allow vibrato [28], micro-tonal notes [5] [17], single note pitch bend [8], and many more musical techniques to be used with MIDI.

Due to the desire for simplicity in the specification as well as the technical limitations of the time, MIDI has its limitations.

Keyboard Bias

While sending keyboard information over a cable has been the largest use case of MIDI from a performance aspect [23], music that relies on notes outside of the standard western 12-tone equal temperament standard [2] and music played by non-keyboard instruments such as trumpets, flutes, or guitars are not well represented in the MIDI format [10]. To allow for non-equal temperament applications the MMA created the MIDI Tuning Standard in 1991[17], which allows users to individually set the frequencies of the 128 possible note numbers. It was used by Yamaha and a couple other manufacturers, but after a couple of years, manufacturers stopped including it, claiming that there wasn't a market for it [21].

Controller Value Resolution

In MIDI messages, the command bytes always have the first bit set to 1 and the data bytes always have the first bit set to 0. Because of this, MIDI only allows 7 bits of information (128 different values) for sending data. While this is enough for encoding note numbers in the 12-tone standard and works for encoding volume information, when trying to encode something like a filter sweep from 100Hz to 10kHz, 128 values is just not enough [3] [10]. MIDI addresses this in several ways.

The pitch bend message natively supports 14 bit resolution and accomplishes this by using the two data bytes for encoding pitch bend data.

For velocity, control change (CC) message 88, the High Resolution Velocity Prefix Message, was created to allow for 14 bit resolution of velocity. By sending a message including the lower 7 bits before sending a Note On message with the higher 7 bits, 14 bit resolution can be achieved [4].

For general CC messages, 14 bit resolution can also be achieved through expanded CC messages sent in pairs of a lower 7 bits and a higher 7 bits. This works by having CC numbers 0-31 adjust the high 7 bits and CC numbers 32-63 adjust the respective low 7 bits which may not allow for continuous and precise values as found with the Continuum [20]. As the higher 7 bits of the arrives, the lower 7 bits are assumed as zero until the lower 7 bits arrives which can lead to a glitch in the audio output when moving from a value like `0x1401` to `0x137f`, which should be down by one step, but will cause an intermediate `0x1300` value.

Similarly to expanded CC messages, RPN and NRPN messages can set 14 bit of data using a series of CC messages. To send an RPN message,

1. a MIDI controller sends two CC message with CC numbers 100 (`0x64`) and 101 (`0x65`) where both of the CC data fields encode the type of RPN message and have the same value. For example `0xB0 0x64 0x00 0xB0 0x65 0x00`, are the first two messages of the RPN for changing the pitch bend range on the MIDI

device where `0xB0` means that this is a CC message on channel 1 and `0x00` is the RPN number for pitch bend sensitivity.

2. Next one or two CC messages are sent to set, increment, or decrement the RPN value specified in the previous messages. To set the value, CC message number 6 will set the high 7 bits (to set the coarse value) and CC number 38 will set the low 7 bits (to set the fine value). For example if the next two messages sent were `0xB0 0x06 0x02 0xB0 0x26 0x03`, the pitch bend range will be ± 2 semitones and 3 cents. CC message 96 will increment the data and CC message 97 will decrement the data.

3. Finally, to end a RPN message, a controller sends CC messages number 100 and 101 with the data values set to 127 (`0x7F`). That would look like this: `0xB0 0x64 0x7F 0xB0 0x65 0x7F`.

This means it takes up to 6 CC messages to send 14 bits of data using RPNs and NRPNs.

Unidirectional Communication

MIDI is a one way communication protocol. This means that a MIDI device has no information about the device that it is sending information to. Two way communication requires two MIDI cables (one output and one input) and there is no standard for how two way communication would work in MIDI 1.0.

2.3 Per Note Expression

The largest modification of the MIDI 1.0 standard from a performance point of view is support for per-note expression, deserving of its own section. The original MIDI specification does not support per-note expression, specifically pitch bend. Pitch bend works best in a monophonic context where only one note is playing at a time. When playing multiple notes at once on a channel, the pitch bend message will affect all notes on that channel. This limitation of MIDI affects channel-wide MIDI messages such as pitch bend and control change messages.

This was unfortunate for instrument designers and MIDI developers since there was no intuitive way to emulate bending a single note like you would on a guitar or get MIDI to accurately react to the expressiveness of controllers such as the Haken Continuum from Lippold Haken [1]. In response to this, many music manufacturers developed systems for their instruments where each note's parameters (pitch, timbre, volume) were individually controllable [9]. Initially many manufacturers used a mix of MIDI and their own protocols, but in 2016 Roli announced a specification for Multi-Dimensional Polyphonic Expression that worked within the context of MIDI so that there could be a standard for per-note expression.

In 2018, this specification became MIDI Polyphonic Expression (MPE). MPE puts each note played on its own individual MIDI channel, allowing up to 15 note polyphony[2] using one of the 16 channels as a master channel that affects all other channels. This and similar methods have been used prior to the ratification of MPE, but it is now an official part of the MIDI specification.

The primary use-case for MPE has been to allow multidimensional controllers to control multiple parameters of individual notes using pitch bend (x-axis), channel pressure (z-axis), and CC messages such as CC #74 (y-axis) for more dimensions of per-note control. This has allowed for standard implementations of non-keyboard controllers such as the Eigenharp Alpha from John Lambert and Eigenlabs[3] or the Linnstrument from Roger Linn [12] [8].

2.4 Conclusion

Since 1983, there have only been a couple of addenda that address MIDI's limitations, including the high resolution velocity prefix (CA-31) [4], updates to the MIDI Tuning Standard [17] to make it easier to define scales outside of the 12-tone scale, and most importantly MIDI Polyphonic Expression.

[1]Can be found at https://www.hakenaudio.com/

[2]Higher note polyphony works in MPE, but all notes on the same channel will be affected by channel wide messages such as pitch bend sacrificing per-note control.

[3]Can be found at http://www.eigenlabs.com/

These solutions stay backwards compatible with the original MIDI specification, but are not always implemented in new MIDI devices or software. Only a handful of controllers support CA-31 and many DAWs do not support the MIDI Tuning due to limited support of Sys-Ex messages. While MPE was ratified and added to the MIDI specification in 2018, the creation of insturments which support it is still on-going. Just this year in 2023, Ableton announced Drift, their first MPE synth as well as MPE support for some of their other plugins [13] and Logic added MPE support for some of their plugins [26]. It seems like we're moving towards a future with more expressive controllers and software implementing specifications to support these controllers, but what if there was another new specification that allowed for even more control.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

What is MIDI 2.0

Announced in 2020, MIDI 2.0 is an update to the MIDI specification that address the limitations of the original MIDI specification (MIDI 1.0) and leverages modern data communication protocols for a more modern specification. MIDI 2.0 aims to expand the set of tools musicians and developers have for processing and creating music while leaving room in the specification for future developments in music technology, while also remaining backward compatible with existing MIDI 1.0 devices. The new specification addresses some of the limitations of the original MIDI specification by adding methods for two way communication between MIDI devices using MIDI Capability Inquiry (MIDI-CI) messages to send and receive information about the devices. The specification also utilizes a new data format called the Universal MIDI Packet (UMP) which allows for data values with higher resolution and more expressivity than MIDI 1.0 messages.

3.1 MIDI Capability Inquiry

This thesis does not focus on MIDI Capability Inquiry, but gives an overview of its current state. MIDI specification M2-101 [16] describes MIDI Capability Inquiry, or MIDI-CI. MIDI-CI defines a mechanism for MIDI devices to communicate bidirectionally and gives the devices extended MIDI capabilities. These include:

- Profile Configuration, a system for setting up a set of rules for how a MIDI device responds to specific MIDI messages.

- Property Exchange, a system for exchanging properties about a MIDI device such as a list of controllers and destinations, a list of programs with names, or other metadata.
- Process Inquiry, a system for discovering the state of specific MIDI messages such as System Messages or Note Data Messages [1].

3.1.1 The Discovery Transaction

MIDI-CI works using Transactions that include an Initiator Device sending an Inquiry message and a Responder device replying to the Inquiry. The first Transaction between MIDI devices to setup the MIDI-CI dialogue is the Discovery Transaction.

