

最新のドキュメントはGitHubにあります。

質問や問題があればGitHubのissueまでご投稿ください。

<https://github.com/kshoji/Unity-MIDI-Plugin-supports>

---

プラグインのインストール方法、機能について説明しています。

## 目次

---

- 目次
- MIDI Plugin for Mobile and Desktop
- プラットフォームで利用可能なMIDIインターフェース
  - 制限事項
    - Android
    - iOS / OSX
    - UWP
    - Windows
  - Linux
    - WebGL
    - Network MIDI (RTP MIDI)
- プラグインのインストール
- ビルドの後処理(PostProcessing)について
  - PostProcessing: iOS
  - PostProcessing: Android
  - PostProcessing: Android, Meta Quest(Oculus Quest)
  - Android: CompanionDeviceManagerを使ってBLE MIDIデバイスを探す
- 機能の実装方法
  - プラグインの初期化
  - プラグインの終了
  - MIDIデバイスの接続・切断のイベントハンドリング
  - deviceIdからMIDIデバイスの情報を取得する
    - GetVendorId / GetProductIdメソッドが利用可能な環境
    - VendorIdの例
    - ProductIdの例
  - MIDIイベントの受信
    - GameObject もしくは C# オブジェクトを使ったMIDIメッセージの受信
    - C# event delegateを利用したMIDIメッセージの受信
  - MIDIイベントの送信
  - Network MIDI (RTP MIDI)による通信を使う
    - ネットワークMIDI (RTP MIDI) サーバーを起動または停止する
    - Network MIDI (RTP MIDI) server
  - シーケンサー機能の利用
    - シーケンサーの作成と開始
    - SMFをシーケンスとして読み出し、再生する
    - シーケンスを記録する
    - シーケンスをSMFとして書き出す

- MIDI 2.0機能の実装方法
  - MIDI 2.0プラグインの初期化
  - MIDI 2.0プラグインの終了
  - MIDI 2.0 デバイスの接続/接続解除イベント処理
    - GameObjectまたはC#オブジェクトを使用してMIDI 2.0接続イベントを受信する
    - C# イベントデリゲートを使用して MIDI 2.0 接続イベントを受信する
  - MIDI 2.0 イベント受信
    - GameObjectまたはC#オブジェクトを使用してMIDI 2.0メッセージを受信する
    - C# イベントデリゲートを使用して MIDI 2.0 メッセージを受信する
  - MIDI 2.0 イベント送信
  - ネットワークMIDI 2.0 (UDP MIDI 2.0) トランスポートの使用
    - ネットワークMIDI 2.0 (UDP MIDI 2.0)サーバの開始
    - ネットワークMIDI 2.0 (UDP MIDI 2.0) サーバーを検索して接続します
  - UmpSequencer機能の使用
    - UmpSequencerの作成と利用開始
    - MIDI 2.0 クリップを UmpSequence として読み込み、再生します。
    - シーケンスを記録する
    - シーケンスをMIDI 2.0クリップファイルに書き込む
    - MIDI 2.0 クリップファイルを SMF に変換します (またはその逆)
- その他の機能、実験的な機能
  - RTP-MIDI 機能
  - MIDI Polyphonic Expression(MPE)機能の利用
    - MPEゾーンの定義
    - MPEイベントの送信
    - MPEイベントの受信
  - Android, iOS, macOS: Nearby Connections MIDIの利用
    - 依存パッケージの追加
    - Scripting Define Symbolの設定
    - Android向けの設定
    - 近隣のデバイスへの広報(Advertise)
    - 広報されたデバイスを見つける
    - MIDIデータをnearbyで送受信する
  - Meta Quest(Oculus Quest)デバイスを使う
    - USB MIDIデバイスと接続する
    - Bluetooth MIDIデバイスと接続する
  - MIDI 2.0に関する実験的な機能
    - ネットワークMIDI 2.0 (UDP MIDI 2.0) 機能
    - MIDIクリップファイル/MIDIコンテナファイルの読み書きサポート
- テストしたデバイス
- 連絡先
  - GitHubでの不具合報告、サポート
  - プラグイン作者
  - 使用した自作のオープンソースソフトウェア
  - 他者が作成したオープンソースソフトウェアパッケージ:
  - 他者提供による、使用したサンプルMIDIデータ
    - Sample SMF

■ Sample MIDI Clip

- バージョン履歴

# MIDI Plugin for Mobile and Desktop

このプラグインはモバイルアプリ(iOS, Android)、デスクトップアプリ(Windows, OSX, Linux)、 WebGLアプリにMIDI送受信の機能を追加します。

現在は「[MIDI 1.0プロトコル](#)」と「[MIDI 2.0プロトコル\(UMP\)](#)」が実装されています。

## プラットフォームで利用可能なMIDIインターフェース

各プラットフォームで対応しているMIDIインターフェースは下記の通りです。

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	○	○	○	○	○	○	△(試験的 に対応)
Android	○	○	△(試験的 に対応)	○	○	○	△(試験的 に対応)
Universal Windows Platform	-	○	△(試験的 に対応)	-	○	-	△(試験的 に対応)
Standalone OSX, Unity Editor OSX	○	○	○	○	○	○	△(試験的 に対応)
Standalone Linux, Unity Editor Linux	○	○	△(試験的 に対応)	-	○	○	△(試験的 に対応)
Standalone Windows, Unity Editor Windows	-	○	△(試験的 に対応)	-	○	-	△(試験的 に対応)
WebGL	○	○	-	-	-	-	-

# 制限事項

## Android

- USB MIDI は API Level 12 (Android 3.1) 以上で利用できます。
- Bluetooth MIDI は API Level 18 (Android 4.3) 以上で利用できます。
- Inter-App MIDI は API Level 23 (Android 6.0) 以上で利用できます。
- MIDI 2.0 は API Level 23 (Android 6.0) 以上で利用できます。
- Mono backend でのビルドでは遅延の問題が発生し、`armeabi-v7a` アーキテクチャしかサポートされません。
  - この問題を解消するためには、Unityの設定 `Project Settings > Player > Configuration > Scripting Backend` を `IL2CPP` に変更します。
- Nearby Connections MIDI の機能を使う場合はAPI Level 28 (Android 9) 以降が必要です。この機能を使う場合、アプリは API Level 33 (Android 13.0) 以上でコンパイルする必要があります。

## iOS / OSX

- iOS 12.0 以上で動作します。
- Bluetooth MIDIはCentralモードのみサポートします。

## UWP

- UWPのバージョン 10.0.10240.0 以上で動作します。
- 現在、MIDI 2.0 (USB) はサポートしていません。
  - Windows向けのMIDI 2.0機能はまもなくリリースされる予定で、実装はその後になります。
- Bluetooth MIDIはサポートしていません。
- Network MIDI(RTP-MIDI) 機能を使う場合は、`Project Settings > Player > Capabilities` の設定にある `PrivateNetworkClientServer` を有効にしてください。

