MIDI 2.0 / UMP ランタイム (Midi2Manager)

このページでは、`jp.kshoji.unity.midi.Midi2Manager` によって提供される MIDI 2.0 ランタイム、Universal MIDI Packet (UMP) のパース、および UDP MIDI 2.0 機能について説明します。

プラットフォームの対応状況や OS の制約については、以下を参照してください:

- プラットフォームと制限事項

主な役割

`Midi2Manager` は以下の役割を担います:

- MIDI 2.0 プラグインバックエンドの初期化と終了処理（プラットフォームに依存）。
- **生の UMP ワード** (`uint[]`) の受信と処理:
  - オプションで「生の UMP」コールバックを転送。
  - メッセージタイプを型付けされたイベントにデコード。
  - 必要に応じてマルチパートメッセージ（SysEx8/テキストなど）を再構築。
- アクティブなバックエンドを通じた UMP メッセージの送信。
- **UDP MIDI 2.0** エンドポイント（サーバー/クライアント）のホストと接続。
- UDP MIDI 2.0 サーバーの検索（LAN ディスカバリ）。

初期化と終了 (推奨パターン)

`Awake` で初期化し、`OnDestroy` で終了します:

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

public sealed class Midi2LifecycleExample : MonoBehaviour
{
    private void Awake()
    {
        // イベントを処理するオブジェクトを登録
        MidiManager.Instance.RegisterEventHandleObject(gameObject);

        // MIDI 2.0 初期化
        MidiManager.Instance.InitializeMidi2(() => { });
    }

    private void OnDestroy()
    {
        // MIDI 2.0 終了
        MidiManager.Instance.TerminateMidi2();
    }
}
```

> このプラグインは、MIDI 2.0 と MIDI 1.0 互換イベントを同じマネージャーインスタンス経由でルーティングするため、API には「MIDI 2.0」と「MIDI 1.0 互換」の両方のハンドラ群が表示されます。

バックエンドの抽象化

MIDI 2.0 プラグインの実装は以下を継承しています:

- `Midi2PluginBase`

実装例:

- `Midi2Plugin.Android.cs`
- `Midi2Plugin.Apple.cs`
- `Midi2Plugin.Linux.cs`
- `Midi2Plugin.Udp.cs` (ネットワークトランスポート)

UMP イベントハンドラインターフェース

UMP/MIDI 2.0 イベントハンドラインターフェースは以下で定義されています:

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

インターフェースは UMP メッセージタイプごとにグループ化されています:

- **Utility (タイプ 0)**
- **System Common & Realtime (タイプ 1)**
- **MIDI 1.0 Channel Voice (タイプ 2)** (互換用)
- **Data 64-bit / SysEx (タイプ 3)**
- **MIDI 2.0 Channel Voice (タイプ 4)**
- **Data 128-bit / SysEx8 / Mixed Data Set (タイプ 5)**
- **Flex Data (タイプ D)**
- **UMP Stream (タイプ F)**

統合インターフェースには以下が含まれます:

- `IMidi2AllEventsHandler`
- `IUmpAllEventsHandler` (UMP からデコードされた MIDI 1.0 + MIDI 2.0 イベントを統合)

C# デリゲートによるイベント受信 (オプション)

ハンドラインターフェースを実装する代わりに、マネージャーのイベント（デリゲートベースのコールバック）を購読することもできます。

これらは以下のような名前になっています:

- `MidiManager.Instance.OnMidi2XXXXEvent`
- および（UMP エンコードされた MIDI 1 互換用） `MidiManager.Instance.OnMidi1XXXXEvent`

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 出力接続: {deviceId}");

    private void OnMidi2NoteOn(string deviceId, int group, int channel, int note,
int velocity, int attributeType, int attributeData)
        => Debug.Log($"MIDI2 NoteOn デバイス:{deviceId} g:{group} ch:{channel} note:
{note} vel:{velocity}");
}
```

MIDI 2.0 メッセージと MIDI 1 互換メッセージの送信

以下のいずれかを送信できます:

- MIDI 2.0 メッセージ (高精度、例: 16ビットベロシティ)
- UMP 内にエンコードされた MIDI 1.0 互換メッセージ (7/14ビット形式)