When a MIDI device powers on, it must create a new random 28 bit ID for itself called an MUID. This device can now send a Discovery Inquiry Message in the form of a MIDI 1.0 System Exclusive Message, as shown in figure 3-1. A detailed description of the contents of this message are as follows:

1. The first byte of a Sys-Ex message is `0xF0`.
2. (`0x7E`) shows that this is the start of a Universal System Exclusive Message.
3. Next is the device ID the message is sent to which in the case of a Discovery Message is the whole MIDI port (`0x7F`)
4. The next two bytes indicate what kind of Universal System Exclusive Message we're sending. This is a MIDI-CI message (`0x0D`) and a Discovery Message (`0x70`).
5. Next is the MIDI-CI Message Version. We can use `0x01` for version 1 of MIDI-CI.
6. The next four bytes are the MUID of our device (Least Significant Byte first).

[1]Process Inquiry was added to the specification in November 2022 [7] and there is very limited information about it.

7. The next four bytes are the MUID of the destination of the message. Since this is a Discovery Message, we want to send it to everything on the network, so we use the Broadcast MUID (`0x7F 0x7F 0x7F 0x7F`).

8. The next four fields are information about the device. We can use (`0x7D 0x00 0x00`) as the device manufacturer (this ID is generally used for educational purposes). We can use anything for the device family and family model number, let's use `0x01 0x00` for the Device Family and `0x01 0x00` for the Device Family Model Number as well. For the Software Revision Level, we can use a version number for this, so I'll use `0x00 0x00 0x00 0x01`

9. Next, a bitmap is used to identify what MIDI-CI messages are supported by the device. A device that supports Profile Configuration and Property Exchange will use `00001100` (`0x0C`) where the 5th bit represents supporting Property Exchange and the 6th bit represents supporting Profile Configuration.

10. The next bytes are the maximum size of System Exclusive Message that our device can receive. This size must be at least 128 bytes to support MIDI-CI and 512 bytes to support Profile Configuration and Property Exchange, so I'll set it to 512 (`00 02 00 00`)

11. The last byte (`7F`) states the ends of the Universal System Exclusive Message.

Replies to Discovery Message

After sending that packet, Device A waits at least 3 seconds for all Replies to the discovery message. Any device on the network can send a Reply to Discovery Message in response to the Discovery Message.

The Reply to Discovery Message format is very similar to the Discovery Message. We adjust the Sub-ID#2 to 71 to indicate that this is a reply message. We include the device's MUID as the Source MUID and we use the initiator's MUID as the destination MUID. We can build the rest of the packet the same way as we built the Discovery Message.

Hex Value	Parameter
F7	Start of System Exclusive Message
7E	Universal System Exclusive Message
7F	Destination Channel 7F = Entire Port
0D	Universal System Exclusive SubID#1 0D = MIDI-CI Message
70	Universal System Exclusive SubID#2 70 = Discovery
01	MIDI-CI Version Number
xx xx xx xx	Source MUID
7F 7F 7F 7F	Destination MUID 7F 7F 7F 7F = Broadcast to All Devices
7D 00 00	Device Manufacturer 7D 00 00 = Educational Purposes
00 01	Device Family Number
00 01	Device Model Number
00 00 00 01	Software Version Number
0C	Capability Inquiries Supported (bitmap) 00001100 = Supports Profile Configuration and Property Exchange
00 02 00 00	Receivable Maximum SysEx Message Size
F7	End of System Exclusive Message

Figure 3-1: Example Discovery Inquiry Message

Hex Value	Parameter
F7	Start of System Exclusive Message
7E	Universal System Exclusive Message
7F	Destination Channel 7F = Entire Port
0D	Universal System Exclusive SubID#1 0D = MIDI-CI Message
71	Universal System Exclusive SubID#2 71 = Reply to Discovery
01	MIDI-CI Version Number
xx xx xx xx	Source MUID
yy yy yy yy	Destination MUID
7D 00 00	Device Manufacturer 7D 00 00 = Educational Purposes
00 01	Device Family Number
00 01	Device Model Number
00 00 00 01	Software Version Number
0C	Capability Inquiries Supported (bitmap) 00001100 = Supports Profile Configuration and Property Exchange
00 02 00 00	Receivable Maximum SysEx Message Size
F7	End of System Exclusive Message

Figure 3-2: Reply to Discovery Inquiry Message

After this transaction, the two devices are aware of each other on the network and can send other MIDI-CI Inquiries such as a Profile Inquiry or a Property Capabilities Inquiry.

3.1.2 Profile Configuration

MIDI specification M2-102 defines Profiles in MIDI-CI [6]. A Profile in MIDI-CI is a defined set of rules for how a MIDI device will respond to a chosen set of MIDI messages. This is similar to General MIDI for MIDI 1.0 but much more customizable. In General MIDI, the 128 program numbers are specified for specific instruments (1 is Acoustic Grand Piano, 74 is flute, 126 is Helicopter, etc) rather than arbitrary instruments defined by the manufacturer. So, when a device sends a program change message 74, the receiving device will play flute sounds [1].

In MIDI 2.0 the Profile specification document defines a receiver device's implementation of specific MIDI messages. For example, a profile for a synthesizer could define control messages for the attack, sustain, decay, and release and how Note On and Off messages respond to these parameters. Within the specification document, there may also be implementation requirements for the receiver such as Minimum Polyphony required, number of MIDI channels supported, or other Non-MIDI data types supported for a profile to work with a receiver.

Note that CC Messages should not gain new definitions and RPN or NRPN messages should control parameters not previously defined. For example, General MIDI 2 defines Reverb Level on CC#91 and it would be beneficial for a Profile that contains controls for reverb to use the same CC. For increased functionality or resolution of an already defined control message, profiles may assign the parameter to a new RPN with approval from the MMA/AMEI or a new NRPN without specific approval.

Profiles may define several levels of compatibility. Within a Profile specification are the set of features that are the minimum requirement to support a Profile and, if necessary, a set of extended features available for a device to utilize within that profile. A partial specification may also be used if a device generally supports a Profile but lacks some part of the defined Minimum Requirement.

Profiles are organized into three categories:

- **Feature Profiles** define features and MIDI implementation requirements that apply across a wide range of musical instrument types such as Per Note Experession, Zone Key Configuration, Real Time Direct Pitch Control, and more. Feature Profiles can also define features that record, edit, or modify MIDI data such as Arpeggiators, Sequencers, or other Rhythm/Music generators. Feature profiles can also define features like Lighting controls, Drone Flight controls, Video Effects, Mixer Controls, DAW Software controls, etc.

- **Instrument Profiles**, much like General MIDI, can define different instruments or devices that Note On/Off messages would adhere to such as Piano, Strings, Brass, Drums, Subtractive Synths, FM Synths, etc.

- **Effect Profiles** define features for different effect types such as Reverb, Chorus, Compressor, Distortion, Delay, etc.

Profile Inquiry Messages

For two MIDI 2.0 devices to utilize Profiles, they first must communicate using MIDI-CI. The first time MIDI 2.0 devices establish a MIDI-CI connection, after the Discovery Transaction they can send inquiry about Profiles. An example Profile Inquiry Message is shown in Figure 3-3. The inquiry can be on the entire MIDI port or on a specific channel.

The receiver responds with a Reply Message that includes a list of Profile IDs that are currently enabled and a list of Profile IDs that are currently disabled, as shown in figure 3-4.

Profile IDs are defined by 5 bytes. If the profile is defined or adopted by AMEI and MMA, they are Standard Defined Profiles and the 5 bytes are `0x7E` to denote that it is a Standard Defined Profile, 1 byte for the Profile Bank, 1 byte for the Profile Number, 1 byte for the Profile Version, and 1 byte for the Profile Level. If the Profile is defined by anyone else, the first three bytes are the Manufacturer SysEx ID, and

Hex Value	Parameter
F7	Start of System Exclusive Message
7E	Universal System Exclusive Message
zz	Destination Channel 7F = Entire Port 00-0F = Channels 1-16
0D	Universal System Exclusive SubID#1 0D = MIDI-CI Message
20	Universal System Exclusive SubID#2 20 = Profile Inquiry
01	MIDI-CI Version Number
xx xx xx xx	Source MUID
yy yy yy yy	Destination MUID
F7	End of System Exclusive Message

Figure 3-3: Profile Inquiry Message

Hex Value	Parameter
F7	Start of System Exclusive Message
7E	Universal System Exclusive Message
zz	Destination Channel 7F = Entire Port 00-0F = Channels 1-16
0D	Universal System Exclusive SubID#1 0D = MIDI-CI Message
21	Universal System Exclusive SubID#2 21 = Reply to Profile Inquiry
01	MIDI-CI Version Number
xx xx xx xx	Source MUID
yy yy yy yy	Destination MUID
01 00	Number of Currently Enabled Profiles Least Significant Byte first
7D 00 00 00 01	Profile ID
00 00	Number of Currently Disabled Profiles Least Significant Byte first
F7	End of System Exclusive Message

Figure 3-4: Reply to Profile Inquiry Message

the last two bytes can be used freely by the manufacturer. In Figure 3-4, the Profile ID uses the Educational Manufacturer SysEx ID and the profile number is set to 1.

