VOCALOID™

Job プラグイン API リファレンスマニュアル

文書番号: VJP-1.0.0.3 / API バージョン: 3.0.1.0

改版履歴

文書番号	API バージョン	変更内容
VJP-1.0.0.0	3.0.0.1	初版作成
VJP-1.0.0.1	3.0.0.1	TIPS を追加 ・ ファイル入出力ライブラリ関数の使い方 ・ 外部プログラムやコマンドを実行する方法
VJP-1.0.0.2	3.0.1.0	ノートの発音記号プロテクトの取得および更新機能を追加。
VJP-1.0.0.3	3.0.1.0	Job プラグインスクリプトの main 関数へ渡される実行環境パラメータテーブル（第 2 パラメータ）に、現在の Job プラグイン API のバージョンを示すフィールドを追加。 API を追加 ・ プリメジャーを取得する ・ オーディオデバイス名を取得する Job プラグイン API 一覧を追加 TIPS を追加 ・ 使用可能な Job プラグイン API を調べる方法

目次

1　はじめに

1.1　Job プラグインとは

Job プラグインは、プログラミング言語 Lua（Ver. 5.1.4）で記述されたスクリプトによって、VOCALOID3 エディタの Musical パート編集機能を、外部からユーザのニーズに応じて自由に拡張することができる仕組みを提供するものです。そして、この仕組みは Job プラグイン API によって提供されます。

Job プラグイン API は、Lua スクリプトから Musical パート情報へのアクセスおよび Musical パート情報更新機能を提供する、一連の関数群によって構成されます。

なお、このドキュメントでは以降、Job プラグインとして記述された Lua スクリプトを、Job プラグインスクリプトと呼ぶことにします。

1.2　バージョン

このドキュメントで解説する Job プラグイン API のバージョンは、3.0.1.0 です。

1.3　Job プラグイン API で出来ることと出来ないこと

1.3.1　Job プラグイン API で出来ること

Job プラグイン API を使えば以下のことが出来ます。

- Musical パートのノートの基本プロパティーの取得と更新
- Musical パートのノートの表情コントロールパラメータの取得と更新
- Musical パートのノートのビブラートタイプの取得と更新
- Musical パートのノートのビブラート長の取得と更新
- Musical パートのコントロールパラメータの取得と更新
- Musical パートのプロパティーの取得と更新
- マスタトラックのテンポの取得
- マスタトラックの拍子の取得
- シーケンスのプロパティーの取得
- WAV パートのプロパティーの取得

1.3.2　Job プラグイン API で出来ないこと

現在のところ、以下のことを行う Job プラグイン API は公開しておりません。

- Musical パートのノートのビブラートコントロールパラメータの取得と更新
- マスタトラックのテンポの更新
- マスタトラックの拍子の更新
- シーケンスのプロパティーの更新
- WAV パートのプロパティーの更新
- 上記以外の情報の取得および更新

2　Job プラグインスクリプトの構造

2.1　文法

Job プラグインスクリプトは、Lua 言語の文法に準拠します。また、Lua 言語が提供する全ての言語機能にアクセスできます。

Lua 言語の文法についての解説は、このドキュメントでは割愛させていただきます。Lua 言語の公式 WEB ページならびに市販の解説書等を参考にしてください。

2.2　エントリポイント関数

Job プラグインスクリプトは、必ず "main" という名前の関数を持たなければなりません。

この関数が、Job プラグインスクリプトのエントリポイントとなります。また、この関数の終了が、当該 Job プラグイン全体の終了になります。

2.2.1　パラメータリスト

"main" 関数は2つのパラメータを持ちます。

第1パラメータは、対象 Musical パートの内の、エディタ上でユーザが選択した始点と終点の時刻およびカレントソングポジションの時刻を格納したテーブル型変数であり、VOCALOID3 エディタから与えられるものです。単位はいずれも Musical パートの先頭を0とする Tick です。なお、ユーザが何も選択せずに Job プラグインを実行した場合には、始点はパートの先頭(0)に、終点はそのパートの末尾イベントの末尾時刻になります。

Job プラグインでは、この始点と終点の時刻の範囲を考慮してデータの更新処理を行うことができます。なお、データの取得可能範囲は Musical パート全体であり、更新可能範囲についても VOCALOID3 エディタ側では制限を設けていませんので、この範囲を超えて更新を行うことも可能です。しかし、この範囲を著しく超えたデータの更新は、これを実行したユーザの意図とは異なる結果になる場合が考えられます。よって、この範囲を超える部分の更新は、たとえばフェードイン／フェードアウト等のマージンとして許容される妥当な範囲内に収まるように考慮した実装を行う必要があります。

一方で、このパラメータをどの様に使用するかは、それぞれの Job プラグインの処理内容に応じて一定の自由度が許容されます。たとえば、以下のような使用方法が考えられます。

✧　指定区間を削除するプラグインの場合(サンプル： DeleteSelection.lua)

VOCALOID3 エディタ上でユーザが選択した始点と終点の時刻の区間を削除区間と定義し、その区間に存在する全てのイベントを削除し、その区間以降の時刻に存在する全てのイベントを削除区間の時間だけ前方へシフトする。

上記を仕様とする Job プラグインは、ユーザが選択した始点と終点の区間を超えてデータが更新されますが、その仕様をユーザに対して明記すれば、このような Job プラグインを作成することは問題無いと考えられます。

✧　指定時刻以降のノートの歌詞を変更する Job プラグインの場合

この場合、VOCALOID3 エディタ上でユーザが選択した始点と終点の時刻は使用せず、ソ

ングポジションの時刻以降のノートの歌詞を変更するという仕様の Job プラグインとして実装することも考えられます。

この場合も、仕様をユーザに対して明記すれば、このような Job プラグインを作成することは問題無いと考えられます。

第2パラメータは、Job プラグインの動作環境についての情報を格納したテーブル型変数であり、以下の情報が含まれます。

✧ 起動された Job プラグインのソースコードが配置されているディレクトリのパス

末尾にデリミタ "¥" が付けられた Job プラグインスクリプトファイルのフルパスです。なお、このパスが Job プラグイン実行時のカレントディレクトリとなります。

✧ 起動された Job プラグインのソースファイル名

拡張子付きの Job プラグインスクリプトファイル名のみであり、フルパスではありません。

✧ Job プラグインが利用可能なテンポラリディレクトリのパス

末尾にデリミタ "¥" が付けられたフルパスです。Job プラグインは、このディレクトリ以下に自由に一時ファイルや一時ディレクトリを作成することが出来ます。

✧ 現在の Job プラグイン API のバージョン

“.”区切りの 4 組の整数で表現された、現在の Job プラグイン API のバージョンです。

なお、Job プラグイン実行時の上記以外のパラメータの取得は、Job プラグイン API を通して行います。

2.2.2 リターンコード

"main" 関数は、必ず 1 つのリターンコードを返さなければなりません。

リターンコードは以下に示すとおりです。

✧ 正常終了させてスクリプトからの更新を Musical パートへ反映させたい場合

0 を返します。

✧ エラーまたはスクリプトからの更新をキャンセルさせたい場合

0 以外の値を返します。

0 以外の場合のリターンコードの値についての規約はありません。Job プラグインごとに自由に決めることができます。しかし、その値によって VOCALOID3 エディタ側の動作をコントロールすることはできません。

2.2.3 サンプルコード

以下に、何も行わない最も単純な Job プラグインスクリプトのエントリポイント関数のサンプルを示します。(サンプル： DoNothing.lua)

```
--
-- 何もしない Job プラグインスクリプト.
--

-- Job プラグインスクリプトのエントリポイント関数.
function main(processParam, envParam)
    -- 実行時に渡されたパラメータの取得.
    beginPosTick = processParam.beginPosTick  -- 選択範囲の始点時刻.
    endPosTick   = processParam.endPosTick    -- 選択範囲の終点時刻.
    songPosTick  = processParam.songPosTick   -- カレントソングポジション時刻.

    -- 実行時に渡された実行環境パラメータを取得する.
    scriptDir  = envParam.scriptDir   -- スクリプトが配置されているディレクトリパス.
    scriptName = envParam.scriptName  -- スクリプトのファイル名.
    tempDir    = envParam.tempDir     -- Job プラグインが利用可能な一時ディレクトリパス.
    apiVersion = envParam.apiVersion  -- 現在の Job プラグイン API のバージョン.

    -- Job プラグインの処理をここ以下で行います.

    -- 正常終了.
    return 0
end
```

