

最新的文档在 GitHub 上。

如果有任何问题或问题，请在 GitHub 上发布问题。

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

本文档解释了如何安装插件，以及如何使用插件的功能。

这是机器翻译的文件。更多详细信息，请参阅原始英文文档。

目录

- 目录
- 移动和桌面的 MIDI 插件
- 适用于平台的 MIDI 接口
 - 关于限制
 - Android
 - iOS / OSX
 - UWP
 - Windows
 - Linux
 - WebGL
 - Network MIDI
- 安装插件
- 关于构建 PostProcessing
 - PostProcessing: iOS
 - PostProcessing: Android
 - PostProcessing: Android, Meta Quest(Oculus Quest)
 - Android: 使用 CompanionDeviceManager 查找 BLE MIDI 设备
- 实现功能
 - 初始化插件
 - 终止插件
 - 连接/断开 MIDI 设备的信号处理
 - 从 deviceId 获取 MIDI 设备信息
 - GetVendorId / GetProductId 方法可以使用的环境
 - VendorId 示例
 - ProductId 示例
 - MIDI 信号接收
 - 使用 GameObject 或 C# 对象接收 MIDI 消息
 - C# 使用信号委托接收 MIDI 信号
 - MIDI 信号发送
 - 使用网络 MIDI (RTP MIDI) 通信
 - 启动或停止网络 MIDI (RTP MIDI) 服务器
 - 网络 MIDI (RTP MIDI) 服务器
 - 使用定序器功能
 - 创建并开始使用 Sequence
 - 将 SMF 读取为 Sequence，并播放它
 - 记录一个 Sequence

- 将序列写入 SMF 文件
- 如何实现 MIDI 2.0 功能
 - MIDI 2.0 插件的初始化
 - MIDI 2.0 插件退役
 - MIDI 2.0 设备连接/断开信号处理
 - 使用 GameObject 或 C# 对象接收 MIDI 2.0 连接信号
 - C# 使用信号委托接收 MIDI 2.0 连接信号
 - MIDI 2.0 信号接收
 - 使用 GameObject 或 C# 对象接收 MIDI 2.0 消息
 - C# 使用信号委托接收 MIDI 2.0 消息
 - MIDI 2.0 信号传输
 - 使用网络 MIDI 2.0 (UDP MIDI 2.0) 传输
 - 启动网络 MIDI 2.0 (UDP MIDI 2.0) 服务器
 - 查找并连接到网络 MIDI 2.0 (UDP MIDI 2.0) 服务器
 - 使用 UmpSequencer 函数
 - UmpSequencer 的生成和利用开始
 - 加载并播放 MIDI 2.0 剪辑作为 UmpSequence。
 - 记录序列
 - 将序列写入 MIDI 2.0 剪辑文件
 - 将 MIDI 2.0 剪辑文件转换为 SMF (反之亦然)
- 其他功能、实验性功能
 - 使用 RTP-MIDI
 - 使用 MIDI Polyphonic Expression (MPE) 功能
 - 定义 MPE 区域
 - 发送 MPE 信号
 - 接收 MPE 信号
 - Android, iOS, macOS: 使用 Nearby Connections MIDI
 - 添加依赖包
 - Scripting Define Symbol 设置
 - Android 设置
 - 向附近的设备发布广告
 - 找到您广告的设备
 - 与 Nearby 的设备发送和接收 MIDI 数据
 - 使用 Meta Quest (Oculus Quest) 设备
 - 连接到 USB MIDI 设备
 - 连接 Bluetooth MIDI 设备
 - 实验性 MIDI 2.0 功能
 - 网络 MIDI 2.0 (UDP MIDI 2.0) 功能
 - 支持读写 MIDI Clip 文件/MIDI Container 文件
- 测试设备
- 联系
 - 在 GitHub 上报告问题
 - 关于插件作者
 - 使用的开源软件由我创建
 - 其他人创建的开源软件包:
 - 其他人使用的示例 MIDI 数据

- Sample SMF
 - Sample MIDI Clip
- 版本历史

移动和桌面的 MIDI 插件

此插件为您的移动应用程序（iOS、Android 平台）、桌面应用程序（Windows、OSX、Linux）和 WebGL 应用程序提供 MIDI 收发功能。

目前已实现[MIDI 1.0 协议](#)和[MIDI 2.0 协议 \(UMP\)](#)。

适用于平台的 MIDI 接口

下面列出了每个平台可用的 MIDI 接口。

平台	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	○	○	○	○	○	○	△(实验特 征)
Android	○	○	△(实验特 征)	○	○	○	△(实验特 征)
Universal Windows Platform	-	○	△(实验特 征)	-	○	-	△(实验特 征)
Standalone OSX, Unity Editor OSX	○	○	○	○	○	○	△(实验特 征)
Standalone Linux, Unity Editor Linux	○	○	△(实验特 征)	-	○	○	△(实验特 征)
Standalone Windows, Unity Editor Windows	-	○	△(实验特 征)	-	○	-	△(实验特 征)
WebGL	○	○	-	-	-	-	-

关于限制

Android

- API 级别 12 (Android 3.1) 或更高版本支持 USB MIDI。
- API 级别 18 (Android 4.3) 或更高版本支持蓝牙 MIDI。
- API 级别 23 (Android 6.0) 或更高版本支持 Inter-App MIDI。
- API 级别 23 (Android 6.0) 或更高版本支持 MIDI 2.0。
- 使用 Mono backend 构建会导致延迟问题，并且仅支持 armeabi-v7a 架构。
 - 要解决此问题，请将 Unity 设置 Project Settings > Player > Configuration > Scripting Backend 更改为 IL2CPP。
- 要使用 Nearby Connections MIDI 功能，需要 API 级别 28 或更高版本。要使用此功能，您的应用必须在 API 级别 33 (Android 13.0) 或更高版本上进行编译。

iOS / OSX

- 支持 iOS 12.0 或更高版本。
- Bluetooth MIDI 支持仅限于 Central 模式。

UWP

- 支持的 UWP 平台版本 10.0.10240.0 或以上。
- 目前不支持 MIDI 2.0 (USB)。
 - 适用于 Windows 的 MIDI 2.0 功能预计将很快发布，随后将实施。
- 不支持 Bluetooth MIDI。
- 要使用网络 MIDI(RTP-MIDI) 功能，请在 Project Settings > Player > Capabilities 设置中启用 PrivateNetworkClientServer。