There are also messages to Enable Profiles, Disable Profiles, Respond to Enable, Respond to Disable, as well as a Profile Specific Data Message to send an arbitrary amount of data to a specific Profile.

3.1.3 Property Exchange

Property Exchange is used to Discover, Get, and Set properties of MIDI 2.0 devices such as device configurations, list of controllers, list of programs, and other metadata. Property Exchange provides generalized access to device properties which can enable devices to auto map controllers, change states, provide visual information about a device, and more without special software.

After a MIDI-CI Discovery Transaction, the MIDI devices will have exchanged their MUIDs, Capabilities (support Profiles and/or Property Exchange), Manufacturer SysEx IDs, and Device Information. Then, an Initiator device can perform an Inquiry of Property Exchange Capabilities, shown in Figure 3-5. This inquiry request includes which channel the message is sent over, the source and destination MUIDs, and the number of simultaneous property exchange requests supported.

The responder device responds with almost the same message (Figure 3-6), confirming that it can exchange property exchange messages and communicating its own number of Simultaneous Property Exchange Requests supported.

Then an Initiator device can perform an Inquiry: Get Property Data (Figure 3-7) with the "ResourceList" Resource to obtain the list of resources that the Responder device has that the Initiatior device can request information about.

The Get Property Data message requires a Request ID, a number from 0 to 127, that is used to associate a reply to the inquiry and allow a device to support simultaneous property exchange requests.

The Get Property Data message also allows for chunking. If the data requested cannot fit inside one SysEx message, it can be sent in multiple chunks each with the same Request ID.

Hex Value	Parameter
F7	Start of System Exclusive Message
7E	Universal System Exclusive Message
zz	Destination Channel 7F = Entire Port 00-0F = Channels 1-16
0D	Universal System Exclusive SubID#1 0D = MIDI-CI Message
30	Universal System Exclusive SubID#2 20 = Property Exchange Capabilities Inquiry
01	MIDI-CI Version Number
xx xx xx xx	Source MUID
yy yy yy yy	Destination MUID
01	Number of Simultaneous Property Exchange Requests Supported
F7	End of System Exclusive Message

Figure 3-5: Property Exchange Capabilities Inquiry Message

Hex Value	Parameter
F7	Start of System Exclusive Message
7E	Universal System Exclusive Message
zz	Destination Channel 7F = Entire Port 00-0F = Channels 1-16
0D	Universal System Exclusive SubID#1 0D = MIDI-CI Message
31	Universal System Exclusive SubID#2 20 = Reply to Property Exchange Capabilities Inquiry
01	MIDI-CI Version Number
xx xx xx xx	Source MUID
yy yy yy yy	Destination MUID
01	Number of Simultaneous Property Exchange Requests Supported
F7	End of System Exclusive Message

Figure 3-6: Reply to Property Exchange Capabilities Inquiry Message

Hex Value	Parameter
F7	Start of System Exclusive Message
7E	Universal System Exclusive Message
zz	Destination Channel 7F = Entire Port 00-0F = Channels 1-16
0D	Universal System Exclusive SubID#1 0D = MIDI-CI Message
30	Universal System Exclusive SubID#2 34 = Get Property Data Inquiry
01	MIDI-CI Version Number
xx xx xx xx	Source MUID
yy yy yy yy	Destination MUID
00	Request ID
nn nn	Length of Following Header Data
data	Header Data
01 00	Number of Chunks in Message Least Significant Byte first
01 00	Number of THIS chunk Least Significant Byte first
00 00	Length of Property Data Get Property Data Inquiry has no Property Data
F7	End of System Exclusive Message

Figure 3-7: Get Property Data Inquiry Message

Hex Value	Parameter
F7	Start of System Exclusive Message
7E	Universal System Exclusive Message
zz	Destination Channel 7F = Entire Port 00-0F = Channels 1-16
0D	Universal System Exclusive SubID#1 0D = MIDI-CI Message
30	Universal System Exclusive SubID#2 35 = Reply to Get Property Data Inquiry
01	MIDI-CI Version Number
xx xx xx xx	Source MUID
yy yy yy yy	Destination MUID
00	Request ID
nn nn	Length of Following Header Data
data	Header Data
01 00	Number of Chunks in Message Least Significant Byte first
01 00	Number of THIS chunk Least Significant Byte first
nn nn	Length of Property Data Get Property Data Inquiry has no Property Data
data	Property Data
F7	End of System Exclusive Message

Figure 3-8: Reply to Get Property Data Inquiry Message

For a Get Property Data Inquiry for the "ResourceList" Resource, the header data is in a JSON format {"resource": "ResourceList"} and the Property Data is empty (as currently defined in MIDI-CI, a Get Property Data Inquiry has no Property Data, so the Length of the Property Data should be set to 0).

The responder device responds with a list of resources that the Initiator Device can request using the Get Property Data Message.

The Header Data of a Reply to Get Property will have a status message in JSON format similar to HTTP status messages like {"status": 200}. The Property Data will have a list of resources that the device supports. If the responder device has 3 resources: DeviceInfo, ChannelList, and CMList, the property data will look like

```
[
    {"resource": "DeviceInfo"},
    {"resource": "ChannelList"},
    {"resource": "CMList"}
]
```

These two inquiries and two replies are the only ones required to support Property Exchange.

It is recommended but optional for a device to support the Get Property Data message with the "DeviceInfo" resource and the "ChannelList" resource to get information about the device and a list of channels that the device supports.

DeviceInfo

An inquiry for DeviceInfo would have Header Data that looks like {"resource": "DeviceInfo"}. The Reply would have Header Data with status and the Property Data might look like this

```
{
    "manufacturerId": [125,0,0],
    "manufacturer": "Educational Use",
    "familyId": [0,0],
    "family": "Example Range",
    "modelId": [48,0],
    "model": "Example Pedal",
    "versionId": [0,0,1,0],
    "version": "1.0"
}
```

These are the required fields in response to a DeviceInfo Inquiry. The manufacturerId denotes the SysEx ID. The familyId, modelId, and versionId can group devices however the manufacturer sees fit.

The response to a DeviceInfo Inquiry can also include a serialNumber as well as links which show other resources that can be requested or set that affect the overall settings of a device.

The above Property Data could have been expanded to

```
{
    "manufacturerId": [125,0,0],
    "manufacturer": "Educational Use",
    "familyId": [0,0],
    "family": "Example Range",
    "modelId": [48,0],
    "model": "Example Pedal",
    "versionId": [0,0,1,0],
    "version": "1.0"
    "serialNumber": "12345678",
    "links": [
        {"resource": "X-SystemSettings"}
    ]
}
```

ChannelList

An inquiry for ChannelList would have Header Data that looks like {"resource": "ChannelList"}. The Reply would have Header Data with status and the Property Data might look like this

```
 [
    {
        "title": "Simple Pedal",
        "channel": 1,
        "links": [
            {"resource": "CMList", "resId": "all"}
        ]
    }
```

```
]
```

where channel 1 describes a Simple Pedal that can be affected by a CMList resource. The CMList resource can return different things, so there is also a resId identifier to select the desired data.

Get Resources

Based on the list of resources a Responder device, an Initiator device can Get or Set information about a resource from the Responder device using the Get Property Data and Set Property Data messages.

Subscribe to Resources

An Initiator device can also Subscribe to a resource if a Responder declares the resource as subscribable and be notified when that resource changes using the Subscribe Property Data message.

