

# Unity PSG Player

Unity上でレトロゲーム機などのPSG（Programmable Sound Generator）音源を鳴らすライブラリです。演奏データはMML（Music Macro Language）のテキストで、楽譜を文字で記述するため、手軽に音楽データを作成できます。

ファミコンの音源に似た（DPCMを除く）表現ができるように設計しています。

- 音色は、矩形波4種類、三角波、ノイズ2種類が鳴らせます。三角波は4bit波形です。
- 演奏表現として、スワイプ、LFO（ビブラート）、音量エンベロープが使えます。

ファミコン音源の完全再現は目標にしていません。（めんどくさいので。）

あくまでも、別途サウンドクリップなどを用意しないで、Unityだけで音を鳴らす目的で制作しています。

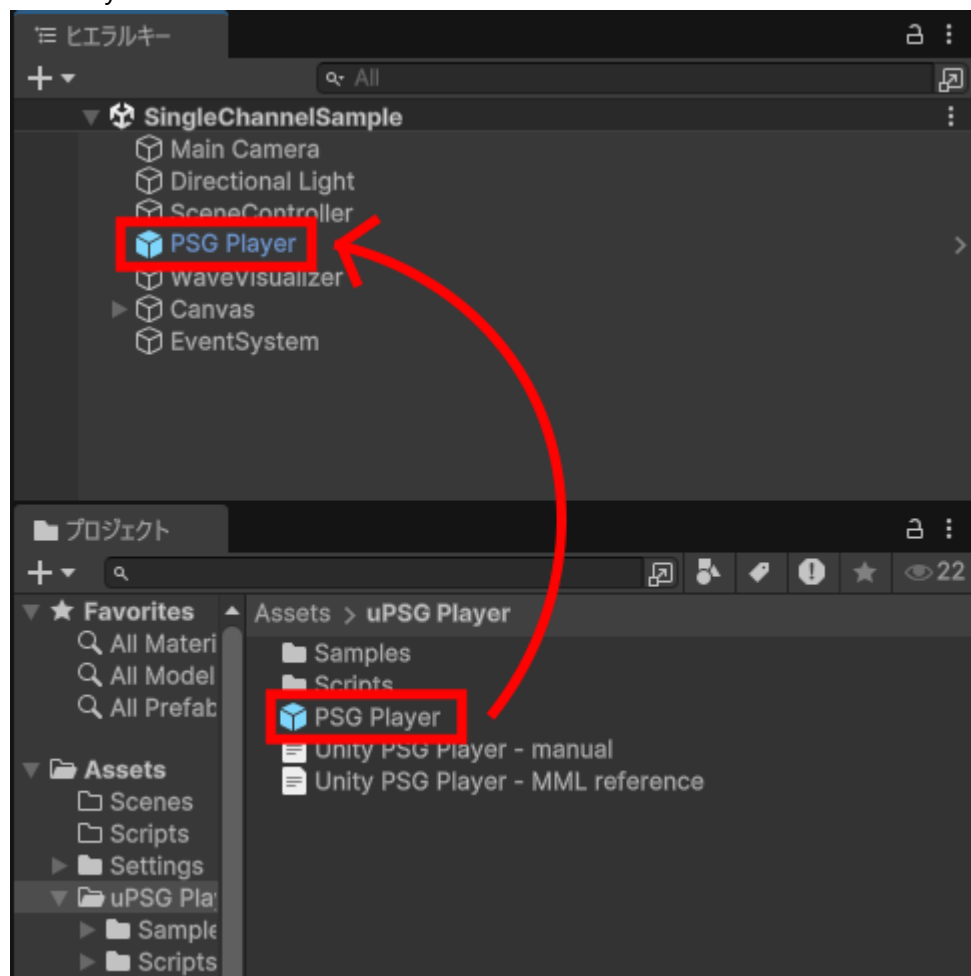
PSG Playerはモノフォニック（単音）で発音します。

ファミコンの4音（DPCMを除く）などを再現するには、PSG Playerを複数設置して同時に再生することで対応します。

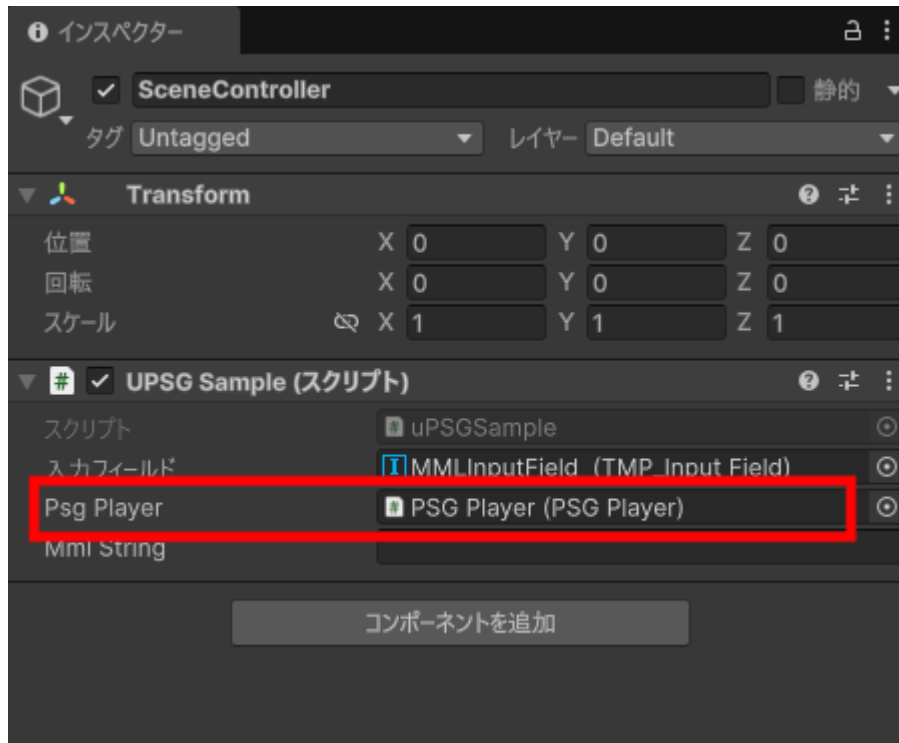
## クイックガイド

### 基本的な使い方

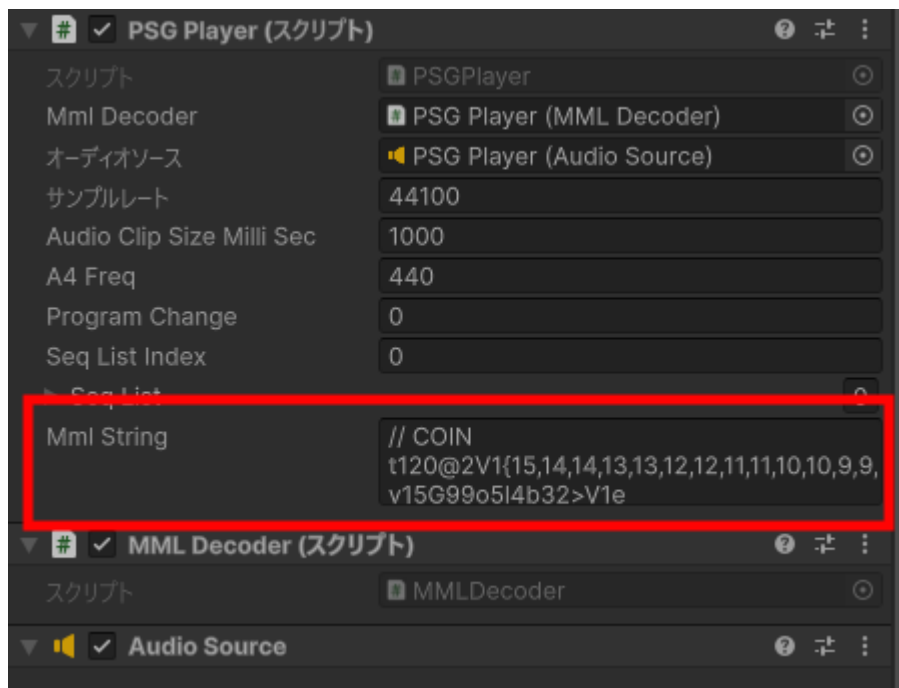
1. PSG Playerプレハブをヒエラルキーに置きます。



2. 操作するスクリプトでPSGPlayerクラス変数を用意し、設置したPSG Playerオブジェクトをアタッチします。