2.3 Job プラグインマニフェスト関数

2.3.1 提供する情報

"manifest"関数は、Job プラグインスクリプト側から VOCALOID3 エディタへ、その Job プラグインが何をするものなのか、製作者、バージョン等の情報を提供するための関数です。**Job プラグインスクリプトは、"main"関数同様、この関数も必ず備えていなければなりません。**

この関数が返すべき値は以下の通りです。いずれも、省略することは出来ません。

- Job プラグインの名称
- 何を行う Job プラグインかを説明するコメント
- 製作者名
- Job プラグインを一意に識別する ID（GUID）
- Job プラグインのバージョン（ドットで連結された4つ組みのバージョン）
- 利用する Job プラグイン API のバージョン（現時点では "3.0.0.1"）

なお、上記の Job プラグインを一意に識別する ID（GUID）は、Microsoft Windows SDK に付属する guidgen.exe 等を使用して生成してください。この際、生成する GUID のフォーマットはレジストリフォーマットにします。

2.3.2 サンプルコード

以下に、Job プラグインスクリプトのプラグインマニフェスト関数のサンプルを示します。

```
--
-- プラグインマニフェスト関数.
--
function manifest()
    myManifest = {
        name          = "神調教プラグイン", -- Job プラグインの名称
        comment       = "神調教します",     -- 何を行う Job プラグインかを説明するコメント
        author        = "プラグイン職人",   -- 製作者名
        pluginID      = "{AE31E35D-2FD4-4608-8D67-C2B45353192A}",  -- Job プラグイン ID
        pluginVersion = "1.0.0.1",  -- Job プラグインのバージョン
        apiVersion    = "3.0.0.1"   -- 利用する Job プラグイン API のバージョン
    }

    return myManifest
end
```

3 Job プラグイン API

3.1 規約

3.1.1 命名規則

Job プラグインスクリプトは、Lua 言語が提供する全ての言語機能にアクセスできます。そこで、これらの名前との衝突を避けるためと、Job プラグイン API であることを明確に分かるようにするために、Job プラグイン API には以下の命名規則を適用します。

- ✧ API 名のプレフィックス

  Job プラグイン API 名には、プレフィックス "VS" を付けます。

- ✧ Lua テーブル名のプレフィックス

  Lua テーブルのテーブル名にも、プレフィックス "VS" を付けます。

- ✧ 変数名

  Lua テーブルのフィールド名等の変数名は、小文字の英字からはじめます。

3.1.2 データ型

Lua 言語には基本的にデータ型はありません。

しかし、VOCALOID3 エディタが扱うデータとの整合性を保つため、ここでは便宜的に以下のデータ型を定義し、このドキュメントではこれらのデータ型を使用して以降の説明をすることにします。

- ✧ VSInt32
  32 ビット符号付整数です。
- ✧ VSFloat
  単精度浮動小数点数です。
- ✧ VSBool
  真偽値です。真のとき VS_TRUE(1)、偽のとき VS_FALSE(0)をとります。
- ✧ VSCString
  定数文字列です。使用可能な文字コードは **UTF-8 のみ**です。

3.1.3 テーブル型の定義方法

Lua 言語にはテーブル型というデータ構造があります。

Job プラグイン API では、テーブル型を Musical パートデータの構造体(ノート情報等)を扱うために使用します。

このドキュメントでは、便宜的にテーブル型の定義方法として C++言語の構造体定義類似の擬似文法を使用して説明します。

たとえば、ノートイベント型の定義は以下のような記述になります。

```
// ノートイベント型.
struct VSLuaNote {
    VSInt32   posTick;      // ノート ON 時刻.
    VSInt32   durTick;      // ノート Duration.
    VSInt32   noteNum;      // 音程.
    VSInt32   velocity;     // ベロシティー.
    VSCString lyric;        // 歌詞.
    VSCString phonemes;     // 発音記号（スペースで区切られる）.
};
```

そして、これを Job プラグインスクリプトでは Lua テーブルとして以下のように扱います。

```
-- ノートイベントの設定.
note = {}                       -- ノートテーブルデータの生成.
note.posTick  = 1920            -- ノート ON 時刻の設定.
note.durTick  = 480             -- ノート Duration の設定.
note.noteNum  = 69              -- 音程の設定.
note.velocity = 64              -- ベロシティーの設定.
note.lyric    = "spring"        -- 歌詞の設定.
note.phonemes = "s p r I N"     -- 発音記号の設定.
```

3.1.4 Job プラグイン API プロトタイプの定義方法

Job プラグイン API のプロトタイプ定義も、C++言語類似の擬似文法を使用します。

たとえば、Lua 言語では、関数が複数の値を返すことができます。このような API のプロトタイプの定義を、このドキュメントでは以下のように行います。

```
// パラメータ入力ダイアログのフィールドから入力された整数値を取得する.
//   パラメータ:
//     fieldName: 取得先のダイアログのフィールド名.
//   戻り値:
//     result: 正常に取得できたとき VS_TRUE, エラーのとき VS_FALSE.
//     value:  パラメータ入力ダイアログのフィールドから入力された整数値.
VSBool result, VSInt32 value VSDlgGetIntValue(VSCString fieldName);
```

上記 API は、1 つの文字列型の引数を取り、真偽型と 32 ビット符号付整数型の 2 つのリターンコードを返す API であることを意味します。

この API の Job プラグインスクリプトでの実際の使用例は以下のようになります。

```
-- パラメータ入力ダイアログからノートの音程の入力値を取得する.
result, noteNum = VSDlgGetIntValue("noteNum")
```

この例は、パラメータ入力ダイアログの"noteNum"という名前のフィールドから 32 ビット符号付整数型の入力値を取得し、その値を noteNum という変数に格納しています。同時に result という真偽型の変数に、この API が成功したか否かの結果を格納しています。

3.1.5 文字コード

Job プラグインスクリプトで使用可能な文字コードは UTF-8 のみです。

3.2 VOCALOID3 エディタとのパラメータの受け渡し API

3.2.1 パラメータ入力ダイアログの動的生成

VOCALOID3 エディタは、Job プラグインスクリプトからのパラメータ受け渡し API コールにし

たがって、Job プラグインスクリプトへのパラメータ入力ダイアログを動的に生成します。

3.2.2 使用するテーブル型と定数の定義

VOCALOID3エディタとのパラメータ受け渡しAPIで使用するテーブル型定義と定数定義を以下に示します。

```
// パラメータ入力ダイアログのフィールドタイプ.
enum VSFlexDlgFieldType {
    FT_INTEGER = 0,     // 整数タイプ.
    FT_BOOL,            // 真偽値タイプ.
    FT_FLOAT,           // 実数タイプ.
    FT_STRING           // 文字列タイプ.
    FT_STRING_LIST      // 文字列リストタイプ（コンボボックス）.
};

// パラメータ入力ダイアログのフィールド定義構造体.
struct VSFlexDlgField {
    VSCString name;         // フィールド名.
    VSCString caption;      // フィールドのキャプション.
    VSCString initialVal;   // フィールドの初期値.
    VSInt32  type;          // フィールドタイプ（FlexDlgFieldType の値をとる）.
};
```

3.2.3 フィールド定義の追加 API

✧ VSBool VSDlgAddField(VSFlexDlgField field)

パラメータ入力ダイアログへ表示する入力フィールド情報を追加します。

なお、フィールドタイプに "FT_STRING_LIST" を指定した場合は、initialVal フィールドにカンマ(",")で区切った文字列を設定して呼び出します。これによって、パラメータ入力ダイアログへコンボボックスフィールドが追加され、コンボボックスの各行へはカンマで区切られた文字列の単位で挿入されます。

```
// パラメータ入力ダイアログにフィールドを追加する.
//   パラメータ: パラメータ入力ダイアログのフィールド定義.
//   戻り値: 正常終了のときVS_TRUE,エラーのときVS_FALSE.
VSBool VSDlgAddField(VSFlexDlgField field);
```

✧ void VSDlgSetDialogTitle(VSCString dlgTitle)

パラメータ入力ダイアログのウィンドウタイトルを設定します。

```
// パラメータ入力ダイアログのウィンドウタイトルを設定する.
//    パラメータ: パラメータ入力ダイアログのウィンドウタイトル文字列.
//    戻り値: なし.
void VSDlgSetDialogTitle(VSCString dlgTitle);
```

3.2.4 ダイアログの表示指示 API

✧ VSInt32 VSDlgDoModal()