The Initiator may start a subscription by sending Header data that includes the resource and the command to start {"command": "start", "resource": "Current Mode"}. The Responder replies to that message with Header data that includes status and a unique subscribeID for the subscription between this Initiator and Responder device {"status": 200, "subscribeId": "sub32847623"}. Then when the resource changes, the Responder can send a subscription message with the command of partial if some part of the resource changed, or full if the entire resource changed. For example Header data could be {"command": "full", "subscribeId": "sub32847623"} and the Property data could be "multichannel". The Initiator may end a subscription the device no longer needs the resource to be synced by sending Header data that includes the command to end and the subscribeId {"command": "end", "subscribeId": "sub32847623"}

3.2 Universal MIDI Packet

Specification M2-104 [25] describes Universal MIDI Packets. UMPs are one, two, three, or four 32 bit words that currently encode Utility Messages, System Real Time Messages, System Common Messages, MIDI 1.0 Channel Voice Messages, MIDI 2.0 Channel Voice Messages, 64 bit Data Messages, and 128 bit Data Messages. Each UMP has fields for Message Type, Group, and Status. The first four bits of every UMP define its Message Type as one of the six above. With four bits, there is room for more message types to be defined by the MMA/AMEI in the future.

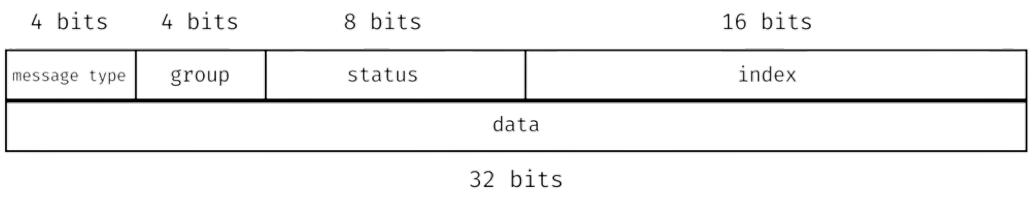
3.2.1 Groups

In MIDI 2.0, Groups are an expansion of MIDI channels [24] and work similarly to how MIDI cables work in USB MIDI 1.0. In the MIDI 2.0 USB specification, the 16 groups are interleaved into a MIDI Stream. Each group is separate from all other groups in terms of response to System Messages. Each group has 16 channels and all channels within a specific group are affect by channel wide MIDI messages.

3.2.2 MIDI 2.0 Channel Voice Messages

MIDI 2.0 Channel Voice Messages are the type of message that MIDI 2.0 devices will send to encode musical information. They are 64 bit messages that include the 4 bit message type, 4 bit group field, 4 bit status field for the type of message, 4 bit channel field, a 16 bit index field, and a 32 bit data field. The specification includes MIDI 2.0 versions of all of the original MIDI 1.0 Channel Voice messages as well as some new messages.

Figure 3-9: 64 bit MIDI 2.0 Channel Voice Message



In the following list we itemize all defined MIDI 2.0 Channel Voice messages and include details on how they are different from their MIDI 1.0 counterparts

- The **Note On** and **Note Off** Messages expands the velocity from 7 to 16 bits and adds attribute type and attribute data fields to address more properties than a MIDI 1.0 Note Off or On message. As of now, there are only a few defined attribute types including No Attribute, Manufacturer Specified, Profile Specified, and Pitch 7.9

- The **Poly Pressure** Message expands the data from 7 bits to 32 bits.

- The **Channel Pressure (Aftertouch)** Message expands the data from 7 bits to 32 bits.

- The **Control Change** Message expands the data from 7 bits to 32 bits.

- The **Pitch Bend** Message expands the data from 14 bits to 32 bits.

- The **Program Change** message combines the Program Change and Bank Select messages from MIDI 1.0 into a unified message with the same number of programs and banks as the original messages.

- The **Registered Controller** and **Assignable Controller** Messages replace the set of CC messages required for RPN and NRPN messages. These new messages are each organized into 128 banks and 128 controllers which result in 16,384 different controllers. The MMA/AMEI defines the Registered Controller Messages and leaves some messages undefined for future use. The Assignable Controller Messages can be assigned to anything.

- Also included are **Relative Registered** and **Assignable Controller** Messages to make relative increases and decreases to Registered Controller and Assignable Controller values.

- **Registered Per-Note Controllers (RPNC)** and **Assignable Per-Note Controller (APNC)** are new in MIDI 2.0. These messages each have 256 different controllers to change how a note is output. For these messages, it makes sense to refer to notes by an index and set their pitch externally using the RPNC Message #3: Pitch 7.25 or the Note On message with Attribute #3: Pitch 7.9. This is explained in more detail in section 3.2.3. The MMA/AMEI defines the Registered Per-Note Controller Messages and leaves some undefined for future use. The Assignable Per-Note Controller Messages can be assigned to anything.

- The **Per-Note Management** Message allows for independent control of multiple notes from the same note index. For example, let's say a note with index zero is playing and an RPNC message is used to pan it to the left. After a note off message at index zero, you want to play another note on index zero and pan it to the right, but not change the pan of the original note as it decays. You can send a per note management message to detach the RPNC message from all previous notes on index zero, play a new note on index zero, and then send a new RPNC message to pan right.

- The **Per-Note Pitch Bend** Message is new in MIDI 2.0 and applies pitch bend to a specific note. Like the Registered and Assignable Per-Note Controller Messages, this message works best when the notes refer to an index.

There are also updates to System Messages, Clock Messages, and System Exclusive Messages, but these messages land outside of the scope of this thesis.

3.2.3 Expansions from MIDI 1.0

Of the updates to the MIDI specification, many increase the resolution of data fields which allow for more detail in MIDI production and performance as well as more realistic sounds from MIDI playback. High resolution virtual instruments such as Ivory 3 German D [2], a virtual grand piano instrument that provides rich and realistic piano tones, have already been created to utilize these expanded data fields. Depending on

[2] Can be found at https://synthogy.com/index.php/products/software-products/ivory-3-german-d

the velocity value, Ivory generates clear high fidelity tones using layering sampling techniques from many different recording of a piano.

In addition, MIDI 2.0 introduces note attributes, per-note control, and new pitch representations.

Note Attributes

The attribute type field in Note On and Off messages allow for an extra arbitrary data field per note played. For example, the attribute could define the waveform of a synth with four possible values for the data field (Sine, Square, Triangle, Saw). This example is implemented in Section 4.5.

Per Note Controller Messages

The new Registered and Assignable Per Note Controller Messages act similarly to Control Change messages except rather than affecting an entire channel, RPNC and APNC messages affect a single note index. In Appendix A of the UMP format specification [25], the MMA specifies a list of RPNC numbers that correspond to functions such as Modulation, Breath, Pan, Reverb, Chorus, etc. All of these messages affect specific notes which allows for per-note expressivity that was previously only possible using MPE.

Pitch Representations

MIDI 2.0 introduces two new pitch representations: Pitch 7.9 and Pitch 7.25. Currently the MMA/AMEI has defined Pitch 7.9 as attribute #3 which allows for 16 bits to represent the pitch of the note in the attribute data field. The first 7 bits denote the semitone as it works for a Note On message without an attribute and the last 9 bits denote a fraction of a semitone with a resolution of $\frac{1}{512}$ semitones (approximately 0.2 cents). When using this attribute, the note number field acts as a note index and doesn't encode any scale or pitch. Pitch 7.25 is defined as RPNC #3. It sets the pitch of a note using the first 7 bits as the semitone and the last 25 bits as a fraction of a semitone with resolution of $\frac{1}{33554432}$ semitones (approximately $2 \cdot 10^{-7}$ cents).

