

# SMF / シーケンシング (jp.kshoji.midisystem)

---

このページでは、以下のディレクトリにある SMF (標準 MIDI ファイル) およびシーケンシング関連のコードについて説明します:

- `Assets/MIDI/Scripts/midisystem/`

名前空間: `jp.kshoji.midisystem`

## このモジュールが提供するもの

- メモリ上の MIDI シーケンスモデル:
  - `Sequence`
  - `Track`
  - `MidiEvent`
- MIDI メッセージの表現:
  - `MidiMessage` (基底)
  - `ShortMessage` (ほとんどのチャンネル/システムメッセージ)
  - `SysexMessage`
  - `MetaMessage`
- SMF ファイルの入出力:
  - `StandardMidiFileReader`
  - `StandardMidiFileWriter`
- シーケンシング用のインターフェースと実装:
  - `ISequencer`, `SequencerImpl`
  - `IReceiver`, `ITransmitter`
  - メタ/コントローラーイベントリスナー

これは Java MIDI API モデル (`Sequence/Track/MidiEvent`) と概念的に似ています。

## StandardMidiFileWriter の動作 (重要な詳細)

**Sequence** を書き出す際:

- トラック数に応じて SMF タイプ 0 およびタイプ 1 をサポートします。
- 以下の情報を含む **MThd** ヘッダーを書き込みます:
  - ファイルタイプ
  - トラック数
  - 分解能 (division type/resolution) のエンコード
- 各トラックについて:
  - 書き込み前にトラック長を計算します。
  - 書き込み時に **システム・リアルタイム** メッセージ (**ShortMessage** ステータス  $\geq$  **0xF8**) をスキップします。
  - 以下を書き込みます:
    - デルタタイム (可変長整数)
    - メッセージバイト (ShortMessage)、または
    - **0xFF** + メタタイプ + 長さ + データ (MetaMessage)、または
    - **0xF0** + 長さ + データ (SysexMessage)
  - **End of Track** メタイベントが確実に存在するようにし、存在しない場合は追加します。

## シーケンサーのワークフロー (再生 / 録音 / 書き出し)

ここでは、同梱されているマニュアルのシーケンサー使用パターンを紹介します。

### シーケンサーの作成とオープン

```
using jp.kshoji.midisystem;

var isSequencerOpened = false;

// シーケンサーを作成してオープンする
var sequencer = new SequencerImpl(() => { isSequencerOpened = true; });
sequencer.Open();
```

```
### SMF を Sequence に読み込んで再生する
```csharp
using System.IO;
using jp.kshoji.midisystem;

// デバイス接続を更新
sequencer.UpdateDeviceConnections();

using var stream = new FileStream(smfPath, FileMode.Open, FileAccess.Read);
sequencer.SetSequence(stream);

sequencer.Start();

// ... 後で停止
sequencer.Stop();
```

### シーケンスを録音する

```
using jp.kshoji.midisystem;

sequencer.UpdateDeviceConnections();

// 分解能 (PPQ) の設定例
sequencer.SetSequence(new Sequence(Sequence.Ppq, 480));

sequencer.StartRecording();

// ... 後で停止
sequencer.Stop();
```

## 録音したシーケンスを SMF ファイルに書き出す

```
using System.IO;
using jp.kshoji.midisystem;

var sequence = sequencer.GetSequence();

if (sequence.GetTickLength() > 0)
{
    using var stream = new FileStream(recordedSmfPath, FileMode.Create,
    FileAccess.Write);
    MidiSystem.WriteSequence(sequence, stream);
}
```

## UMP ツールとの関係

`Assets/MIDI/Scripts/UmpSequencer/` にある UMP ツール群は、以下の相互変換を提供します:

- SMF `Sequence` (MIDI 1.0 表現)
- UMP `UmpSequence` (MIDI 2.0 パケット指向の表現)

以下のような場合に使用します:

- MIDI ファイル (SMF) のインポート/エクスポート。
- MIDI 2.0 ワークフロー用の UMP クリップへの変換。
- コンバーターがサポートしている範囲での、MIDI 2.0 クリップファイルと SMF の相互変換。

## 実用的なガイドンス

- 単純に「MIDI ファイルを読み込んでメッセージを送信する」場合は、通常以下の手順を踏みます:
  1. SMF を `Sequence` に読み込む。
  2. ティックごとにイベントを反復処理する（またはシーケンサーを使用する）。
  3. `MidiManager` (MIDI 1.0) を介して送信、または UMP に変換して `Midi2Manager` (MIDI 2.0) を介して送信。
- 精密なタイミングやトランスポート制御（再生/停止/ループ）が必要な場合は、以下を検討してください:
  - `SequencerImpl` (SMF 形式のシーケンシング)。
  - UMP 中心の開発であれば、MIDI 2.0 UMP クリップシーケンサーのワークフロー（[MIDI 2.0 / UMP](#) を参照）。

## ソースコード実装例 (SMF → MIDI 出力)

以下を使用してください:

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

テスト方法:

1. `Assets/StreamingAssets/` ディレクトリに `.mid` ファイルを配置します（ディレクトリがない場合は作成してください）。
2. `streamingAssetsMidiFileName` を設定します（例: `test.mid`）。
3. コンポーネントを `GameObject` にアタッチして Play モードに入ります。

この例では `StandardMidiFileReader` を使用してファイルを読み込み、イベントを反復処理して `MidiManager` 経由で基本的なメッセージを送信します。