The latest document is on the GitHub.
If you have any questions or issues, please post an issue on GitHub.
https://github.com/kshoji/Unity-MIDI-Plugin-supports

This document explains how to install the plugin, and to use the plugin's features.

Table of Contents

MIDI Plugin for Mobile and Desktop

This plugin provides MIDI transceiving features to your mobile app(iOS, Android, Universal Windows Platform), desktop app(Windows, OSX, Linux), and WebGL app.
Currently implemented `MIDI 1.0` and `MIDI 2.0(UMP)` protocols.

Available MIDI interfaces for platforms

The available MIDI interfaces for each platforms are listed below.

Platform	Bluetooth MIDI	USB MIDI	Network MIDI (RTP-MIDI)	Nearby Connections MIDI	Inter-App MIDI	USB MIDI 2.0	Network MIDI 2.0(UDP MIDI 2.0)
iOS	○	○	○	○	○	○	△(experimental)
Android	○	○	△(experimental)	○	○	○	△(experimental)
Universal Windows Platform	-	○	△(experimental)	-	○	-	△(experimental)
Standalone OSX, Unity Editor OSX	○	○	○	○	○	○	△(experimental)
Standalone Linux, Unity Editor Linux	○	○	△(experimental)	-	○	○	△(experimental)
Standalone Windows, Unity Editor Windows	-	○	△(experimental)	-	○	-	△(experimental)
WebGL	○	○	-	-	-	-	-

About limitations

Android

- USB MIDI is supported API Level 12 (Android 3.1) or above.
- Bluetooth MIDI is supported API Level 18 (Android 4.3) or above.
- Inter-App MIDI is supported API Level 23 (Android 6.0) or above.
- MIDI 2.0 is supported API Level 23 (Android 6.0) or above.
- Build with `Mono backend` causes latency issue, and it supports only `armeabi-v7a` architecture.
  - To fix this problem, change Unity's `Project Settings > Player > Configuration > Scripting Backend` config to `IL2CPP`.
- Nearby Connections MIDI feature requires API Level 28 (Android 9) or above. And with this feature, the app should be compiled with API Level 33 (Android 13.0) or above.

iOS / OSX

- Supported iOS version 12.0 or above.
- Bluetooth MIDI support is Central mode only.

UWP

- Supported UWP platform version 10.0.10240.0 or above.
- Currently, MIDI 2.0 (USB) is not supported.
  - MIDI 2.0 functionality for Windows is expected to be released soon, with implementation following after that.
- Bluetooth MIDI is not supported.
- To use Network MIDI(RTP-MIDI) feature, enable `PrivateNetworkClientServer` at `Project Settings > Player > Capabilities` setting.

Windows

- Bluetooth MIDI is not supported.
- MIDI 2.0 (USB) is not supported.

Linux

- When MIDI1 and MIDI2 protocols coexist on one USB device, only MIDI 2.0 port will be found.

WebGL

- Supporting MIDI devices are depend on the running OS/Browser environment.
- WebGL may not access to another server resources with UnityWebRequest, so put resource files(such as SMF) into `StreamingAssets`.
- WebGL can't natively use raw UDP/TCP connections, and therefore can't use network MIDI (RTP MIDI, UDP MIDI 2.0).
- Currently, WebGL cannot handle USB MIDI 2.0 devices, nor network MIDI 2.0, so MIDI 2.0 functions are not available in WebGL except for reading and writing MIDI Clip files.
- You *should* modify `index.html` file of `WebGLTemplates` directory like below. This enables to access Unity's runtime from `unityInstance` variable.
  - Or, copy `MIDI/Samples/WebGLTemplates` files to `Assets/WebGLTemplates`, and select `Default-MIDI` or `Minimal-MIDI` template from `Project Settings > Player > Resolution and Presentation > WebGL Template` setting.
  - For more information, see Unity official document of WebGL Templates.

Part of original code:

```
script.onload = () => {
createUnityInstance(canvas, config, (progress) => {
    progressBarFull.style.width = 100 * progress + "%";
}).then((unityInstance) => {
```

Modified: add global `unityInstance` variable.

```
var unityInstance = null; // <- HERE
script.onload = () => {
createUnityInstance(canvas, config, (progress) => {
    progressBarFull.style.width = 100 * progress + "%";
}).then((unityInst) => { // <- HERE
    unityInstance = unityInst; // <- HERE
```

Network MIDI (RTP MIDI)

- The error correction(RTP MIDI Journaling) protocol is not supported on experimental support specified above.

How to install plugin

1. Import the unitypackage from the Asset Store view.
    - If the package has been updated, some older file versions are installed at `Assets/MIDI/Plugins/Android`. If you find them, please remove all older versions' aar files.
2. Select the app's platform; iOS, Android, Standalone, Web, and UWP. And then build the sample app.
    - The sample scenes are found at Assets/MIDI/Samples directory.
3. If the Nearby Connections' dependency has been installed, you may need to update it to the latest version.

About build PostProcessing, and additional script symbols

PostProcessing: iOS

- Additional framework will be automatically added while building postprocess.
    - Additional frameworks: `CoreMIDI framework`, `CoreAudioKit.framework`
- `Info.plist` will be automatically adjusted while building postprocess.
    - Additional property: `NSBluetoothAlwaysUsageDescription`

PostProcessing: Android

- `AndroidManifest.xml` will be automatically adjusted while building postprocess.
    - Additional permissions:
        - `android.permission.BLUETOOTH`
        - `android.permission.BLUETOOTH_ADMIN`
        - `android.permission.ACCESS_FINE_LOCATION`
        - `android.permission.BLUETOOTH_SCAN`
        - `android.permission.BLUETOOTH_CONNECT`
        - `android.permission.BLUETOOTH_ADVERTISE`
    - Additional feature:
        - `android.hardware.bluetooth_le`
        - `android.hardware.usb.host`

PostProcessing: Android, Meta Quest(Oculus Quest)

If you want to use the USB MIDI feature on Meta Quest(Oculus Quest) device, please UNCOMMENT below to detect USB MIDI device connections.

The part of `PostProcessBuild.cs`

```
public class ModifyAndroidManifest : IPostGenerateGradleAndroidProject
{
    public void OnPostGenerateGradleAndroidProject(string basePath)
    {
        :

        // ⚠ NOTE: If you want to use the USB MIDI feature on Meta Quest
2(Oculus Quest 2), please UNCOMMENT below to detect USB MIDI device connections.
        // androidManifest.AddUsbIntentFilterForOculusDevices();
```

Android: Use CompanionDeviceManager to find BLE MIDI devices

You can use CompanionDeviceManager for BLE MIDI device connection on Android.