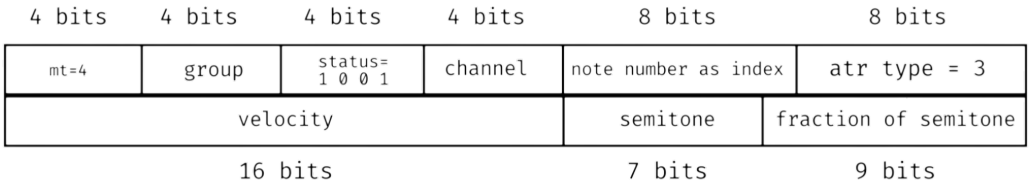


Figure 3-10: Pitch 7.9 in a Note On Message

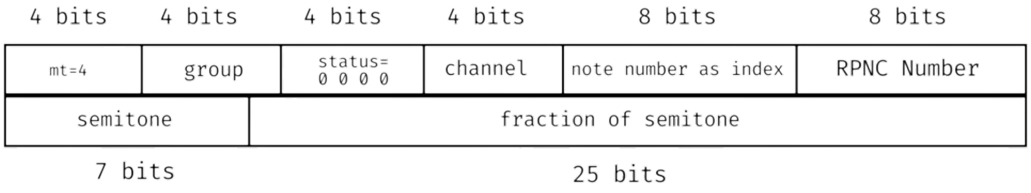


Figure 3-11: Pitch 7.25 in a Registered Per Note Controller Message

3.2.4 Ways to use the new messages

To give an example of how to use the new messages we can look at various ways to define the pitch of a note.

- Like in MIDI 1.0, the Note On message can be used with the pitch semitone as the note number and a velocity value [3]:

  `Note On 60, Vel:` $2^{16}-1$`, Attr Type: 0, Attr Data: 0`

  This would play the loudest possible Middle C.

- The Note On Message could also be used with attribute type #3 and the pitch information in the attribute data in the Pitch 7.9 format. In this case, the pitch of the note would be determined by the Pitch 7.9 value, and the MIDI note number would as the note index.

  `Note On 0, Vel:` $2^{16}-1$`, Attr Type: 3, Attr Data: 60 « 9`

  This would sound the same as the previous message.

- You could utilize the fractional semitone:

  `Note On 0, Vel:` $2^{16}-1$`, Attr Type: 3, Attr Data: 60 « 9 | 1 « 8`

  This would sound a note halfway between Middle C and C#.

[3]With 16 bit resolution for velocity, $2^{16}-1$ is the largest decimal value in 16 bits

- You could use the RPNC message to set the pitch in the Pitch 7.25 format:

  `RPNC 3, Note Number: 0, Data: 60 ≪ 25`

  `Note On 0, Vel:` $2^{16}-1$`, Attr Type: 0, Attr Data: 0`

  This would sound the same as the first two messages.

We could also use the different defined RPNC messages to express two notes of the same pitch differently:

`RPNC 3, Note Number: 0, Data: 60 ≪ 25`

`RPNC 3, Note Number: 1, Data: 60 ≪ 25`

`RPNC 10, Note Number: 0, Data: 0`

`RPNC 10, Note Number: 1, Data:` $2^{32}-1$

`RPNC 91, Note Number: 0, Data:` $2^{32}-1$

`RPNC 93, Note Number: 1, Data:` $2^{32}-1$

`Note On 0, Vel:` $2^{16}-1$`, Attr Type: 0, Attr Data: 0`

`Note On 1, Vel:` $2^{16}-1$`, Attr Type: 0, Attr Data: 0`

Here we defined two different notes with the same pitch of 60, one note is panned to the left and one to the right, one note has full reverb and one note full chorus, and then played the notes out.

If we wanted to encode more information about a note, we can also use an attribute type of 1 and the attribute data field. For example if the attribute type encoded the waveform of the note, attribute data of 1 could use a sine wave and attribute data of 2 could use a square wave when synthesizing the note.[4]

`RPNC 3, Note Number: 0, Data: 60 ≪ 25`

`RPNC 3, Note Number: 1, Data: 60 ≪ 25`

`Note On 0, Vel:` $2^{16}-1$`, Attr Type: 1, Attr Data: 1`

`Note On 1, Vel:` $2^{16}-1$`, Attr Type: 1, Attr Data: 2`

[4]For a description of the implementation of this example see 4.5.

3.3 MIDI 2.0 and MPE

The new RPNC and APNC messages enable per-note expressivity in a cleaner format than MPE does in MIDI 1.0. Since MIDI 2.0 is backwards compatible with MIDI 1.0 (and MPE), MIDI devices that send MPE information can work within a MIDI 2.0 environment by utilizing a profile that specifies zoning and channeling splitting as MPE does; however, new MIDI devices will most likely only use the new UMP messages for improved expressivity in MIDI devices.

Chapter 4

Implementing MIDI 2.0

This section describes the implementation of a Python library that creates and sends MIDI 2.0 UMP packets using human readable Python code. The library serves to display examples of using the new MIDI messages and creative use cases of expressive musical applications that are not possible (or very difficult to implement) with MIDI 1.0.

The Python library communicates with the MIDI bus in Apple's operating system using CoreMIDI to send and receive MIDI 2.0 packets. I developed this implementation using MacOS since Apple has already developed a MIDI 2.0 API and the Mac Operating System currently supports MIDI 2.0 natively. In fact, a few applications on MacOS such as Logic and MultiTrackStudio support receiving MIDI 2.0 UMP packets. Google also developed a MIDI 2.0 API for Android, and Microsoft is currently in the working with funding from AMEI to develop an open source MIDI 2.0 driver for Windows.

While this library integrates support for CoreMIDI, the functionality of the code and the structures within the library are platform independent. When the MIDI 2.0 API for Windows is publicly available, with minor additions to communicate with the Windows API, the library will work on Windows. Code and instructions for building are on Github: `https://github.com/jshjulian/py-midi2`

4.1 Overview of Implementation

The library defines:

- Universal MIDI Packet Creation
- Communication with MacOS and CoreMIDI
- API for easily using new features such as Attributes and Pitch 7.9
- Synthesizer Playground: A creative application that utilizes Attributes and Registered Per Note Controllers.
- MIDI Keyboard: A creative application that utilizes Pitch 7.9, fractional semitones and Registered Per Note Controllers

4.2 UMP Structures

Every MIDI 1.0 and MIDI 2.0 Channel Voice message described in specification M2-104 [25] is defined by the UniversalMIDIPacket class. The MIDIMessageCreator class is used to pass in parameters corresponding to a MIDI message and return a UniversalMIDIPacket object. The UniversalMIDIPacket class has methods to convert an object to a list of 32 bit integers that can be passed to the operating system MIDI bus.

```python
    def midi2_0_note_on(note_num: int, vel:int, atribute_type:int=0, atribute_data:int=0, group:int=0, channel:int=0):
        """
        MIDI 2.0 Note On Message

        Parameters
        ----------
        note_num : int
          Note Number (7 bits)
        vel : int
          Note Velocity (16 bits)
        attribute_type : int, optional
          Atribute Type (8 bits), Default: 0 for no Attribute Type
        attribute_data : int, optional
          Attribute Data (16 bits), Default: 0 for no Attribute Data
        group : int, optional
          Group (4 bits) default: group 1
        channel : int, optional
          Channel (4 bits) default: channel 1
        """
        status = int('1001', 2) << 4 | int(channel)
        index = int(note_num) << 8 | int(atribute_type)
        data = int(vel) << 16 | int(atribute_data)
        return UniversalMIDIPacket(name="MIDI 2.0 Note On Message",
                                   message_type=4,
                                   group=group,
                                   status=status,
                                   index=index,
                                   data=data)
```

Listing 4.1: MIDI 2.0 Note On example

4.3 Communicating with MacOS

Apple provides documentation for the CoreMIDI framework and provides a CoreMIDI.h file which gives access to MIDI Sources and Destinations for sending and receiving MIDI data to and from the MIDI bus. I wrote a C extension file that receives Python objects, converts them to lists of 32 bit integers, and sends them to the MIDI bus as well as code that takes lists of 32 bit integers from the MIDI bus, converts them to Python objects, and sends them to Python. From Python, we can call a function that sends a list of 32 bit integers as MIDI data that is received by the C code and sent to the MIDI bus using CoreMIDI.