## Windows

- Bluetooth MIDIはサポートしていません。
- MIDI 2.0 (USB) はサポートしていません。

## Linux

- MIDI1とMIDI2のプロトコルが1つのデバイスで共存している場合、MIDI2のポートのみが見つかります。

## WebGL

- サポートされるMIDIデバイスはOSやブラウザに依存します。
- WebGLはUnityWebRequestを使って他のサーバーのリソースにアクセスできない場合があるので、SMFなどのリソースファイルを [StreamingAssets](#) に置いてください。
- WebGLは生のUDP/TCP接続を利用できないので、ネットワークMIDIの機能(RTP MIDI, UDP MIDI 2.0)が利用できません。
- 現在、WebGLはUSB MIDI 2.0のデバイスを処理できないため、MIDI 2.0の機能はMIDI Clipファイルの読み書きを除いて利用できません。
- [WebGLTemplates](#) ディレクトリの `index.html` ファイルを下記のように変更する必要があります。`unityInstance` 変数を経由してUnityのランタイムにアクセスできようとしています。
  - もしくは、[MIDI/Samples/WebGLTemplates](#) のファイルを [Assets/WebGLTemplates](#) にコピーし、[Project Settings > Player > Resolution and Presentation > WebGL Template](#) の設定から、[Default-MIDI](#) か [Minimal-MIDI](#) のテンプレートを選択します。
  - 詳しくは、[Unity公式のWebGL Templatesドキュメント](#) を参照してください。

オリジナルコードの抜粋:

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

修正後: グローバル変数 `unityInstance` を追加

```
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)

- エラー訂正(RTP MIDIジャーナリング)プロトコルは上記に示された「試験的」のプラットフォームでは実装されていません。

# プラグインのインストール

---

1. アセットストアViewからunitypackageをインポートします。
  - パッケージを更新した場合、[Assets/MIDI/Plugins/Android](#)に古いバージョンのファイルがある場合があります。その場合は古いバージョンのaarファイルをすべて削除してください。
2. プラットフォーム(iOSやAndroid)を選択して、サンプルアプリをビルドします。
  - サンプルシーンは Assets/MIDI/Samples ディレクトリにあります。
3. Nearby Connectionsのパッケージがインストールされていれば、最新版への更新が必要になる場合があります。

## ビルドの後処理(PostProcessing)について

---

### PostProcessing: iOS

- ビルドの後処理によって、追加のフレームワークが自動的に追加されます。
  - 追加されるフレームワーク: [CoreMIDI framework](#), [CoreAudioKit.framework](#)
- [Info.plist](#) が自動的に調整されます。
  - 追加されるプロパティ: [NSBluetoothAlwaysUsageDescription](#)

### PostProcessing: Android

- ビルドの後処理によって、[AndroidManifest.xml](#) が自動的に調整されます。
  - 追加されるパーミッション:
    - [android.permission.BLUETOOTH](#)
    - [android.permission.BLUETOOTH\\_ADMIN](#)
    - [android.permission.ACCESS\\_FINE\\_LOCATION](#)
    - [android.permission.BLUETOOTH\\_SCAN](#)
    - [android.permission.BLUETOOTH\\_CONNECT](#)
    - [android.permission.BLUETOOTH\\_ADVERTISE](#)
  - 追加されるfeature:
    - [android.hardware.bluetooth\\_le](#)
    - [android.hardware.usb.host](#)

## PostProcessing: Android, Meta Quest(Oculus Quest)

Oculus Quest 2 でUSB MIDI機能を使う場合は、接続を検知するために下記のコードをコメントアウト解除する必要があります。

PostProcessBuild.cs の一部

```
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: CompanionDeviceManagerを使ってBLE MIDIデバイスを探す

Androidでは、[CompanionDeviceManager](#)を使ってBLE MIDIデバイスの接続を見つけることができます。

この機能を有効にするには、[Scripting Define Symbols](#) 設定に `FEATURE_ANDROID_COMPANION_DEVICE` を追加します。

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

⚠️ NOTE: この機能を使うことで、Meta(Oculus) QuestデバイスではBluetooth MIDIデバイスを見つけて接続できます。

# 機能の実装方法

基本的なMIDI機能の使い方について解説しています。

## プラグインの初期化

1. MonoBehaviourの `Awake` メソッドから `MidiManager.Instance.InitializeMidi` メソッドを呼び出します。

- `MidiManager` という名前の GameObject が、ヒエラルキーの `DontDestroyOnLoad` の下に自動的に作成されます。

△ NOTE: EventSystem コンポーネントが既に他の場所に存在する場合には、  
`gameObject.AddComponent<EventSystem>()` メソッドの呼出を  
`MidiManager.Instance.InitializeMidi` メソッドから削除してください。

2. (BLE MIDI のみ)

- `MidiManager.Instance.StartScanBluetoothMidiDevices` メソッドを呼び出して、周囲の BLE MIDI デバイスを探します。
  - このメソッドは `InitializeMidi` メソッドのコールバックAction内から呼ばれるべきです。

```
private void Awake()
{
    // このgameObject でMIDIイベントを受信します。
    // gameObject は IMidiXXXXEventHandler を実装する必要があります。
    MidiManager.Instance.RegisterEventHandleObject(gameObject);

    // MIDI 機能の初期化
    MidiManager.Instance.InitializeMidi(() =>
    {
#if (UNITY_ANDROID || UNITY_IOS || UNITY_WEBGL) && !UNITY_EDITOR
        // Bluetooth MIDIデバイスのスキャンを開始
        MidiManager.Instance.StartScanBluetoothMidiDevices(0);
#endif
    });
}
```

## プラグインの終了

1. MonoBehaviour の `OnDestroy` メソッド内で `MidiManager.Instance.TerminateMidi` を呼び出します。

- このメソッドはMIDI機能を使い終わって、シーンが終了する際に呼ばれるべきです。

```
private void OnDestroy()
{
    // 全てのMIDI機能を停止する。
```

```
        MidiManager.Instance.TerminateMidi();  
    }  
}
```

## MIDIデバイスの接続・切断のイベントハンドリング

1. `MidiManager.Instance.RegisterEventHandleObject` メソッドを用いてイベントを受信するための `GameObject` を登録します。
2. イベントを受信するため、インターフェース `IMidiDeviceEventHandler` を実装します。
  - 新しいMIDIデバイスが接続された際には、 `OnMidiInputDeviceAttached`, `OnMidiOutputDeviceAttached` が呼ばれます。
  - MIDIデバイスが接続解除された際には、 `OnMidiInputDeviceDetached`, `OnMidiOutputDeviceDetached` が呼ばれます。

```
public void OnMidiInputDeviceAttached(string deviceId)  
{  
    // MIDI受信デバイスが接続された。  
}  
  
public void OnMidiOutputDeviceAttached(string deviceId)  
{  
    // MIDI送信デバイスが接続された。  
    Debug.Log($"MIDI device attached. deviceId: {deviceId}, name:  
{MidiManager.Instance.GetDeviceName(deviceId)}");  
}  
  
public void OnMidiInputDeviceDetached(string deviceId)  
{  
    // MIDI受信デバイスの接続が解除された。  
}  
  
public void OnMidiOutputDeviceDetached(string deviceId)  
{  
    // MIDI送信デバイスの接続が解除された。  
    Debug.Log($"MIDI device detached. deviceId: {deviceId}, name:  
{MidiManager.Instance.GetDeviceName(deviceId)}");  
}
```