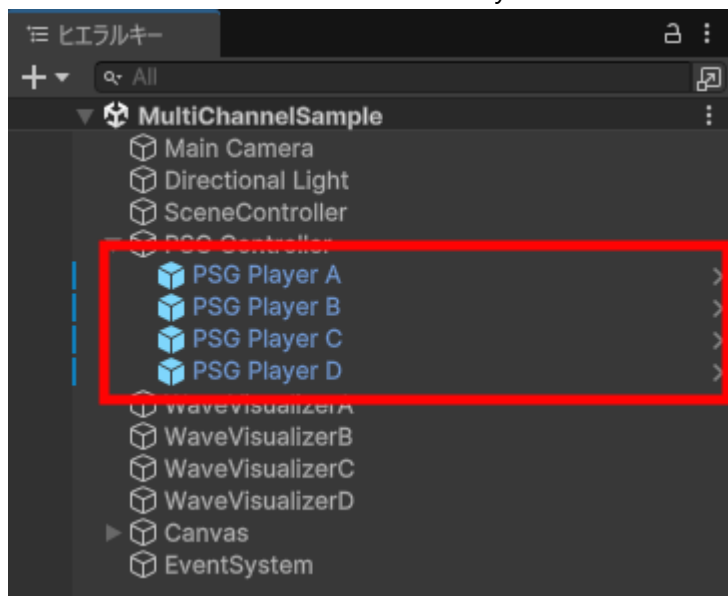


3. PSGPlayerのmmlString変数に記述したMMLが、`Play()`で再生されます。  
MMLについては「[MMLリファレンス](#)」を参照してください。

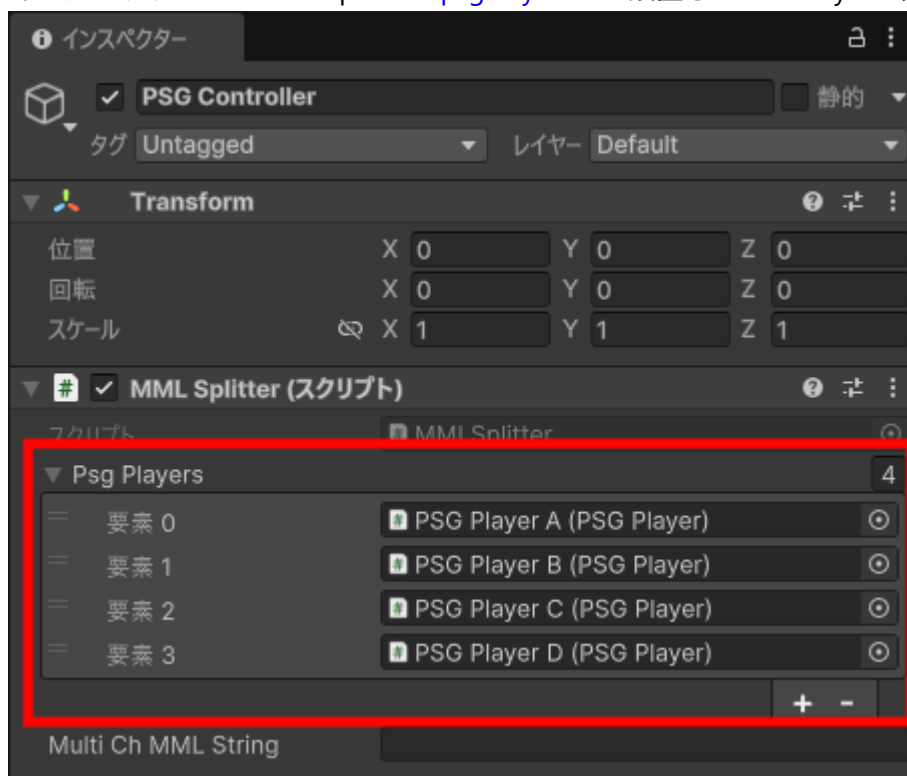


## マルチチャンネルの使い方

1. 必要なチャンネル数に合わせてPSG Playerプレハブを置きます。



2. 適当なゲームオブジェクトにMMLSplitterスクリプトをアタッチします。
3. インスペクターからMMLSplitterのpsgPlayersに、設置したPSG Playerを割り当てます。