パラメータ入力ダイアログを表示させます。

モーダルダイアログとして表示するため、OK または Cancel ボタンが押されるまで制御は戻りません。

```
// モーダルダイアログを表示してパラメータの入力を行う.
//    パラメータ: なし
//    戻り値: OK ボタン押下のとき IDOK(1), Cancel ボタン押下のとき IDCANCEL(2).
VSInt32 VSDlgDoModal();
```

3.2.5 入力パラメータの取得 API

✧ VSBool result, VSInt32 value VSDlgGetIntValue(VSCString fieldName)

パラメータ入力ダイアログのフィールドから入力された整数値を取得します。

```
// パラメータ入力ダイアログのフィールドから入力された整数値を取得する.
//    パラメータ:
//       fieldName: 取得先のダイアログのフィールド名.
//    戻り値:
//       result: 正常に取得できたとき VS_TRUE, エラーのとき VS_FALSE.
//       value:  パラメータ入力ダイアログのフィールドから入力された整数値.
VSBool result, VSInt32 value VSDlgGetIntValue(VSCString fieldName);
```

✧ VSBool result, VSBool value VSDlgGetBoolValue(VSCString fieldName)

パラメータ入力ダイアログのフィールドから入力された真偽値を取得します。

```
// パラメータ入力ダイアログのフィールドから入力された真偽値を取得する.
//    パラメータ:
//       fieldName: 取得先のダイアログのフィールド名.
//    戻り値:
//       result: 正常に取得できたとき VS_TRUE, エラーのとき VS_FALSE.
//       value:  パラメータ入力ダイアログのフィールドから入力された真偽値.
VSBool result, VSBool value VSDlgGetBoolValue(VSCString fieldName);
```

✧ VSBool result, VSFloat value VSDlgGetFloatValue(VSCString fieldName)

パラメータ入力ダイアログのフィールドから入力された浮動小数点数値を取得します。

```
// パラメータ入力ダイアログのフィールドから入力された浮動小数点数値を取得する.
//   パラメータ:
//     fieldName: 取得先のダイアログのフィールド名.
//   戻り値:
//     result: 正常に取得できたとき VS_TRUE, エラーのとき VS_FALSE.
//     value:  パラメータ入力ダイアログのフィールドから入力された浮動小数点数値.
VSBool result, VSFloat value VSDlgGetFloatValue(VSCString fieldName);
```

✧ VSBool result, VSCString value VSDlgGetStringValue(VSCString fieldName)

パラメータ入力ダイアログのフィールドから入力された文字列値を取得します。

フィールドタイプに "FT_STRING_LIST" が指定されたフィールドの場合は、コンボボックスの選択行に格納されている文字列が取得されます。

```
// パラメータ入力ダイアログのフィールドから入力された文字列値を取得する.
//   パラメータ:
//     fieldName: 取得先のダイアログのフィールド名.
//   戻り値:
//     result: 正常に取得できたとき VS_TRUE, エラーのとき VS_FALSE.
//     value:  パラメータ入力ダイアログのフィールドから入力された文字列値.
VSBool result, VSCString value VSDlgGetStringValue(VSCString fieldName);
```

3.2.6 サンプルプログラム

✧ DialogSample.lua

3.3 ノートイベントの取得および編集 API

3.3.1 使用するテーブル型の定義

ノートイベントの取得および編集 API で使用するテーブル型定義を以下に示します。

```
// ノートイベント型.
struct VSLuaNote {
    VSInt32   posTick;      // ノート ON 時刻.
    VSInt32   durTick;      // ノート Duration.
    VSInt32   noteNum;      // 音程.
    VSInt32   velocity;     // ベロシティー.
    VSCString lyric;        // 歌詞.
    VSCString phonemes;     // 発音記号（スペースで区切られる）.
    VSBool    phLock;       // 発音記号プロテクトフラグ（省略可.デフォルト：VS_FALSE）.
};
```

```
// 表情コントロールパラメータを含むノートイベント型.
struct VSLuaNoteEx {
    VSInt32   posTick;      // ノート ON 時刻.
    VSInt32   durTick;      // ノート Duration.
    VSInt32   noteNum;      // 音程.
    VSInt32   velocity;     // ベロシティー.
    VSCString lyric;        // 歌詞.
    VSCString phonemes;     // 発音記号（スペースで区切られる）.
    VSBool    phLock;       // 発音記号プロテクトフラグ（省略可.デフォルト：VS_FALSE）.

    VSInt32 bendDepth;      // ベンドデプス（0 ～ 100）.
    VSInt32 bendLength;     // ベンドレングス（0 ～ 100）.
    VSBool  risePort;       // 上行形でのポルタメント付加フラグ.
    VSBool  fallPort;       // 下行形でのポルタメント付加フラグ.
    VSInt32 decay;          // ディケイ（0 ～ 100）.
    VSInt32 accent;         // アクセント（0 ～ 100）.
    VSInt32 opening;        // オープニング（0 ～ 127）.

    // ビブラート長（整数で丸めたパーセント値）.
    VSInt32 vibratoLength;
    // ビブラートタイプ（ビブラートの種類）.
    //    0:ビブラートはかけない.
    //    1 ～  4:Normal
    //    5 ～  8:Extreme
    //    9 ～ 12:Fast
    //   13 ～ 16:Slight
    VSInt32 vibratoType;
};
```

ここで、phLock フィールドは、API バージョン 3.0.1.0 で追加されました。これは、ノートの発音記号をプロテクトするフラグです。VS_TRUE を設定すると、そのノートの発音記号はプロテクトされます。VS_FALSE を設定すると、そのノートの発音記号のプロテクトが解除されます。

この発音記号のプロテクトは、VOCALOID3 エディタから歌詞を編集したときに実行される発音記号の自動変換機能から、ノートの発音記号をプロテクトするものです。Job プラグインからの変更に対してもプロテクトするものではありません。したがって、Job プラグインからは、このプロテクトが掛けられているノートに対しても発音記号を変更することが出来ます。プロテクトが掛けら

れたノートに対して変更するかどうかは、それぞれの Job プラグインの用途に応じて Job プラグインの裁量に委ねられます。

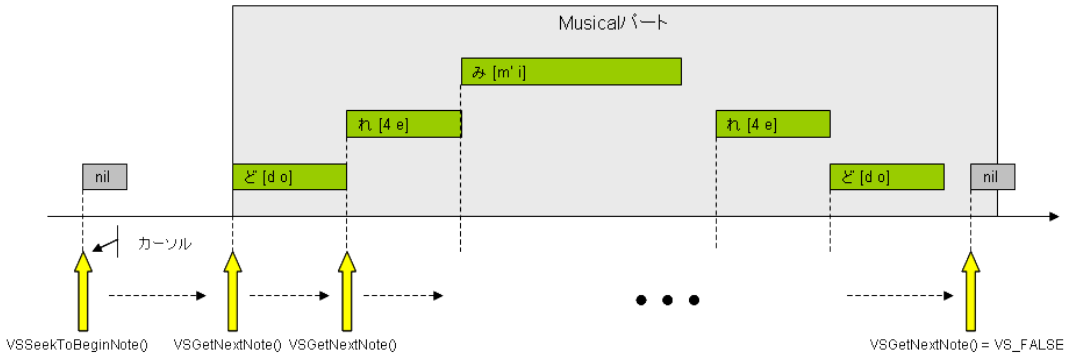
3.3.2 カーソルによる順次アクセス API

ノートイベントへのアクセスは、カーソルによる Musical パート先頭からの順次アクセスにより行うことができます。

まず、VSSeekToBeginNote() API により、Musical パートの先頭ノートの1つ前の仮想的なノートにカーソルを位置づけます。

次に、VSGetNextNote() API により、Musical パートの先頭ノートへカーソルを位置づけ、そのノートの値を取得することができます。以降、VSGetNextNote() API を繰り返しコールすることにより、順次次のノートを Musical パートの末尾ノートまで取得することができます。

Musical パートの末尾ノートの取得後に VSGetNextNote() API をコールした場合、この API は VS_FALSE(0)を返し、カーソルをその末尾ノートの1つ後ろの仮想的なノートに位置づけます。



カーソルによる順次アクセス

✧ void VSSeekToBeginNote()

カーソルを Musical パートの先頭ノートの1つ前の仮想的なノートに位置づけます。

```
// カーソルを Musical パートの先頭ノートの１つ前に位置づける.
//    パラメータ: なし.
//    戻り値: なし.
void VSSeekToBeginNote();
```

✧ VSBool result, VSLuaNote note VSGetNextNote()

