Exploring MIDI Specs with JUCE
CSC497 PROGRESS REPORT #1

Matthew Haussmann
UVic | MUCS | V00827158
haussmannm@gmail.coM
February 26, 2021

1. INTRODUCTION

I seek to develop a utility application which displays incoming and outgoing MIDI messages in human-readable text. This program will include a prototype implementation of Property Exchange, which is part of the recent MIDI 2.0 Capability Inquiry (MIDI-CI) specification. Property Exchange is a method for getting and setting various data (called Resources) between two devices.

The application will be implemented in the JUCE C++ framework. JUCE and the Projucer app help audio developers create multi-format audio applications with ease by handling much of the background work and many library implementations of useful functions and packages. It is able to output many plug-in types, including VST, VST3, AU, AAX, or as a standalone program. JUCE is used under a free educational license.

MIDI 2.0 (2.0) is a recent expansion to the MIDI 1.0 (1.0) specification which was first proposed in 1983.[1] There have been several updates or other expansions to the MIDI protocol over the years, such as MIDI Polyphonic Expression (MPE) or the specifications for different transports such as MIDI BLE for Bluetooth. MIDI 2.0 has been touted as the “the biggest advance in music technology in decades" by the Midi Manufacturers’ Association (MMA.[2] However, due to how recently the specifications were released, there are only a couple items currently available on the market that have any 2.0 capabilities, and the USB Implementers’ Forum (USB-IF) only released the transport specification in June 2020 making it the first and only transport that currently supports 2.0. The 2.0 specification was adopted January 19, 2020 by the MMA, only a short time before.[3]

The advances of 2.0 needed to be widely adoptable, thus 2.0 is fully backwards compatible, and achieves this through MIDI-CI. Unlike 1.0, 2.0 is a bidirectional connection and conversation between devices, which provides the infrastructure for MIDI-CI to be implemented. MIDI-CI has 3 P's: Profile configuration, Property exchange, and Protocol Negotiation. Profile configuration aims to create automatic and standard mappings between virtual instruments or effects and the devices we control them with. Property exchange can be viewed as a more detailed list of capabilities and controllable parameters and settings. Protocol negotiation is the talk between devices that decides what protocol to use, where two 2.0 devices will know what the capabilities of the other is. Upon encountering a MIDI 1.0 device, a 2.0 device will decide to speak 1.0 with that device. 2.0 has room for future expansion, with data fields left unspecified in order to future-proof the spec along with its backwards compatibility.

A new communication protocol was proposed along with 2.0, called the Universal Midi Packet (UMP), which can contain every type of 1.0 and 2.0 messages that have been specified. Another advance is the option to prepend a new Jitter Reduction (JR) timestamp to improve timing accuracy; this mechanism uses a sender-based clocking between both 2.0 devices in communication. The UMP is of various sizes depending on the message, with the largest being 128-bit. Compared to the original 1.0 message maximum of 24-bit this is a huge difference which allows for the new capabilities that 2.0 has to offer.[4]

There are expanded tuning capabilities, articulation properties, more bits for velocity (2^16-1 vs. 2^7-1 velocity values), per-note controls (e.g. per-note pitch bend), and a simplification of many old 1.0 abilities and communications, such as SysEx messages. 1.0 has 16 MIDI channels, and 2.0 introduces 16 groups of 16 channels, for a total of 256 MIDI channels. Rules have also been developed for how 1.0 and 2.0 messages are translated when one device doesn't speak 2.0. So 2.0 controllers will have greater expressive capabilities, and the framework provided by 2.0 should increase the devices’ usability for greater ease of use. Thus, the hope is more time spent in a fluid, expressive, creative mindset as opposed to dealing with technical processes and setup.