4. MMLSplitterのmultiChMMLString変数にMMLを入れて、SplitMML()で各チャンネルにMMLを分配し、PlayAllChannels()で再生されます。

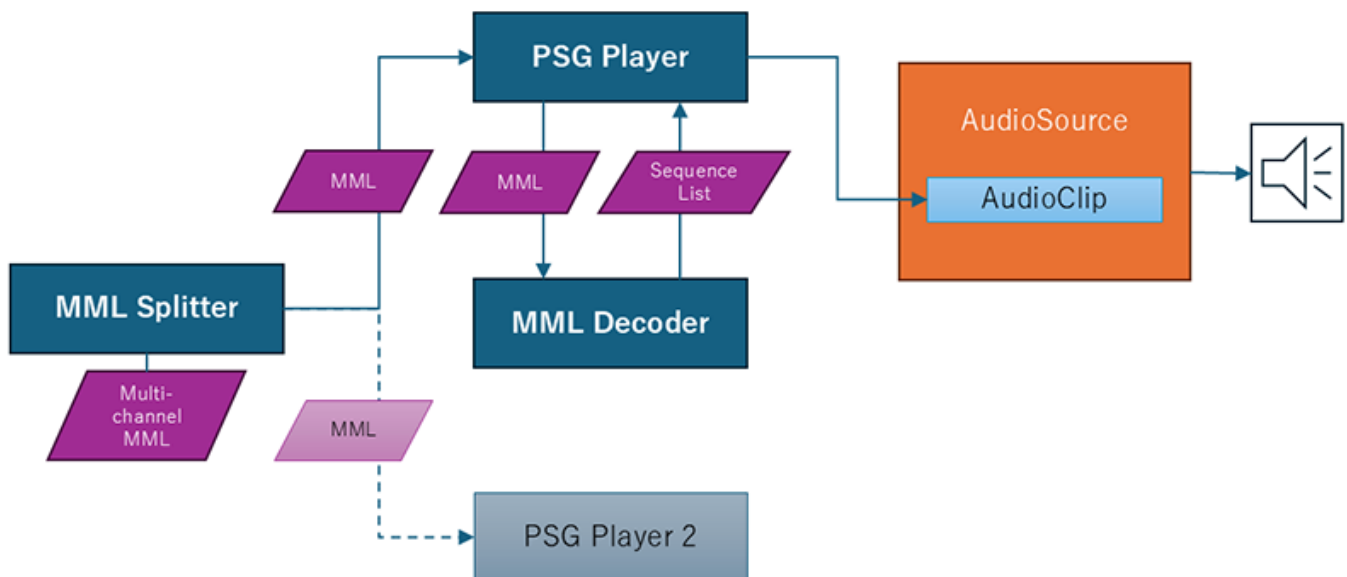


## 構成

Unity PSG Playerは、

- シーケンスデータに沿って音を生成する「**PSG Player**」
- MMLをシーケンスデータに変換する「**MML Decoder**」
- MMLを分割する「**MML Splitter**」

で構成されています。



単音で使用する場合はMML Splitterは必要ありません。

また、PSG Playerで生成した音を鳴らすためにAudioSourceコンポーネントが必要になります。

## 概要

PSG Playerは矩形波（パルス波）、三角波、ノイズを単音（モノフォニック）で生成します。生成した音はAudioClipとしてAudioSourceのリソースにアタッチされ、ストリーム再生します。ストリーム再生中のAudioClipは、必要なバッファ分のデータを順次要求し、その都度PSG Playerが波形データを計算します。

PSG Playerは、List配列のシーケンスデータを順次処理して、音程や音量を計算します。  
MMLからシーケンスデータへの変換はMML Decoderで行います。

また、マルチチャンネルMMLを振り分けるのにMML Splitterを使います。  
MMLの行ごとに処理し、行頭の文字で振り分けるチャンネルを決定します。  
詳しくは「[MMLリファレンス](#)」を参照してください。  
MML SplitterはMMLの分配の他に、各PSG Playerのコントロールをまとめて行うことができます。