To enable this feature, add `FEATURE_ANDROID_COMPANION_DEVICE` to the `Scripting Define Symbols` setting.

```
Project Settings > Other Settings > Script Compilation > Scripting Define Symbols
```

> ⚠ NOTE: The Meta(Oculus) Quest devices can use this feature to find and connect the Bluetooth MIDI devices.

How to implement MIDI features

Describes how to use basic MIDI features.

Initializing MIDI Plugin

1. Call `MidiManager.Instance.InitializeMidi` method on `Awake` method in MonoBehaviour.
    - A GameObject named `MidiManager` will be created in `DontDestroyOnLoad` at the hierarchy view.

      > ⚠ NOTE: If the EventSystem component already exists in another place, remove gameObject.AddComponent<EventSystem>() method calling at MidiManager.Instance.InitializeMidi method.

2. (BLE MIDI only)
    - Call `MidiManager.Instance.StartScanBluetoothMidiDevices` method to scan BLE MIDI devices.
        - This method should be called in the `InitializeMidi` method's callback action.

```
private void Awake()
{
    // Receive MIDI data with this gameObject.
    // The gameObject should implement IMidiXXXXXEventHandler.
    MidiManager.Instance.RegisterEventHandleObject(gameObject);

    // Initialize MIDI feature
    MidiManager.Instance.InitializeMidi(() =>
    {
#if (UNITY_ANDROID || UNITY_IOS || UNITY_WEBGL) && !UNITY_EDITOR
        // Start scan Bluetooth MIDI devices
        MidiManager.Instance.StartScanBluetoothMidiDevices(0);
#endif
    });
}
```

Terminating MIDI Plugin

1. Call `MidiManager.Instance.TerminateMidi` method on `OnDestroy` method in `MonoBehaviour`.
    - This method should be called on finishing Scene or finishing to use MIDI functions.

```
private void OnDestroy()
{
    // Shutdown all MIDI features
    MidiManager.Instance.TerminateMidi();
}
```

MIDI device attaching/detaching event handling

1. Implement interface `IMidiDeviceEventHandler` for event receiving.
    - Detect new connection:
        - `OnMidiInputDeviceAttached`, `OnMidiOutputDeviceAttached` will be called on a new MIDI device connected.
    - Detect disconnection:
        - `OnMidiInputDeviceDetached`, `OnMidiOutputDeviceDetached` will be called on a MIDI device disconnected.
2. Call `MidiManager.Instance.RegisterEventHandleObject` method to register GameObject to receive event.
3. When a MIDI device connected or disconnected, the implemented method will be called.

```
public void OnMidiInputDeviceAttached(string deviceId)
{
    // A new MIDI receiving device has been connected.
}

public void OnMidiOutputDeviceAttached(string deviceId)
{
    // A new MIDI sending device has been connected.
    Debug.Log($"MIDI device attached. deviceId: {deviceId}, name:
{MidiManager.Instance.GetDeviceName(deviceId)}");
}

public void OnMidiInputDeviceDetached(string deviceId)
{
    // A MIDI receiving device has been disconnected.
}

public void OnMidiOutputDeviceDetached(string deviceId)
{
    // A MIDI sending device has been disconnected.
    Debug.Log($"MIDI device detached. deviceId: {deviceId}, name:
{MidiManager.Instance.GetDeviceName(deviceId)}");
}
```

All codes are found at `Assets/MIDI/Samples/Scripts/MidiSampleScene.cs` file.

Get MIDI device information with the deviceId

- Call `MidiManager.Instance.GetDeviceName(string deviceId)` method to get the device name of the specified device id.
- Call `MidiManager.Instance.GetVendorId(string deviceId)` method to get the vendor id of the specified device id.
    - Some platform or kinds of MIDI connection(BLE MIDI, RTP MIDI) are not supported, so the empty string will be returned with these environments.
- Call `MidiManager.Instance.GetProductId(string deviceId)` method to get the product id of the specified device id.
    - Some platform or kinds of MIDI connection(BLE MIDI, RTP MIDI) are not supported, so the empty string will be returned with these environments.

After the device being disconnected, these method returns empty string.

Availability of GetVendorId / GetProductId method

Platform	**Bluetooth MIDI**	**USB MIDI**	**Network MIDI (RTP-MIDI)**	**Nearby Connections MIDI**
iOS	○	○	-	-
Android	○	○	-	-
Universal Windows Platform	-	○	-	-
Standalone OSX, Unity Editor OSX	○	○	-	-
Standalone Linux, Unity Editor Linux	-	-	-	-
Standalone Windows, Unity Editor Windows	-	○	-	-
WebGL	-	△ (GetVendorId only)	-	-

VendorId examples

Since the API differs depending on the platform, the obtained VendorId will differ.

Platform	**Bluetooth MIDI**	**USB MIDI**	**Network MIDI (RTP-MIDI)**	**Nearby Connections MIDI**
iOS	QUICCO SOUND Corp.	Generic	-	-

Platform	Bluetooth MIDI	USB MIDI	Network MIDI (RTP-MIDI)	Nearby Connections MIDI
Android	QUICCO SOUND Corp.	1410	-	-
Universal Windows Platform	-	VID_0582	-	-
Standalone OSX, Unity Editor OSX	QUICCO SOUND Corp.	Generic	-	-
Standalone Linux, Unity Editor Linux	-	-	-	-
Standalone Windows, Unity Editor Windows	-	1	-	-
WebGL	QUICCO SOUND Corp.	Microsoft Corporation	-	-

ProductId examples

Since the API differs depending on the platform, the obtained ProductId will differ.