Windows

- 不支持 Bluetooth MIDI。
- 目前不支持 MIDI 2.0 (USB)。

Linux

- 如果 MIDI1 和 MIDI2 协议共存于一台设备上，则只会找到 MIDI2 端口。

WebGL

- 支持的 MIDI 设备取决于运行的操作系统/浏览器环境。
- WebGL 可能无法通过 UnityWebRequest 访问其他服务器资源，因此请将资源文件（例如 SMF）放入 **StreamingAssets**。
- WebGL 不能使用原始 UDP/TCP 连接，因此网络 MIDI 功能（RTP MIDI、UDP MIDI 2.0）不可用。
- 目前，WebGL 无法处理 USB MIDI 2.0 设备，因此除了读取和写入 MIDI Clip 文件之外，MIDI 2.0 功能不可用。
- 您 应该修改 **WebGLTemplates** 目录的 **index.html** 文件，如下所示。这可以从 **unityInstance** 变量访问 Unity 的运行时。
 - 或者，将 **MIDI/Samples/WebGLTemplates** 文件复制到 **Assets/WebGLTemplates**，然后从 **Project Settings > Player > Resolution and Presentation > WebGL Template** 设置中选择 **Default-MIDI** 或 **Minimal-MIDI** 模板。
 - 有关更多信息，请参阅 [WebGL 模板的 Unity 官方文档](#)。

摘自原始代码:

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

修改的: 添加 **unityInstance** 全局变量。

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

Network MIDI

- 上面指定的实验支持不支持纠错（RTP MIDI Journaling）协议。

安装插件

1. 从 Asset Store 视图导入 unitypackage。
 - 如果您更新了软件包，则 Assets/MIDI/Plugins/Android 中可能有旧版本的文件。在这种情况下，请删除所有旧版本的aar文件。
2. 选择应用的平台；iOS 或安卓。并构建示例应用程序。
 - 示例场景位于 Assets/MIDI/Samples 目录中。
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。

关于构建 PostProcessing

PostProcessing: iOS

- 在构建后期处理时会自动添加额外的框架。
 - 附加框架：CoreMIDI.framework、CoreAudioKit
- Info.plist 将在构建后期处理时自动调整。
 - 附加属性：NSBluetoothAlwaysUsageDescription

PostProcessing: Android

- AndroidManifest.xml 将在构建后处理时自动调整。
 - 附加权限：
 - android.permission.BLUETOOTH
 - android.permission.BLUETOOTH_ADMIN
 - android.permission.ACCESS_FINE_LOCATION
 - android.permission.BLUETOOTH_SCAN
 - android.permission.BLUETOOTH_CONNECT
 - android.permission.BLUETOOTH_ADVERTISE
 - 附加功能：
 - android.hardware.bluetooth_le
 - android.hardware.usb.host

PostProcessing: Android, Meta Quest(Oculus Quest)

- 如果您想在 Oculus(Meta) Quest 2 上使用 USB MIDI 功能, 请在下方取消注释以检测 USB MIDI 设备连接。

PostProcessBuild.cs 的部分

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

        // ⚠ NOTE: Oculus Quest 2 取消注释此行
        // androidManifest.AddUsbIntentFilterForOculusDevices();
```

Android: 使用 CompanionDeviceManager 查找 BLE MIDI 设备

您可以使用 [CompanionDeviceManager](#) 在 Android 上连接 BLE MIDI 设备。

要启用此功能, 请将 `FEATURE_ANDROID_COMPANION_DEVICE` 添加到 [Scripting Define Symbols](#) 设置。

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

⚠ NOTE: 此功能允许您的 Meta(Oculus) Quest 设备发现并连接到蓝牙 MIDI 设备。

实现功能

解释如何使用基本 MIDI 功能。

初始化插件

1. 调用 `MidiManager.Instance.InitializeMidi` 方法 `Awake` 在 `MonoBehaviour` 中

- 一个名为 `MidiManager` 的游戏对象将 `DontDestroyOnLoad` 在层次视图

△ NOTE: 如果 `EventSystem` 组件已经存在于另一个代码中, 请删除在 `MidiManager.Instance.InitializeMidi` 方法中调用的 `gameObject.AddComponent()` 方法。

2. (仅限BLE MIDI)

- 调用 `MidiManager.Instance.StartScanBluetoothMidiDevices` 方法来扫描 BLE MIDI 设备。
 - 此方法应在 `InitializeMidi` 方法的回调操作中调用。

```
private void Awake()
{
    // 接收此游戏对象上的 MIDI 信号。
    // gameObject 必须实现 IMidiXXXXEventHandler
    MidiManager.Instance.RegisterEventHandleObject(gameObject);

    // 初始化 MIDI 功能
    MidiManager.Instance.InitializeMidi(() =>
    {
#if (UNITY_ANDROID || UNITY_IOS || UNITY_WEBGL) && !UNITY_EDITOR
        // 开始扫描 Bluetooth MIDI 设备
        MidiManager.Instance.StartScanBluetoothMidiDevices(0);
#endif
    });
}
```

终止插件

1. 调用 `MidiManager.Instance.TerminateMidi` 方法 `OnDestroy` 在 `MonoBehaviour` 中

- 在完成场景或完成使用 MIDI 功能时应调用此方法。

```
private void OnDestroy()
{
    // 停止所有 MIDI 功能。
    MidiManager.Instance.TerminateMidi();
}
```

连接/断开 MIDI 设备的信号处理

1. 使用 `MidiManager.Instance.RegisterEventHandleObject` 方法注册一个 `GameObject` 以接收信号。
2. 实现接口 `IMidiDeviceEventHandler` 来接收信号。
 - 连接新 MIDI 设备时会调用 `OnMidiInputDeviceAttached`、`OnMidiOutputDeviceAttached`。
 - 当 MIDI 设备断开连接时，会调用 `OnMidiInputDeviceDetached`、`OnMidiOutputDeviceDetached`。