効果音をPSGで鳴らす場合、レトロゲーム機などではBGMの1チャンネルの演奏を中断してリアルタイムで効果音を再生していましたが、PSG Playerでは先読みして演奏した音がバッファリングされる仕様なので、効果音をタイミングよく再生することができません。  
PSG Playerで効果音を再現するには、効果音専用のPSG Playerを別で用意し、BGMの1チャンネルをミュートすることで対応してください。

なお、マルチチャンネルでPSG Playerを使う場合、AudioMixerを利用すると音量の調整などがしやすくなります。

## サンプルレートについて

PSG Playerでは生成するAudioClipのサンプルレートを設定できます。  
サンプルレートは1秒間に入るサンプル数です。  
1サンプル毎に処理を行うので、サンプルレートを高くするとCPU負荷も高くなります。

また、サンプルレートは生成する音の周波数の上限にも影響します。  
音色によって1波長に必要なサンプル数は以下の通りです。

音色	サンプル数
矩形波	2サンプル
25%(75%)パルス波	4サンプル
12.5%パルス波	8サンプル
三角波	32サンプル

問題なく出せる音の周波数は、サンプルレートを上記のサンプル数で割った値となるので、特に三角波は最高音程が低くなります。

v0.9.2beta三角波の生成ロジックを少し変更したので、矩形波と同程度の高音まで出せるようになりました。

※あくまでも期待通りの音程が出せる上限であり、高音になると音色は崩れていきます。  
デフォルトのサンプリングレートでは、ちゃんとした音が出るのはおよそ7オクターブ目(o7)ぐらいまでです。  
※ノイズは独特な処理をしているので、サンプルレートが低いと高音で音量が下がります。

## PSG Playerスクリプトリファレンス

### 変数・Public関数一覧

- [変数](#)

- [mmlDecoder](#)
- [audioSource](#)
- [sampleRate](#)
- [audioClipSizeMilliSec](#)
- [a4Freq](#)
- [tickPerNote](#)
- [programChange](#)
- [seqListIndex](#)
- [seqList](#)
- [mmlString](#)
- [Public関数](#)
  - [Play\(\)](#)
  - [Play\(string \\_mmlString\)](#)
  - [DecodeMML\(\)](#)
  - [PlayDecoded\(\)](#)
  - [PlaySequence\(\)](#)
  - [Stop\(\)](#)
  - [IsPlaying\(\)](#)
  - [Mute\(bool isOn\)](#)
  - [ExportSeqJson\(bool \\_prettyPrint\)](#)
  - [DecodeAndExportSeqJson\(bool \\_prettyPrint\)](#)
  - [GetSeqJson\(\)](#)
  - [ImportSeqJson\(string \\_jsonString\)](#)
  - [SetSeqJson\(SeqJson \\_seqJson\)](#)
  - [RenderSequenceTodClipData\(\)](#)
  - [ExportRenderedAudioClip\(\)](#)

---

## 変数

---

### mmlDecoder

```
[SerializeField] private MMLDecoder mmlDecoder;
```

MMLをシーケンスデータに変換するMML Decoderコンポーネントを登録します。

---

### audioSource

```
[SerializeField] private AudioSource audioSource;
```

生成したAudioClipを渡すAudioSourceを登録します。

---

## sampleRate

```
public int sampleRate = 32000;
```

AudioClipのサンプルレートを設定します。  
デフォルトは32000Hz（32kHz）です。

---

## audioClipSizeMilliSec

```
public int audioClipSizeMilliSec = 1000;
```

AudioClipの長さを設定します。  
デフォルトは1000msec（1秒）です。

---

## a4Freq

```
public float a4Freq = 440f;
```

音階の基準となるオクターブ4のラ（o4a）の音の周波数を設定します。  
デフォルトは440Hzです。  
この変数はMMLのコマンドで変更することができます。

---

## tickPerNote

```
public int tickPerNote = 960;
```

1拍（4分音符）の分解能を設定します。  
音長はこの分解能に基づいたティック数に変換され、実際の音の長さ（秒）はテンポとこの分解能から計算されます。

(例：8分音符は480ティックとなり、テンポ120の場合音の長さは  
 $60[\text{sec}] / 120[\text{notePerMin}] * 480[\text{tick}] / 960[\text{tickPerNote}] = 0.25[\text{sec}]$   
で0.25秒となります。)