Platform	Bluetooth MIDI	USB MIDI	Network MIDI (RTP-MIDI)	Nearby Connections MIDI
iOS	mi.1	USB2.0-MIDI	-	-
Android	mi.1	298	-	-
Universal Windows Platform	-	PID_012A	-	-
Standalone OSX, Unity Editor OSX	mi.1	USB2.0-MIDI	-	-
Standalone Linux, Unity Editor Linux	-	-	-	-
Standalone Windows, Unity Editor Windows	-	102	-	-
WebGL	mi.1	UM-ONE	-	-

MIDI Event Receiving

Receive MIDI message using GameObject, or C# object

1. Implement event receiving interface written in `IMidiEventHandler.cs` source code, named like `IMidiXXXXXEventHandler`.
    - If you want to receive a Note On event, implement the `IMidiNoteOnEventHandler` interface.
2. Call `MidiManager.Instance.RegisterEventHandleObject` method to register GameObject, or C# object to receive MIDI event.
3. When a MIDI event received, the implemented method will be called.

```
public class MidiSampleScene : MonoBehaviour, IMidiAllEventsHandler,
IMidiDeviceEventHandler
{
    private void Awake()
    {
        // Receive MIDI data with this gameObject.
        // The gameObject should implement IMidiXXXXXEventHandler.
        MidiManager.Instance.RegisterEventHandleObject(gameObject);
        ...
```

MIDI event receiving:

```
public void OnMidiNoteOn(string deviceId, int group, int channel, int note, int
velocity)
{
    // Note On event has been received.
    Debug.Log($"OnMidiNoteOn channel: {channel}, note: {note}, velocity:
{velocity}");
}

public void OnMidiNoteOff(string deviceId, int group, int channel, int note, int
velocity)
{
    // Note Off event has been received.
    Debug.Log($"OnMidiNoteOff channel: {channel}, note: {note}, velocity:
{velocity}");
}
```

All codes are found at `Assets/MIDI/Samples/Scripts/MidiSampleScene.cs` file.

Receive MIDI message using C# event delegate

Another way, you can use C# event delegates to receive MIDI events, named like `MidiManager.Instance.OnMidiXXXXEvent`.

```
void OnMidiNoteOn(string deviceId, int group, int channel, int note, int velocity)
{
    // Note On event has been received.
    Debug.Log($"OnMidiNoteOn channel: {channel}, note: {note}, velocity:
{velocity}");
}

...

MidiManager.Instance.OnMidiNoteOnEvent += OnMidiNoteOn;
```

MIDI Event Sending

1. Call the `MidiManager.Instance.SendMidiXXXXXX` method somewhere.
   Like this:

```
// Send Note On message
MidiManager.Instance.SendMidiNoteOn("deviceId", 0/*groupId*/, 0/*channel*/,
60/*note*/, 127/*velocity*/);
```

2. `deviceId` can be obtained from `MidiManager.Instance.OutputDeviceIdSet` property (type: HashSet<string>).

```
deviceIds = MidiManager.Instance.OutputDeviceIdSet().ToArray();

...

if (GUILayout.Button("NoteOn"))
{
    // Send Note On message
    MidiManager.Instance.SendMidiNoteOn(deviceIds[deviceIdIndex], 0, (int)channel,
(int)noteNumber, (int)velocity);
}
```

All codes are found at `Assets/MIDI/Samples/Scripts/MidiSampleScene.cs` file.

Using Network MIDI (RTP MIDI) Transport

This feature is an experimental feature for non-iOS platforms.

Start or stop the Network MIDI (RTP MIDI) server

- Start RTP-MIDI session:
    - Call `MidiManager.Instance.StartRtpMidiServer` method with session name and TCP port number to start RTP-MIDI session accepting.
    - This will start listening to the TCP port with the specified port number. The another computer can connect with the app.
- Stop RTP-MIDI session:
    - Call `MidiManager.Instance.StopRtpMidiServer` method to stop RTP-MIDI session communications.

```
#if !UNITY_IOS && !UNITY_WEBGL
    // Start RTP MIDI server with session name "RtpMidiSession", and listen with
port 5004.
    MidiManager.Instance.StartRtpMidiServer("RtpMidiSession", 5004);

...

    // Stop the session
    MidiManager.Instance.StartRtpMidiServer(5004);
#endif
```

Connect to Network MIDI (RTP MIDI) server

RTP MIDI feature doesn't have the advertising / discovering function, so RTP-MIDI client should know the RTP-MIDI server's address and port.

- Connect another RTP-MIDI running computer:
    - Call `MidiManager.Instance.ConnectToRtpMidiServer` method to start connection with another computer.

```
#if !UNITY_WEBGL || UNITY_EDITOR
// Connect to another machine's RTP-MIDI session
MidiManager.Instance.ConnectToRtpMidiServer("RtpMidiSession", 5004, new
IPEndPoint(IPAddress.Parse("192.168.0.111"), 5004));

...

// Disconnect from the session
MidiManager.Instance.DisconnectFromListener(5004, new
IPEndPoint(IPAddress.Parse("192.168.0.111"), 5004));
#endif
```

When connection was established, the callback method `OnMidiInputDeviceAttached`, `OnMidiOutputDeviceAttached` will be called.
You can send or receive MIDI messages with using the deviceId argument of these callbacks.

Use Sequencer features

The library has a sequencer feature, ported from `javax.sound.midi` package classes.

- The sequencer can read / write Standard MIDI File(SMF).
- The sequencer can record MIDI events from MIDI device to the track object.
  - The recorded track can export to SMF data.
- The sequencer can play the recorded track, or read SMF data.

Creating and start using a Sequencer

```
var isSequencerOpened = false;
// Create a new Sequence instance, and open it
var sequencer = new SequencerImpl(() => { isSequencerOpened = true; });
sequencer.Open();
```

All codes are found at `Assets/MIDI/Samples/Scripts/MidiSampleScene.cs` file.

Read SMF as a Sequence, and play it

```
// Apply currently connected MIDI input/output devices to the sequencer
sequencer.UpdateDeviceConnections();

// Load a SMF from FileStream, and set it to Sequencer
using var stream = new FileStream(smfPath, FileMode.Open, FileAccess.Read);
sequencer.SetSequence(stream);
// Start to play sequence
sequencer.Start();

...

// Stop playing
sequencer.Stop();
```

Record a sequence

```
// Apply currently connected MIDI input/output devices to the sequencer
sequencer.UpdateDeviceConnections();

// Add a new sequence to record
sequencer.SetSequence(new Sequence(Sequence.Ppq, 480));
// Start recording MIDI data to the sequence
sequencer.StartRecording();
```

```
...

// Stop recording
sequencer.Stop();
```

Write the sequence to a SMF file

```
var sequence = sequencer.GetSequence();
// Check the sequence length
if (sequence.GetTickLength() > 0)
{
    // Write the sequence to a SMF file
    using var stream = new FileStream(recordedSmfPath, FileMode.Create,
FileAccess.Write);
    MidiSystem.WriteSequence(sequence, stream);
}
```

How to implement MIDI 2.0 features

Describes how to use basic MIDI 2.0 features.