例:

```
// MIDI 2.0 Note On (16ビットベロシティ)
MidiManager.Instance.SendMidi2NoteOn(
    "deviceId",
    0 /*group*/,
    0 /*channel*/,
    60 /*note*/,
    65535 /*velocity*/,
    0 /*attributeType*/,
    0 /*attributeData*/
);

// UMP としてエンコードされた MIDI 1.0 互換 Note On (7ビットベロシティ)
MidiManager.Instance.SendMidi1NoteOn(
    "deviceId",
    0 /*group*/,
    0 /*channel*/,
    60 /*note*/,
    127 /*velocity*/
);
```

UDP MIDI 2.0 機能

UDP MIDI 2.0 トランスポート（ネットワーク MIDI 2.0）は `Midi2Plugin.Udp.cs` で実装され、`MidiManager` / `Midi2Manager` を介して利用できます。

UDP MIDI 2.0 サーバーの起動

サーバーは以下のパラメータで起動できます:

- エンドポイント名
- mDNS サービスインスタンス名
- UDP ポート
- オプションの NAK/エラーコールバック
- オプションの認証（共有シークレットまたはユーザー認証リスト）

例（シークレットやパスワードはプレースホルダです）:

```
#if !UNITY_WEBGL || UNITY_EDITOR
// UDP MIDI 2.0 サーバーの起動
MidiManager.Instance.StartUdpMidi2Server(
    12345,
    "MyMidi2Endpoint",
    "MyMIDI2Service"
);

// NAK ハンドリング付き
MidiManager.Instance.StartUdpMidi2Server(
    12345,
    "MyMidi2Endpoint",
    "MyMIDI2Service",
    onErrorAction: nak =>
    {
        Debug.LogError($"UDP MIDI エラー。ステータス:{nak.NakStatus}, コマンド:
{nak.CommandHeader:x8}, メッセージ:{nak.Message}");
    }
);

// 共有シークレット認証付き（プレースホルダ）
MidiManager.Instance.StartUdpMidi2Server(
    12345,
    "MyMidi2Endpoint",
    "MyMIDI2Service",
    authenticationSecret: "<shared-secret>"
);

// ユーザー認証付き（プレースホルダ）
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
```

検索と接続

```
#if !UNITY_WEBGL || UNITY_EDITOR
// 検索開始
MidiManager.Instance.StartDiscoverUdpMidi2Server();

// 見つかったサービス（サービスインスタンス名）を取得
var discovered = MidiManager.Instance.GetDiscoveredUdpMidi2Servers();

// 最初に見つかったサーバーに接続
if (discovered.Count > 0)
{
    MidiManager.Instance.ConnectToUdpMidi2Server(discovered[0], "UDP-MIDI2-
Client");
}
#endif
```

制限事項:

- UDP MIDI 2.0 はまだ試験的な機能です。他の実装との相互運用性は異なる場合があります。

UmpSequencer (MIDI 2.0 クリップ・シーケンシング)

このプロジェクトには、MIDI 2.0 シーケンサーワークフローが含まれています（SMF シーケンシングとは別個のものです）：

- MIDI 2.0 クリップファイル (`.midi2`) の読み書き。
- UMP イベントをシーケンスモデルに記録。
- 記録されたクリップ/シーケンスの再生。
- SMF (MIDI 1.0 `Sequence`) と UMP クリップシーケンスの相互変換。

参照:

- SMF / シーケンシング (SMF + 変換に関する注意点)

クリップ/コンテナ形式に関する注意

- MIDI 2.0 クリップファイルのサポートはありますが、クリップファイル自体はまだ普及していません。
- MIDI 2.0 コンテナファイルのサポートはありますが、コンテナ仕様はドラフト段階であり、変更される可能性があります。
    - コンテナワークフローは試験的なものと考えてください。

ソースコード実装例 (MIDI 2.0 クイックスタート)

以下を使用してください:

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

これを GameObject にアタッチして Play モードに入ります。 以下の動作が行われます:

- MIDI 2.0 の初期化 (`InitializeMidi2`)
- MIDI 2.0 デバイスの接続/切断をログ出力
- MIDI 2.0 Note On イベントをログ出力