コードの全容は `Assets/MIDI/Samples/Scripts/MidiSampleScene.cs` ファイルにあります。

## deviceIdからMIDIデバイスの情報を取得する

- `MidiManager.Instance.GetDeviceName(string deviceId)` メソッドを呼び出して、指定されたデバイスIDからデバイス名を取得します。
- `MidiManager.Instance.GetVendorId(string deviceId)` メソッドを呼び出して、指定されたデバイスIDからベンダーIDを取得します。
  - いくつかのプラットフォーム・MIDIの接続種別(BLE MIDI, RTP MIDI)ではサポートされていません。これらの環境では空文字列が返却されます。
- `MidiManager.Instance.GetProductId(string deviceId)` メソッドを呼び出して、指定されたデバイスIDからプロダクトIDを取得します。
  - いくつかのプラットフォーム・MIDIの接続種別(BLE MIDI, RTP MIDI)ではサポートされていません。これらの環境では空文字列が返却されます。

デバイスが切断された場合には、これらのメソッドは空文字列を返却します。

GetVendorId / GetProductIdメソッドが利用可能な環境

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の例

APIが異なるため、プラットフォームによって取得されるVendorIdが異なります。

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

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

APIが異なるため、プラットフォームによって取得されるProductIdが異なります。

<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イベントの受信

GameObject もしくは C# オブジェクトを使ったMIDIメッセージの受信

1. `IMidiEventHandler.cs` に定義されている受信インターフェースを実装します。実装するクラス名は→ のようなものです `IMidiXXXXEventHandler`.
  - Note On イベントを受信したい場合には、 `IMidiNoteOnEventHandler` インタフェースを実装します。
2. イベント受信をするGameObjectを登録するため、  
`MidiManager.Instance.RegisterEventHandleObject` メソッドを呼び出します。
3. MIDIイベントを受信したら、実装したメソッドが呼ばれます。

```
public class MidiSampleScene : MonoBehaviour, IMidiAllEventsHandler,
IMidiDeviceEventHandler
{
    private void Awake()
    {
        // このgameObjectでMIDIイベントを受信します。
        // gameObjectは IMidiXXXXEventHandler を実装する必要があります。
        MidiManager.Instance.RegisterEventHandleObject(gameObject);
        ...
    }
}
```

MIDIイベントの受信:

```
public void OnMidiNoteOn(string deviceId, int group, int channel, int note, int
velocity)
{
    // Note Onのイベントを受信した。
    Debug.Log($"OnMidiNoteOn channel: {channel}, note: {note}, velocity:
{velocity}");
}

public void OnMidiNoteOff(string deviceId, int group, int channel, int note, int
velocity)
{
    // Note Offのイベントを受信した。
    Debug.Log($"OnMidiNoteOff channel: {channel}, note: {note}, velocity:
{velocity}");
}
```

コードの全容は `Assets/MIDI/Samples/Scripts/MidiSampleScene.cs` ファイルにあります。

C# event delegateを利用したMIDIメッセージの受信

別のある方法として、`MidiManager.Instance.OnMidiXXXXEvent` のような名前の C# の event delegate を使って、 MIDI イベントを受信することができます。

```

void OnMidiNoteOn(string deviceId, int group, int channel, int note, int velocity)
{
    // Note Onのイベントを受信した。
    Debug.Log($"OnMidiNoteOn channel: {channel}, note: {note}, velocity:
{velocity}");
}

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

```

## MIDIイベントの送信

- コードのどこかで `MidiManager.Instance.SendMidiXXXXXX` メソッドを呼び出します。一例:

```

// Note Onメッセージを送信
MidiManager.Instance.SendMidiNoteOn("deviceId", 0/*groupId*/, 0/*channel*/,
60/*note*/, 127/*velocity*/);

```

- 指定する `deviceId` は `MidiManager.Instance.DeviceIdSet` プロパティから取得できます。(型は `HashSet<string>` です)

```

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

...

if (GUILayout.Button("NoteOn"))
{
    // Note Onメッセージを送信
    MidiManager.Instance.SendMidiNoteOn(deviceIds[deviceIdIndex], 0, (int)channel,
(int)noteNumber, (int)velocity);
}

```

コードの全容は `Assets/MIDI/Samples/Scripts/MidiSampleScene.cs` ファイルにあります。

## Network MIDI (RTP MIDI)による通信を使う

iOS以外のプラットフォームでは試験的な機能です。

### ネットワークMIDI（RTP MIDI）サーバーを起動または停止する

- RTP-MIDIセッションを開始する:
  - セッション名とTCPポート番号を指定して `MidiManager.Instance.StartRtpMidiServer` メソッドを呼び出すことで、RTP-MIDIセッションの受け付けが開始されます。
  - これで指定したTCPポート番号を使用したサーバが開始されます。別のコンピュータはアプリに接続することができます。
- RTP-MIDIセッションを停止する:
  - `MidiManager.Instance.StopRtpMidiServer` メソッドを呼び出すと、RTP-MIDIセッションの通信を終了します。

```
#if !UNITY_IOS && !UNITY_WEBGL
    // セッション名「RtpMidiSession」、ポート5004を利用して RTP MIDI サーバーを開始します。
    MidiManager.Instance.StartRtpMidiServer("RtpMidiSession", 5004);

    ...
    // セッションを終了します。
    MidiManager.Instance.StartRtpMidiServer(5004);
#endif
```

### Network MIDI (RTP MIDI) server

RTP MIDI機能は 広報(advertise)・発見(discover)の仕組みがないため、クライアントはRTP MIDIサーバのアドレスとポートを知っている必要があります。

- 別のRTP-MIDIが動作しているコンピュータへ接続する:
  - `MidiManager.Instance.ConnectToRtpMidiServer` メソッドを呼び出すとRTP-MIDIサーバに接続します。

```
#if !UNITY_WEBGL || UNITY_EDITOR
    // 別マシンの RTP-MIDI セッションに接続する。
    MidiManager.Instance.ConnectToRtpMidiServer("RtpMidiSession", 5004, new
    IPPEndPoint(IPAddress.Parse("192.168.0.111"), 5004));

    ...
    // セッションから切断します。
    MidiManager.Instance.DisconnectFromListener(5004, new
    IPPEndPoint(IPAddress.Parse("192.168.0.111"), 5004));
#endif
```

接続が確立されると、コールバック `OnMidiInputDeviceAttached` と `OnMidiOutputDeviceAttached` が呼ばれます。コールバックの引数の `deviceId` を利用してMIDIメッセージの送受信ができます。