カーソルを次のノートへ進め、そのノートの値を取得します。

```
// カーソルを次のノートへ進め, そのノートの値を取得する.
//   パラメータ: なし.
//   戻り値:
//     result: 正常に取得できたとき VS_TRUE, エラーのとき VS_FALSE.
//     note:   ノートの値.
VSBool result, VSLuaNote note VSGetNextNote();
```

3.3.3 ノートの更新 API

✧ VSBool VSUpdateNote(VSLuaNote note)

パラメータで渡したノートの値でノートを更新します。

パラメータへ渡せるノートは、あらかじめ VSGetNextNote() API で取得したノートでなければなりません。すなわち、VSGetNextNote() API でノートを取得し、そのノートのフィールドに対して更新すべき値を設定し、その後それを VSUpdateNote() API のパラメータとして API をコールしてノートの値を更新する手順になります。

```
// ノートの値を更新する.
//   パラメータ: 更新する値が設定されたノート.
//   戻り値: 正常に更新できたとき VS_TRUE, エラーのとき VS_FALSE.
VSBool VSUpdateNote(VSLuaNote note);
```

3.3.4 ノートの追加 API

✧ VSBool VSInsertNote(VSLuaNote note)

パラメータで渡したノートの値でノートを追加します。

パラメータで渡すノートの各フィールドは、適切な値で設定されていなければなりません。

ただし、phLock フィールドは省略することが出来ます。省略した場合、デフォルト値として VS_FALSE(発音記号をプロテクトしない)が設定されます。

phLock 以外のフィールドは省略することが出来ません。

```
// ノートを追加する.
//   パラメータ: 追加するノート.
//   戻り値: 正常に追加できたとき VS_TRUE, エラーのとき VS_FALSE.
VSBool VSInsertNote(VSLuaNote note);
```

3.3.5 ノートの削除 API

✧ VSBool VSRemoveNote(VSLuaNote note)

パラメータで渡したノートを削除します。

パラメータへ渡せるノートは、あらかじめ VSGetNextNote() API で取得したノートでなければなりません。すなわち、VSGetNextNote() API で削除すべきノートを取得し、それを

VSRemoveNote () API のパラメータとして API をコールしてノートを削除する手順になります。

```
// ノートを削除する.
//   パラメータ: 削除対象ノート.
//   戻り値: 正常に削除できたとき VS_TRUE, エラーのとき VS_FALSE.
VSBool VSRemoveNote(VSLuaNote note);
```

3.3.6 ノートの表情コントロールパラメータに関する API

✧ VSBool result, VSLuaNoteEx noteEx VSGetNextNoteEx()

カーソルを次のノートへ進め、そのノートの表情コントロールパラメータを含むノートの値を取得します。

```
// カーソルを次のノートへ進め、表情コントロールパラメータを含むノートの値を取得する.
//   パラメータ: なし.
//   戻り値:
//     result: 正常に取得できたとき VS_TRUE, エラーのとき VS_FALSE.
//     noteEx: 表情コントロールパラメータを含むノートの値.
VSBool result, VSLuaNoteEx noteEx VSGetNextNoteEx();
```

✧ VSBool VSUpdateNoteEx(VSLuaNoteEx noteEx)

パラメータで渡した表情コントロールパラメータを含むノートの値を更新します。

パラメータへ渡せるノートは、あらかじめ VSGetNextNoteEx () API で取得した表情コントロールパラメータを含むノートでなければなりません。

```
// 表情コントロールパラメータを含むノートの値を更新する.
//   パラメータ: 表情コントロールパラメータを含むノート.
//   戻り値: 正常に更新できたとき VS_TRUE, エラーのとき VS_FALSE.
VSBool VSUpdateNoteEx(VSLuaNoteEx noteEx);
```

✧ VSBool VSInsertNoteEx(VSLuaNoteEx noteEx)

パラメータで渡した表情コントロールパラメータを含むノートを追加します。

パラメータで渡すノートの各フィールドは、適切な値で設定されていなければなりません。

ただし、phLock フィールドは省略することが出来ます。省略した場合、デフォルト値として VS_FALSE(発音記号をプロテクトしない)が設定されます。

phLock 以外のフィールドは省略することが出来ません。

```
// 表情コントロールパラメータを含むノートを追加する.
//   パラメータ: 追加する表情コントロールパラメータを含むノート.
//   戻り値: 正常に追加できたとき VS_TRUE,エラーのとき VS_FALSE.
VSBool VSInsertNoteEx(VSLuaNoteEx noteEx);
```

3.3.7 サンプルプログラム

- ✧ NoteSample1.lua
- ✧ NoteSample2.lua

3.4 コントロールパラメータの取得および編集 API

3.4.1 使用するテーブルの型と定数の定義

コントロールパラメータの取得および編集 API で使用するテーブル型および定数の定義を以下に示します。

```
// コントロールパラメータの種類（文字列）.
//   "DYN": ダイナミクス
//   "BRE": ブレシネス
//   "BRI": ブライトネス
//   "CLE": クリアネス
//   "GEN": ジェンダーファクター
//   "PIT": ピッチベンド
//   "PBS": ピッチベンドセンシティビティー
//   "POR": ポルタメントタイミング

// コントロールパラメータブレークポイント型
struct VSLuaControl {
    VSInt32   posTick;  // コントロールパラメータブレークポイントの位置.
    VSInt32   value;    // コントロールパラメータブレークポイントの値.
    VSCString type;     // コントロールパラメータの種類.
};
```

3.4.2 時刻指定によるランダムアクセス方式

コントロールパラメータは、実装上は時間軸上に離散的に存在するブレークポイントとして存在していますが、概念的には時間軸に対して連続的に変化する性質を持っています。この場合、データのアクセス方式としては、ノートのような順次アクセス方式よりも、時刻指定によるランダムアクセス方式のほうがより直感的で分かりやすいと考えられます。

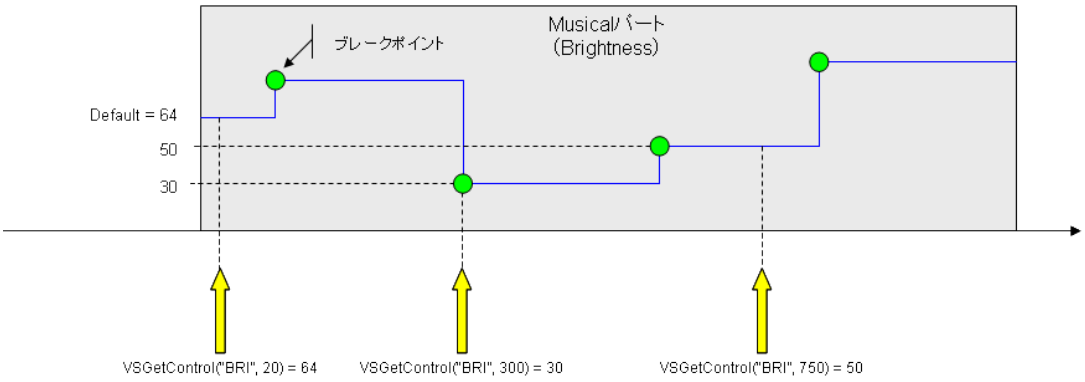
そこで、コントロールパラメータに対しては、時刻指定によるランダムアクセス方式による API

を導入することにします。

下図に、この方式の API を使用した場合の例を示します。

コントロールパラメータの値は、ブレークポイントとブレークポイントとの間が下図の要領で直線で補間されるため、Musical パート内の任意の時刻に必ず値が存在します。このため、時刻指定によるランダムアクセス方式が可能になります。

- ✧ 先頭ブレークポイントより前の時刻を指定した場合
  API は、そのコントロールパラメータのデフォルト値を返します。
- ✧ ブレークポイント上の時刻を指定した場合
  API は、そのブレークポイントの値を返します。
- ✧ ブレークポイント間および末尾ブレークポイントより後の時刻を指定した場合
  API は、その時刻より前に存在する最大時刻のブレークポイントの値を返します。



時刻指定によるランダムアクセス方式

3.4.3 時刻指定によるランダムアクセス方式 API

- ✧ VSBool result, VSInt32 value VSGetControlAt(VSCString controlType, VSInt32 posTick)
  Musical パートの指定した時刻の指定したコントロールパラメータの値を取得します。

```
// コントロールパラメータを取得する.
//   パラメータ:
//     controlType: 取得するコントロールパラメータの種類.
//     posTick:     取得するコントロールパラメータの時刻.
//                    パートの先頭を 0 とした Tick を単位とする時刻.
//   戻り値:
//     result: 正常に取得できたとき VS_TRUE, エラーのとき VS_FALSE.
//     value:  コントロールパラメータの値.
VSBool result, VSInt32 value VSGetControlAt(VSCString controlType, VSInt32 posTick);
```