---

## programChange

```
public int programChange;
```

生成する音色の番号です。

番号	波形
0	パルス波（12.5%）
1	パルス波（25%）
2	矩形波（50%）
3	パルス波（75%）
4	三角波
5	ノイズ
6	短周期ノイズ

この変数はMMLのコマンドで変更することができます。

### seqListIndex

```
[SerializeField] private int seqListIndex = 0;
```

シーケンスデータの処理中の位置です。  
主にデバッグ用途で表示します。

### seqList

```
[SerializeField] private List<SeqEvent> seqList = new();
```

シーケンスデータのList配列です。  
主にデバッグ用途で表示します。

### mmlString

```
[Multiline] public string mmlString = "";
```

演奏するMML文字列です。  
この変数をMML Decoderに渡してシーケンスデータに変換します。

### Public関数



---

## Play()

```
public void Play();
```

- パラメーター：なし

[mmlString](#)のMML文字列をシーケンスデータに変換して、再生を開始します。

---

## Play(string \_mmlString)

```
public void Play(string _mmlString);
```

- パラメーター：**\_mmlString** MML文字列

パラメーターの引数を[mmlString変数](#)に渡して、シーケンスデータに変換して再生を開始します。

---

## DecodeMML()

```
public bool DecodeMML();
```

- パラメーター：なし
- 戻り値：デコード成功で**True**

[mmlString](#)のMML文字列をMML Decoderに渡してシーケンスデータに変換します。

---

## PlayDecoded()

```
public void PlayDecoded();
```

- パラメーター：なし

デコード済みのシーケンスデータを再生します。  
変換処理を行わないので、CPU負荷の軽減が期待できます。

---

## PlaySequence()

```
public void PlaySequence();
```

- パラメーター：なし

デコード済みのシーケンスデータを再生します。

[PlayDecoded\(\)](#)と同じです。

---

## Stop()

```
public void Stop();
```

- パラメーター：なし

再生中の音を停止します。

---

## IsPlaying()

```
public bool IsPlaying();
```

- パラメーター：なし
- 戻り値：再生中ならTrue

[AudioSource](#)の再生状況を返します。

---

## Mute(bool isOn)

```
public void Mute(bool isOn);
```

- パラメーター：**isOn** Trueならミュートオン

ミュートをオンにすると、[AudioSource](#)をミュートにした上で、生成されるサンプルを0（無音）にします。  
[False](#)で解除すると、[AudioSource](#)は即時にミュート解除されますが、サンプルは次のノートイベントが発生するまで無音を継続します。

ただし、バッファ済みのサンプルより早くミュート解除した場合は、すでに生成されたサンプルが発音されます。

---

## ExportSeqJson(bool \_prettyPrint)

```
public string ExportSeqJson(bool _prettyPrint)
```

### v0.9.3beta

- パラメーター：\_prettyPrint **True**で改行やインデントが有効（省略時**False**）
- 戻り値：**JSON形式の文字列**

デコードされたMMLのシーケンスデータをJSON化して文字列として出力します。  
シーケンスデータが無い（[DecodeMML\(\)](#)や[Play\(\)](#)を行ってない）場合はNullが戻ります。  
JSONの中身は、[tickPerNote](#)の値と[seqList](#)を組み合わせたものになります。

---

#### DecodeAndExportSeqJson(bool \_prettyPrint)

```
public string DecodeAndExportSeqJson(bool _prettyPrint)
```

### v0.9.3beta

- パラメーター：\_prettyPrint **True**で改行やインデントが有効（省略時**False**）
- 戻り値：**JSON形式の文字列**

MMLをデコードしてからシーケンスデータをJSON化して出力します。

---

#### GetSeqJson()

```
public SeqJson GetSeqJson()
```

### v0.9.3beta

- パラメーター：なし
- 戻り値：**SeqJsonクラスオブジェクト**

デコードされたMMLのシーケンスデータをSeqJsonクラスオブジェクトとして出力します。  
[ExportSeqJson\(\)](#)でJSON化する前のデータです。  
主にMML SplitterでマルチチャンネルJSONをエクスポートする際に利用します。

