MIDI 1.0 运行时 (MidiManager)

本页面介绍了围绕 `jp.kshoji.unity.midi.MidiManager` 和 `IMidiPlugin` 后端接口构建的 MIDI 1.0 运行时 API。

> 注意：本项目支持多种传输协议（原生平台 MIDI、WebGL MIDI、RTP-MIDI、Nearby Connections 等）。它们都通过 `IMidiPlugin` 和 `MidiManager` 提供一致的表面 API。

有关安装/构建要求，请参阅：

- 入门指南
- 构建后处理与脚本定义符号
- 平台与限制

核心职责

`MidiManager` 负责：

- 初始化与终止 MIDI 插件后端。
- 接收 MIDI 1.0 消息（音符、控制器、程序变更、触后、弯音、SysEx 等）。
- 通过 **Unity EventSystem** 将消息分发到实现了对应接口的 Unity 对象。
- 通过当前后端发送 MIDI 1.0 消息。

初始化 / 终止（推荐模式）

在 `Awake` 中初始化，在 `OnDestroy` 中终止：

```
using UnityEngine;
using jp.kshoji.unity.midi;

public sealed class Midi1LifecycleExample : MonoBehaviour
{
    private void Awake()
    {
        MidiManager.Instance.RegisterEventHandleObject(gameObject);
        MidiManager.Instance.InitializeMidi(() =>
        {
#if (UNITY_ANDROID || UNITY_IOS || UNITY_WEBGL) && !UNITY_EDITOR
            // 仅限支持 BLE MIDI 的平台：初始化完成后开始扫描。
            MidiManager.Instance.StartScanBluetoothMidiDevices(0);
#endif
        });
    }

    private void OnDestroy()
    {
        MidiManager.Instance.TerminateMidi();
    }
}
```

> 如果您的场景中已经存在 `EventSystem` 并且看到重复添加的警告/错误， 请移除管理器初始化代码中自动添加 `EventSystem` 的部分。

后端抽象：IMidiPlugin

`IMidiPlugin` 定义了依赖于平台/传输的具体实现：

- 生命周期：
  - `InitializeMidi(Action initializeCompletedAction)`
  - `TerminateMidi()`
  - (编辑器) `PlayModeStateChanged(...)`
- 可选的 BLE 扫描/广播（由平台编译符号控制）。
- 设备元数据：
  - `GetDeviceName(deviceId)`
  - `GetVendorId(deviceId)`
  - `GetProductId(deviceId)`
- MIDI 1.0 发送方法：
  - 音符开启/关闭 (Note On/Off), 控制器 (CC), 程序变更 (PC), 触后 (Aftertouch), 弯音 (Pitch Bend)
  - SysEx 以及系统实时/常用消息（可用性因平台而异）

具体实现位于：

- `MidiPlugin.Android.cs`, `MidiPlugin.Apple.cs`, `MidiPlugin.Linux.cs`, `MidiPlugin.Windows.cs`, `MidiPlugin.WebGL.cs`
- `MidiPlugin.RtpMidi.cs` (RTP-MIDI 传输)
- `MidiPlugin.Nearby.cs` (Nearby 传输)

设备元数据：名称 / VendorId / ProductId

- `MidiManager.Instance.GetDeviceName(deviceId)`
- `MidiManager.Instance.GetVendorId(deviceId)`
- `MidiManager.Instance.GetProductId(deviceId)`

重要提示：

- 部分平台/传输不支持 Vendor/Product ID，此时可能会返回空字符串。
- 设备断开连接后，这些方法可能会返回空字符串。

GetVendorId / GetProductId 的可用性

平台	蓝牙 MIDI	USB MIDI	网络 MIDI (RTP-MIDI)	Nearby Connections MIDI
iOS	○	○	-	-
Android	○	○	-	-
UWP (通用 Windows 平台)	-	○	-	-
独立 macOS / Unity 编辑器 macOS	○	○	-	-
独立 Linux / Unity 编辑器 Linux	-	-	-	-
独立 Windows / Unity 编辑器 Windows	-	○	-	-
WebGL	-	△ (仅限 GetVendorId)	-	-

VendorId 示例

由于底层平台 API 的差异，在不同操作系统上观察到的 `VendorId` 字符串可能不同。

平台	蓝牙 MIDI	USB MIDI	网络 MIDI (RTP-MIDI)	Nearby Connections MIDI
iOS	QUICCO SOUND Corp.	Generic	-	-
Android	QUICCO SOUND Corp.	1410	-	-
UWP	-	VID_0582	-	-
独立 macOS / Unity 编辑器 macOS	QUICCO SOUND Corp.	Generic	-	-

平台	蓝牙 MIDI	USB MIDI	网络 MIDI (RTP-MIDI)	Nearby Connections MIDI
独立 Linux / Unity 编辑器 Linux	-	-	-	-
独立 Windows / Unity 编辑器 Windows	-	1	-	-
WebGL	QUICCO SOUND Corp.	Microsoft Corporation	-	-

ProductId 示例

`ProductId` 情况类似。

平台	蓝牙 MIDI	USB MIDI	网络 MIDI (RTP-MIDI)	Nearby Connections MIDI
iOS	mi.1	USB2.0-MIDI	-	-
Android	mi.1	298	-	-
UWP	-	PID_012A	-	-
独立 macOS / Unity 编辑器 macOS	mi.1	USB2.0-MIDI	-	-
独立 Linux / Unity 编辑器 Linux	-	-	-	-
独立 Windows / Unity 编辑器 Windows	-	102	-	-
WebGL	mi.1	UM-ONE	-	-

