

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](#)
- [Available MIDI interfaces for platforms](#)
  - [About limitations](#)
- [How to install plugin](#)
- [About build PostProcessing and additional script symbols](#)
  - [PostProcessing: iOS](#)
  - [PostProcessing:Android](#)
  - [Android: Use CompanionDeviceManager to find BLE MIDI devices](#)
- [How to implement features](#)
  - [Initializing Plugin](#)
  - [Terminating Plugin](#)
  - [MIDI device attaching/detaching event handling](#)
  - [Get MIDI device information with the deviceId](#)
    - [Availability of GetVendorId / GetProductId method](#)
    - [VendorId examples](#)
    - [ProductId examples](#)
  - [MIDI Event Receiving](#)
  - [MIDI Event Sending](#)
  - [Use sequencer features](#)
    - [Creating and start using a Sequencer](#)
    - [Read SMF as a Sequence, and play it](#)
    - [Record a sequence](#)
    - [Write the sequence to a SMF file](#)
- [Miscellaneous, or experimental features](#)
  - [RTP-MIDI feature](#)
  - [Using MIDI Polyphonic Expression\(MPE\) feature](#)
    - [Defines MPE zones](#)
    - [Sending MPE events to device](#)
    - [Receiving MPE events from device](#)
  - [Android: Use CompanionDeviceManager to find BLE MIDI devices](#)
  - [Android, iOS, macOS: Using Nearby Connections MIDI](#)
    - [Add dependency package](#)
    - [Specify Scripting Define Symbol](#)
    - [Setting for Android](#)
    - [Advertise the nearby devices](#)
    - [Discover the advertising devices](#)
    - [Sending / Receiving MIDI data with nearby.](#)

- [Work with Meta Quest\(Oculus Quest\) devices](#)
  - [Connect to USB MIDI devices](#)
  - [Connect to Bluetooth MIDI devices](#)
- [Tested devices](#)
- [Contacts](#)
  - [Report issues on GitHub](#)
  - [About plugin author](#)
  - [Used Open Source Softwares created by me:](#)
  - [Used example MIDI data by others](#)
- [Version History](#)

# MIDI Plugin for Mobile and Desktop

This plugin provides MIDI transeiving features to your mobile app(iOS, Android, Universal Windows Platform), desktop app(Windows, OSX, Linux), and WebGL app.  
Currently implemented MIDI 1.0 protocol only.

## 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
iOS	○	○	○	○
Android	○	○	△(experimental)	○
Universal Windows Platform	-	○	△(experimental)	-
Standalone OSX, Unity Editor OSX	○	○	○	○
Standalone Linux, Unity Editor Linux	○	○	△(experimental)	-
Standalone Windows, Unity Editor Windows	-	○	△(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.
- 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 11.0 or above.
- Bluetooth MIDI support is Central mode only.

## UWP

- Supported UWP platform version 10.0.10240.0 or above.
- 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.

## 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`.
- 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

- 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 or Android. and build the sample app.
  - The sample scene is 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`
- 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 features

Describes how to use basic MIDI features.

### Initializing 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()` 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.
3. (RTP-MIDI only)
  - Call `MidiManager.Instance.StartRtpMidiServer` method with session name and udp port number to start RTP-MIDI session accepting.

```
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
    });
}
```

```
#endif
});

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

## Terminating 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. Call `MidiManager.Instance.RegisterEventHandleObject` method to register GameObject to receive event.
2. Implement interface `IMidiDeviceEventHandler` for event receiving.
  - `OnMidiInputDeviceAttached`, `OnMidiOutputDeviceAttached` will be called on a new MIDI device connected.
  - `OnMidiInputDeviceDetached`, `OnMidiOutputDeviceDetached` will be called on a MIDI device disconnected.

```
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.
    receivedMidiMessages.Add($"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.
    receivedMidiMessages.Add($"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.

<b>Platform</b>	<b>Bluetooth MIDI</b>	<b>USB MIDI</b>	<b>Network MIDI (RTP-MIDI)</b>	<b>Nearby Connections MIDI</b>
iOS	QUICCO SOUND Corp.	Generic	-	-
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.

<b>Platform</b>	<b>Bluetooth MIDI</b>	<b>USB MIDI</b>	<b>Network MIDI (RTP-MIDI)</b>	<b>Nearby Connections MIDI</b>
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

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 to receive event.
3. On receiving a MIDI event, 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.
    receivedMidiMessages.Add($"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.
    receivedMidiMessages.Add($"OnMidiNoteOff channel: {channel}, note: {note},
velocity: {velocity}");
}
```

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

## 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.

## 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(smfpPath, 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);  
}
```

# Miscellaneous, or experimental features

---

Describes how to use some non-basic features.

Contains experimental features.

## RTP-MIDI feature

Experimental feature for non-iOS platforms.

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

```
// Starts to listen UDP 5004 port with session name "RtpMidiSession".
MidiManager.Instance.StartRtpMidiServer("RtpMidiSession", 5004);
...
// Stop the session
MidiManager.Instance.StopRtpMidi(5004);

// Connect to another machine's RTP-MIDI session
MidiManager.Instance.ConnectToRtpMidiClient("RtpMidiSession", 5004, new
IPEndPoint(IPAddress.Parse("192.168.0.111"), 5004));
```



# 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(upper 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. On receiving a MPE event, 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
receivedMidiMessages.Add($"OnMpeZoneDefined managerChannel: {managerChannel},
memberChannelCount: {memberChannelCount}");
}

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

public void OnMpeNoteOff(string deviceId, int channel, int note, int velocity)
{
    // Note Off event has been received.
    receivedMidiMessages.Add($"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

# 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)

# 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](mailto:0x0badc0de@gmail.com)
- github: <https://github.com/kshoji>

## Used Open Source Softwares 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 example MIDI data by others

Specified as UnityWebRequest's URL source. The SMF binary file is not included. The original website below has no [https](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.



# Version History

---

- v1.0 Initial release
- v1.1 Update release
  - 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
  - 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
  - Fix sequencer thread remains after closing
  - Fix Android ProgramChange message failure
  - Fix System exclusive logging issue
  - Fix ThreadInterruptedException issue on UWP
  - Fix SMF reading/writing issues around System exclusive
  - Some performance improvements
- v1.3.0 Update release
  - 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
  - [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
  - Fixed Android compile error
  - Fixed MIDI event order while playing SMF sequence
- v1.3.3 Bugfix release
  - Fixed WebGL MIDI sending failure
- v1.3.4 Bugfix release

- 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
  - 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
  - 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
  - Fixed: WebGL Sample scene fails to initialize
- v1.4.3 Bugfix release
  - 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
  - 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
  - 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
  - Add: MIDI Polyphonic Expression feature (currently, experimental)
  - Totally refactored internal implements
  - Updated: Improved SMF reading compatibility
  - Fixed: SMF playback fails around the first MIDI event
  - Fixed: Android input device initialization issue
- v1.5.1 Bugfix release
  - 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