Initializing MIDI 2.0 Plugin

1. Call `MidiManager.Instance.InitializeMidi2` method on `Awake` method in MonoBehaviour.
    - A GameObject named `MidiManager` will be created in `DontDestroyOnLoad` at the hierarchy view.

> ⚠ NOTE: If the EventSystem component already exists in another place, remove gameObject.AddComponent<EventSystem>() method calling at MidiManager.Instance.InitializeMidi2 method.

Because MIDI 2.0 events are converted to MIDI 1.0 events and vice versa, methods for both MIDI1 and 2 exist on the MidiManager class.

```
private void Awake()
{
    // Receive MIDI data with this gameObject.
    // The gameObject should implement IMidi2XXXXXEventHandler.
    MidiManager.Instance.RegisterEventHandleObject(gameObject);

    // Initialize MIDI feature
    MidiManager.Instance.InitializeMidi2(() => { });
}
```

Terminating MIDI 2.0 Plugin

1. Call `MidiManager.Instance.TerminateMidi2` method on `OnDestroy` method in `MonoBehaviour`.
    - This method should be called on finishing Scene or finishing to use MIDI functions.

```
private void OnDestroy()
{
    // Shutdown all MIDI features
    MidiManager.Instance.TerminateMidi2();
}
```

MIDI 2.0 device attaching/detaching event handling

Receive MIDI 2.0 connection event using GameObject, or C# object

1. Implement interface `IMidi2DeviceEventHandler` for event receiving.
    - `OnMidi2InputDeviceAttached`, `OnMidi2OutputDeviceAttached` will be called on a new MIDI 2.0 device connected.

    - `OnMidi2InputDeviceDetached`, `OnMidi2OutputDeviceDetached` will be called on a MIDI 2.0 device disconnected.
2. Call `MidiManager.Instance.RegisterEventHandleObject` method to register GameObject or C# object to receive event.
3. When a MIDI device connected or disconnected, the implemented method will be called.

```
public void OnMidi2InputDeviceAttached(string deviceId)
{
    // A new MIDI 2.0 receiving device has been connected.
}

public void OnMidi2OutputDeviceAttached(string deviceId)
{
    // A new MIDI 2.0 sending device has been connected.
    Debug.Log($"MIDI device attached. deviceId: {deviceId}, name:
{MidiManager.Instance.GetDeviceName(deviceId)}");
}

public void OnMidi2InputDeviceDetached(string deviceId)
{
    // A MIDI 2.0 receiving device has been disconnected.
}

public void OnMidi2OutputDeviceDetached(string deviceId)
{
    // A MIDI 2.0 sending device has been disconnected.
    Debug.Log($"MIDI 2.0 device detached. deviceId: {deviceId}, name:
{MidiManager.Instance.GetDeviceName(deviceId)}");
}
```

All codes are found at `Assets/MIDI/Samples/Scripts/Midi2SampleScene.cs` file.

Receive MIDI 2.0 connection event using C# event delegate

Another way, you can use C# event delegates to receive MIDI events, named like `MidiManager.Instance.OnMidi2XXXXEvent`.

```
void OnMidi2OutputDeviceAttached(string deviceId)
{
    // MIDI 2.0 output device connected event has been received.
    Debug.Log($"OnMidi2OutputDeviceAttached deviceId: {deviceId});
}

...

MidiManager.Instance.OnMidi2OutputDeviceAttachedEvent +=
OnMidi2OutputDeviceAttached;
```

MIDI 2.0 Event Receiving

Receive MIDI 2.0 message using GameObject, or C# object

1. Implement event receiving interface written in `IMidi2EventHandler.cs` source code, named like `IMidi2XXXXXEventHandler` or `IMidi1XXXXXEventHandler`.
    - If you want to receive a Note On event, implement the `IMidi2NoteOnEventHandler` interface.
2. Call `MidiManager.Instance.RegisterEventHandleObject` method to register GameObject or C# object to receive event.
3. When a MIDI event received, the implemented method will be called.

```
public class Midi2SampleScene : MonoBehaviour, IMidi2AllEventsHandler,
IMidi2DeviceEventHandler
{
    private void Awake()
    {
        // Receive MIDI data with this Unity gameObject.
        // The gameObject should implement IMidiXXXXXEventHandler.
        MidiManager.Instance.RegisterEventHandleObject(gameObject);

        // Receive MIDI data with this C# object.
        // The object's class should implement IMidiXXXXXEventHandler.
        MidiManager.Instance.RegisterEventHandleObject(obj);

        ...
```

MIDI event receiving:

```
public void OnMidi2NoteOn(string deviceId, int group, int channel, int note, int
velocity, int attributeType, int attributeData)
{
    // Note On event has been received.
    Debug.Log($"OnMidi2NoteOn channel: {channel}, note: {note}, velocity:
{velocity}, attributeType: {attributeType}, attributeData: {attributeData}");
}

public void OnMidi2NoteOff(string deviceId, int group, int channel, int note, int
velocity, int attributeType, int attributeData)
{
    // Note Off event has been received.
    Debug.Log($"OnMidiNoteOff channel: {channel}, note: {note}, velocity:
{velocity}, attributeType: {attributeType}, attributeData: {attributeData}");
}
```

All codes are found at `Assets/MIDI/Samples/Scripts/Midi2SampleScene.cs` file.

Receive MIDI 2.0 message using C# event delegate

Another way, you can use event delegates to receive MIDI events, named like `MidiManager.Instance.OnMidi2XXXXEvent` or `MidiManager.Instance.OnMidi1XXXXEvent`.

```
void OnMidi2NoteOn(string deviceId, int group, int channel, int note, int
velocity, int attributeType, int attributeData)
{
    // Note On event has been received.
    Debug.Log($"OnMidi2NoteOn channel: {channel}, note: {note}, velocity:
{velocity}, attributeType: {attributeType}, attributeData: {attributeData}");
}

...

MidiManager.Instance.OnMidi2NoteOnEvent += OnMidi2NoteOn;
```

MIDI 2.0 Event Sending

1. Call the `MidiManager.Instance.SendMidi2XXXXXX` or `MidiManager.Instance.SendMidi1XXXXXX` method somewhere.
   (`SendMidi1XXXXXX` methods are MIDI 1 compatibility features using MIDI 2.0 protocol, the precision of values are 7-bits or 14-bits.)