## シーケンサー機能の利用

`javax.sound.midi` パッケージから移植されたシーケンサー機能です。

- シーケンサーはStandard MIDI File(SMF)の読み書きができます。
- シーケンサーはMIDIイベントをTrackオブジェクトに記録できます。
  - 記録されたTrackはSMFとしてエクスポートできます。
- シーケンサーは記録したTrackやSMFを再生できます。

### シーケンサーの作成と開始

```
// 新しいシーケンサーのインスタンスを作成し、開く
var isSequencerOpened = false;
var sequencer = new SequencerImpl(() => { isSequencerOpened = true; });
sequencer.Open();
```

コードの全容は `Assets/MIDI/Samples/Scripts/MidiSampleScene.cs` ファイルにあります。

### SMFをシーケンスとして読み出し、再生する

```
// シーケンサーに、現在接続されているMIDI入出力デバイスの情報を反映します。
sequencer.UpdateDeviceConnections();

// FileStreamからSMFを読み込み、シーケンサーに設定します。
using var stream = new FileStream(smfPath, FileMode.Open, FileAccess.Read);
sequencer.SetSequence(stream);
// シーケンスを再生します。
sequencer.Start();

...
// 再生を停止します。
sequencer.Stop();
```

### シーケンスを記録する

```
// シーケンサーに、現在接続されているMIDI入出力デバイスの情報を反映します。
sequencer.UpdateDeviceConnections();

// 記録するための新しいシーケンスを追加します。
sequencer.SetSequence(new Sequence(Sequence.Ppq, 480));
// シーケンスへのMIDIデータの記録を開始します。
sequencer.StartRecording();
```

```
...
```

```
// 記録を停止します。  
sequencer.Stop();
```

## シーケンスをSMFとして書き出す

```
var sequence = sequencer.GetSequence();  
// シーケンスの長さをチェック  
if (sequence.GetTickLength() > 0)  
{  
    // シーケンスをSMFとして書き出します。  
    using var stream = new FileStream(recordedSmfPath, FileMode.Create,  
    FileAccess.Write);  
    MidiSystem.WriteSequence(sequence, stream);  
}
```

# MIDI 2.0機能の実装方法

MIDI 2.0 の基本機能の使用方法を説明します。

## MIDI 2.0プラグインの初期化

1. MonoBehaviour の `Awake` メソッド内から `MidiManager.Instance.InitializeMidi2` メソッドを呼び出します。

- `MidiManager` という名前の GameObject が、ヒエラルキーの `DontDestroyOnLoad` の下に自動的に作成されます。

△ NOTE: EventSystem コンポーネントが既に他の場所に存在する場合には、  
`gameObject.AddComponent<EventSystem>()` メソッドの呼出を  
`MidiManager.Instance.InitializeMidi` メソッドから削除してください。

MIDI 2.0のイベントをMIDI 1.0に変換(もしくはその逆)を行っているため、 `MidiManager` クラス上にMIDI1/2用の両方のメソッドが存在します。

```
private void Awake()
{
    // この gameObject でMIDIデータを受信する。
    // gameObjectは IMidi2XXXXEventHandler を実装する必要があります。
    MidiManager.Instance.RegisterEventHandleObject(gameObject);

    // MIDI機能を初期化
    MidiManager.Instance.InitializeMidi2(() => { });
}
```

## MIDI 2.0プラグインの終了

1. MonoBehaviour の `OnDestroy` メソッド内で `MidiManager.Instance.TerminateMidi2` を呼び出します。

- このメソッドはMIDI機能を使い終わって、シーンが終了する際に呼ばれるべきです。

```
private void OnDestroy()
{
    // 全てのMIDI機能を停止する。
    MidiManager.Instance.TerminateMidi2();
}
```

## MIDI 2.0 デバイスの接続/接続解除イベント処理

GameObjectまたはC#オブジェクトを使用してMIDI 2.0接続イベントを受信する

1. イベント受信用のインターフェース `IMidi2DeviceEventHandler` を実装します。

- 新しい MIDI 2.0 デバイスが接続されると、`OnMidi2InputDeviceAttached`、`OnMidi2OutputDeviceAttached` が呼び出されます。
- MIDI 2.0 デバイスが切断されると、`OnMidi2InputDeviceDetached`、`OnMidi2OutputDeviceDetached` が呼び出されます。

- `MidiManager.Instance.RegisterEventHandleObject` メソッドを呼び出して、イベントを受信するための `GameObject` または C# オブジェクトを登録します。
- MIDI デバイスが接続または切断されると、実装されたメソッドが呼び出されます。

```

public void OnMidi2InputDeviceAttached(string deviceId)
{
    // 新しい MIDI 2.0 受信デバイスが接続された。
}

public void OnMidi2OutputDeviceAttached(string deviceId)
{
    // 新しい MIDI 2.0 送信デバイスが接続された。
    Debug.Log($"MIDI device attached. deviceId: {deviceId}, name: {MidiManager.Instance.GetDeviceName(deviceId)}");
}

public void OnMidi2InputDeviceDetached(string deviceId)
{
    // MIDI 2.0 受信デバイスが接続解除された。
}

public void OnMidi2OutputDeviceDetached(string deviceId)
{
    // MIDI 2.0 送信デバイスが接続解除された。
    Debug.Log($"MIDI 2.0 device detached. deviceId: {deviceId}, name: {MidiManager.Instance.GetDeviceName(deviceId)}");
}

```

コードの全容は `Assets/MIDI/Samples/Scripts/Midi2SampleScene.cs` ファイルにあります。

### C# イベントデリゲートを使用して MIDI 2.0 接続イベントを受信する

別 の方法として、C# イベントデリゲートを使用して、`MidiManager.Instance.OnMidi2XXXXEvent` のような名前の MIDI イベントを受信することもできます。

```

void OnMidi2OutputDeviceAttached(string deviceId)
{
    // 新しい MIDI 2.0 送信デバイスが接続された。
    Debug.Log($"OnMidi2OutputDeviceAttached deviceId: {deviceId}");
}

...

MidiManager.Instance.OnMidi2OutputDeviceAttachedEvent +=
    OnMidi2OutputDeviceAttached;

```



## MIDI 2.0 イベント受信

GameObjectまたはC#オブジェクトを使用してMIDI 2.0メッセージを受信する

1. `IMidi2EventHandler.cs` ソースコードに記述されている、`IMidi2XXXXXEventHandler` または `IMidi1XXXXXEventHandler` のような名前のイベント受信インターフェイスを実装します。
  - Note On イベントを受信する場合は、`IMidi2NoteOnEventHandler` インターフェイスを実装します。
2. `MidiManager.Instance.RegisterEventHandleObject` メソッドを呼び出して、イベントを受信するための GameObject または C# オブジェクトを登録します。
3. MIDI イベントを受信すると、実装されたメソッドが呼び出されます。

```
public class Midi2SampleScene : MonoBehaviour, IMidi2AllEventsHandler,
IMidi2DeviceEventHandler
{
    private void Awake()
    {
        // この Unity ゲームオブジェクトで MIDI データを受信します。
        // gameObject は IMidiXXXXXEventHandler を実装する必要があります。
        MidiManager.Instance.RegisterEventHandleObject(gameObject);

        // この C# オブジェクトを使用して MIDI データを受信します。
        // オブジェクトのクラスは IMidiXXXXXEventHandler を実装する必要があります。
        MidiManager.Instance.RegisterEventHandleObject(obj);

        ...
    }
}
```