---

#### ImportSeqJson(string \_jsonString)

```
public bool ImportSeqJson(string _jsonString)
```

### v0.9.3beta

- パラメーター：**\_jsonString** JSON形式の文字列
- 戻り値：インポート成功で**True**

JSON形式の文字列をシーケンスデータとしてインポートします。  
この際、[tickPerNote](#)の値も読み込まれます。

---

### SetSeqJson(SeqJson \_seqJson)

```
public bool SetSeqJson(SeqJson _seqJson)
```

#### v0.9.3beta

- パラメーター：**\_seqJson** SeqJsonクラスオブジェクト
- 戻り値：インポート成功で**True**

SeqJsonクラスオブジェクトから直接[tickPerNote](#)の値とシーケンスデータを読み込みます。  
主にMML SplitterでマルチチャンネルJSONをインポートする際に利用します。

---

### RenderSequenceTodClipData()

```
public float[] RenderSequenceTodClipData()
```

#### v0.9.4beta

- パラメーター：なし
- 戻り値：**float配列で構成されるサンプルデータ**

シーケンス全体の波形データを生成します。

[sampleRate](#)で指定されたサンプルレートで1サンプル毎に計算し、各サンプルをfloat配列にして返します。  
サンプルレートが高いほど、そしてシーケンスが長いほど、レンダリングに時間がかかります。  
また、シーケンスのループコマンド（[MMLのループ"L"](#)）は、レンダリング時は無効になります。

---

### ExportRenderedAudioClip()

```
public AudioClip ExportRenderedAudioClip()
```

#### v0.9.4beta

- パラメーター：なし
- 戻り値：**AudioClip**

シーケンス全体の波形データ生成し、AudioClipにしてエクスポートします。

AudioClipのサンプルレートは[sampleRate](#)で指定した値になります。

効果音などレスポンスを重視したい場合に、この関数でAudioClipを先に用意しておくことができます。  
逆にBGMなど長いシーケンスの場合は、レンダリングに時間がかかるので注意が必要です。

---

# MML Splitterスクリプトリファレンス

## [MML Splitter] 変数・Public関数一覧

- [変数](#)
  - [psgPlayers](#)
  - [multiChMMLString](#)
- [Public関数](#)
  - [SplitMML\(\)](#)
  - [SplitMML\(string \\_multiChMMLString\)](#)
  - [SetAllChannelsSampleRate\(int \\_rate\)](#)
  - [SetAllChannelClipSize\(int \\_msec\)](#)
  - [PlayAllChannels\(\)](#)
  - [PlayAllChannelsDecoded\(\)](#)
  - [PlayAllChannelsSequence\(\)](#)
  - [DecodeAllChannels\(\)](#)
  - [StopAllChannels\(\)](#)
  - [IsAnyChannelPlaying\(\)](#)
  - [MuteChannel\(int channel, bool isMute\)](#)
  - [ExportMultiSeqJson\(bool \\_prettyPrint\)](#)
  - [DecodeAndExportMultiSeqJson\(bool \\_prettyPrint\)](#)
  - [ImportMultiSeqJson\(string \\_jsonString\)](#)
  - [ExportMixedAudioClip\(int \\_sampleRate\)](#)

---

## [MML Splitter] 変数

---

### **psgPlayers**

```
[SerializeField] private PSGPlayer[] psgPlayers;
```

MMLを分割送信するPSG Playerコンポーネントをチャンネルの数だけ登録します。

---

### **multiChMMLString**

```
public string multiChMMLString;
```

分割送信する元のMML文字列を登録します。

---

## [MML Splitter] Public関数

---

## SplitMML()

```
public void SplitMML();
```

- パラメーター：なし

multiChMMLStringのMML文字列を分割して、psgPlayersに登録されてるPSG Playerに送信します。  
送信チャンネルの振り分けについては「[MMLリファレンス](#)」を参照してください。