Property Exchange works with a payload of data in the form of a JSON file, and over 1.0 transport it uses SysEx messages. SysEx stands for System Exclusive, and is one of the original ways that 1.0 allows manufacturers to implement their own unique message data within a MIDI message.[5]

There are currently two developer tools that have been created to aid in development of 2.0-ready software and hardware. There are called MIDI 2.0 Scope, and MIDI-CI Workbench, both developed by individual

members of the 2.0 spec development team. However, neither of these tools are available to to public yet, and are only available to corporate members of the MMA until the MMA can "validate interoperability with real shipping products."[6]

2. GOAL

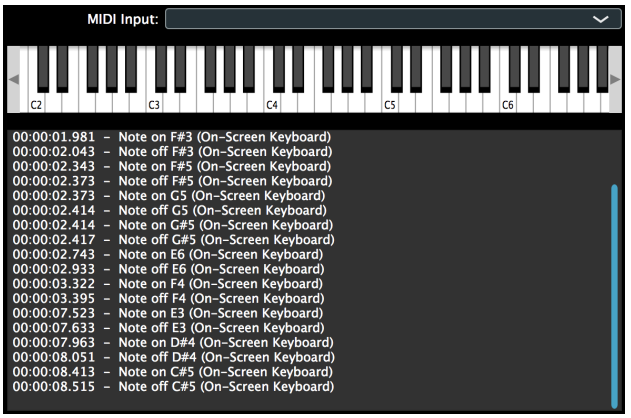
To develop to the 2.0 specification will require a strong understanding of 1.0 messages, data, and methods of implementation. Programming MIDI messages was my first experience programming, way back in middle school on my Yamaha Clavinova electric keyboard on a barebones MIDI event editor with some audio effects and basic mixing capabilities including level, pan, and a master EQ. That began my fascination with MIDI. Moving beyond programming the data into programming the software itself is the process. C++ is common in audio development so JUCE is an appropriate vehicle. I am aiming to understand each MIDI message type and its use, and how they are packaged and sent over transports.

I have been advised by members of the 2.0 development team that Property Exchange and Profiles work over 1.0 transports,[7] but I want to understand why that is the case. So, by understanding the structure of how MIDI data is managed, delivered, and received then I can begin to build my own implementation of Property Exchange based on the how JUCE has implemented the current 1.0 functionality. This will be based on reading the official documents released by the MMA outline the specifications for Property Exchange, including the white paper on the common practices.

As there basically no hardware devices available for 2.0, my implementation will be all virtual, but the 1.0 component of the application will function with hardware.

3. PLAN & FUTURE ACTION

For my application, I have begun work with JUCE tutorials to implement and understand MIDI messaging.[8] Much of the basic MIDI functionality is handled by JUCE classes and methods, so the what I have so far is a working app that displays most incoming MIDI 1.0 messages to the app's console, with a selection box for the available MIDI devices and a graphical keyboard in case there are no hardware devices attached. See image (1) above.



(1) GUI for the MIDI utility app

The timestamps are calculated with a difference between when the application was launched, and when the incoming messages are received.

I have discovered system API's for MIDI, and have looked into Apple's CoreMidi API which has adopted 2.0 capabilities with their newest operating system MacOS 11.0, called "Big Sur."[10] JUCE has not released updated 2.0 wrapper functions for operating system API's yet. In an online forum, a JUCE admin mentioned they were waiting for system API's to be released before JUCE releases their wrapper functions.9 In a comment to one of my questions on midi.org, one of the 2.0 development team members mentioned that he hopes Microsoft will release their support for 2.0 in 2021,[4] so JUCE's support will likely follow that release.

With this in mind, I had initially planned to not rely on any preexisting 2.0 code. However, I looked into JUCE's library implementation of the CoreMidi framework (folder entitled CoreMidi.framework) and discovered a header file called "MIDICapabilityInquiry.h"[11] This file seems to be the beginning of JUCE integrating 2.0 into its framework. I have not digested it's contents, but the file says © 2018 by Apple inc. See (2) below.

```
1  /*
2   File:       CoreMIDI/MIDICapabilityInquiry.h
3
4   Contains:   API for MIDI Capability Inquiry (MIDI-CI)
5
6   Copyright:  (c) 2018 by Apple Inc., all rights reserved.
7
8   Bugs?:      For bug reports, consult the following page on
9   the World Wide Web:
10
11  http://developer.apple.com/bugreporter/
12  */
13
14 // This API requires the modern Objective-C runtime.
15 #if !defined(MIDICapabilityInquiry_h) && defined(__OBJC2__)
16 #define MIDICapabilityInquiry_h
17
18 #import <CoreMIDI/MIDIServices.h>
19 #import <Foundation/Foundation.h>
20 #import <stdint.h>
```

(2) First lines of MIDICapabilityInquiry.h

I plan to delve into this file to see if it will be possible to integrate it into my project.