```
public void OnMidiInputDeviceAttached(string deviceId)
{
    // 已连接 MIDI 接收设备。
}

public void OnMidiOutputDeviceAttached(string deviceId)
{
    // 已连接 MIDI 传输设备。
    Debug.Log($"MIDI device attached. deviceId: {deviceId}, name: {MidiManager.Instance.GetDeviceName(deviceId)}");
}

public void OnMidiInputDeviceDetached(string deviceId)
{
    // MIDI 接收设备已断开连接。
}

public void OnMidiOutputDeviceDetached(string deviceId)
{
    // MIDI 发送设备已断开连接。
    Debug.Log($"MIDI device detached. deviceId: {deviceId}, name: {MidiManager.Instance.GetDeviceName(deviceId)}");
}
```

所有代码都在 `Assets/MIDI/Samples/Scripts/MidiSampleScene.cs` 中。

从deviceId获取MIDI设备信息

- 调用 `MidiManager.Instance.GetDeviceName(string deviceId)` 方法从指定的设备ID中获取设备名称。
- 调用 `MidiManager.Instance.GetVendorId(string deviceId)` 方法从指定的设备ID中获取供应商ID。
 - 某些平台/MIDI 连接类型（BLE MIDI、RTP MIDI）不支持它。在这些环境中将返回空字符串。
- 调用 `MidiManager.Instance.GetProductId(string deviceId)` 方法从指定的设备ID中获取产品ID。
 - 某些平台/MIDI 连接类型（BLE MIDI、RTP MIDI）不支持它。在这些环境中将返回空字符串。

如果设备断开连接，这些方法将返回空字符串。

GetVendorId / GetProductId方法可以使用的环境

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示例

不同平台由于API不同，获取到的VendorId也不同。

Platform	Bluetooth MIDI	USB MIDI	Network MIDI (RTP-MIDI)	Nearby Connections MIDI
iOS	QUICCO SOUND Corp.	Generic	-	-
Android	QUICCO SOUND Corp.	1410	-	-

Platform	Bluetooth MIDI	USB MIDI	Network MIDI (RTP-MIDI)	Nearby Connections MIDI
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示例

不同平台由于API不同，获取到的ProductId也不同。

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 信号接收

使用 GameObject 或 C# 对象接收 MIDI 消息

1. 实现信号接收接口，用 `IMidiEventHandler.cs` 源码编写，命名为 `IMidiXXXXEventHandler`。
 - 如果要接收 Note On 信号，请实现 `IMidiNoteOnEventHandler` 接口。
2. 调用 `MidiManager.Instance.RegisterEventHandleObject` 方法注册 GameObject 接收信号；
3. 收到 MIDI 信号后，将调用已实现的方法。

```
public class MidiSampleScene : MonoBehaviour, IMidiAllEventsHandler,
IMidiDeviceEventHandler
{
    private void Awake()
    {
        // 接收此游戏对象上的 MIDI 信号。
        // gameobject 必须实现 IMidiXXXXEventHandler
        MidiManager.Instance.RegisterEventHandleObject(gameObject);
        ...
    }
}
```

接收 MIDI 信号：

```
public void OnMidiNoteOn(string deviceId, int group, int channel, int note, int
velocity)
{
    // 收到 Note On 信号。
    Debug.Log($"OnMidiNoteOn channel: {channel}, note: {note}, velocity:
{velocity}");
}

public void OnMidiNoteOff(string deviceId, int group, int channel, int note, int
velocity)
{
    // 收到 Note Off 信号。
    Debug.Log($"OnMidiNoteOff channel: {channel}, note: {note}, velocity:
{velocity}");
}
```

所有代码都在 `Assets/MIDI/Samples/Scripts/MidiSampleScene.cs` 中。

C# 使用信号委托接收 MIDI 信号

或者，您可以使用名称类似“`MidiManager.Instance.OnMidiXXXXEvent`”的 C# 信号委托来接收 MIDI 信号。

```
void OnMidiNoteOn(string deviceId, int group, int channel, int note, int velocity)
{
    // 收到 Note On 信号。
    Debug.Log($"OnMidiNoteOn channel: {channel}, note: {note}, velocity:
```

```
{velocity}");  
}  
  
...  
  
MidiManager.Instance.OnMidiNoteOnEvent += OnMidiNoteOn;
```

MIDI 信号发送

1. 调用 `MidiManager.Instance.SendMidiXXXXXX` 方法。

像这样:

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

2. `deviceId` 可以从 `MidiManager.Instance.DeviceIdSet` 属性 (类型 : `HashSet<string>`) 中获取。

```
deviceIds = MidiManager.Instance.OutputDeviceIdSet().ToArray();  
  
...  
  
if (GUILayout.Button("NoteOn"))  
{  
    // Note On发送消息  
    MidiManager.Instance.SendMidiNoteOn(deviceIds[deviceIdIndex], 0, (int)channel,  
(int)noteNumber, (int)velocity);  
}
```

所有代码都在 `Assets/MIDI/Samples/Scripts/MidiSampleScene.cs` 中。

使用网络 MIDI (RTP MIDI) 通信

这是 iOS 以外平台上的一项实验性功能。

启动或停止网络 MIDI (RTP MIDI) 服务器

- 启动 RTP-MIDI 会话:
 - 您可以通过调用“MidiManager.Instance.StartRtpMidiServer”方法并指定会话名称和 TCP 端口号来开始接受 RTP-MIDI 会话。
 - 服务器将使用指定的 TCP 端口号启动。另一台计算机可以连接到该应用程序。
- 停止 RTP-MIDI 会话:
 - 调用MidiManager.Instance.StopRtpMidiServer方法将结束RTP-MIDI会话的通信。

```
#if !UNITY_IOS && !UNITY_WEBGL
    // 使用会话名称“RtpMidiSession”和端口 5004 启动 RTP MIDI 服务器。
    MidiManager.Instance.StartRtpMidiServer("RtpMidiSession", 5004);

    ...

    // 结束会话。
    MidiManager.Instance.StopRtpMidiServer(5004);
#endif
```

网络 MIDI (RTP MIDI) 服务器

RTP MIDI 功能没有广告/发现机制，因此客户端必须知道 RTP MIDI 服务器的地址和端口。

- 连接到另一台运行 RTP-MIDI 的计算机:
 - 调用MidiManager.Instance.ConnectToRtpMidiServer方法将连接到RTP-MIDI服务器。

```
#if !UNITY_WEBGL || UNITY_EDITOR
    // 连接到另一台机器上的 RTP-MIDI 会话。
    MidiManager.Instance.ConnectToRtpMidiServer("RtpMidiSession", 5004, new
        IPEndPoint(IPAddress.Parse("192.168.0.111"), 5004));

    ...