---

## SplitMML(string \_multiChMMLString)

```
public void SplitMML(string _multiChMMLString);
```

- パラメーター：\_multiChMMLString マルチチャンネルのMML文字列

パラメーターの引数をmultiChMMLString変数に渡して、MMLをPSG Playerに分割送信します。

---

## SetAllChannelsSampleRate(int \_rate)

```
public void SetAllChannelsSampleRate(int _rate);
```

- パラメーター：\_rate サンプルレート

全てのPSG Playerのサンプルレートを設定します（単位Hz）。

---

## SetAllChannelClipSize(int \_msec)

```
public void SetAllChannelClipSize(int _msec);
```

- パラメーター：\_msec AudioClipの長さ

全てのPSG PlayerのAudioClip長さを設定します（単位ミリ秒）。

---

## PlayAllChannels()

```
public void PlayAllChannels();
```

- パラメーター：なし

全てのPSG Playerで同時にMMLをデコードして再生します。

マルチチャンネルMMLの分割送信は行わないので、事前に[SplitMML\(\)](#)を実行してください。

---

### **PlayAllChannelsDecoded()**

```
public void PlayAllChannelsDecoded();
```

- パラメーター：なし

全てのPSG Playerで同時にデコード済みのシーケンスデータを再生します。

---

### **PlayAllChannelsSequence()**

```
public void PlayAllChannelsSequence()
```

- パラメーター：なし

全てのPSG Playerで同時にシーケンスデータを再生します。

[PlayAllChannelsDecoded\(\)](#)と同じです。

---

### **DecodeAllChannels()**

```
public void DecodeAllChannels()
```

- パラメーター：なし

全てのPSG PlayerでMMLをシーケンスデータにデコードします。

マルチチャンネルMMLの分割送信は行わないので、事前に[SplitMML\(\)](#)を実行してください。

---

### **StopAllChannels()**

```
public void StopAllChannels();
```

- パラメーター：なし

全てのPSG Playerの再生を停止します。

---

## IsAnyChannelPlaying()

```
public bool IsAnyChannelPlaying();
```

- パラメーター：なし
- 戻り値：いずれかのPSG Playerが再生中ならTrue

各PSG PlayerのAudioSourceのうち、どれか一つでも再生中ならTrueを返します。

---

## MuteChannel(int channel, bool isMute)

```
public void MuteChannel(int channel, bool isMute);
```

- パラメーター：**channel** 対象チャンネル  
**isMute** Trueならミュートオン

指定したチャンネルをミュートします。

---

## ExportMultiSeqJson(bool \_prettyPrint)

```
public string ExportMultiSeqJson(bool _prettyPrint)
```

### v0.9.3beta

- パラメーター：**\_prettyPrint** Trueで改行やインデントが有効（省略時False）
- 戻り値：**JSON形式の文字列**

各PSG PlayerのデコードされたMMLのシーケンスデータをまとめてJSON化して文字列として出力します。JSONの中身は、各PSG Playerで出力したSeqJsonクラスオブジェクトをリスト化したものになります。

---

## DecodeAndExportMultiSeqJson(bool \_prettyPrint)

```
public string DecodeAndExportMultiSeqJson(bool _prettyPrint)
```

### v0.9.3beta

- パラメーター：**\_prettyPrint** Trueで改行やインデントが有効（省略時False）
- 戻り値：**JSON形式の文字列**



各PSG PlayerのMMLをデコードしてからシーケンスデータをJSON化して出力します。  
マルチチャンネルMMLの分割送信は行わないので、事前に[SplitMML\(\)](#)を実行してください。

---

### ImportMultiSeqJson(string \_jsonString)

```
public void ImportMultiSeqJson(string _jsonString)
```

#### v0.9.3beta

- パラメーター： **\_jsonString** JSON形式の文字列

JSON形式の文字列をマルチチャンネルのシーケンスデータとしてインポートします。

---

### ExportMixedAudioClip(int \_sampleRate)

```
public AudioClip ExportMixedAudioClip(int _sampleRate)
```

#### v0.9.4beta

- パラメーター： **\_sampleRate** AudioClipのサンプルレート
- 戻り値： **AudioClip**

各PSG Playerでレンダリングした波形データをミックスしてAudioClipをエクスポートします。  
混合した波形データがサンプル値の範囲を超えないように、チャンネル数で割った値で合成します。  
そのため、単音の最大音量はチャンネル数によって減少します。  
また、全チャンネルでサンプルレートを揃える必要があるので、この関数を使用後は各PSG Playerの [sampleRate](#)が引数で指定した値になります。

全部のPSG Playerでレンダリングした後にミックスするので、特に長いシーケンスの場合は処理に時間がかかるので注意してください。

---