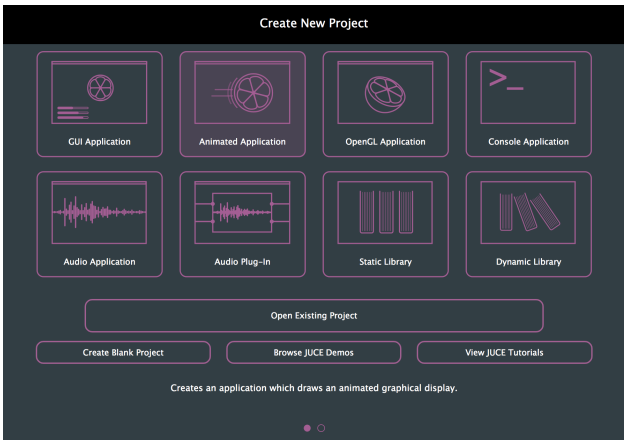
There are a couple more tutorials that JUCE offers that will help me understand how MIDI events are created and handled, so I will move forward with incorporating those too.

Once these are completed I believe I will have a better understanding of how 1.0 messages are handled, which will set me up to begin diving into the developer white papers and specification papers. There were five papers for 2.0 published by the MMA and the Association of Musical Electronics Industry (AMEI) in February 2020. I will search the three of the papers related to Property Exchange for relevant materials and begin working from there.

4. INITIAL RESULTS

The step of getting a working application to display 1.0 messages came easily with JUCE tutorials, however the exact implementation structure is obfuscated by JUCE's API so I seek to dive deeper into the 1.0 messages. Having a working prototype brings good hope that I will be able to incorporate further functionality into the application, and having leads to follow up on is helpful for the direction of my research efforts.

Further reports will include descriptions of the code I am using as I come to understand more in-depth how JUCE functionality fits together. Without JUCE and the Projucer (3) I would be without much support. As I am still waiting to be registered in the CSC497 course I am thankful for my project supervisor's support and interest in the project, as well as his support throughout my entire degree.



(3) The Projucer launchpad GUI

5. REFERENCES

[1] Kent, M., Bomers, F., & Porter, B. (2019, November 27). Mike Kent, Florian Bomers, & Brett Porter - youtube.com. Retrieved January 5, 2021, from https://www.youtube.com/watch?v=K2dAIvrI8zg

[2] Rogerson, B. (2020, December 26). MIDI 2.0 spec confirmed: "the biggest advance in music technology in decades". Retrieved December 28, 2020, from https://www.musicradar.com/news/midi-20-spec-confirmed-the-biggest-advance-in-music-technology-in-decades

[3] USB-IF Publishes USB Device Class Specification for MIDI Devices v2.0. (2020, July 16). *Www.usb.org*. Joe Balich. Retrieved from https://www.usb.org/

[4] Balster, E. Retrieved February 16, 2021, from https://imitone.com/discover-midi/

[5] AMEI, & MMA. (2020, February 20). Official MIDI Specifications. Retrieved February 19, 2021, from https://www.midi.org/specifications/midi-2-0-specifications/property-exchange-specifications

[6] Porter, B. (2021, January 27). Re: Looking for Project Ideas: is there MIDI 2.0 software or programming language support? [Online forum comment]. Retrieved February 19, 2021, from https://www.midi.org/forum/7601-looking-for-project-ideas-is-there-midi-2-0-software-or-programming-language-support#reply-7632

[7] Bomers, F. (221AD, January 18). *Looking for Project Ideas: is there MIDI 2.0 software or programming language support?*

[8] Tutorial: Handling MIDI events. Retrieved February 23, 2021, from https://docs.juce.com/master/tutorial_handling_midi_events.html

[9] Walker, T. H. (2020, July 7). Core MIDI. Retrieved January 14, 2021, from https://developer.apple.com/documentation/coremidi?language=objc

[10] Ryanblock, & t0m. (2020, July 14). MIDI 2.0 Support in JUCE. Retrieved February 19, 2021, from https://forum.juce.com/t/midi-2-0-support-in-juce/40711

[11] Apple. (2018). MIDICapabilityInquiry.h. Apple Inc.