4.4 MIDI 2.0 Python API

With the ability to create UMPs and convert them into 32bit integers and send them to the MIDI bus, I built an API on top of those modules to easily send MIDI data in Python. In Code Listing 4.2, we create a MIDI2 object that has a name, source, destination, and MUID, the only things required for a fresh MIDI 2.0 device. We can use the `rpnc` method to send a Registered Per Note Controller message given parameters or we can use the `rpnc_pitch_7_25` method to specfically use Pitch 7.25.

```
from ump import MIDIMessageCreator as m
import random
class MIDI2:
  def __init__(self, name="Python MIDI 2.0"):
    self.name = name
    self.src = MIDISource(name + " src")
    self.dst = MIDIDestination(name + " dst")
    self.muid = random.getrandbits(32)
    self.muid_bytes = [
      self.muid     & int("ff", 16),
      self.muid>>8  & int("ff", 16),
      self.muid>>16 & int("ff", 16),
      self.muid>>24 & int("ff", 16)
    ]

  def rpnc(self, note_idx, num, data, group=0, channel=0):
    msg = m.midi2_0_reg_per_note(note_idx, num, data, group,
channel)
    self.src.send(msg())

  def rpnc_pitch_7_25(self, note_idx, pitch, frac_of_semitone,
group=0, channel=0):
    semitones = int(frac_of_semitone * (2**25-1))
    data = pitch << 25 | semitones
    self.rpnc(note_idx, 3, data, group, channel)
```

Listing 4.2: RPNC Pitch 7.25 example

4.5 Use Case: Synthesizer Playground

The new attribute field in Note On and Note Off messages allows each note to have an extra piece of data to determine how it will play out. This is different from MIDI 1.0 where we only can send pitch and velocity information with a Note On or Note Off message.

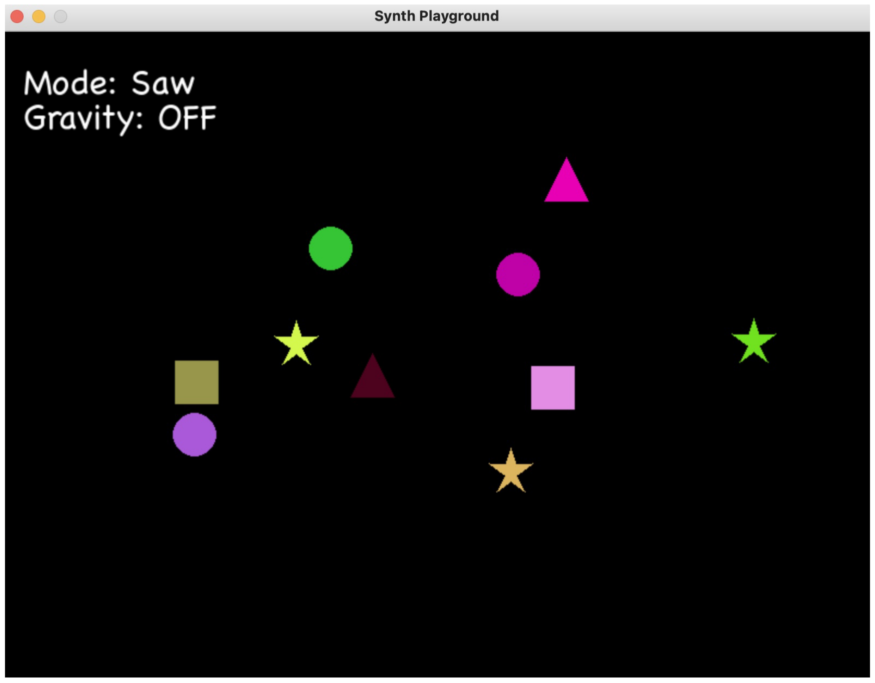
As an example of this, as the developer of a MIDI 2.0 device, I can choose to denote that the attribute data will determine the waveform of an individual synthesizer note. In this example, I specify that an attribute data value of 1 represents a Sine wave, 2 represents a Square wave, 3 represents a Triangle Wave, and 4 represents a Sawtooth wave. I can then send Note On and Note off messages with the attribute type set to 1 (which indicates that the interpretation of the attribute data is Manufacturer specified) and play notes where the attribute data specifies the waveform for that note.

This example is implemented in the "Synthesizer Playground", a Python app that uses the MIDI 2.0 library and a GUI interface for generating music (see screenshot in Figure 4-1). In this app, a user can select a waveform using the number keys: 1,2,3,4. Once selected, the user can click on the screen and create a note that uses the selected waveform. The user can remove/stop notes using the q,w,e,r keys that correspond to Sine, Square, Triangle, and Sawtooth. With the spacebar, the user can enable or disable gravity which makes all of the notes fall off the screen. As the notes fall, RPNC messages are sent to change the volume and pan of the notes.

4.6 Use Case: MIDI 2.0 Keyboard

The new MIDI messages allow for new musical systems that were not easy or intuitive to create using the MIDI 1.0 protocol. Specifically, the addition of 7.9 and 7.25 pitch representations make it very easy to play notes outside of the 12-tone scale by allowing note on messages and RPNC messages to set pitch using semitones and fractions of semitones. With the per-note controller messages, it may make sense to use the note

Figure 4-1: Screenshot from Synth Playground



number field in note on and note off messages as a note index which does not imply scale or pitch.

To demonstrate this, I created the example application "The MIDI 2.0 Keyboard". In this example we use the computer keyboard to trigger MIDI notes. The pitch of these notes can be set arbitrarily in order to create different scales by utilizing the fractional semitones of pitch 7.25. To illustrate this I created a couple of presets implementing different tunings including:

- A 24 tone equal temperament preset with the tones shown in Figure 4-2 [1].
- A 31 tone equal temperament preset
- An 18 tone equal temperament preset
- A preset that makes each row on the keyboard an octave of different lengths in equal temperament. Q-P is 10-TET, A-L is 9-TET, and Z-M is 7-TET. If we set the start of the scale to Middle C or MIDI note number 60, We can easily

[1]Original source image was accessed from `https://commons.wikimedia.org/wiki/File:Blank_BRSB_Keyboard_Layout.svg` on 05/18/23

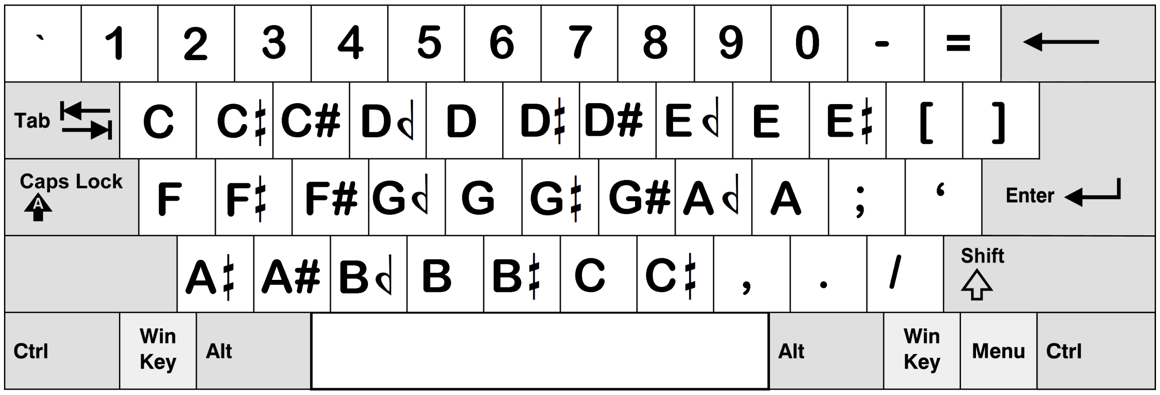


Figure 4-2: 24 TET tones shown on a keyboard

calculate the semitone and fractional semitone required for each scale by taking the note number, dividing it by the length of the scale, and multiplying by 12. For example, the letter V is at index 3 of the third row of the keyboard. It is $\frac{3}{7}$ through the row and should sound a little sharper than an E.