- ✧ VSBool VSUpdateControlAt(VSCString controlType, VSInt32 posTick, VSInt32 value)
  Musical パートの指定した時刻の指定したコントロールパラメータの値を更新します。

```
// コントロールパラメータの値を更新する.
//   パラメータ:
//     controlType: 取得するコントロールパラメータの種類.
//     posTick:     更新するコントロールパラメータの時刻.
//                    パートの先頭を 0 とした Tick を単位とする時刻.
//     value:       更新するコントロールパラメータの値.
//   戻り値: 正常に更新できたとき VS_TRUE,エラーのとき VS_FALSE.
VSBool VSUpdateControlAt(VSCString controlType, VSInt32 posTick, VSInt32 value);
```

3.4.4 カーソルによる順次アクセス方式 API

コントロールパラメータに対しても、その値の変化点がどこにあるのかを知りたい場合があります。このような場合、コントロールパラメータのブレークポイントを先頭から順次調べていくほうが効率的です。

そこで、コントロールパラメータに対してもカーソルによる順次アクセス API を用意します。

✧ VSBool VSSeekToBeginControl(VSCString controlType)

カーソルを Musical パートの指定した先頭コントロールパラメータブレークポイントの1つ前に位置付けます。

```
// カーソルを Musical パートの指定した先頭コントロールパラメータブレークポイントの
// 1つ前に位置づける.
//   パラメータ:
//     controlType: 取得するコントロールパラメータの種類.
//   戻り値: 正常に位置づけられたとき VS_TRUE,エラーのとき VS_FALSE.
VSBool VSSeekToBeginControl(VSCString controlType);
```

✧ VSBool result, VSLuaControl control VSGetNextControl(VSCString controlType)

カーソルを Musical パートの指定したコントロールパラメータブレークポイントの次のコントロールパラメータブレークポイントへ進め、その値を取得します。

```
// カーソルを次のコントロールパラメータブレークポイントへ進め,その値を取得する.
//   パラメータ:
//     controlType: 取得するコントロールパラメータの種類.
//   戻り値:
//     result:  正常に取得できたとき VS_TRUE,エラーのとき VS_FALSE.
//     control: コントロールパラメータブレークポイントの値.
VSBool result, VSLuaControl control VSGetNextControl(VSCString controlType);
```

✧ VSBool VSUpdateControl(VSLuaControl control)

パラメータで渡したコントロールイベントの値でコントロールイベントを更新します。

パラメータへ渡せるコントロールイベントは、あらかじめ VSGetNextControl() API で取得したものでなければなりません。すなわち、VSGetNextControl() API でコントロールイベントを取得し、そのコントロールイベントのフィールドに対して更新すべき値を設定し、その後それを VSUpdateControl() API のパラメータとして API をコールしてコントロールイベントの値を更新する手順になります。

```
// コントロールイベントの値を更新する.
//    パラメータ: 更新する値が設定されたコントロールイベント.
//    戻り値: 正常に更新できたとき VS_TRUE, エラーのとき VS_FALSE.
VSBool VSUpdateControl(VSLuaControl control);
```

✧ VSBool VSInsertControl(VSLuaControl control)

パラメータで渡したコントロールイベントの値でコントロールイベントを追加します。

パラメータで渡すコントロールイベントの各フィールドは、適切な値で設定されていなければなりません。(全てのフィールドは省略できません。)

```
// コントロールイベントを追加する.
//    パラメータ: 追加するコントロールイベント.
//    戻り値: 正常に追加できたとき VS_TRUE, エラーのとき VS_FALSE.
VSBool VSInsertControl(VSLuaControl control);
```

✧ VSBool VSRemoveControl(VSLuaControl control)

パラメータで渡したコントロールイベントを削除します。

パラメータへ渡せるコントロールイベントは、あらかじめ VSGetNextControl() API で取得したものでなければなりません。すなわち、VSGetNextControl() API で削除すべきコントロールイベントを取得し、それを VSRemoveControl() API のパラメータとして API をコールしてコントロールイベントを削除する手順になります。

```
// コントロールイベントを削除する.
//    パラメータ: 削除対象コントロールイベント.
//    戻り値: 正常に削除できたとき VS_TRUE, エラーのとき VS_FALSE.
VSBool VSRemoveControl(VSLuaControl control);
```

3.4.5 デフォルト値の取得 API

✧ VSBool result, VSInt32 value VSGetDefaultControlValue(VSCString controlType)

指定したコントロールパラメータタイプのデフォルト値を取得します。

```
// 指定したコントロールパラメータタイプのデフォルト値を取得する.
//   パラメータ:
//     controlType: 取得するコントロールパラメータの種類.
//   戻り値:
//     result: 正常に取得できたとき VS_TRUE, エラーのとき VS_FALSE.
//     value:  コントロールパラメータのデフォルト値.
VSBool result, VSInt32 value VSGetDefaultControlValue(VSCString controlType);
```

3.4.6　サンプルプログラム

- ✧ ControlSample1.lua
- ✧ ControlSample2.lua
- ✧ ControlSample3.lua

3.5 マスタトラック情報の取得 API

3.5.1　使用するテーブル型の定義

マスタトラック情報の取得 API で使用するテーブル型定義を以下に示します。

なお、マスタトラック情報の時刻は、すべてシーケンス頭を原点とするグローバル Tick で表現されますので注意が必要です。(Musical パート情報で扱われる時刻は、Musical パート頭を原点とするパート相対 Tick です。)

```
// テンポイベント型.
struct VSLuaTempo {
    VSInt32   posTick;      // テンポ時刻（グローバル Tick）.
    VSFloat   tempo;        // テンポ値（BPM）.
};

// 拍子イベント型.
struct VSLuaTimeSig {
    VSInt32   posTick;      // 拍子時刻（グローバル Tick）.
    VSInt32   numerator;    // 拍子の分子.
    VSInt32   denominator;  // 拍子の分母.
};
```

3.5.2　カーソルによる順次アクセス API

マスタトラックのテンポイベントおよび拍子イベントへのアクセスは、カーソルによるシーケンス先頭からの順次アクセスにより行うことができます。

まず、VSSeekToBeginTempo() API により、シーケンスの先頭テンポの1つ前の仮想的なテ

ンポにカーソルを位置づけます。

次に、VSGetNextTempo() API により、シーケンスの先頭テンポへカーソルを位置づけ、そのテンポの値を取得することができます。以降、VSGetNextTempo() API を繰り返しコールすることにより、順次次のテンポをシーケンスの末尾テンポまで取得することができます。

シーケンスの末尾テンポの取得後に VSGetNextTempo() API をコールした場合、この API は VS_FALSE(0)を返し、カーソルをその末尾テンポの1つ後ろの仮想的なテンポに位置づけます。

上記動作は、拍子イベントに対しても同様です。

✧ void VSSeekToBeginTempo()

テンポカーソルをシーケンスの先頭テンポの1つ前に位置付けます。

```
// テンポカーソルをシーケンスの先頭テンポの１つ前に位置づける.
//   パラメータ: なし.
//   戻り値: なし.
void VSSeekToBeginTempo();
```

✧ void VSSeekToBeginTimeSig()

拍子カーソルをシーケンスの先頭拍子の1つ前に位置付けます。

```
// 拍子カーソルをシーケンスの先頭拍子の１つ前に位置づける.
//   パラメータ: なし.
//   戻り値: なし.
void VSSeekToBeginTimeSig();
```

✧ VSBool result, VSLuaTempo tempo VSGetNextTempo()

テンポカーソルを次のテンポへ進め、そのテンポの値を取得します。

```
// カーソルを次のテンポへ進め,そのテンポの値を取得する.
//   パラメータ: なし.
//   戻り値:
//     result: 正常に取得できたとき VS_TRUE,エラーのとき VS_FALSE.
//     tempo:  テンポの値.
VSBool result, VSLuaTempo tempo VSGetNextTempo();
```

✧ VSBool result, VSLuaTimeSig timeSig VSGetNextTimeSig()

拍子カーソルを次の拍子へ進め、その拍子の値を取得します。

```
// カーソルを次の拍子へ進め,その拍子の値を取得する.
//   パラメータ: なし.
//   戻り値:
//     result:  正常に取得できたとき VS_TRUE,エラーのとき VS_FALSE.
//     timeSig: 拍子の値.
VSBool result, VSLuaTimeSig timeSig VSGetNextTimeSig();
```