Like this:

```
// Send Note On message with MIDI 2.0 event: velocity is 16 bits.
MidiManager.Instance.SendMidi2NoteOn("deviceId", 0/*groupId*/, 0/*channel*/,
60/*note*/, 65535/*velocity*/, 0/*attributeType*/, 0/*attribute*/);

// Send Note On message with MIDI 1.0 event: velocity is 7 bits.
MidiManager.Instance.SendMidi1NoteOn("deviceId", 0/*groupId*/, 0/*channel*/,
60/*note*/, 127/*velocity*/);
```

2. `deviceId` can be obtained from `MidiManager.Instance.Midi2OutputDeviceIdSet` property (type: HashSet<string>).

```
deviceIds = MidiManager.Instance.MidiOutputDeviceIdSet().ToArray();

...

if (GUILayout.Button("NoteOn"))
{
    // Send Note On message
    MidiManager.Instance.SendMidi2NoteOn(deviceIds[deviceIdIndex], 0,
(int)channel, (int)noteNumber, (int)velocity, (int)attributeType, (int)attribute);
}
```

All codes are found at `Assets/MIDI/Samples/Scripts/Midi2SampleScene.cs` file.

Using Network MIDI 2.0 (UDP MIDI 2.0) Transport

Start the Network MIDI 2.0 (UDP MIDI 2.0) server

- Call `MidiManager.Instance.StartUdpMidi2Server` method with `MIDI endpoint name`, `mDNS service instance name` and `udp port number` to start UDP-MIDI session accepting.
    - If you want to use an error handling action(called when NAK command has received), call `MidiManager.Instance.StartUdpMidi2Server` with `onErrorAction` argument.
    - If you want to use authentication feature with shared secret(passphrase), call `MidiManager.Instance.StartUdpMidi2Server` with `authenticationSecret` argument.
    - If you want to use authentication feature with user authentication(account name / password), call `MidiManager.Instance.StartUdpMidi2Server` with `userAuthentications` argument.

```
#if !UNITY_WEBGL || UNITY_EDITOR
    // Start UDP MIDI 2.0 server with session name: "MyMidi2Endpoint", with mDNS
service instance name: "MyMIDI2Service", and listen with port 12345.
    MidiManager.Instance.StartUdpMidi2Server(12345, "MyMidi2Endpoint",
"MyMIDI2Service");

    // with error handling
    MidiManager.Instance.StartUdpMidi2Server(12345, "MyMidi2Endpoint",
"MyMIDI2Service", onErrorAction: nak =>
    {
        // prints error information
        Debug.LogError($"UDP MIDI error. status:{nak.NakStatus}, command:
{nak.CommandHeader:x8}, message: {nak.Message}");
    });

    // with shared secret
    MidiManager.Instance.StartUdpMidi2Server(12345, "MyMidi2Endpoint",
"MyMIDI2Service", authenticationSecret: "MyUDP-MIDI2Secret");

    // with user authentication
    MidiManager.Instance.StartUdpMidi2Server(12345, "MyMidi2Endpoint",
"MyMIDI2Service", userAuthentications: new List<KeyValuePair<string, string>>()
    {
        new KeyValuePair<string, string>("accountname1", "password1"),
        new KeyValuePair<string, string>("accountname2", "password2"),
    }););
#endif
```

Search, and connect to Network MIDI 2.0 (UDP MIDI 2.0) server

Call `MidiManager.Instance.StartDiscoverUdpMidi2Server()` method to find Network MIDI 2.0 (UDP MIDI 2.0) server from the same network.

```
#if !UNITY_WEBGL || UNITY_EDITOR
// Start to find server
MidiManager.Instance.StartDiscoverUdpMidi2Server();

...

// Stop to find
MidiManager.Instance.StopDiscoverUdpMidi2Server();
#endif
```

The all discovered servers can obtain from `MidiManager.Instance.GetDiscoveredUdpMidi2Servers()` method. This returns the `service instance name` of the server(the list of string), you can connect the server with this `service instance name`.

```
#if !UNITY_WEBGL || UNITY_EDITOR
// Get discovered servers
var discoverdServers = MidiManager.Instance.GetDiscoveredUdpMidi2Servers();

// Connect to first discovered server, with client name "UDP-MIDI2-Client"
if (discoverdServers.Count > 0)
{
    MidiManager.Instance.ConnectToUdpMidi2Server(discoverdServers[0], "UDP-MIDI2-Client");
}
#endif
```

When connection was established, the callback method `OnMidi2InputDeviceAttached`, `OnMidi2OutputDeviceAttached` will be called.
You can send or receive with using the deviceId argument of these callbacks.

Using UmpSequencer features

The library has a sequencer for MIDI 2.0 feature.

- The sequencer can read / write MIDI 2.0 Clip file(.midi2 file).
- The sequencer can record MIDI 2.0 events from MIDI 2.0 device to the UmpSequence object.
    - The recorded UmpSequence can export to MIDI 2.0 Clip file data.
- The sequencer can play the recorded UmpSequence, or read MIDI 2.0 Clip file data.

Creating and start using a UmpSequencer

```
var isSequencerOpened = false;
// Create a new Sequence instance, and open it
var sequencer = new UmpSequencer(() => { isSequencerOpened = true; });
sequencer.Open();
```

All codes are found at `Assets/MIDI/Samples/Scripts/Midi2SampleScene.cs` file.