    // 断开会话。
    MidiManager.Instance.DisconnectFromListener(5004, new
        IPEndPoint(IPAddress.Parse("192.168.0.111"), 5004));
#endif
```

一旦建立连接，就会调用回调“OnMidInputDeviceAttached”和“OnMidOutputDeviceAttached”。您可以使用回调的 deviceId 参数发送和接收 MIDI 消息。

使用定序器功能

音序器功能从 `javax.sound.midi` 包移植。

- 音序器可以读取和写入标准 MIDI 文件 (SMF)。
- 音序器可以将 MIDI 信号记录到 Track 对象。
 - 录制的 Track 可以导出为 SMF。
- 音序器可以播放录制的 Track 和 SMF。

创建并开始使用 Sequence

```
// 创建并打开一个新的定序器实例
var isSequencerOpened = false;
var sequencer = new SequencerImpl(() => { isSequencerOpened = true; });
sequencer.Open();
```

所有代码都在 `Assets/MIDI/Samples/Scripts/MidiSampleScene.cs` 中。

将 SMF 读取为 Sequence，并播放它

```
// 反映音序器上当前连接的 MIDI 输入/输出设备的信息。
sequencer.UpdateDeviceConnections();

// 从 FileStream 加载 SMF 并将其设置到定序器。
using var stream = new FileStream(smfPath, FileMode.Open, FileAccess.Read);
sequencer.SetSequence(stream);
// 播放序列。
sequencer.Start();

...
// 停止播放。
sequencer.Stop();
```

记录一个 Sequence

```
// 反映音序器上当前连接的 MIDI 输入/输出设备的信息。
sequencer.UpdateDeviceConnections();

// 添加新的录制序列。
sequencer.SetSequence(new Sequence(Sequence.Ppq, 480));
// 开始将 MIDI 数据记录到序列中。
sequencer.StartRecording();

...
// 停止录制。
sequencer.Stop();
```

将序列写入 SMF 文件

```
var sequence = sequencer.GetSequence();
// 检查序列长度
if (sequence.GetTickLength() > 0)
{
    // 将序列导出为 SMF。
    using var stream = new FileStream(recordedSmfPath, FileMode.Create,
FileAccess.Write);
    MidiSystem.WriteSequence(sequence, stream);
}
```

如何实现 MIDI 2.0 功能

了解如何使用 MIDI 2.0 的基本功能。

MIDI 2.0 插件的初始化

- 从 MonoBehaviour 的 `Awake` 方法中调用 `MidiManager.Instance.InitializeMidi2` 方法。
 - 一个名为“MidiManager”的游戏对象将在层次结构视图中的“DontDestroyOnLoad”下自动创建。

△ NOTE: 如果您的 EventSystem 组件已存在于其他地方, 请从 `MidiManager.Instance.InitializeMidi` 方法中删除 `gameObject.AddComponent<EventSystem>()` 方法调用。

由于 MIDI 2.0 信号转换为 MIDI 1.0 并且反之亦然, 因此 MidiManager 类同时具有针对 MIDI1 和 MIDI2 的方法。

```
private void Awake()
{
    // 接收此游戏对象上的 MIDI 数据。
    // gameObject 必须实现 IMidi2XXXXEventHandler。
    MidiManager.Instance.RegisterEventHandleObject(gameObject);

    // 初始化 MIDI 函数
    MidiManager.Instance.InitializeMidi2(() => { });
}
```

MIDI 2.0 插件退役

- 在 MonoBehaviour 的 `OnDestroy` 方法中调用 `MidiManager.Instance.TerminateMidi2`。
 - 当您使用完 MIDI 功能并且场景即将结束时, 应调用此方法。

```
private void OnDestroy()
{
    // 禁用所有 MIDI 功能。
    MidiManager.Instance.TerminateMidi2();
}
```

MIDI 2.0 设备连接/断开信号处理

使用 GameObject 或 C# 对象接收 MIDI 2.0 连接信号

- 实现接口“IMidi2DeviceEventHandler”用于接收信号。
 - 当连接新的 MIDI 2.0 设备时, 会调用 `OnMidi2InputDeviceAttached` 和 `OnMidi2OutputDeviceAttached`。

- 当 MIDI 2.0 设备断开连接时，会调用 `OnMidi2InputDeviceAttached` 和 `OnMidi2OutputDeviceAttached`。

2. 通过调用“`MidiManager.Instance.RegisterEventHandleObject`”方法注册一个 `GameObject` 或 C# 对象来接收信号。

3. 当 MIDI 设备连接或断开连接时，将调用已实现的方法。

```
public void OnMidi2InputDeviceAttached(string deviceId)
{
    // 已连接新的 MIDI 2.0 接收设备。
}

public void OnMidi2OutputDeviceAttached(string deviceId)
{
    // 已连接新的 MIDI 2.0 发送设备。
    Debug.Log($"MIDI device attached. deviceId: {deviceId}, name: {MidiManager.Instance.GetDeviceName(deviceId)}");
}

public void OnMidi2InputDeviceDetached(string deviceId)
{
    // MIDI 2.0 接收设备已断开连接。
}

public void OnMidi2OutputDeviceDetached(string deviceId)
{
    // MIDI 2.0 发送设备已断开连接。
    Debug.Log($"MIDI 2.0 device detached. deviceId: {deviceId}, name: {MidiManager.Instance.GetDeviceName(deviceId)}");
}
```

所有代码都在 `Assets/MIDI/Samples/Scripts/Midi2SampleScene.cs` 中。

C# 使用信号委托接收 MIDI 2.0 连接信号

或者，您可以使用 C# 信号委托来接收名称类似“`MidiManager.Instance.OnMidi2XXXXEvent`”的 MIDI 信号。

```
void OnMidi2OutputDeviceAttached(string deviceId)
{
    // 已连接新的 MIDI 2.0 发送设备。
    Debug.Log($"OnMidi2OutputDeviceAttached deviceId: {deviceId}");
}

...