4.7 Conclusion

This chapter presented implementation details of a Python library I created that enables users to easily create and sends Universal MIDI Packets to the MIDI bus in the MacOS operating system using Apple's CoreMIDI framework. This chapter also walked through some examples using the Python library and the new features of MIDI 2.0 include note attributes, 7.9 and 7.25 pitch representations, and per note controllers.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 5

Analysis of MIDI 2.0

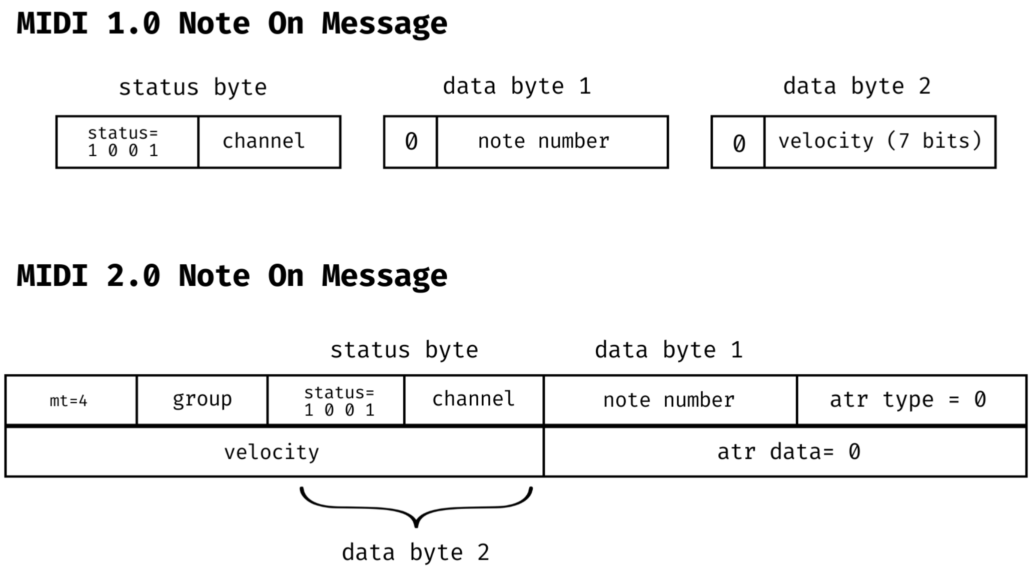
The MIDI 2.0 specification provides many new tools for production, performance, and digital instrument design. This chapter looks at how these tools will affect how digital music and digital instruments are made, with a particular focus on the impact of the Universal MIDI Packet (UMP).

5.1 Universal MIDI Packet

All of the data in a MIDI 1.0 Packet still exists in a Universal MIDI Packet with additional information included depending on the message. Status numbers for MIDI 1.0 Channel Voice Messages like the Note On message or Control Change message are the same in MIDI 2.0, and are contained within the UMP as shown in Figure 5-1. The data byte(s) also exists within the UMP.

For all UMPs, the additional information added includes expanded data fields which are covered in more detail in this section. This section also covers how UMPs allow for new MIDI messages and pitch representations, and how the per-Note controller messages allow per-note expressivity rather than only channel wide expressivity in MIDI 1.0 or up to 16 note expressivity in MPE.

Figure 5-1: MIDI 1.0 bytes in Universal MIDI Packets



5.1.1 Higher Resolution Data Format

UMPs improve on the limited data format in MIDI 1.0 by increasing the data field of most of the MIDI messages from 7 bits to 32 bits. This increase in the range of possible data values from 128 values to over 4 billion values gives more control to musicians and developers for expressing volume, pitch, timbre, and any other musical techniques.

With modern high speed communication protocols like USB, Thunderbolt, and Ethernet, MIDI developers can feel safe in terms of bandwidth when sending MIDI messages as 64 bit UMPs are comparable and possibly more compact than other streamed data such as OSC [27], another protocol for connecting sound sources and computers, and tiny compared to other streamed data formats such as video. UMPs are still much larger than MIDI 1.0 packets, so we can fit a lot more data about musical events than we could in the past.

5.1.2 Note on and Off Attribute Field

The new attribute field in Note On and Off message allows for even more data to be packed into musical events. The possibilities are endless with different drumming techniques, string articulations, ASDR, pitch information, and more. As more virtual instruments are built to support MIDI 2.0, we'll hear a wide dynamic range in audio, more realistic sounds and tones, and we will be able to more faithfully reproduce acoustic music in MIDI.

5.1.3 Pitch Representation and Micro-Tunings

Pitch 7.9 and Pitch 7.25 provide a non integer representation of pitch without using frequency values which makes it very easy to build new scales outside of the 12 tone equal temperament scale while still using the knowledge of the 12 tone scale. With $\frac{1}{512}$ of a semitone resolution with Pitch 7.9 and $\frac{1}{33554432}$ of a semitone resolution with Pitch 7.25, pitch changes can be as seamless as if we were using frequency values. Pitch 7.9 and Pitch 7.25 will make it easy to create microtonal and isomorphic keyboards such as the Lumatone Keyboard [1] or the Dodeka Keyboard [2] and integrate those natively into Digital Audio Workstations with MIDI 2.0 support.

5.1.4 More Expressivity with Per-Note Controllers

RPNC and APNC messages increase the possible expressive note polyphony from 16 using MPE to 128 by using the note number field as an index. By utilizing the 16 channels, we can have $16 \cdot 128 = 2,048$ expressive note polyphony and if we also the 16 groups, we can have $16 * 2048 = 32,768$ expressive note polyphony, which is a lot of individually modifiable notes. This along with the new pitch representations allows MIDI to move away from 12 tone scales and keyboard representations. With the ability to reference notes by an index, notes are no longer tied to their pitch and can be expressed on a note by note basis. This also allows for multiple instance of

[1]Can be found at https://www.lumatone.io/

[2]Can be found at https://www.dodekamusic.com/products/dodeka-keyboard/

the same note to be playing at the same time and each individually controllable.

5.1.5 Current State of UMPs

The specification leaves room for many more MIDI messages, attribute types, and RPNC parameters, so as the specification evolves and more people are using MIDI 2.0, I anticipate new use cases supported by new messages. As of today, very few DAWs and virtual instruments support MIDI 2.0 UMPs and of the ones that do, none of them support the new pitch representations. If, from MIDI 2.0, only UMPs were fully supported, that would already by a major improvement from the current state of MIDI.

5.2 Hopes and Possibilities for MIDI-CI

In this early stage of MIDI 2.0, MIDI-CI is still being developed. No publicly available software supports even the Discovery Transaction of MIDI-CI. I originally intended to implement MIDI-CI in my Python Library, but with no MIDI 2.0 device to communicate with, it was difficult to setup, test, and analyze. Only when we see MIDI 2.0 devices communicating with MIDI-CI can we know how it will be best used, but we can still speculate on its benefits.

5.2.1 Profiles

The most successful profile came even before MIDI 2.0 in the form of General MIDI (GM) in 1991. With General MIDI, MIDI devices are equipped with 128 different instruments as presets such that any MIDI file played back on a MIDI device using General MIDI will sound roughly the same since the program change and bank select message will choose the same type of instrument such as Honky-tonk Piano or Choir Aahs and any CC or RPN message will react as specified in the GM specification [1]. Similarly, Profiles create a shared knowledge of how MIDI devices should respond to certain messages, and with a profile specification document this knowledge can take many forms: instrument settings like General MIDI, effects setting like reverb or parametric equalizers using control change messages, or many other parameters. As the MMA and AMEI define profiles, we will have a better understanding for how profiles will be defined and how they will be used.

5.2.2 Property Exchange

Property Exchange will enable auto-configuration of MIDI devices without prior knowledge of the device or special software. The ability to send information about the properties of a MIDI device was possible in MIDI 1.0 using Sys-Ex messages. In MIDI 2.0 Property Exchange still uses Sys-Ex messages, but with the well-defined specification devices will be able to auto-map faders and knobs, send and save banks and presets, and more. Without any devices that support Property Exchange, its difficult to envision its impact, but I anticipate Property Exchange to be a very important and useful tool in the future of MIDI.

5.3 Conclusion

Some of the tools that UMPs provide exist in MIDI 1.0. Higher resolution data could be achieved with multiple CC messages or RPN/NRPN messages and alternate tunings exist using the MIDI Tuning Standard, but these solutions are rarely used in real world applications because they can be awkward and are not always supported. MIDI 2.0 puts all of these tools together, front and center.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 6

Conclusion

MIDI 2.0 aims to add functionality and increase expressivity to the MIDI specification while staying backwards compatible with the original MIDI specification. Even though it is still in the early stages of development, MIDI-CI sets up a system for MIDI devices to communicate how they work through Profile Configuration and Property Exchange. These standards will speed up workflows and our understanding of others MIDI devices as the specification for MIDI-CI becomes more refined.

The addition of the Universal MIDI Packet enables more expressivity than previous versions MIDI through the new Per-Note Controller Messages, Attribute field in Note On and Note Off messages, and new pitch representations. With higher resolution messages and new ways to form old messages, MIDI 2.0 improves on MIDI 1.0 and addresses many of the limitations of the original specification.

6.1 Contributions

While much of the MIDI 2.0 specification is still under development, this thesis covers key features of MIDI 2.0.