3.5.3 時刻指定によるランダムアクセス API

マスタトラックのテンポイベントおよび拍子イベントへのアクセスも、時刻指定によるランダムアクセス API が用意されます。

その仕組みはコントロールパラメータの時刻指定によるランダムアクセス API と同様です。

しかし、コントロールパラメータのそれとの唯一の違いは、マスタトラックの時刻の指定は、常にシーケンス頭を原点とするグローバル Tick である点です。よって、Musical パート内のコントロールパラメータの値等との関連付けを行う場合は、グローバル時刻と Musical パート内ローカル時刻との変換が必要になりますので注意が必要です。

✧ VSBool result, VSFloat tempo VSGetTempoAt(VSInt32 posTick)

指定した時刻のテンポの値を取得します。

```
// 指定した時刻のテンポの値を取得する.
//   パラメータ:
//     posTick: 取得するテンポの時刻.
//                シーケンスの先頭を 0 とした Tick を単位とする時刻.
//   戻り値:
//     result:  正常に取得できたとき VS_TRUE,エラーのとき VS_FALSE.
//     tempo:   テンポの値 (BPM).
VSBool result, VSFloat tempo VSGetTempoAt(VSInt32 posTick);
```

✧ VSBool result, VSInt32 numerator, VSInt32 denominator VSGetTimeSigAt(VSInt32 posTick)

指定した時刻の拍子の値を取得します。

```
// 指定した時刻の拍子の値を取得する.
//   パラメータ:
//     posTick: 取得する拍子の時刻.
//                  シーケンスの先頭を 0 とした Tick を単位とする時刻.
//   戻り値:
//     result:  正常に取得できたとき VS_TRUE, エラーのとき VS_FALSE.
//     numerator:   拍子の分子.
//     denominator: 拍子の分母.
VSBool result, VSInt32 numerator, VSInt32 denominator VSGetTimeSigAt(VSInt32 posTick);
```

3.5.4 シーケンス全体の情報取得 API

✧ VSCString VSGetSequenceName()

現在開いているシーケンスファイル名を取得します。

```
// シーケンスファイル名を取得する.
//   パラメータ: なし.
//   戻り値: 現在開いているシーケンスファイル名.
VSCString VSGetSequenceName();
```

✧ VSCString VSGetSequencePath()

現在開いているシーケンスファイルパスを取得します。

```
// シーケンスファイルパスを取得する.
//   パラメータ: なし.
//   戻り値: 現在開いているシーケンスファイルパス.
VSCString VSGetSequencePath();
```

✧ VSInt32 VSGetResolution()

現在開いているシーケンスの時間分解能を取得します。

```
// 時間分解能を取得する.
//   パラメータ: なし.
//   戻り値: 現在開いているシーケンスの時間分解能.
VSInt32 VSGetResolution();
```

✧ VSInt32 VSGetPreMeasure()

現在開いているシーケンスのプリメジャーを取得します。単位は小節です。

```
// プリメジャーを取得する.
//    パラメータ: なし.
//    戻り値: 現在開いているシーケンスのプリメジャー. 単位は小節.
VSInt32 VSGetPreMeasure();
```

✧ VSInt32 VSGetPreMeasureInTick()

現在開いているシーケンスのプリメジャーを、Tick を単位として取得します。

これは、1 小節目の先頭時刻を意味します。

```
// プリメジャーを取得する.
//    パラメータ: なし.
//    戻り値: 現在開いているシーケンスのプリメジャー. 単位は Tick.
VSInt32 VSGetPreMeasureInTick();
```

3.5.5 サンプルプログラム

✧ MasterTrackSample.lua

3.6 Musical パート情報の取得および編集 API

3.6.1 使用するテーブル型の定義

Musical パート情報の取得および編集 API で使用するテーブル型定義を以下に示します。

```
// パート型.
struct VSLuaMusicalPart {
    VSInt32   posTick;  // パートの位置（グローバル Tick）.
    VSInt32   playTime; // パートの最大再生時間.
    VSInt32   durTick;  // パートの Duration.
    VSCString name;     // パート名.
    VSCString comment;  // コメント.
};

// バーチャル Singer 型.
struct VSLuaMusicalSinger {
    VSInt32  vBS;           // バーチャルバンクセレクト.
    VSInt32  vPC;           // バーチャルプログラムチェンジ.

    // 音色パラメータ.
    VSInt32 breathiness;    // ブレシネス.
    VSInt32 brightness;     // ブライトネス.
    VSInt32 clearness;      // クリアネス.
    VSInt32 genderFactor;   // ジェンダーファクター.
    VSInt32 opening;        // オープニング.

    VSCString compID;       // コンポーネント ID.
};
```

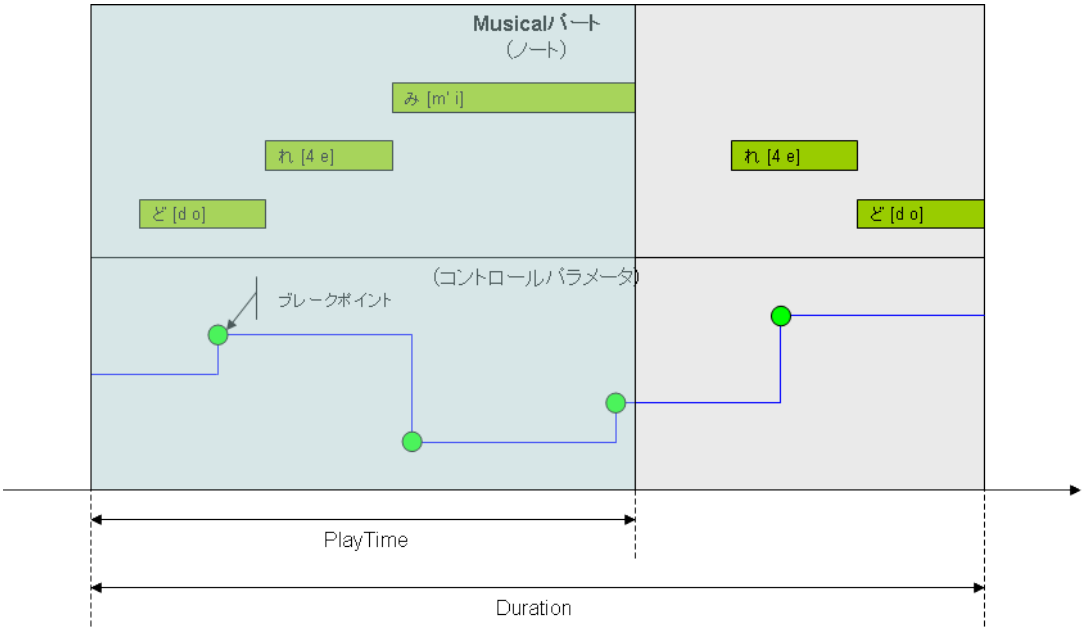
3.6.2 PlayTime と Duration について

Musical パートを含め、パートには PlayTime と Duration という2つの長さを意味するデータが存在します。

PlayTime は、実際に再生される時間（Tick）を意味します。

Duration は、そのパートの先頭時刻とパート末尾イベントの末尾時刻との距離を意味します。単位は、Musical パートでは Tick、WAV パートの場合は Sec です。

下図に、PlayTime と Duration の関係を示します。



PlayTime と Duration の関係

3.6.3 Musical パートの情報取得 API

✧ VSBool result, VSLuaMusicalPart part VSGetMusicalPart()

現在処理対象の Musical パート情報を取得します。

```
// 現在処理対象の Musical パート情報を取得する.
//   パラメータ: なし.
//   戻り値:
//     result: 正常に取得できたとき VS_TRUE, エラーのとき VS_FALSE.
//     part:   現在処理対象の Musical パート情報.
VSBool result, VSLuaMusicalPart part VSGetMusicalPart();
```

3.6.4 Musical パートの情報更新 API

✧ VSBool VSUpdateMusicalPart(VSLuaMusicalPart part)

パラメータで渡した Musical パートの値で Musical パートを更新します。

パラメータへ渡せる Musical パートは、あらかじめ VSGetMusicalPart() API で取得した Musical パートでなければなりません。すなわち、VSGetMusicalPart() API で Musical パートを取得し、そのフィールドに対して更新すべき値を設定し、その後それを VSUpdateMusicalPart() API のパラメータとして API をコールして Musical パートの値を更新する手順になります。

```
// 現在処理対象の Musical パート情報を更新する.
//   パラメータ: 更新する Musical パート情報.
//   戻り値: 正常に更新できたとき VS_TRUE, エラーのとき VS_FALSE.
VSBool VSUpdateMusicalPart(VSLuaMusicalPart part);
```