MIDIイベントの受信:

```
public void OnMidi2NoteOn(string deviceId, int group, int channel, int note, int
velocity, int attributeType, int attributeData)
{
    // Note On イベントを受信
    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 イベントを受信
    Debug.Log($"OnMidiNoteOff channel: {channel}, note: {note}, velocity:
{velocity}, attributeType: {attributeType}, attributeData: {attributeData}");
}
```

コードの全容は `Assets/MIDI/Samples/Scripts/Midi2SampleScene.cs` ファイルにあります。

## C# イベントデリゲートを使用して MIDI 2.0 メッセージを受信する

別 の方法として、イベント デリゲートを使用して、`MidiManager.Instance.OnMidi2XXXXEvent` または `MidiManager.Instance.OnMidi1XXXXEvent` のような名前の MIDI イベントを受信することもできます。

```
void OnMidi2NoteOn(string deviceId, int group, int channel, int note, int velocity, int attributeType, int attributeData)
{
    // Note On イベントを受信
    Debug.Log($"OnMidi2NoteOn channel: {channel}, note: {note}, velocity:
{velocity}, attributeType: {attributeType}, attributeData: {attributeData}");
}

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

## MIDI 2.0 イベント送信

- コードのどこかで `MidiManager.Instance.SendMidi2XXXXXX` または `MidiManager.Instance.SendMidi1XXXXXX` メソッドを呼び出します。  
(`SendMidi1XXXXXX` メソッドは MIDI 2.0 プロトコルを使用した MIDI 1 互換のための機能であり、値の精度は 7 ビットまたは 14 ビットです。)

例:

```
// Note On メッセージをMIDI 2.0で送信: velocityは 16 ビット
MidiManager.Instance.SendMidi2NoteOn("deviceId", 0/*groupId*/, 0/*channel*/,
60/*note*/, 65535/*velocity*/, 0/*attributeType*/, 0/*attribute*/);

// Note On メッセージをMIDI 1.0で送信: velocityは 7 ビット
MidiManager.Instance.SendMidi1NoteOn("deviceId", 0/*groupId*/, 0/*channel*/,
60/*note*/, 127/*velocity*/);
```

- `deviceId` は `MidiManager.Instance.Midi2OutputDeviceIdSet` プロパティ (型: `HashSet<string>`) から取得できます。

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

...
if (GUILayout.Button("NoteOn"))
{
    // Note On メッセージを送信
    MidiManager.Instance.SendMidi2NoteOn(deviceIds[deviceIdIndex], 0,
(int)channel, (int)noteNumber, (int)velocity, (int)attributeType, (int)attribute);
}
```

コードの全容は `Assets/MIDI/Samples/Scripts/Midi2SampleScene.cs` ファイルにあります。

## ネットワークMIDI 2.0 (UDP MIDI 2.0) トランスポートの使用

### ネットワークMIDI 2.0 (UDP MIDI 2.0)サーバの開始

- UDP-MIDI セッションの受け入れを開始するには、`MIDI エンドポイント名`、`mDNS サービス インスタンス名`、`udp ポート番号` を指定して `MidiManager.Instance.StartUdpMidi2Server` メソッドを呼び出します。
  - エラー処理用のAction (NAK コマンドを受信したときに呼び出される) を使用する場合は、`onErrorAction` 引数を指定して `MidiManager.Instance.StartUdpMidi2Server` を呼び出します。
  - 共有シークレット (パスフレーズ) を使用した認証機能を使用する場合は、`authenticationSecret` 引数を指定して `MidiManager.Instance.StartUdpMidi2Server` を呼び出します。
  - ユーザー認証(アカウント名/パスワード)による認証機能を使用する場合は、`userAuthentications` 引数を指定して `MidiManager.Instance.StartUdpMidi2Server` を呼び出します。

```
#if !UNITY_WEBGL || UNITY_EDITOR
    // セッション名「MyMidi2Endpoint」、mDNS サービス インスタンス名「MyMIDI2Service」、ポート 12345
    // で UDP MIDI 2.0 サーバーを起動します。
    MidiManager.Instance.StartUdpMidi2Server(12345, "MyMidi2Endpoint",
    "MyMIDI2Service");

    // エラー処理ありの場合
    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}");
    });

    // パスフレーズありの場合
    MidiManager.Instance.StartUdpMidi2Server(12345, "MyMidi2Endpoint",
    "MyMIDI2Service", authenticationSecret: "MyUDP-MIDI2Secret");

    // ユーザー認証ありの場合
    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
```

ネットワークMIDI 2.0 (UDP MIDI 2.0) サーバーを検索して接続します

同じネットワークからネットワーク MIDI 2.0 (UDP MIDI 2.0) サーバーを検索するには、  
`MidiManager.Instance.StartDiscoverUdpMidi2Server()` メソッドを呼び出します。

```
#if !UNITY_WEBGL || UNITY_EDITOR
// サーバーの検索を開始
MidiManager.Instance.StartDiscoverUdpMidi2Server();

...
// サーバーの検索を停止
MidiManager.Instance.StopDiscoverUdpMidi2Server();
#endif
```

検出されたすべてのサーバーは、`MidiManager.Instance.GetDiscoveredUdpMidi2Servers()` メソッドから取得できます。これはサーバーの サービス インスタンス名 (文字列のリスト) を返します。この サービス インスタンス名 を使用してサーバーに接続できます。

```
#if !UNITY_WEBGL || UNITY_EDITOR
// 検出されたサーバーを取得する
var discoveredServers = MidiManager.Instance.GetDiscoveredUdpMidi2Servers();

// 最初に検出されたサーバーにクライアント名「UDP-MIDI2-Client」で接続します。
if (discoveredServers.Count > 0)
{
    MidiManager.Instance.ConnectToUdpMidi2Server(discoveredServers[0], "UDP-MIDI2-
Client");
}
#endif
```

接続が確立されると、コールバック メソッド `OnMidi2InputDeviceAttached`、  
`OnMidi2OutputDeviceAttached` が呼び出されます。  
これらのコールバックの `deviceId` 引数を使用して送信または受信できます。

## UmpSequencer機能の使用

ライブラリには MIDI 2.0 機能用のシーケンサーがあります。

- シーケンサーは MIDI 2.0 クリップ ファイル (.midi2 ファイル) を読み書きできます。
- シーケンサーは、MIDI 2.0 デバイスからの MIDI 2.0 イベントを `UmpSequence` オブジェクトに記録できます。
  - 記録された `UmpSequence` は MIDI 2.0 クリップ ファイルデータにエクスポートできます。
- シーケンサーは記録された `UmpSequence` を再生したり、MIDI 2.0 クリップ ファイルデータを読み取ったりできます。