- Chapter 2 covers MIDI 1.0, its limitations including keyboard/12 tone bias, controller value resolution, unidirectional communication, and per note expression.

- Chapter 3 covers the MIDI 2.0 specification, how it addresses MIDI 1.0's limitations and how it improves MIDI. This includes an overview of MIDI Capability Inquiry, Profiles, and Property Exchange as well as an overview of Universal MIDI Packets and its new features including note attributes, new pitch representations, and per note controllers.

- Chapter 4 covers my implementation of MIDI 2.0 in Python focusing on UMPs. The chapter includes the Synth Playground and MIDI Keyboard as creative musical examples using note attributes, pitch 7.25, and per note controllers.

- Chapter 5 covers my analysis of MIDI 2.0, its tools, and how it will affect the future of digital music and digital instrument design.

6.2 Future Work

Since MIDI 2.0 is still fairly new, not widely adopted, and partially still in development, there will be many changes and additions to the specification as time goes on.

When Windows and Linux support for MIDI 2.0 is publicly available, I'd like to add support for those operating systems in my library.

As MIDI 2.0 devices are developed and there are more examples of MIDI-CI, I'm excited to see more research and use cases of Profiles and Property Exchange and I would like to implement MIDI-CI in my library. Especially after the MMA/AMEI defines more profiles, I would be interested in seeing the development of profiles for independent musicians and developers.

When more software implements UMPs, I will be interested in seeing what attributes independent developers use and what attributes the MMA/AMEI define in the specification. And with the increased polyphonic expression from Per-Note Controllers, additional creative applications using MIDI 2.0 can be created, allowing for computer music we've never seen before.

New ways of representing pitch allows for broader access to alternative tuning systems, I hope MIDI 2.0 enables the creation of more microtonal music and microtonal music controllers.

6.3 Conclusion

The goal of this research was to analyze and implement MIDI 2.0 in order to learn more about its capabilities, to help facilitate future research in music technology and digital instrument music design, and to support musicians in the use of the MIDI 2.0 spec as it currently stands. As a musician and developer myself, I'm excited for what MIDI 2.0 has to offer and I will definitely continue using and researching it in the future.

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

[1] *About General MIDI.* Jan. 3, 2012. URL: `https://web.archive.org/web/20120103100025/http://www.midi.org/techspecs/gm.php` (visited on 04/05/2023).

[2] *An Introduction To OSC | Linux Journal.* URL: `https://www.linuxjournal.com/content/introduction-osc` (visited on 02/23/2023).

[3] M. Bye. “MIDI Polyphonic Expression: A Bridge Between Acoustic and Electronic Instrument Experience?” MA thesis. 2018. URL: `https://www.duo.uio.no/handle/10852/62802` (visited on 02/22/2023).

[4] *CC #88 High Resolution Velocity Prefix (CA-031).* The MIDI Manufacturers Association and The Association of Musical Electronics Industry.

[5] S. Çelik. “Micro-MIDI: A Real Time, Dynamic Microtonal MIDI Application”. In: *International Journal of Engineering and Science Invention* (Sept. 2016). ISSN: 2319-6734. URL: `https://www.researchgate.net/publication/348296976_Micro-MIDI_A_Real_Time_Dynamic_Microtonal_MIDI_Application` (visited on 04/11/2023).

[6] *Common Rules for MIDI-CI Profiles.* M2-102-UM, Version 1.1. The MIDI Manufacturers Association. Feb. 2020.

[7] *Details about MIDI 2.0™, MIDI-CI, Profiles and Property Exchange (Updated November 2022).* URL: `https://www.midi.org/midi-articles/details-about-midi-2-0-midi-ci-profiles-and-property-exchange` (visited on 04/04/2023).

[8] Gaz Williams. *The Gaz Williams Show - Geert Bevin and the story of MPE.* June 2, 2021. URL: https://www.youtube.com/watch?v=SEAp_ffF8EY (visited on 02/09/2023).

[9] JUCE. *MPE: Making MIDI more expressive, Ben Supper (ROLI).* Nov. 18, 2016. URL: https://www.youtube.com/watch?v=n1Hu2LhbtbA (visited on 02/09/2023).

[10] D. Kopf. *An update to a 37-year-old digital protocol could profoundly change the way music sounds.* Quartz. Jan. 30, 2020. URL: https://qz.com/1788828/how-will-midi-2-0-change-music (visited on 04/05/2023).

[11] P. D. Lehrman. "What is MIDI?" In: *MIDI for the Professional.* Amsco Publications, 2017.

[12] R. Linn. "LinnStrument and other new expressive musical controllers". In: *The Journal of the Acoustical Society of America* 134.5 (Nov. 1, 2013), p. 4053. ISSN: 0001-4966. DOI: 10.1121/1.4830792. URL: https://doi.org/10.1121/1.4830792 (visited on 05/18/2023).

[13] *Live 11.3 is Now in Public Beta | Ableton.* URL: https://www.ableton.com/en/blog/live-113-is-now-in-public-beta/ (visited on 05/10/2023).

[14] G. Loy. "Musicians Make a Standard: The MIDI Phenomenon". In: *Computer Music Journal* 9.4 (1985), pp. 8–26. ISSN: 01489267, 15315169. URL: http://www.jstor.org/stable/3679619 (visited on 05/16/2023).

[15] P. Manning. *Electronic and Computer Music.* Fourth Edition. Oxford, New York: Oxford University Press, Mar. 27, 2013. ISBN: 978-0-19-974639-2.

[16] *MIDI Capability Inquiry (MIDI-CI).* M2-101-UM, Version 1.1. The MIDI Manufacturers Association. Feb. 2020.

[17] *MIDI Tuning Standard.* In: *Wikipedia.* Page Version ID: 1129769518. Dec. 27, 2022. URL: https://en.wikipedia.org/w/index.php?title=MIDI_tuning_standard&oldid=1129769518 (visited on 05/10/2023).

[18] R. A. Moog. "MIDI: Musical Instrument Digital Interface". In: *Journal of The Audio Engineering Society* 34 (1986), pp. 394–404. URL: `https://moogfoundation.org/wp-content/uploads/5267-1.pdf` (visited on 05/16/2023).

[19] F. R. Moore. "The Dysfunctions of MIDI". In: *Computer Music Journal* 12.1 (1988), pp. 19–28. ISSN: 01489267, 15315169. URL: `http://www.jstor.org/stable/3679834` (visited on 05/16/2023).

[20] *MPE+*. Haken Audio. URL: `https://www.hakenaudio.com/mpe` (visited on 02/09/2023).

[21] A. Reuter. *"Equal temperament is the McDonald's of tuning" – A conversation with Khyam Allami*. Passive/Aggressive. URL: `https://passiveaggressive.dk/equal-temperament-is-the-mcdonalds-of-tuning-a-conversation-with-khyam-allami/` (visited on 05/18/2023).

[22] sonicstate. *NAMM 2023 - AmeNote - ProtoZOA - MIDI 2.0*. Apr. 15, 2023. URL: `https://www.youtube.com/watch?v=IrAYZ8NmhiA` (visited on 05/16/2023).

[23] *The Complete MIDI 1.0 Detailed Specification*. version 96.1. The MIDI Manufacturers Association. 1996.

[24] The MIDI Association. *MIDI 2.0 Protocol Messages and the Universal MIDI Packet NAMM 2021 Session*. Jan. 22, 2021. URL: `https://www.youtube.com/watch?v=Kky1nlwz8-8?t=180` (visited on 04/19/2023).

[25] *Univerisal MIDI Packet (UMP) and MIDI 2.0 Protocol*. M2-104-UM, Version 1.1. The MIDI Manufacturers Association. Feb. 2020.

[26] *Use MPE with software instruments in Logic Pro*. Apple Support. URL: `https://support.apple.com/guide/logicpro/use-mpe-with-software-instruments-lgcp8f599497/mac` (visited on 05/18/2023).

[27] D. Wessel and M. Wright. "Problems and prospects for intimate musical control of computers". In: *Computer music journal* 26.3 (2002), pp. 11–22.

[28] A. Yang. *Generation of Vibrato in Expressive Performances in MIDI*. USC. 2004. URL: `http://www-classes.usc.edu/engr/ise/599muscog/2004/projects/yang/` (visited on 04/11/2023).