事件分发模型 (Unity EventSystem)

处理器接口

MIDI 1.0 事件处理器接口定义在：

- `Assets/MIDI/Scripts/IMidiEventHandler.cs`
- `Assets/MIDI/Scripts/IMidiDeviceEventHandler.cs`

这些是细分化的接口，例如：

- `IMidiNoteOnEventHandler` (音符开启)
- `IMidiControlChangeEventHandler` (控制器变更)
- `IMidiSystemExclusiveEventHandler` (系统排他消息)
- ……以及用于时钟、同步、节拍选择等系统消息的处理器。

此外还提供聚合接口以简化实现：

- `IMidiPlayingEventsHandler` (包含音符与通道消息)
- `IMidiSystemEventsHandler` (包含系统消息与 SysEx)
- `IMidiAllEventsHandler` (包含所有 MIDI 1.0 消息)
- ……以及用于 Start/Stop/Clock、Song Select 等的系统消息处理器。

此外还有聚合接口：

- `IMidiPlayingEventsHandler` (音符 + 通道事件)
- `IMidiSystemEventsHandler` (系统与 SysEx 事件)
- `IMidiAllEventsHandler` (所有事件)

注册处理器

`MidiManager` 提供了注册/注销事件处理器对象的方法。预期用法如下：

- 在 Unity 组件/对象上实现一个或多个 `IMidi*EventHandler` 接口。
- 将该对象注册到 `MidiManager`。
- 当有传入的 MIDI 消息时，`MidiManager` 会执行所有匹配处理器接口的回调。

通过 C# 事件委托接收事件（可选）

作为接口的替代方案，您可以订阅名为以下格式的 C# 事件：

- `MidiManager.Instance.OnMidiXXXXEvent`

示例：

```
using UnityEngine;
using jp.kshoji.unity.midi;

public sealed class Midi1DelegateExample : MonoBehaviour
{
    private void OnEnable()
    {
        MidiManager.Instance.OnMidiNoteOnEvent += OnMidiNoteOn;
    }

    private void OnDisable()
    {
        MidiManager.Instance.OnMidiNoteOnEvent -= OnMidiNoteOn;
    }

    private void OnMidiNoteOn(string deviceId, int group, int channel, int note,
int velocity)
        => Debug.Log($"NoteOn 设备:{deviceId} 通道:{channel} 音符:{note} 力度:
{velocity}");
}
```

蓝牙 MIDI 从机模式 (仅限 Android)

除了扫描/连接 BLE MIDI 设备外，Android 还可以将您的应用广播为 **BLE MIDI 从机 (Peripheral)**。

API 表面：

- 开始广播：`MidiManager.Instance.StartAdvertisingBluetoothMidiDevice()`
- 停止广播：`MidiManager.Instance.StopAdvertisingBluetoothMidiDevice()`

注意：

- 此功能仅**限 Android**。
- 广播与扫描是独立的；一旦对端连接成功，您仍可以使用正常的 MIDI 发送/接收功能。
- 如果您还在使用高级 Android BLE 配对工作流，请参阅： 构建后处理与脚本定义符号 中的 `FEATURE_ANDROID_COMPANION_DEVICE`。

发送 MIDI 1.0 消息

`MidiManager` 为常用消息提供了封装方法：

- 音符开启/关闭: `SendMidiNoteOn`, `SendMidiNoteOff`
- 控制器变更 (CC): `SendMidiControlChange`
- 程序变更 (PC): `SendMidiProgramChange`
- 触后 (Aftertouch): `SendMidiPolyphonicAftertouch`, `SendMidiChannelAftertouch`
- 弯音 (Pitch Wheel): `SendMidiPitchWheel`
- 系统排他消息 (SysEx): `SendMidiSystemExclusive`
- 系统消息：时钟、启动/继续/停止、重置等。

参数范围

- `group`: 0–15 (MIDI 1.0 驱动通常会忽略此参数，但在 MIDI 2.0 传输中有效)
- `channel`: 0–15
- `note`: 0–127
- `velocity`: 0–127
- `pressure`: 0–127
- `amount` (弯音): 0–16383 (8192 为中心值)
- `SysEx`: 字节数组，通常以 `0xF0` 开始并以 `0xF7` 结束。

RTP-MIDI 辅助方法（如果已启用）

使用 RTP-MIDI 插件时，`MidiManager` 会暴露以下方法：

- 按端口启动/停止 RTP-MIDI 服务器
- 连接/断开终端 (Endpoints)
- 查询监听器是否正在运行

另请参阅：

- 传输协议与平台说明

源代码示例实现 (MIDI 1.0 快速入门)

创建一个新脚本（或使用提供的脚本）：

- `Assets/MIDI/Samples/Scripts/DocumentationExamples/Midi1QuickStartExample.cs`

将其附加到 GameObject 上，进入播放模式，然后连接 MIDI 设备。
它将执行以下操作：

- 记录设备连接/断开日志
- 记录传入的 Note On 消息
- （可选）向第一个输出设备发送测试音符