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

MIDI 2.0 信号接收

使用 GameObject 或 C# 对象接收 MIDI 2.0 消息

1. 实现在 `IMidi2EventHandler.cs` 源代码中编写的信号接收接口，例如 `IMidi2XXXXXEventHandler` 或 `IMidi1XXXXXEventHandler`。
 - 如果您想接收 Note On 信号，请实现 `IMidi2NoteOnEventHandler` 接口。
2. 通过调用“`MidiManager.Instance.RegisterEventHandleObject`”方法注册一个 GameObject 或 C# 对象来接收信号。
3. 当接收到 MIDI 信号时，将调用实现的方法。

```
public class Midi2SampleScene : MonoBehaviour, IMidi2AllEventsHandler,
IMidi2DeviceEventHandler
{
    private void Awake()
    {
        // 这个 Unity GameObject 将接收 MIDI 数据。
        // gameObject 必须实现 IMidiXXXXXEventHandler。
        MidiManager.Instance.RegisterEventHandleObject(gameObject);

        // 使用此 C# 对象接收 MIDI 数据。
        // 该对象的类必须实现IMidiXXXXXEventHandler。
        MidiManager.Instance.RegisterEventHandleObject(obj);

        ...
    }
}
```

接收 MIDI 信号：

```
public void OnMidi2NoteOn(string deviceId, int group, int channel, int note, int
velocity, int attributeType, int attributeData)
{
    // 接收“Note On”信号
    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)
{
    // 接收“Note Off”信号
    Debug.Log($"OnMidiNoteOff channel: {channel}, note: {note}, velocity:
{velocity}, attributeType: {attributeType}, attributeData: {attributeData}");
}
```

所有代码都在 `Assets/MIDI/Samples/Scripts/Midi2SampleScene.cs` 中。

C# 使用信号委托接收 MIDI 2.0 消息

或者，您可以使用 C# 信号委托来接收名称类似 `MidiManager.Instance.OnMidi2XXXXEvent` 或 `MidiManager.Instance.OnMidi1XXXXEvent` 的 MIDI 信号。

```
void OnMidi2NoteOn(string deviceId, int group, int channel, int note, int velocity, int attributeType, int attributeData)
{
    // 接收“Note On”信号
    Debug.Log($"OnMidi2NoteOn channel: {channel}, note: {note}, velocity: {velocity}, attributeType: {attributeType}, attributeData: {attributeData}");
}

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

MIDI 2.0 信号传输

- 在代码中的某处调用 `MidiManager.Instance.SendMidi2XXXXXX` 或 `MidiManager.Instance.SendMidi1XXXXXX` 方法。
(`SendMidi1XXXXXX` 方法是使用 MIDI 2.0 协议的 MIDI 1 兼容函数，值可以具有 7 位或 14 位精度。)

例：

```
// 通过 MIDI 2.0 发送 Note On 消息：velocity 为 16 bits
MidiManager.Instance.SendMidi2NoteOn("deviceId", 0/*groupId*/, 0/*channel*/,
60/*note*/, 65535/*velocity*/, 0/*attributeType*/, 0/*attribute*/);

// 在 MIDI 1.0 中发送 Note On 消息：velocity 为 7 bits
MidiManager.Instance.SendMidi1NoteOn("deviceId", 0/*groupId*/, 0/*channel*/,
60/*note*/, 127/*velocity*/);
```

- 可以从 `MidiManager.Instance.Midi2OutputDeviceIdSet` 属性（类型：`HashSet<string>`）中获取 `deviceId`。

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

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

所有代码都在 `Assets/MIDI/Samples/Scripts/Midi2SampleScene.cs` 中。

使用网络 MIDI 2.0 (UDP MIDI 2.0) 传输

启动网络 MIDI 2.0 (UDP MIDI 2.0) 服务器

- 要开始接受 UDP-MIDI 会话，请使用 `MIDI 端点名称`、`mDNS 服务实例名称` 和 `udp 端口号` 调用 `MidiManager.Instance.StartUdpMidi2Server` 方法。
 - 如果您想使用错误处理操作（收到 NAK 命令时调用），请使用 `onErrorAction` 参数调用 `MidiManager.Instance.StartUdpMidi2Server`。
 - 如果您想使用共享密钥（密码）的身份验证功能，请使用 `authenticationSecret` 参数调用 `MidiManager.Instance.StartUdpMidi2Server`。
 - 如果您想使用用户身份验证（帐户名/密码），请使用 `userAuthentications` 参数调用 `MidiManager.Instance.StartUdpMidi2Server`。

```
#if !UNITY_WEBGL || UNITY_EDITOR
    // 启动一个 UDP MIDI 2.0 服务器，会话名称为"MyMidi2Endpoint"，mDNS 服务实例名称
    // 为"MyMIDI2Service"，端口为 12345。
    MidiManager.Instance.StartUdpMidi2Server(12345, "MyMidi2Endpoint",
    "MyMIDI2Service");

    // 具有错误处理功能
    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}");
    });

    // 如果有密码
    MidiManager.Instance.StartUdpMidi2Server(12345, "MyMidi2Endpoint",
    "MyMIDI2Service", authenticationSecret: "MyUDP-MIDI2Secret");

    // 具有用户身份验证
    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
```

查找并连接到网络 MIDI 2.0 (UDP MIDI 2.0) 服务器

要搜索同一网络上的网络 MIDI 2.0 (UDP MIDI 2.0) 服务器，请调用 `MidiManager.Instance.StartDiscoverUdpMidi2Server()` 方法。

```
#if !UNITY_WEBGL || UNITY_EDITOR
    // 开始搜索服务器
    MidiManager.Instance.StartDiscoverUdpMidi2Server();
```

```

...
// 停止搜索服务器
MidiManager.Instance.StopDiscoverUdpMidi2Server();
#endif

```

可以通过 `MidiManager.Instance.GetDiscoveredUdpMidi2Servers()` 方法检索所有已发现的服务器。这将返回服务器的“服务实例名称”（字符串列表）。您现在可以使用此“服务实例名称”连接到服务器。

```

#if !UNITY_WEBGL || UNITY_EDITOR
// 获取发现的服务器
var discoveredServers = MidiManager.Instance.GetDiscoveredUdpMidi2Servers();