3.6.5 Musical パートのバーチャル Singer 情報取得 API

✧ VSBool result, VSLuaMusicalSinger singer VSGetMusicalPartSinger()

現在処理対象の Musical パートのバーチャル Singer 情報を取得します。Musical パートのバーチャル Singer は、Musical パートに必ず1つのみ存在します。

```
// 現在処理対象の Musical パートのバーチャル Singer 情報を取得する.
//   パラメータ: なし.
//   戻り値:
//     result: 正常に取得できたとき VS_TRUE, エラーのとき VS_FALSE.
//     singer: 現在処理対象の Musical パートのバーチャル Singer 情報.
VSBool result, VSLuaMusicalSinger singer VSGetMusicalPartSinger();
```

3.6.6 サンプルプログラム

MusicalPartSample.lua

3.7 WAV パート情報の取得 API

3.7.1 使用するテーブル型の定義

WAV パート情報の取得 API で使用するテーブル型定義を以下に示します。

```
// WAV パート型.
struct VSLuaWAVPart {
    VSInt32   posTick;      // パートの位置（グローバル Tick）.
    VSInt32   playTime;     // パートの最大再生時間.
    VSInt32   sampleRate;   // サンプリング周波数.
    VSInt32   sampleReso;   // 量子化ビット数.
    VSInt32   channels;     // チャンネル数.
    VSCString name;         // パート名.
    VSCString comment;      // コメント.
    VSCString filePath;     // WAV ファイルの絶対パス.
};
```

3.7.2 WAV パート情報取得 API

✧ VSBool result, VSLuaWAVPart wavPart VSGetStereoWAVPart()

ステレオトラックの WAV パート情報を取得します。

ステレオトラックの WAV パートは最大で 1 つまでしか持てませんので、この API ではその唯一の WAV パートを取得します。(ステレオトラックに WAV パートが存在する場合)

```
// ステレオトラックの WAV パート情報を取得する.
//   パラメータ: なし.
//   戻り値:
//     result:  正常に取得できたとき VS_TRUE, エラーのとき VS_FALSE.
//     wavPart: ステレオトラックの WAV パート情報.
VSBool result, VSLuaWAVPart wavPart VSGetStereoWAVPart();
```

✧ void VSSeekToBeginMonoWAVPart()

モノラルトラックの WAV パートカーソルを、先頭 WAV パートの1つ前に位置付けます。

```
// モノラルトラックの WAV パートカーソルを先頭 WAV パートの１つ前に位置づける.
//   パラメータ: なし.
//   戻り値: なし.
void VSSeekToBeginMonoWAVPart();
```

✧ VSBool result, VSLuaWAVPart wavPart VSGetNextMonoWAVPart()

モノラルトラックの WAV パートカーソルを次の WAV パートへ進め、その WAV パート情報を取得します。

```
// モノラルトラックの WAV パートカーソルを次へ進め、その WAV パート情報を取得する.
//   パラメータ: なし.
//   戻り値:
//     result:  正常に取得できたとき VS_TRUE, エラーのとき VS_FALSE.
//     wavPart: モノラルトラックの WAV パート情報.
VSBool result, VSLuaWAVPart wavPart VSGetNextMonoWAVPart();
```

3.7.3 サンプルプログラム

✧ WAVPartSample.lua

3.8 処理中の状態表示 API

✧ VSInt32 VSMessageBox(VSCString message, VSUint32 type)

メッセージをメッセージボックスへ表示します。

```
// メッセージをメッセージボックスへ表示する.
//   パラメータ:
//     message: 表示する文字列.
//     type: メッセージボックスのスタイル.
//             MB_OK               = 0
//             MB_OKCANCEL         = 1
//             MB_ABORTRETRYIGNORE = 2
//             MB_YESNOCANCEL      = 3
//             MB_YESNO            = 4
//             MB_RETRYCANCEL      = 5
//   戻り値: メッセージボックスからの結果コード.
//             IDOK     = 1
//             IDCANCEL = 2
//             IDABORT  = 3
//             IDRETRY  = 4
//             IDIGNORE = 5
//             IDYES    = 6
//             IDNO     = 7
VSInt32 VSMessageBox(VSCString message, VSUint32 type);
```

3.9 オーディオデバイス情報の取得 API

✧ VSCString VSGetAudioDeviceName()

現在使用しているオーディオデバイスのデバイス名を取得します。

```
// 現在使用中のオーディオデバイスのデバイス名を取得する.
//   パラメータ: なし.
//   戻り値: 現在使用中のオーディオデバイスのデバイス名.
VSCString VSGetAudioDeviceName();
```

4 Job プラグインの登録および Job プラグインスクリプトの修正

4.1 Job プラグインの登録

VOCALOID3 エディタの「ジョブ」 -> 「Job プラグインを管理」メニューを選択することで、Job プラグインスクリプトを管理するダイアログが開きます。

ダイアログには、現在登録されている Job プラグインスクリプトの一覧が表示されます。

ここから、新たな Job プラグインスクリプトの追加および登録済み Job プラグインスクリプトの削

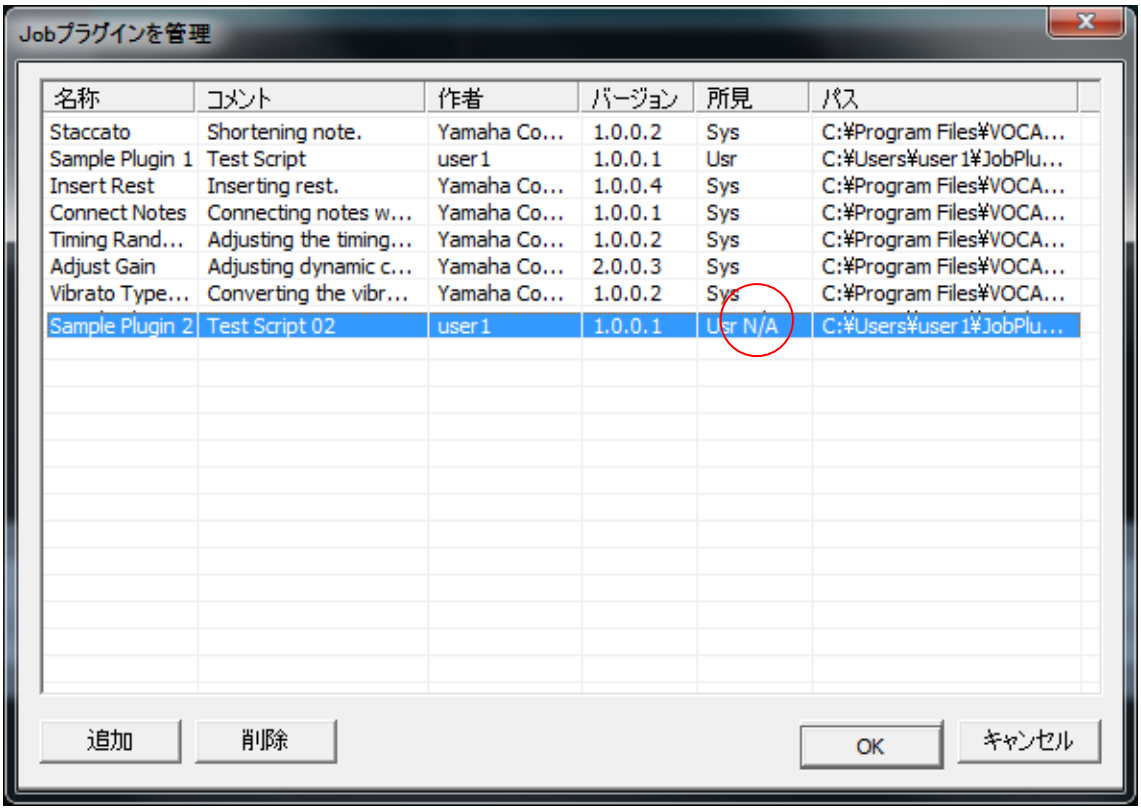
除を行えます。

ダイアログ上の「追加」ボタンを押すことで、ファイル選択ダイアログが表示され、ここで追加したい Job プラグインスクリプトファイルを選択することで追加することが出来ます。

