# MIDI 2.0 / UMP 运行时 (Midi2Manager)

本页面介绍了由 `jp.kshoji.unity.midi.Midi2Manager` 提供的 MIDI 2.0 运行时、通用 MIDI 数据包 (UMP) 解析以及 UDP MIDI 2.0 功能。

有关各平台的可用性和操作系统限制，请参阅：

- 平台与限制

## 核心职责

`Midi2Manager` 负责：

- 初始化/终止 MIDI 2.0 插件后端（取决于平台）。
- 接收 **原始 UMP 字组**（`uint[]`）并：
  - （可选）转发“原始 UMP”回调，
  - 将消息类型解码为强类型事件，
  - 在适用时重新组装多部分消息（例如 SysEx8/文本）。
- 通过活动后端发送 UMP 消息。
- 托管并连接到 **UDP MIDI 2.0** 终端（服务端/客户端）。
- 发现 UDP MIDI 2.0 服务器（局域网发现）。

## 初始化 / 终止（推荐模式）

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

public sealed class Midi2LifecycleExample : MonoBehaviour
{
    private void Awake()
    {
        MidiManager.Instance.RegisterEventHandleObject(gameObject);

        // MIDI 2.0 初始化
        MidiManager.Instance.InitializeMidi2(() => { });
    }

    private void OnDestroy()
    {
        MidiManager.Instance.TerminateMidi2();
    }
}
```

> 本插件通过同一个管理器实例路由 MIDI 2.0 和 MIDI 1.0 兼容事件， 因此您会在 API 中看到“MIDI 2”和“MIDI 1 兼容”两类处理器家族。

## 后端抽象

MIDI 2.0 插件实现派生自：

- `Midi2PluginBase`

具体实现包括：

- `Midi2Plugin.Android.cs`
- `Midi2Plugin.Apple.cs`
- `Midi2Plugin.Linux.cs`
- `Midi2Plugin.Udp.cs` (网络传输)

# UMP 事件处理器接口

UMP/MIDI 2.0 事件处理器接口定义在：

- `Assets/MIDI/Scripts/IMidi2EventHandler.cs`
- `Assets/MIDI/Scripts/IMidi2DeviceEventHandler.cs`

接口按 UMP 消息类型分组：

- **实用工具 (类型 0)**
- **系统常用与实时 (类型 1)**
- **MIDI 1.0 通道语音 (类型 2)** (兼容性消息)
- **64 位数据 / SysEx (类型 3)**
- **MIDI 2.0 通道语音 (类型 4)**
- **128 位数据 / SysEx8 / 混合数据集 (类型 5)**
- **Flex Data (类型 D)**
- **UMP 流 (类型 F)**

聚合接口包括：

- `IMidi2AllEventsHandler`
- `IUmpAllEventsHandler` (涵盖 UMP 中解码出的 MIDI 1.0 和 MIDI 2.0 事件)

## 通过 C# 事件委托接收事件（可选）

除了实现处理器接口外，您还可以订阅管理器事件（基于委托的回调）。

它们的命名格式如下：

- `MidiManager.Instance.OnMidi2XXXXEvent`
- 以及（针对 UMP 编码的 MIDI 1 兼容消息）`MidiManager.Instance.OnMidi1XXXXEvent`

示例：

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

public sealed class Midi2DelegateExample : MonoBehaviour
{
    private void OnEnable()
    {
        MidiManager.Instance.OnMidi2OutputDeviceAttachedEvent +=
OnMidi2OutputDeviceAttached;
        MidiManager.Instance.OnMidi2NoteOnEvent += OnMidi2NoteOn;
    }

    private void OnDisable()
    {
        MidiManager.Instance.OnMidi2OutputDeviceAttachedEvent -=
OnMidi2OutputDeviceAttached;
        MidiManager.Instance.OnMidi2NoteOnEvent -= OnMidi2NoteOn;
    }

    private void OnMidi2OutputDeviceAttached(string deviceId)
        => Debug.Log($"MIDI 2.0 输出已连接: {deviceId}");

    private void OnMidi2NoteOn(string deviceId, int group, int channel, int note,
int velocity, int attributeType, int attributeData)
        => Debug.Log($"MIDI2 NoteOn 设备:{deviceId} 组:{group} 通道:{channel} 音符:
{note} 力度:{velocity}");
}
```