// 连接到找到的第一个客户端名称为“UDP-MIDI2-Client”的服务器。
if (discoveredServers.Count > 0)
{
    MidiManager.Instance.ConnectToUdpMidi2Server(discoveredServers[0], "UDP-MIDI2-
Client");
}
#endif

```

一旦连接建立，回调方法 `OnMidi2InputDeviceAttached`, `OnMidi2OutputDeviceAttached` 将被调用。您可以使用这些回调的 `deviceID` 参数发送或接收。

使用 UmpSequencer 函数

该库有一个用于 MIDI 2.0 功能的序列器。

- 该音序器可以读取和写入 MIDI 2.0 剪辑文件 (.midi2 文件)。
- 序列器可以将来自 MIDI 2.0 设备的 MIDI 2.0 信号录制到 `UmpSequence` 对象中。
 - 记录的 `UmpSequences` 可以导出为 MIDI 2.0 剪辑文件数据。
- 该音序器可以播放录制的 `UmpSequences` 并读取 MIDI 2.0 剪辑文件数据。

UmpSequencerの作成と利用開始

```

var isSequencerOpened = false;
// 创建并打开一个新的序列实例。
var sequencer = new UmpSequencer(() => { isSequencerOpened = true; });
sequencer.Open();

```

所有代码都在 `Assets/MIDI/Samples/Scripts/Midi2SampleScene.cs` 中。

加载并播放 MIDI 2.0 剪辑作为 `UmpSequence`。

```
// 从 FileStream 读取 MIDI 剪辑并将其设置为 UmpSequencer
using var stream = new FileStream(midiClipPath, FileMode.Open, FileAccess.Read);
sequencer.SetSequence(stream);
// 开始播放序列
sequencer.StartPlaying();

...
// 停止播放
sequencer.StopPlaying();
```

记录序列

```
// 设置新的序列进行录制
sequencer.SetSequence(new Sequence());
// 开始将 MIDI 数据录制到序列中
sequencer.StartRecording();

...
// 停止录制
sequencer.StopRecording();
```

将序列写入 MIDI 2.0 剪辑文件

要将 MIDI 2.0 剪辑文件数据写入流，请使用 [UmpSequenceWriter.WriteSequence](#) 方法。

```
// 从序列器获取 UmpSequence
var sequence = sequencer.GetSequence();

using (var stream = new FileStream("recorded.mid2", FileMode.Create,
FileAccess.Write))
{
    UmpSequenceWriter.WriteSequence(new List<UmpSequence>(){sequence}, stream);
}
```

将 MIDI 2.0 剪辑文件转换为 SMF（反之亦然）

要将 UMP 序列转换为 MIDI 序列，请使用 [SequenceConverter.ConvertUmpSequence\(Sequence\)](#) 方法。

```
// 从序列器获取 UmpSequence
var sequence = sequencer.GetSequence();

// 转换为 MIDI 序列并写入 SMF
using (var stream = new FileStream("recorded.smf", FileMode.Create,
FileAccess.Write))
```

```
{  
    // 写入 SMF  
    // 第二个参数 : fileType : 0 ( 单轨 SMF )  
    new  
StandardMidiFileWriter().Write(SequenceConverter.ConvertUmpSequence(sequence), 0,  
stream);  
}
```

要将 MIDI 序列转换为 UMP 序列，请使用“SequenceConverter.ConvertSequence(UmpSequence)”方法。

```
// 从 MIDI 1.0 序列器获取序列  
var sequence = sequencer.GetSequence();  
  
// 转换为 UMP 序列并写入 MIDI 剪辑文件  
using (var stream = new FileStream("recorded.midi2", FileMode.Create,  
FileAccess.Write))  
{  
    // 写入 MIDI 剪辑文件  
    UmpSequenceWriter.WriteSequence(SequenceConverter.ConvertSequence(sequence),  
stream);  
}
```

其他功能、实验性功能

本节介绍如何使用一些高级功能。

还包括实验性功能。

使用 RTP-MIDI

非 iOS 平台的实验功能。

使用 MIDI Polyphonic Expression (MPE) 功能

此功能目前处于实验阶段，尚未经过充分测试。

如果您发现任何错误，请[将问题添加到支持存储库](#)。

定义 MPE 区域

在使用 MPE 功能之前，您必须首先为您的 MIDI 设备定义一个“MPE 区域”。

```
// 设置 MPE 区域  
MpeManager.Instance.SetupMpeZone(deviceId, managerChannel, memberChannelCount);
```

- **managerChannel** 0 : 下区, 15 : 上区
- **memberChannelCount** 0 : 禁用该区域中的 MPE 功能。 1 至 15 : 启用 MPE。

- 如果同时配置了下限和上限区域，并且 `memberChannelCount` 之和超过 14，则第一个定义的区域将减少。

设置示例：

首先，定义下部区域。

```
// 定义一个具有 10 个成员频道的下层区域。
// 下层区域的管理者通道为 0 。
MpeManager.Instance.SetupMpeZone(deviceId, 0, 10);
```

	下区：10个会员频道		普通频道	
ch#	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15			

接下来，添加一个包含 7 个成员频道的上部区域。

```
// 添加一个包含 7 个成员频道的上部区域。
// 上层区域的管理员通道为 15 。
MpeManager.Instance.SetupMpeZone(deviceId, 15, 7);
```

下区会员频道由10个减少至7个。

	下区：7个会员频道		上区：7个会员频道	
ch#	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15			

接下来，删除下部区域。

```
// 要删除现有区域，请将成员数指定为 0。
MpeManager.Instance.SetupMpeZone(deviceId, 0, 0);
```

	普通频道		上区：7个会员频道	
ch#	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15			

发送 MPE 信号

要发送 MPE 信号，请调用 `MpeManager.Instance.SendMpeXXXXXX` 方法。

一例：

```
// Note On发送消息
// 参数 masterChannel 为 0 (下区) 或 15 (上区) 。
```

```
MpeManager.Instance.SendMpeNoteOn(deviceId, masterChannel, noteNumber, velocity);
```

接收 MPE 信号

当收到区域配置信号时，MPE 区域会自动配置。

1. 实现 `IMpeEventHandler.cs` 中定义的接收接口。实现类名称类似于 → `IMpeXXXXEventHandler`。
 - 如果您想接收 Note On 信号，请实现 `IMpeNoteOnEventHandler` 接口。
2. 调用 `MidiManager.Instance.RegisterEventHandleObject` 方法来注册 `GameObject` 以接收信号。
3. 当接收到 MPE 信号时，将调用已实现的方法。
 - 定义区域时会调用 `OnMpeZoneDefined` 方法。

```
public class MpeSampleScene : MonoBehaviour, IMpeAllEventsHandler,
IMidiDeviceEventHandler
{
    private void Awake()
    {
        // 接收此游戏对象上的 MPE 信号。
        // gameObject 必须实现 IMpeXXXXEventHandler。
        MidiManager.Instance.RegisterEventHandleObject(gameObject);
        ...
    }
}
```

接收 MPE 信号:

```
public void OnMpeZoneDefined(string deviceId, int managerChannel, int
memberChannelCount)
{
    // 收到区域定义信号。
    // managerChannel 0: 下区，15: 上区
    // memberChannelCount 0: 区域删除，1-15: 区域定义
    Debug.Log($"OnMpeZoneDefined managerChannel: {managerChannel}, "
    "memberChannelCount: {memberChannelCount}");
}

public void OnMpeNoteOn(string deviceId, int channel, int note, int velocity)
{
    // 收到 Note On 信号。
    Debug.Log($"OnMpeNoteOn channel: {channel}, note: {note}, velocity: "
    "{velocity}");
}

public void OnMpeNoteOff(string deviceId, int channel, int note, int velocity)
{
    // 收到 Note Off 信号。
    Debug.Log($"OnMpeNoteOff channel: {channel}, note: {note}, velocity: "
    "{velocity}");
}
```

Android, iOS, macOS: 使用Nearby Connections MIDI

使用 Google 的 Nearby Connections 库发送和接收 MIDI
(目前这是一个专有的实现，与类似的库不兼容。)

添加依赖包

打开 Unity 的 Package Manager 视图，按左上角的 + 按钮，然后选择 Add package from git URL... 项。指定以下 URL。

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

或者，

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

⚠ NOTE: 如果之前安装了依赖包，您可能需要将它们更新到最新版本。

Scripting Define Symbol设置

要启用 Nearby Connections MIDI 功能，请在 Player Settings 中添加 Scripting Define Symbol。

`ENABLE_NEARBY_CONNECTIONS`

Android设置

将 Unity 的 Project Settings > Player > Identification > Target API Level 设置为 API Level 33 或更高版本。

向附近的设备发布广告

调用 `MidiManager.Instance.StartNearbyAdvertising()` 方法将您的设备通告给附近的设备。

要停止广告，请调用 `MidiManager.Instance.StopNearbyAdvertising()` 方法。

```
if (isNearbyAdvertising)
{
    if (GUILayout.Button("Stop advertise Nearby MIDI devices"))
    {
        // 停止广告。
        MidiManager.Instance.StopNearbyAdvertising();
        isNearbyAdvertising = false;
    }
}
else
{
    if (GUILayout.Button("Advertise Nearby MIDI devices"))
    {
        // 开始向附近的设备投放广告。
        MidiManager.Instance.StartNearbyAdvertising();
        isNearbyAdvertising = true;
    }
}
```

找到您广告的设备

调用 `MidiManager.Instance.StartNearbyDiscovering()` 方法来发现附近连接 MIDI 设备。

要停止搜索, 请调用 `MidiManager.Instance.StopNearbyDiscovering()` 方法。

```
if (isNearbyDiscovering)
{
    if (GUILayout.Button("Stop discover Nearby MIDI devices"))
    {
        // 停止发现设备。
        MidiManager.Instance.StopNearbyDiscovering();
        isNearbyDiscovering = false;
    }
}
else
{
    if (GUILayout.Button("Discover Nearby MIDI devices"))
    {
        // 开始搜索您附近广告中的设备。
        MidiManager.Instance.StartNearbyDiscovering();
        isNearbyDiscovering = true;
    }
}
```

与Nearby的设备发送和接收 MIDI 数据

发送和接收MIDI数据的方法与普通MIDI相同。

使用 Meta Quest (Oculus Quest) 设备

目前，我使用 Oculus Quest 2 作为我的测试设备。

连接到 USB MIDI 设备

取消下面的注释以检测 USB MIDI 设备连接。

摘自 [PostProcessBuild.cs](#)：

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

        // ⚠ NOTE: 如果您想在 Meta Quest 2 (Oculus Quest 2) 中使用 USB MIDI 功能。
        // 请取消下面的注释以检测 USB MIDI 设备的连接。
        // androidManifest.AddUsbIntentFilterForOculusDevices();
    }
}
```

连接 Bluetooth MIDI 设备

[CompanionDeviceManager](#) 可用于连接 Android BLE MIDI 设备。

要启用此功能，请将 `FEATURE_ANDROID_COMPANION_DEVICE` 添加到 [Scripting Define Symbols](#) 设置中。

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

实验性 MIDI 2.0 功能

网络 MIDI 2.0 (UDP MIDI 2.0) 功能

此功能实现了 [Network MIDI 2.0 \(UDP\) Transport Specification](#)。

由于目前没有其他实现，因此无法保证相互兼容性。

如果您使用不同的实现方式并且无法通过 UDP MIDI 2.0 进行通信，请告知我们。

支持读写 MIDI Clip文件/MIDI Container文件

MIDI 剪辑文件规范当前已作为发行版本发布。

不过，MIDI Clip 文件尚未得到广泛应用，而且样本文件也只有少数，因此尚不清楚它们是否符合兼容性规范。

MIDI Container文件规范处于草案状态。

我们目前不建议使用此功能，因为规格将来可能会发生变化。

测试设备

- 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 设备:
 - Quicco mi.1 (BLE MIDI)
 - Miselu C.24 (BLE MIDI)
 - TAHORNG Elefue (BLE MIDI)
 - Roland UM-ONE (USB MIDI)
 - **⚠ NOTE:**此设备不适用于 iOS。
 - Gakken NSX-39 (USB-MIDI)
 - MacOS Audio MIDI Setup (RTP-MIDI)
- MIDI 2.0 设备:
 - 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](#)

联系

在 GitHub 上报告问题

- GitHub 支持仓库: <https://github.com/kshoji/Unity-MIDI-Plugin-supports>
 - 搜索和报告问题: <https://github.com/kshoji/Unity-MIDI-Plugin-supports/issues>

关于插件作者

- Kaoru Shoji/庄司 薫 : 0x0badc0de@gmail.com
- github: <https://github.com/kshoji>

使用的开源软件由我创建

- 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>

其他人创建的开源软件包:

- 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

其他人使用的示例 MIDI 数据

指定为 UnityWebRequest 的 URL 源。不包括 SMF 二进制文件。

Sample SMF

由于原网站不提供 [https](#) 服务，因此示例代码指定了另一个网站的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