ここで、Job プラグインスクリプトの追加処理が、VOCALOID3 エディタ内部でどの様に行われているかの概要を以下に説明します。

まず、指定された Job プラグインスクリプトファイルをロードし、その Job プラグインスクリプトの Job プラグインマニフェストを取得します。

そして、取得した Job プラグインマニフェストを検証し、正当な Job プラグインであると判定された場合に限り、VOCALOID3 エディタが内部で管理する Job プラグインライブラリへの登録を行います。このとき、Job プラグインスクリプトファイルの SHA-1 ハッシュコードを計算し、これも合わせて Job プラグインライブラリへ登録します。



Job プラグインスクリプト管理ダイアログ

4.2 Job プラグインスクリプトの修正

VOCALOID3 エディタは、登録された Job プラグインの情報を、Job プラグインライブラリという内部データベースによって管理しています。

そして、もし Job プラグインの登録時に算出した Job プラグインスクリプトファイルの SHA-1 ハッシュコードがその後の編集によって変化した場合、VOCALOID3 エディタはセキュリティー対策上、一律に「悪意の改ざん」とみなして実行できない状態にしています。

この場合、Job プラグインスクリプト管理ダイアログ上の「所見」フィールドに「N/A」というマークが付けられますので、これによってその Job プラグインが編集によって実行できない状態になったことが分かります。

そこで、Job プラグインを登録後にその Job プラグインスクリプトを修正した場合、まず Job プラグインスクリプト管理ダイアログからそれを削除し、あらためて登録を行うという手順が必要になります。

なお、Job プラグインの削除は、Job プラグインスクリプト管理ダイアログから目的の Job プラグインをリストから選択し、「削除」ボタンを押すことで行います。

5 Job プラグイン API 一覧

API 名	対応 API バージョン
VSDlgSetDialogTitle	3.0.0.1 ～
VSDlgAddField	3.0.0.1 ～
VSDlgGetIntValue	3.0.0.1 ～
VSDlgGetBoolValue	3.0.0.1 ～
VSDlgGetFloatValue	3.0.0.1 ～
VSDlgGetStringValue	3.0.0.1 ～
VSDlgDoModal	3.0.0.1 ～
VSMessageBox	3.0.0.1 ～
VSGetSequenceName	3.0.0.1 ～
VSGetSequencePath	3.0.0.1 ～
VSGetResolution	3.0.0.1 ～
VSGetPreMeasure	3.0.1.0 ～
VSGetPreMeasureInTick	3.0.1.0 ～
VSSeekToBeginTempo	3.0.0.1 ～
VSGetNextTempo	3.0.0.1 ～
VSSeekToBeginTimeSig	3.0.0.1 ～
VSGetNextTimeSig	3.0.0.1 ～
VSGetTempoAt	3.0.0.1 ～
VSGetTimeSigAt	3.0.0.1 ～
VSGetMusicalPart	3.0.0.1 ～
VSUpdateMusicalPart	3.0.0.1 ～
VSGetMusicalPartSinger	3.0.0.1 ～
VSSeekToBeginNote	3.0.0.1 ～
VSGetNextNote	3.0.0.1 ～

VSGetNextNoteEx	3.0.0.1 ～
VSUpdateNote	3.0.0.1 ～
VSUpdateNoteEx	3.0.0.1 ～
VSInsertNote	3.0.0.1 ～
VSInsertNoteEx	3.0.0.1 ～
VSRemoveNote	3.0.0.1 ～
VSGetControlAt	3.0.0.1 ～
VSUpdateControlAt	3.0.0.1 ～
VSSeekToBeginControl	3.0.0.1 ～
VSGetNextControl	3.0.0.1 ～
VSUpdateControl	3.0.0.1 ～
VSInsertControl	3.0.0.1 ～
VSRemoveControl	3.0.0.1 ～
VSGetDefaultControlValue	3.0.0.1 ～
VSSeekToBeginMonoWAVPart	3.0.0.1 ～
VSGetNextMonoWAVPart	3.0.0.1 ～
VSGetStereoWAVPart	3.0.0.1 ～
VSGetAudioDeviceName	3.0.1.0 ～

6 TIPS

6.1 ファイル入出力ライブラリ関数の使い方

3.1.5 節で述べたとおり、Job プラグインスクリプトの中で使用可能な文字コードは UTF-8 のみです。このため、日本語など ASCII 文字以外の文字を含むファイルパスを、Lua のファイル入出力ライブラリ関数へ渡す場合には注意が必要です。

具体的には、次のファイル入出力ライブラリ関数を使用する場合に注意が必要です。

- io.open()
- io.input()
- io.output()

✧ パラメータ入力ダイアログから取得したファイルパスを渡す場合

パラメータ入力ダイアログから VSDlgGetStringValue() によって取得した文字列は、あらかじめ UTF-8 でエンコードされています。したがって、この文字列は、このままファイルパスとしてファイル入出力ライブラリ関数へ渡すことができます。

以下に使用例を示します。

```
-- パラメータ入力ダイアログにフィールドを追加する.
field = {}
field.name       = "outFilePath"
field.caption    = "出力ファイルパス"
field.initialVal = ""
field.type       = 3
dlgStatus = VSDlgAddField(field)

-- パラメータ入力ダイアログから出力先ファイルパスを取得する.
dlgStatus = VSDlgDoModal()
dlgStatus, outFilePath = VSDlgGetStringValue("outFilePath")

-- ファイル出力.
outFile, errMsg = io.open(outFilePath, "w")
if (outFile) then
  outFile:write("Hello, World!!¥n")
end
```

✧ 直接ファイルパスを渡す場合

3.1.5 節で述べたとおり、Job プラグインスクリプトは UTF-8 でエンコードされたファイルとして保存しなければなりません。そして、UTF-8 でエンコードされている限り、日本語など ASCII 文字以外の文字を含むファイルパスを、直接リテラルとしてファイル入出力ライブラリ関数へ渡すことができます。

以下に使用例を示します。

```
function main(processParam, envParam)
  -- 指定したパスを出力ファイルへ割り当てる.
  io.output(envParam.tempDir .. "¥¥こんにちは.txt")

  -- ファイルへ出力する.
  io.write("Hello, World!!¥n")

  return 0
end
```

6.2 外部プログラムやコマンドを実行する方法

Job プラグインスクリプトから、外部プログラムおよび DOS コマンドを実行することができます。ここでは、その扱い方法を説明します。

本節の説明は、以下の Lua ライブラリ関数に適用されます。

- os.execute()

✧ 外部プログラムのパスの取り扱い方法

外部プログラムのパスとして使用可能な文字コードは UTF-8 のみです。また、その取り扱い方法は上記 5.1 節と同様です。

以下に使用例を示します。

```
-- メモ帳を起動する.
os.execute("notepad.exe")
```

※ os.execute() 関数は、起動した外部プログラムが終了するまで制御を戻しません。

✧ DOS コマンドの実行方法

DOS コマンドを実行する場合は、os.execute() 関数の引数として次のようにコマンド文字列を渡します。

```
CMD /C "DOS コマンド"
```

※ DOS コマンドをダブルクォーテーションで囲みます。

以下に使用例を示します。

```
-- 作成するディレクトリのパス.
newDir = "C:¥¥Temp¥¥myDir"

-- ディレクトリを作成する.
-- CMD /C "MKDIR "C:¥Temp¥myDir""
os.execute("CMD /C ¥"MKDIR ¥"" .. newDir .. "¥"¥"")
```

6.3 使用可能な Job プラグイン API を調べる方法

Job プラグイン API は、Lua の "_G" というグローバル変数の一覧を収めたテーブルに格納されています。そこで、次のような関数を Job プラグインスクリプトに定義することで、現在実行中の環境で使用可能な Job プラグイン API を調べることが出来ます。

```
-- Job プラグイン API が利用可能かどうかを調べる.
--   引数  apiName: 調べる Job プラグイン API 名
--   戻り値  利用可能: true  利用不可: false
function isAvailableAPI(apiName)
  local isAvailable = false
  local key, val
  for key, val in pairs(_G) do
    if ((key == apiName) and (type(val) == "function")) then
      -- apiName に一致する関数が存在する（利用可能）.
      isAvailable = true
      break
    end
  end

  return isAvailable
end
```

この関数を、たとえば次のように使用します。

```
function main(processParam, envParam)
  -- シーケンス名を取得する.
  sequenceName = ""
  if (isAvailableAPI("VSGetSequenceName")) then
    sequenceName = VSGetSequenceName()
  else
    -- シーケンス名取得 API を使用できないので終了する.
    VSMessageBox("VSGetSequenceName is NOT available.", 0)
    return 1
  end

  -- メインルーチン.
  status = doMain()
  return status
end
```