## UmpSequencerの作成と利用開始

```
var isSequencerOpened = false;
// 新しいシーケンスインスタンスを作成して開きます
var sequencer = new UmpSequencer(() => { isSequencerOpened = true; });
sequencer.Open();
```

コードの全容は [Assets/MIDI/Samples/Scripts/Midi2SampleScene.cs](#) ファイルにあります。

MIDI 2.0 クリップを `UmpSequence` として読み込み、再生します。

```
// FileStreamからMIDIクリップを読み込み、UmpSequencerに設定する
using var stream = new FileStream(midiClipPath, FileMode.Open, FileAccess.Read);
sequencer.SetSequence(stream);
// シーケンスの再生を開始
sequencer.StartPlaying();

...
// 再生を停止
sequencer.StopPlaying();
```

シーケンスを記録する

```
// 記録するための新しいシーケンスを設定する
sequencer.SetSequence(new Sequence());
// シーケンスにMIDIデータの記録を開始する
sequencer.StartRecording();

...
// 記録を停止する
sequencer.StopRecording();
```

シーケンスをMIDI 2.0クリップファイルに書き込む

MIDI 2.0 クリップ ファイルデータをストリームに書き込むには、`UmpSequenceWriter.WriteSequence` メソッドを使用します。

```
// シーケンサーからUmpSequenceを取得する
var sequence = sequencer.GetSequence();

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

MIDI 2.0 クリップファイルを SMF に変換します (またはその逆)

UMP シーケンスを MIDI シーケンスに変換するには、  
`SequenceConverter.ConvertUmpSequence(Sequence)` メソッドを使用します。

```
// シーケンサーからUmpSequenceを取得する
var sequence = sequencer.GetSequence();

// MIDIシーケンスに変換し、SMFに書き込む
using (var stream = new FileStream("recorded.smf", FileMode.Create,
FileAccess.Write))
{
    // SMFに書き込む
    // 2番目の引数: fileType: 0 (シングルトラックSMF)
    new
StandardMidiFileWriter().Write(SequenceConverter.ConvertUmpSequence(sequence), 0,
stream);
}
```

MIDI シーケンスを UMP シーケンスに変換するには、

`SequenceConverter.ConvertSequence(UmpSequence)` メソッドを使用します。

```
// MIDI 1.0シーケンサーからシーケンスを取得する
var sequence = sequencer.GetSequence();

// UMPシーケンスに変換し、MIDIクリップファイルに書き込む
using (var stream = new FileStream("recorded.midi2", FileMode.Create,
FileAccess.Write))
{
    // MIDIクリップファイルに書き込む
    UmpSequenceWriter.WriteSequence(SequenceConverter.ConvertSequence(sequence),
stream);
}
```

## その他の機能、実験的な機能

いくつかの応用的な機能の使用方法について説明します。  
実験的な機能も含まれています。

### RTP-MIDI 機能

iOS以外のプラットフォームでの試験的な機能です。

### MIDI Polyphonic Expression(MPE)機能の利用

現在、実験的に提供される機能で、充分にはテストされていません。  
もし不具合を見付けた場合、[サポートのリポジトリにIssueを追加してください。](#)

## MPEゾーンの定義

MPE機能を使う前に、最初にMIDIデバイスに対して MPEゾーン を定義する必要があります。

```
// MPE ゾーンをセットアップ  
MpeManager.Instance.SetupMpeZone(deviceId, managerChannel, memberChannelCount);
```

- `managerChannel` 0: lowerゾーン, 15: upperゾーン
- `memberChannelCount` 0: そのゾーンでMPE機能を無効にする。1から15: MPEを有効にする。
  - lowerゾーンとupperゾーンの両方が設定され、`memberChannelCount` の合計が14を超過した場合、最初に定義されたゾーンが縮小されます。

設定の例:

まず最初に、lowerゾーンを定義します。

```
// メンバー・チャンネルが10個のlowerゾーンを定義します。  
// lowerゾーンのマネージャーチャンネルは 0 です。  
MpeManager.Instance.SetupMpeZone(deviceId, 0, 10);
```

	Lowerゾーン: メンバー・チャンネル 10個	通常のチャンネル	
ch#	0   1   2   3   4   5   6   7   8   9   10   11   12   13   14   15		

次に、メンバー・チャンネルが7個のupperゾーンを追加します。

```
// メンバー・チャンネルが7個のupperゾーンを追加します。  
// upperゾーンのマネージャーチャンネルは 15 です。  
MpeManager.Instance.SetupMpeZone(deviceId, 15, 7);
```

lowerゾーンのメンバー・チャンネルは10から7に縮められます。

	Lowerゾーン: メンバー・ch. 7個	Upperゾーン: メンバー・ch. 7個	
ch#	0   1   2   3   4   5   6   7   8   9   10   11   12   13   14   15		

さらに続いて、lowerゾーンを削除します。

```
// 既存のゾーンを削除するには、メンバー数に0を指定します。  
MpeManager.Instance.SetupMpeZone(deviceId, 0, 0);
```

通常のチャンネル	Upperゾーン: メンバーch. 7個
ch#  0   1   2   3   4   5   6   7   8   9   10   11   12   13   14   15	

## MPEイベントの送信

MPEイベントを送信するには、 `MpeManager.Instance.SendMpeXXXXXX` メソッドを呼びます。

一例:

```
// Note Onメッセージを送信
// 引数 masterChannel は 0(lowerゾーン)、15(upperゾーン)のいずれかです。
MpeManager.Instance.SendMpeNoteOn(deviceId, masterChannel, noteNumber, velocity);
```

## MPEイベントの受信

MPEゾーンはゾーン設定イベント受信時に自動的に設定されます。

1. `IMpeEventHandler.cs` に定義されている受信インターフェースを実装します。実装するクラス名は→の  
ようなものです `IMpeXXXXEventHanlder`.
  - Note On イベントを受信したい場合には、 `IMpeNoteOnEventHandler` インタフェースを実装し  
ます。
2. イベント受信をするGameObjectを登録するため、  
`MidiManager.Instance.RegisterEventHandleObject` メソッドを呼び出します。
3. MPEイベントを受信したら、実装したメソッドが呼ばれます。
  - ゾーンが定義されたときは `OnMpeZoneDefined` メソッドが呼ばれます。

```
public class MpeSampleScene : MonoBehaviour, IMpeAllEventsHandler,
IMidiDeviceEventHandler
{
    private void Awake()
    {
        // このgameObject でMPE イベントを受信します。
        // gameObject は IMpeXXXXEventHanlder を実装する必要があります。
        MidiManager.Instance.RegisterEventHandleObject(gameObject);
        ...
    }
}
```

MPEイベントの受信:

```
public void OnMpeZoneDefined(string deviceId, int managerChannel, int
memberChannelCount)
{
    // ゾーン定義のイベントを受信した。
    // managerChannel 0: Lowerゾーン、15: upperゾーン
    // memberChannelCount 0: ゾーン削除、1-15: ゾーン定義
    Debug.Log($"OnMpeZoneDefined managerChannel: {managerChannel},
```

```
memberChannelCount: {memberChannelCount}");  
}  
  
public void OnMpeNoteOn(string deviceId, int channel, int note, int velocity)  
{  
    // Note Onのイベントを受信した。  
    Debug.Log($"OnMpeNoteOn channel: {channel}, ⚠ NOTE: {note}, velocity:  
{velocity}");  
}  
  
public void OnMpeNoteOff(string deviceId, int channel, int note, int velocity)  
{  
    // Note Offのイベントを受信した。  
    Debug.Log($"OnMpeNoteOff channel: {channel}, ⚠ NOTE: {note}, velocity:  
{velocity}");  
}
```

# Android, iOS, macOS: Nearby Connections MIDIの利用

GoogleのNearby Connectionsライブラリを使ったMIDIの送受信です。  
(現状ではこれは独自実装のため、類似のライブラリとの互換性はありません)

## 依存パッケージの追加

UnityのPackage Managerビューを開き、左上にある + ボタンを押し、 Add package from git URL... メニューを選択します。

以下のURLを指定します。

`ssh://git@github.com/kshoji/Nearby-Connections-for-Unity.git`

もしくは、

`git+https://github.com/kshoji/Nearby-Connections-for-Unity`

⚠ NOTE: 依存パッケージが過去にインストールされていれば、最新版への更新が必要になる場合があります。

## Scripting Define Symbolの設定

Nearby Connections MIDIの機能を有効にするには、Player settingsにてScripting Define Symbolを追加します。

`ENABLE_NEARBY_CONNECTIONS`

## Android向けの設定

Unityの Project Settings > Player > Identification > Target API Level 設定を API Level 33 以上に設定します。

## 近隣のデバイスへの広報(Advertise)

近隣のデバイスに対して自分のデバイスを広報するために、

`MidiManager.Instance.StartNearbyAdvertising()` メソッドを呼びます。

広報を停止するには、`MidiManager.Instance.StopNearbyAdvertising()` メソッドを呼びます。

```
if (isNearbyAdvertising)
{
    if (GUILayout.Button("Stop advertise Nearby MIDI devices"))
    {
        // 広報を停止します。
        MidiManager.Instance.StopNearbyAdvertising();
        isNearbyAdvertising = false;
    }
}
else
{
    if (GUILayout.Button("Advertise Nearby MIDI devices"))
    {
        // 近隣のデバイスに広報を開始します。
        MidiManager.Instance.StartNearbyAdvertising();
        isNearbyAdvertising = true;
    }
}
```

```
    }  
}
```

## 広報されたデバイスを見つける

Nearby Connections MIDIデバイスを見つけるために、

`MidiManager.Instance.StartNearbyDiscovering()` メソッドを呼びます。

探索を止めるには、`MidiManager.Instance.StopNearbyDiscovering()` メソッドを呼びます。

```
if (isNearbyDiscovering)  
{  
    if (GUILayout.Button("Stop discover Nearby MIDI devices"))  
    {  
        // デバイスの探索を停止します。  
        MidiManager.Instance.StopNearbyDiscovering();  
        isNearbyDiscovering = false;  
    }  
}  
else  
{  
    if (GUILayout.Button("Discover Nearby MIDI devices"))  
    {  
        // 近隣で広報しているデバイスの探索を開始します。  
        MidiManager.Instance.StartNearbyDiscovering();  
        isNearbyDiscovering = true;  
    }  
}
```

## MIDIデータをnearbyで送受信する

MIDIデータの送受信方法については通常のMIDIと同様です。

## Meta Quest(Oculus Quest)デバイスを使う

現在、テストデバイスとしてはOculus Quest 2を使用しています。

### USB MIDIデバイスと接続する

USB MIDI デバイスの接続を検出するには、以下のコメントを解除してください。

`PostProcessBuild.cs` より抜粋:

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

        // ⚠ NOTE: Meta Quest 2(Oculus Quest 2) で USB MIDI 機能を使用したい場合は、
        USB MIDI デバイスの接続を検出するために以下のコメントを解除してください。
        // androidManifest.AddUsbIntentFilterForOculusDevices();
```

### Bluetooth MIDIデバイスと接続する

AndroidのBLE MIDIデバイスの接続に[CompanionDeviceManager](#)が使えます。

この機能を有効にするには、[Scripting Define Symbols](#) の設定に  
`FEATURE_ANDROID_COMPANION_DEVICE` を追加します。

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

## MIDI 2.0に関する実験的な機能

### ネットワークMIDI 2.0（UDP MIDI 2.0）機能

この機能は [Network MIDI 2.0 \(UDP\) Transport Specification](#) を実装します。

現時点では他の実装がないため、相互の互換性は保証できません。

別の実装を使用していて、UDP MIDI 2.0 経由で通信できない場合は、お知らせください。

### MIDIクリップファイル/MIDIコンテナファイルの読み書きサポート

MIDI クリップ ファイル仕様は現在リリース バージョンとして公開されています。

しかし、MIDI クリップ ファイルはまだ普及しておらず、サンプル ファイルも少ないため、互換性仕様を満たしているかどうかはわかりません。

MIDI コンテナ ファイルの仕様はドラフト状態です。

将来的に仕様が変更される可能性があるため、現時点ではこの機能の使用はお勧めしません。

# テストしたデバイス

---

- 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:** このデバイスはiOSでは動きませんでした。
  - 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](#)

# 連絡先

---

## GitHubでの不具合報告、サポート

- GitHubのサポートリポジトリ <https://github.com/kshoji/Unity-MIDI-Plugin-supports>
  - 問題の検索と報告: <https://github.com/kshoji/Unity-MIDI-Plugin-supports/issues>

## プラグイン作者

- Kaoru Shoji/庄司 薫 : [0x0badc0de@gmail.com](mailto:0x0badc0de@gmail.com)
- github: <https://github.com/kshoji>

## 使用した自作のオープンソースソフトウェア

- 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>

## 他者が作成したオープンソースソフトウェアパッケージ:

- 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

## 他者提供による、使用したサンプルMIDIデータ

いずれも、UnityWebRequestのURLとして指定しています。SMFやMIDI Clipのバイナリデータ自体はパッケージには含まれていません。

### Sample SMF

オリジナルのウェブサイトが <https://bitmidi.com/> のサービスを提供していないため、サンプルコードでは別のサイトのURLを指定しています。(<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

- MIDIクリップのテストファイルは、このgithubリポジトリから取得しました。: [test-midi-files](#).
  - これらのファイルは [MIT ライセンス](#) に基づいてライセンスされています。