- MIDI 剪辑测试文件取自此 GitHub 存储库: [test-midi-files](#).
 - 这些文件是根据 [MIT 许可证](#) 授权的。

版本历史

- v1.0 初始版本: 2021/08/07
- v1.1 更新版本: 2022/04/11
 - 添加 MIDI 音序器（播放/录制 MIDI 序列）功能
 - 添加 SMF 读/写功能
 - 添加 BLE MIDI 外设功能在 Android 上
 - 修复 USB MIDI 接收问题 在 Android 上
 - 修复 BLE MIDI 在 Android / iOS 上发送问题 在
 - 上修复 BLE MIDI 接收问题 (NoteOn with velocity = 0)
- v1.2.0 更新版本: 2022/08/17
 - 添加对 Android 或其他平台的实验性 RTP-MIDI 支持。
 - 在通用 Windows 平台 (UWP) 上添加 USB MIDI 支持。
 - 添加 Android 12 的新蓝牙权限支持。
 - 修复 iOS、Android 上的 MIDI 收发性能改进。
 - 修复示例场景多次附加时的 EventSystem 重复错误。
 - 修复 Android BLE MIDI 关于固定时间戳的问题。
- v1.2.1 修正版本: 2022/08/29
 - 修复排序器线程在关闭后仍然存在
 - 修复 Android ProgramChange 消息失败
 - 修复 System Exclusive 日志记录问题
 - 修复 UWP 上的 ThreadInterruptException 问题
 - 修复围绕 System Exclusive 的 SMF 读/写问题
 - 一些性能改进
- v1.3.0 更新版本: 2022/10/13
 - 添加对 Standalone OSX、Windows、Linux 的平台支持
 - 添加对 WebGL 的平台支持
 - 添加对 Unity Editor OSX、Windows、Linux 的支持
 - 将 Sequencer 实现从 Thread 更改为 Coroutine
 - 修复 iOS/OSX 设备附加/分离问题
- v1.3.1 修正版本: 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 权限请求问题
 - 添加对 Android CompanionDeviceManager 支持
- v1.3.2 修正版本: 2023/05/19
 - 修复 Android 上的编译错误
 - 修复 播放 SMF 时 MIDI 信号的顺序
- v1.3.3 修正版本: 2023/05/25
 - 修复 WebGL 上 MIDI 发送失败
- v1.3.4 修正版本: 2023/06/05

- 修复：与之前连接的设备ID相同但设备名称不同的设备时，获取到错误的设备名称。
- iOS: 将“完成”按钮添加到 BLE MIDI 搜索弹出框
- Sample scene: BLE MIDI 扫描功能仅限 Android/iOS
- MidiManager 单例模式细化
- v1.4.0 更新版本: 2023/12/07
 - 追加: 对 Android、iOS、macOS 的 Nearby Connections MIDI 支持
 - 追加: 对 WebGL 的 蓝牙 LE MIDI 支持
 - 修复: 了在 iOS 设备上连接/断开设备时错误的回调。
- v1.4.1 更新版本: 2024/02/22
 - 修复: [Linux平台下链接错误](#)
 - 追加: WebGL 平台添加了对供应商名称和设备名称的支持
 - 追加: [iOS/MacOS/Linux/Android 上的应用程序间 MIDI 连接（Virtual MIDI）支持](#)
 - 修复: 了示例场景中的内存泄漏
- v1.4.2 修正版本: 2024/02/27
 - 修复: WebGL Sample scene 初始化失败
- v1.4.3 修正版本: 2024/03/12
 - 修复: 如果蓝牙关闭，Android 插件初始化失败
 - 修复: 在 Android 14 上无法打开 USB MIDI 设备
 - 修复: Windows 插件无法更新 MIDI 设备
 - 修复: 退出 MidiManager 时 Linux 插件崩溃
 - 修复: Unity 编辑器在停止游戏后停止 MIDI 功能
 - 追加: 获取输入/输出deviceld的能力（与所有deviceld获取方法分开添加）
 - 追加: 能够指定与 MIDI 音序器的设备连接
 - 追加: 接收 MIDI 音序器播放结束信号的回调
 - 追加: 改进了 MIDI 音序器信号计时的准确性
 - 追加: 现在可以以微秒为单位指定 MIDI 音序器播放位置。
- v1.4.4 修正版本: 2024/04/09
 - 修复: Android插件加载失败时初始化中断
 - 修复: Android蓝牙MIDI无法发送MIDI消息
 - 追加: Android平台支持ProGuard minify配置
- v1.4.5 修正版本: 2024/04/13
 - 修复: Android Bluetooth MIDI 在重负载时无法发送
 - 更新: 支持使用Android新的蓝牙LE API
 - 修复: Android 上 CompanionDeviceManager 初始化的问题。从该版本开始，此功能需要 [ACCESS_FINE_LOCATION](#) 权限。
- v1.5.0 修复: 2024/06/05
 - 追加: MIDI 和弦表达功能（目前处于实验状态）
 - 完全重构内部实现
 - 更新: 改进了 SMF 加载兼容性
 - 修复: SMF 播放在接近开始信号时失败
 - 修复: Android 输入设备初始化失败
- v1.5.1 修正版本: 2024/09/05
 - 修复: Unity 编辑器不适用于 12.3 之前的旧 macOS 版本
 - 已调整为适用于 macOS 10.13 或更高版本
 - 修复: 导入示例场景时出现重复的 GUID
- v2.0.0 更新版本: 2025/05/05

- 追加: MIDI 2.0 支持 iOS、macOS 独立版、Android 和独立 Linux
- 追加: 网络 MIDI 2.0 (UDP MIDI 2.0) 功能
- 追加: 支持导入/导出 MIDI 2.0 剪辑/容器文件
 - MIDI 2.0 容器文件格式目前处于草案规范阶段，因此将来可能会发生变化。
- 追加: C# 使用信号委托接收 MIDI
- 追加: 使用 C# 对象进行 MIDI 接收
- 修复: 定义符号 ENABLE_NEARBY_CONNECTIONS 时出现编译错误
- 修复: Linux 平台上未收到某些信号 (TuneRequest、TimingClock、Start、Stop、Continue、ActiveSensing、Reset) 。
- 修复: 某些应用程序间 MIDI 设备无法在 Linux 平台上连接。
- 修复: 即使停止播放模式后，编辑器中仍然接收 MIDI 消息