## 发送 MIDI 2.0 vs MIDI 1 兼容消息

您可以选择发送：

- MIDI 2.0 消息（更高精度，例如 16 位力度）
- 编码在 UMP 中的 MIDI 1.0 兼容消息（7/14 位风格）

示例：

```
// MIDI 2.0 Note On (16 位力度)
MidiManager.Instance.SendMidi2NoteOn(
    "deviceId",
    0 /*group*/,
    0 /*channel*/,
    60 /*note*/,
    65535 /*velocity*/,
    0 /*attributeType*/,
    0 /*attributeData*/
);

// 编码为 UMP 的 MIDI 1.0 兼容 Note On (7 位力度)
MidiManager.Instance.SendMidi1NoteOn(
    "deviceId",
    0 /*group*/,
    0 /*channel*/,
    60 /*note*/,
    127 /*velocity*/
);
```

# UDP MIDI 2.0 功能

UDP MIDI 2.0 传输（网络 MIDI 2.0）由 `Midi2Plugin.Udp.cs` 实现，并经由 `MidiManager` / `Midi2Manager` 暴露。

## 启动 UDP MIDI 2.0 服务器

您可以启动一个具有以下参数的服务器：

- 终端名称 (Endpoint name)
- mDNS 服务实例名称
- UDP 端口
- 可选的 NAK/错误回调
- 可选的身份验证（共享密钥或用户认证列表）

示例（包含密钥/密码占位符）：

```
#if !UNITY_WEBGL || UNITY_EDITOR
// 启动 UDP MIDI 2.0 服务器
MidiManager.Instance.StartUdpMidi2Server(
    12345,
    "MyMidi2Endpoint",
    "MyMIDI2Service"
);

// 带有 NAK 处理的启动
MidiManager.Instance.StartUdpMidi2Server(
    12345,
    "MyMidi2Endpoint",
    "MyMIDI2Service",
    onErrorAction: nak =>
    {
        Debug.LogError($"UDP MIDI 错误。状态:{nak.NakStatus}, 命令:
{nak.CommandHeader:x8}, 消息:{nak.Message}");
    }
);

// 使用共享密钥认证（占位符）
MidiManager.Instance.StartUdpMidi2Server(
    12345,
    "MyMidi2Endpoint",
    "MyMIDI2Service",
    authenticationSecret: "<shared-secret>"
);

// 使用用户认证（占位符）
MidiManager.Instance.StartUdpMidi2Server(
    12345,
    "MyMidi2Endpoint",
    "MyMIDI2Service",
    userAuthentications: new
```

```
System.Collections.Generic.List<System.Collections.Generic.KeyValuePair<string,
string>>
    {
        new("<account-1>", "<password-1>"),
        new("<account-2>", "<password-2>"),
    }
);
#endif
```

发现并连接

```
#if !UNITY_WEBGL || UNITY_EDITOR
// 开始发现
MidiManager.Instance.StartDiscoverUdpMidi2Server();

// 读取已发现的服务（服务实例名称）
var discovered = MidiManager.Instance.GetDiscoveredUdpMidi2Servers();

// 连接到发现的第一个服务器
if (discovered.Count > 0)
{
    MidiManager.Instance.ConnectToUdpMidi2Server(discovered[0], "UDP-MIDI2-
Client");
}
#endif
```

限制：

- UDP MIDI 2.0 仍被视为实验性功能；与其他协议栈的互操作性可能有所不同。

# UmpSequencer (MIDI 2.0 剪辑序列化)

该项目包含一套 MIDI 2.0 序列化工作流（独立于 SMF 序列化）：

- 读写 MIDI 2.0 剪辑文件 (`.midi2`)
- 可以将 UMP 事件录制到序列模型中
- 可以播放录制的剪辑/序列
- 可以在 SMF (MIDI 1.0 `Sequence`) 和 UMP 剪辑序列之间进行转换

另请参阅：

- SMF / 序列化（关于 SMF 与转换的说明）

## 剪辑/容器格式说明

- 虽然支持 MIDI 2.0 剪辑文件 (Clip file)，但该文件格式尚未广泛普及。
- 虽然支持 MIDI 2.0 容器文件 (Container file)，但容器规范目前仍处于草案阶段，可能会发生变化。
    - 请将容器工作流视为实验性功能。

# 源代码示例实现 (MIDI 2.0 快速入门)

请参考：

- `Assets/MIDI/Samples/Scripts/DocumentationExamples/Midi2QuickStartExample.cs`

将其附加到 GameObject 上并进入播放模式。
它将执行以下操作：

- 初始化 MIDI 2.0 (`InitializeMidi2`)
- 记录 MIDI 2.0 设备连接/断开日志
- 记录 MIDI 2.0 Note On 事件