# バージョン履歴

---

- v1.0 初期リリース: 2021/08/07
- v1.1 更新リリース: 2022/04/11
  - 追加: MIDI シーケンサ(MIDIシーケンスの再生・録音)
  - 追加: SMFの読み取り・書き出し
  - 追加: AndroidでのBLE MIDI Peripheral機能
  - 修正: AndroidでのUSB MIDI受信時の問題
  - 修正: Android・iOSでのBLE MIDI送信時の問題
  - 修正: AndroidでのBLE MIDI送信(velocity = 0でのNoteOn)の問題
- v1.2.0 更新リリース: 2022/08/17
  - 追加: Androidほかプラットフォーム向けの、試験的な RTP-MIDIサポート
  - 追加: Universal Windows Platform(UWP)向けのUSB MIDIサポート
  - 追加: Android 12の新しいBluetoothパーミッションのサポート
  - 修正: iOS, AndroidでのMIDI送受信のパフォーマンス向上
  - 修正: シーンが複数回追加された際にEventSystemが複数個作成されるエラー
  - 修正: AndroidのBLE MIDIでのタイムスタンプの問題
- v1.2.1 バグ修正: 2022/08/29
  - 修正: シーケンサーのスレッドが閉じたあとも残る問題
  - 修正: AndroidでのProgramChangeメッセージの受信が失敗する
  - 修正: サンプルシーンでのSystem exclusiveのログが正しく表示されない
  - 修正: UWPでThreadInterruptedExceptionが発生する
  - 修正: SMFでSystem exclusiveを読み書きすると起きる問題
  - 修正: いくつかのパフォーマンス改善
- v1.3.0 更新リリース: 2022/10/13
  - 追加: Standalone OSX, Windows, Linuxプラットフォーム対応
  - 追加: WebGLプラットフォーム対応
  - 追加: Unity Editor OSX, Windows, Linux対応
  - 変更: シーケンサーの実装を Thread から Coroutine に
  - 修正: iOS/OSX でのデバイス接続・切断時の問題
- v1.3.1 バグ修正: 2023/05/18
  - [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のパーミッション要求についての問題を解消
  - AndroidのCompanionDeviceManager経由での接続をサポート
- v1.3.2 バグ修正: 2023/05/19
  - Androidのコンパイルエラーを修正
  - SMF再生時のMIDIイベントの順序が正しくなるよう修正
- v1.3.3 バグ修正: 2023/05/25
  - WebGLのMIDI送信失敗を修正
- v1.3.4 バグ修正: 2023/06/05

- 過去に接続していたデバイスIDと同じで、デバイス名が違うものが接続されたときに、間違ったデバイス名を取得する不具合を修正
  - iOS: BLE MIDIデバイス検索のポップアップに「完了」ボタンを追加
  - サンプルシーン: BLE MIDIデバイスの検索は Android/iOS のみ使えるよう調整
  - MidiManager のシングルトンの設計を調整
- v1.4.0 更新リリース: 2023/12/07
    - 追加: Android, iOS, macOS向けの Nearby Connections MIDI サポート
    - 追加: WebGL向けの Bluetooth LE MIDI サポート
    - 修正: iOSデバイスでの デバイス接続/切断時のコールバックが間違っていたのを修正
  - v1.4.1 更新リリース: 2024/02/22
    - 修正: Linuxプラットフォームでのリンクエラーを修正
    - 追加: WebGLプラットフォームでベンダ名/デバイス名をサポート
    - 追加: iOS/MacOS/Linux/Android プラットフォームでアプリ間MIDI接続(仮想MIDI)をサポート
    - 修正: サンプルシーンのメモリリークを修正
  - v1.4.2 バグ修正: 2024/02/27
    - 修正: WebGLのサンプルシーンの初期化が失敗する
  - v1.4.3 バグ修正: 2024/03/12
    - 修正: Bluetooth がオフの場合、Android プラグインの初期化が失敗する
    - 修正: Android 14 で USB MIDI デバイスを開けない
    - 修正: Windows プラグインがMIDIデバイスの更新に失敗する
    - 修正: Linux プラグインが MidiManager の終了時にクラッシュする
    - 修正: ゲームプレイを停止した後、Unity エディターが MIDI 機能を停止する
    - 追加: 入出力の deviceld を取得する機能 (全 deviceld の取得メソッドとは別に追加)
    - 追加: MIDI シーケンサーへのデバイス接続を指定する機能
    - 追加: MIDI シーケンサーの再生終了イベントを受信するコールバック
    - 追加: MIDI シーケンサーのイベント タイミングの精度を向上
    - 追加: MIDI シーケンサーの再生位置をマイクロ秒時間で指定できるように
  - v1.4.4 バグ修正: 2024/04/09
    - 修正: Androidプラグインのロードが失敗した時に初期化が中断される
    - 修正: AndroidのBluetooth MIDIがMIDIメッセージを送信できない
    - 追加: AndroidプラットフォームでProGuard minify設定のサポート
  - v1.4.5 バグ修正: 2024/04/13
    - 修正: AndroidのBluetooth MIDIが高負荷のときに送信に失敗する
    - 更新: Androidの新しいBluetooth LEのAPIを使うよう対応
    - 修正: AndroidのCompanionDeviceManager初期化に関する問題を修正。本バージョンから、この機能には **ACCESS\_FINE\_LOCATION** パーミッションが必要になります。
  - v1.5.0 更新リリース: 2024/06/05
    - 追加: MIDI Polyphonic Expression機能(現在は実験的な状態での提供です)
    - 内部実装を全面的にリファクタリング
    - 更新: SMF読み込みの互換性を向上
    - 修正: SMFの再生が冒頭のイベント付近で失敗する
    - 修正: Androidの入力デバイスの初期化に失敗する
  - v1.5.1 バグ修正: 2024/09/05
    - 修正 12.3より前の古いmacOSバージョンでUnityエディタが動かない
      - macOS 10.13以降で動作するよう調整
    - 修正 サンプルシーンのインポート時にGUIDが重複している

- v2.0.0 更新リリース: 2025/05/05
  - 追加: iOS、macOS スタンドアロン、Android、スタンドアロン Linux 向けの MIDI 2.0 サポート
  - 追加: ネットワークMIDI 2.0(UDP MIDI 2.0)機能
  - 追加: MIDI 2.0 クリップ/コンテナファイルのインポート/エクスポートのサポート
    - MIDI 2.0 コンテナ ファイル形式は現在ドラフト状態の仕様であるため、将来変更される予定です。
  - 追加: C# イベント デリゲートによる MIDI 受信
  - 追加: C#オブジェクトによるMIDI受信
  - 修正: シンボルレ ENABLE\_NEARBY\_CONNECTIONS が定義されている場合の、コンパイルエラー
  - 修正: Linux プラットフォームで一部のイベント (TuneRequest、TimingClock、Start、Stop、Continue、ActiveSensing、Reset) が受信されない。
  - 修正: Linuxプラットフォームで一部のアプリ間MIDIデバイスが接続されない
  - 修正: 再生モードを停止した後でもエディターでMIDIメッセージを受信し続ける