Read MIDI 2.0 Clip as a UmpSequence, and play it

```
// Load a MIDI Clip from FileStream, and set it to UmpSequencer
using var stream = new FileStream(midiClipPath, FileMode.Open, FileAccess.Read);
sequencer.SetSequence(stream);
// Start to play sequence
sequencer.StartPlaying();

...

// Stop playing
sequencer.StopPlaying();
```

Record a sequence

```
// Add a new sequence to record
sequencer.SetSequence(new Sequence());
// Start recording MIDI data to the sequence
sequencer.StartRecording();

...

// Stop recording
sequencer.StopRecording();
```

Write the sequence to a MIDI 2.0 Clip file

To write MIDI 2.0 Clip file data to stream, use `UmpSequenceWriter.WriteSequence` method.

```
// Get a UmpSequence from the sequencer
var sequence = sequencer.GetSequence();

using (var stream = new FileStream("recorded.midi2", FileMode.Create,
FileAccess.Write))
{
    UmpSequenceWriter.WriteSequence(new List<UmpSequence>(){sequence}, stream);
}
```

Convert MIDI 2.0 Clip file to SMF, or vice versa

To convert UMP sequence to MIDI sequence, use `SequenceConverter.ConvertUmpSequence(Sequence)` method.

```
// Get a UmpSequence from the sequencer
var sequence = sequencer.GetSequence();

// Convert to MIDI sequence, and write it to SMF
using (var stream = new FileStream("recorded.smf", FileMode.Create,
FileAccess.Write))
{
    // write to SMF
    // second argument: fileType: 0 (Single track SMF)
    new
StandardMidiFileWriter().Write(SequenceConverter.ConvertUmpSequence(sequence), 0,
stream);
}
```

To convert MIDI sequence to UMP sequence, use `SequenceConverter.ConvertSequence(UmpSequence)` method.

```
// Get a Sequence from the MIDI 1.0 sequencer
var sequence = sequencer.GetSequence();

// Convert to UMP sequence, and write it to MIDI Clip file
using (var stream = new FileStream("recorded.midi2", FileMode.Create,
FileAccess.Write))
{
    // write to MIDI Clip file
    UmpSequenceWriter.WriteSequence(SequenceConverter.ConvertSequence(sequence),
stream);
}
```

Miscellaneous, or experimental features

Describes how to use some non-basic features.
Contains experimental features.

RTP-MIDI feature

This feature is an experimental feature for non-iOS platforms.

Using MIDI Polyphonic Expression(MPE) feature

This feature is currently experimental. These functions may not be well tested.
If you encountered any bugs on this feature, please report a issue to the support repository.

Defines MPE zones

Before starting to use MPE features, at first, the `zone` should be defined to the MIDI device.

```
// Setup a MPE zone
MpeManager.Instance.SetupMpeZone(deviceId, managerChannel, memberChannelCount);
```

- `managerChannel` 0: lower zone, 15: upper zone
- `memberChannelCount` 0: MPE feature disabled at the zone, From 1 to 15: MPE enabled.
    - If both lower and upper zones are configured and the sum of `memberChannelCount` exceeds 14, the former defined zone will be reduced.

Setup example)
At first, defines lower zone:

```
// Define a lower zone with 10 member channels.
// The manager channel of lower zone is 0.
MpeManager.Instance.SetupMpeZone(deviceId, 0, 10);
```

```
   | Lower Zone: member count: 10                | Normal channels   |
ch#| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10| 11| 12| 13| 14| 15|
```

Then, defines Upper Zone with memberChannelCount: 7

```
// Add a upper zone with 7 member channels.
// The manager channel of upper zone is 15.
MpeManager.Instance.SetupMpeZone(deviceId, 15, 7);
```

The number of member channels in the lower zone is reduced from 10 to 7.

```
   | Lower Zone: member count: 7   | Upper Zone: member count: 7   |
ch#| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10| 11| 12| 13| 14| 15|
```

And then, removes Lower Zone:

```
// To remove the existing zone, Specify its member count to 0.
MpeManager.Instance.SetupMpeZone(deviceId, 0, 0);
```

```
   | Normal channels               | Upper Zone: member count: 7   |
ch#| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10| 11| 12| 13| 14| 15|
```

Sending MPE events

To send MPE events, Call the `MpeManager.Instance.SendMpeXXXXXX` method somewhere.
Like this:

```
// Send Note On message
// The masterChannel argument is 0(lower zone), or 15(uppper zone).
MpeManager.Instance.SendMpeNoteOn(deviceId, masterChannel, noteNumber, velocity);
```

Receiving MPE events

The MPE zone is automatically configured when the MPE zone configure event is received.

1. Implement event receiving interface written in `IMpeEventHandler.cs` source code, named like `IMpeXXXXXEventHandler`.
    - If you want to receive a Note On event, implement the `IMpeNoteOnEventHandler` interface.
2. Call `MidiManager.Instance.RegisterEventHandleObject` method to register GameObject to receive event.
3. When a MPE event received, the implemented method will be called.
    - When zone defined, `OnMpeZoneDefined` method will be called.

```
public class MpeSampleScene : MonoBehaviour, IMidiDeviceEventHandler,
IMpeAllEventsHandler
{
    private void Awake()
    {
        // Receive MPE/MIDI data with this gameObject.
        // The gameObject should implement IMpeXXXXXEventHandler.
        MidiManager.Instance.RegisterEventHandleObject(gameObject);
        ...
```

MPE event receiving:

```
public void OnMpeZoneDefined(string deviceId, int managerChannel, int
memberChannelCount)
{
    // Zone defined event has been received.
    // managerChannel 0: lower zone, 15: upper zone
    // memberChannelCount 0: zone removed, 1-15: zone defined
    Debug.Log($"OnMpeZoneDefined managerChannel: {managerChannel},
memberChannelCount: {memberChannelCount}");
}

public void OnMpeNoteOn(string deviceId, int channel, int note, int velocity)
{
    // Note On event has been received.
    Debug.Log($"OnMpeNoteOn channel: {channel}, note: {note}, velocity:
{velocity}");
}
```

```
public void OnMpeNoteOff(string deviceId, int channel, int note, int velocity)
{
    // Note Off event has been received.
    Debug.Log($"OnMpeNoteOff channel: {channel}, note: {note}, velocity:
{velocity}");
}
```

Android, iOS, macOS: Using Nearby Connections MIDI

The MIDI transfer using Google's Nearby Connections library.
(At this stage, this is a proprietary implementation, and has no interoperability with other similar libraries.)

Add dependency package

Open the Unity's Package Manager view, push the + button on top-left corner, and select `Add package from git URL...` menu.
Then specify the URLs below:
`ssh://git@github.com/kshoji/Nearby-Connections-for-Unity.git`
or
`git+https://github.com/kshoji/Nearby-Connections-for-Unity`

> ⚠ NOTE: If the dependency was previously installed, you may need to update it to the latest version.

Specify Scripting Define Symbol

To enable Nearby Connections MIDI feature, add a Scripting Define Symbol to Player settings.
`ENABLE_NEARBY_CONNECTIONS`

Setting for Android

Change Unity's `Project Settings > Player > Identification > Target API Level` to `API Level 33` or higher.

Advertise the nearby devices

To advertise the device to nearby, call the `MidiManager.Instance.StartNearbyAdvertising()` method.
To stop the advertising, call the `MidiManager.Instance.StopNearbyAdvertising()` method.

```
if (isNearbyAdvertising)
{
    if (GUILayout.Button("Stop advertise Nearby MIDI devices"))
    {
        // Stop advertising
        MidiManager.Instance.StopNearbyAdvertising();
        isNearbyAdvertising = false;
    }
}
else
{
    if (GUILayout.Button("Advertise Nearby MIDI devices"))
    {
        // Start advertising to nearby
        MidiManager.Instance.StartNearbyAdvertising();
        isNearbyAdvertising = true;
    }
}
```

Discover the advertising devices

To discover the Nearby Connections MIDI devices, call the `MidiManager.Instance.StartNearbyDiscovering()` method.
To stop the discovering, call the `MidiManager.Instance.StopNearbyDiscovering()` method.

```
if (isNearbyDiscovering)
{
    if (GUILayout.Button("Stop discover Nearby MIDI devices"))
    {
        // Stop discovering nearby devices
        MidiManager.Instance.StopNearbyDiscovering();
        isNearbyDiscovering = false;
    }
}
else
{
    if (GUILayout.Button("Discover Nearby MIDI devices"))
    {
        // Start discovering nearby advertising devices
        MidiManager.Instance.StartNearbyDiscovering();
        isNearbyDiscovering = true;
    }
}
```

Sending / Receiving MIDI data with nearby

The method of sending and receiving MIDI data is the same as normal MIDI.

Work with Meta Quest(Oculus Quest) devices

Currently, using Oculus Quest 2 as test device.

Connect to USB MIDI devices

Please UNCOMMENT below to detect USB MIDI device connections.

The part of `PostProcessBuild.cs`

```
public class ModifyAndroidManifest : IPostGenerateGradleAndroidProject
{
    public void OnPostGenerateGradleAndroidProject(string basePath)
    {
        :

        // ⚠ NOTE: If you want to use the USB MIDI feature on Meta Quest 2(Oculus Quest 2), please UNCOMMENT below to detect USB MIDI device connections.
        // androidManifest.AddUsbIntentFilterForOculusDevices();
```

Connect to Bluetooth MIDI devices

Use CompanionDeviceManager for BLE MIDI device connection on Android.

To enable this feature, add `FEATURE_ANDROID_COMPANION_DEVICE` to the `Scripting Define Symbols` setting.

```
Project Settings > Other Settings > Script Compilation > Scripting Define Symbols
```

Experimental features around MIDI 2.0

Network MIDI 2.0 (UDP MIDI 2.0) feature

This feature implements Network MIDI 2.0 (UDP) Transport Specification.
Currently, there are no other implementations, so mutual compatibility cannot be ensured.
If you use a different implementation and are unable to communicate via UDP MIDI 2.0, please let me know.

MIDI Clip file / MIDI Container file read & write support

MIDI Clip file specification is currently on published as release version. But, MIDI Clip files are not yet widespread, and there are few sample files, so I am not sure if they meet the compatibility specifications.

MIDI Container file specification is in the draft state. I do not currently recommend using this feature as the specifications are likely to change in the future.

Tested devices

- Android: Pixel 7, Oculus Quest2
- iOS: iPod touch 7th gen
- UWP/Standalone Windows/Unity Editor Windows: Surface Go 2
- Standalone OSX/Unity Editor OSX: Mac mini 3,1
- Standalone Linux/Unity Editor Linux: Ubuntu 22.04 on VirtualBox
- MIDI devices:
  - Quicco mi.1 (BLE MIDI)
  - Miselu C.24 (BLE MIDI)
  - TAHORNG Elefue (BLE MIDI)
  - Roland UM-ONE (USB MIDI)
    - ⚠ NOTE: This device didn't work with iOS.
  - Gakken NSX-39 (USB-MIDI)
  - MacOS Audio MIDI Setup (RTP-MIDI)
- MIDI 2.0 devices:
  - Self-built AmeNote Protozoa device (USB MIDI 2.0)
  - Software projects related MIDI 2.0 developing, testing
    - MIDI 2.0 Workbench
    - Sample code from Apple: Incorporating MIDI 2 into your apps
    - Test MIDI Clip files: test-midi-files

Contacts

Report issues on GitHub

- GitHub supports repository: https://github.com/kshoji/Unity-MIDI-Plugin-supports
  - Search and report issues: https://github.com/kshoji/Unity-MIDI-Plugin-supports/issues

About plugin author

- Kaoru Shoji : 0x0badc0de@gmail.com
- github: https://github.com/kshoji

Used Open Source Software packages created by plugin author:

- Android Bluetooth MIDI library: https://github.com/kshoji/BLE-MIDI-for-Android
- Android USB MIDI library: https://github.com/kshoji/USB-MIDI-Driver
- Unity MIDI Plugin Android (Inter App MIDI): https://github.com/kshoji/Unity-MIDI-Plugin-Android-Inter-App
- iOS MIDI library: https://github.com/kshoji/Unity-MIDI-Plugin-iOS
- MidiSystem for .NET(sequencer, SMF importer/exporter): https://github.com/kshoji/MidiSystem-for-.NET
- RTP-MIDI for .NET: https://github.com/kshoji/RTP-MIDI-for-.NET
- Unity MIDI Plugin UWP: https://github.com/kshoji/Unity-MIDI-Plugin-UWP
- Unity MIDI Plugin Linux: https://github.com/kshoji/Unity-MIDI-Plugin-Linux
- Unity MIDI Plugin OSX: https://github.com/kshoji/Unity-MIDI-Plugin-OSX

Used Open Source Software packages created by others:

- Network MIDI 2.0(UDP MIDI 2.0) Discovery feature
  - net-mdns 0.27.0
  - net-dns 2.0.1
  - SimpleBase 2.1.0, modified to compile with older version's Unity
  - net-commons-common-logging 3.4.1

Used example MIDI data by others

Specified as UnityWebRequest's URL source. The SMF / MIDI Clip binary file is not included.

Sample SMF

The original website below has no `https` services, so the sample code specifies the another site's URL(https://bitmidi.com/uploads/14947.mid).

- Prelude and Fugue in C minor BWV 847 Music by J.S. Bach
  - The MIDI, audio(MP3, OGG) and video files of Bernd Krueger are licensed under the cc-by-sa Germany License.
  - This means, that you can use and adapt the files, as long as you attribute to the copyright holder

    - Name: Bernd Krueger
    - Source: http://www.piano-midi.de
    - The distribution or public playback of the files is only allowed under identical license conditions.

Sample MIDI Clip

- The MIDI Clip test file is came from this github repository: test-midi-files.
    - These files are licenced under the MIT licence.

Version History

- v1.0 Initial release: 7 Aug 2021
- v1.1 Update release: 11 Apr 2022
  - Add MIDI sequencer(playing / recording MIDI sequence) feature
  - Add SMF reading / writing feature
  - Add BLE MIDI Peripheral feature on Android
  - Fix USB MIDI receiving issues on Android
  - Fix BLE MIDI sending issues on Android / iOS
  - Fix BLE MIDI receiving issue(NoteOn with velocity = 0) on Android
- v1.2.0 Update release: 17 Aug 2022
  - Add experimental RTP-MIDI support for Android, or other platforms.
  - Add USB MIDI support for Universal Windows Platform(UWP).
  - Add Android 12's new Bluetooth permissions support.
  - Fix MIDI tranceiving performance improvement on iOS, Android.
  - Fix EventSystem duplication error when the sample scene appended multiple times.
  - Fix Android BLE MIDI's issue around fixed timestamp.
- v1.2.1 Bugfix release: 29 Aug 2022
  - Fix sequencer thread remains after closing
  - Fix Android ProgramChange message failure
  - Fix System exclusive logging issue
  - Fix ThreadInterruptException issue on UWP
  - Fix SMF reading/writing issues around System exclusive
  - Some performance improvements
- v1.3.0 Update release: 13 Oct 2022
  - Add platform support for Standalone OSX, Windows, Linux
  - Add platform support for WebGL
  - Add support for Unity Editor OSX, Windows, Linux
  - Changed Sequencer implementation from Thread to Coroutine
  - Fix iOS/OSX device attaching/detaching issue
- v1.3.1 Bugfix release: 18 May 2023
  - Issue connecting to Quest 2 via cable
  - Sample scene stops working.
  - Byte is obsolete on android
  - Any way of negotiating MTU?
  - Can't get it to work on iOS
  - Have errors with sample scene
  - Android permissions requesting issue
  - Add: Android CompanionDeviceManager support
- v1.3.2 Bugfix release: 19 May 2023
  - Fixed Android compile error
  - Fixed MIDI event order while playing SMF sequence
- v1.3.3 Bugfix release: 25 May 2023
  - Fixed WebGL MIDI sending failure
- v1.3.4 Bugfix release: 5 Jun 2023

  - Fixed: The wrong device name was acquired when a device with the same device ID as the previously connected device but with a different device name.
  - iOS: Add 'Done' button to the BLE MIDI searching popover
  - Sample scene: BLE MIDI Scan feature is Android/iOS only
  - MidiManager singleton pattern refined
- v1.4.0 Update release: 7 Dec 2023
  - Add: Nearby Connections MIDI feature for Android, iOS, macOS
  - Add: Bluetooth LE MIDI feature for WebGL
  - Fixed: iOS device attach/detach callback has mismatches
- v1.4.1 Update release: 22 Feb 2024
  - Fixed: The link error at Linux platform
  - Add: Vendor name / device name support for WebGL platform.
  - Add: Inter App MIDI connections(virtual MIDI) support for iOS/MacOS/Linux/Android
  - Fixed: Memory leak issue on Sample scene.
- v1.4.2 Bugfix release: 27 Feb 2024
  - Fixed: WebGL Sample scene fails to initialize
- v1.4.3 Bugfix release: 12 Mar 2024
  - Fixed: Android plugin initialization fails when Bluetooth is off
  - Fixed: Fails opening USB MIDI devices on Android 14
  - Fixed: Windows plugin fails updating MIDI devices
  - Fixed: Linux plugin crashes at terminating MidiManager
  - Fixed: Unity Editor stops MIDI feature after stopping the game play
  - Added: Feature to get input/output deviceIds(separated from getting all deviceId set method)
  - Added: Feature to specify device connections to the MIDI sequencer
  - Added: The callback to receive MIDI sequencer playback finished event
  - Added: MIDI sequencer's event timing accuracy improved
  - Added: MIDI sequencer's playback position can be specified with the microseconds time
- v1.4.4 Bugfix release: 9 Apr 2024
  - Fixed: The initialization is interrupted when Android plug-in fails loading.
  - Fixed: Android Bluetooth MIDI can't send MIDI messages.
  - Added: The ProGuard minify configuration support for Android platform.
- v1.4.5 Bugfix release: 13 Apr 2024
  - Fixed: Android Bluetooth MIDI lose sending MIDI messages under high load.
  - Updated: Use Android's newer Bluetooth LE APIs.
  - Fixed: Android CompanionDeviceManager initialization issues. From this version, the feature requests `ACCESS_FINE_LOCATION` permission.
- v1.5.0 Update release: 5 Jun 2024
  - Add: MIDI Polyphonic Expression feature (currently, experimental)
  - Totally refactored internal implements
  - Updated: Improved SMF reading compativility
  - Fixed: SMF playback fails around the first MIDI event
  - Fixed: Android input device initialization issue
- v1.5.1 Bugfix release: 5 Sept 2024
  - Fixed: Unity Editor doesn't work on older macOS versions prior to 12.3
    - Adjusted to work with macOS 10.13 and later
  - Fixed: Duplicate GUIDs when importing sample scenes

- v2.0.0 Update release: 5 May 2025
    - Added: MIDI 2.0 support for iOS, macOS Standalone, Android, Standalone Linux.
    - Added: Network MIDI 2.0(UDP MIDI 2.0) feature.
    - Added: MIDI 2.0 Clip / Container file importing / exporting support.
        - MIDI 2.0 Container file format is currently draft-state specification, so this will change at future.
    - Added: MIDI receiving by C# event delegates.
    - Added: MIDI receiving with C# object.
    - Fixed: Compile error when symbol ENABLE_NEARBY_CONNECTIONS defined.
    - Fixed: Some events(TuneRequest, TimingClock, Start, Stop, Continue, ActiveSensing, Reset) are not received on Linux platform.
    - Fixed: Some inter-app MIDI devices not attached on Linux platform.
    - Fixed: Still receiving MIDI messages in Editor even after stopping Play mode