JAKLD-SICP の MIDI 機能利用マニュアル

新名庸生

2009 年 10 月 12 日

概要

JAKLD-SICP は、Java のライブラリを利用した MIDI 機能を提供しています。このマニュアルでは、JAKLD-SICP が備えている MIDI 関連のプリミティブ関数、音楽を記述するために用意された音楽記述言語 (MML の一種) とその利用方法などを解説します。

目次

1 基本的な手順

1.1 簡単な音楽を書いてみる

ここでは JAKLD-SICP で音楽を記述し、再生する手順を紹介します。手順の大まかな流れは以下のようになっています。

1. シンセサイザとシーケンサを定義し、2つをつなぐ
2. シーケンスを用意し、それにトラックをつくる
3. トラックに演奏データを書き込み、シーケンスをシーケンサにセットする
4. シーケンサを起動する
5. シンセサイザとシーケンサをクローズする

語句について簡単に説明します。シンセサイザは音を合成する装置、シーケンサは演奏データを再生する装置、シーケンスは演奏データです。トラックはシーケンスに演奏データを書き込むための領域の単位です。例えば、3種類の楽器からなる曲の場合、3つのトラックをシーケンスに作成し、各トラックにそれぞれの楽器のための演奏データを書き込む、という使い方をします。シンセサイザとシーケンサは、使い終わったらクローズして解放します。

では実際にコードを書いてみます。

```
;;1
(define synth (get-synthesizer)) ; シンセサイザの取得
(define seqr  (get-sequencer))   ; シーケンサの取得
(set-receiver seqr synth)        ; シーケンサとシンセサイザをつなぐ
;;2
(define seq (sequence 384))      ; シーケンスの生成
(define trk (create-track seq))  ; シーケンスにトラックをつくる
;;3
(set-mml trk "l8g4>c4<b>cd4cdeefe<a4>ddc4cc<b4ab>c2.")
                                 ; トラックに MML で書いた演奏データを書き込む
(set-sequence seqr seq)          ; シーケンスをシーケンサにセットする
;;4
(start-sequencer seqr)           ; シーケンサを走らせる（音楽の再生）
;;5
(close-device synth)             ; シンセサイザを解放する
(close-device seqr)              ; シーケンサを解放する
```

以上のコードを JAKLD-SICP で評価すると「大きな古時計」の一節が演奏されます。コードは前述の説明そのままという感じなので、分かりにくい箇所はないと思います。

少し補足をしますと、シーケンスを生成するときに関数 sequence に渡している引数の単位は PPQ(Pulse Per Quarter note) で、4分音符をいくつに分割するかを指定します。楽譜の音符は2分音符、4分音符、8分音符…のように音の長さが2分の1ずつになっていますので PPQ は2のべき乗であったほうがよいのですが、3連符などのように音を3分割する場合もありますからここでは 128×3 で 384 と指定しています。

set-mml は第 1 引数に演奏データを書き込むトラック、その後に 0 個以上の MML の文字列をとります。(この例では 1 つです。0 個の場合は何もおこりません。) MML の文法については後で詳しく解説します。なお、MML を簡単に再生するための関数 play-mml というものも定義してあります。これは上のコードを 1 つの関数にまとめたもので、0 個以上の MML の文字列を引数にとります。例えば、先ほどの MML は次のようにしても演奏できます。

```
(play-mml "l8g4>c4<b>cd4cdeefe<a4>ddc4cc<b4ab>c2.")
```

play-mml を使った場合、シンセサイザやシーケンスなどを直接扱うことはできませんし、トラックも 1 つだけですが、MML を手軽に鳴らすことができます。

MML を使えば、ここに出てくるプリミティブ関数だけを使って音楽の記述・再生ができますが、JAKLD-SICP では、MIDI で規定されているメッセージなどをベースにしたプリミティブ関数が定義されており、これらを使うことで自分好みの音楽記述言語をつくることができます。また、自分で作成したシーケンスを MIDI ファイルに変換する関数や既存の MIDI ファイルを JAKLD-SICP で再生するための関数などもプリミティブ関数として定義されています。プリミティブ関数についても後の章で解説します。

2 MML

2.1 MML とは

前章でも書いたように、JAKLD-SICP では MML という言語を使って音楽を記述することができます。MML は Music Macro Language の略で、比較的広く使われている音楽記述言語です。ただ、MML というのは言語のカテゴリーであり、明確な仕様があるわけではありません。詳細は各 MML 処理系によって異なっています。JAKLD-SICP で使用できる MML は、MML コンパイラ mml2mid で定義されている MML のサブセットに一部変更を加えたものとなっています。mml2mid の MML については MML 実習マニュアル [2] で詳説されています。以下、MML と書いた場合は JAKLD-SICP が提供している MML をさすことにします。

2.2 JAKLD-SICP で MML を使う方法

JAKLD-SICP で MML を使うための関数には set-mml と play-mml の 2 つがあります。

set-mml は第 1 引数にトラック、それ以降に 0 個以上の MML データ (MML の文法に沿った文字列) をとり、渡されたデータを左から順に解析しながらトラックに書き込んでいきます。渡す MML データが 0 個の場合は無意味な操作となります。複数個のデータを 1 つにまとめて渡しても、1 つのデータをばらして渡しても動作は同じです。例えば前章の、

```
(set-mml trk "l8g4>c4<b>cd4cdeefe<a4>ddc4cc<b4ab>c2.")
```

は、

```
(set-mml trk "l8g4>c4<b>c" "d4cdeefe<a4>ddc4cc<b4ab>c2.")
```

としても、

```
(set-mml trk "l8g4>c4<b>c" "d4cdeefe<a4>" "ddc4cc<b4ab>c2.")
```

としても同じです。長い MML データは区切りのいいところで分割して渡したほうが見やすくてよいと思います。

ただ、set-mml を用いて MML データを再生するには、事前にシンセサイザやシーケンサ、シーケンスを定義し、シーケンスにトラックを作成しなければならず、MML データをちょっと再生してみたい場合には不向きです。そういう場合には play-mml が便利です。play-mml は 0 個以上の MML データを引数にとります。再生に必要な定義を内部で行い、作成したトラックに set-mml で MML データを書き込んで再生し、終わるとデバイスをクローズします。全部内部でやってしまうので細かい操作はできませんが、単に MML を再生したい場合には便利です。

```
(play-mml "l8g4>c4<b>cd4cdeefe<a4>ddc4cc<b4ab>c2.")
```

次の節からは MML の文法について説明していきます。

2.3 基本構造

MML データはコマンドの列になっています。コマンドは、

コマンド名 引数, 引数, … , 引数

という形をしており、２つ目以降の引数はカンマの後に書きます。ほとんどのコマンドは０または１引数ですが、２つ以上の引数をとるコマンド (例えば音符コマンド) もあります。コマンドとコマンドの間にある、スペースや改行などの空白文字は無視されます。また、大文字と小文字は区別されます。

2.4 音

2.4.1 音の高さ

■音名　MML で音を出すには音符コマンドを使います。ド・レ・ミ・ファ・ソ・ラ・シに対応する音符コマンドはそれぞれ c・d・e・f・g・a・b です。

例

```
"cdefedc"
```

■オクターブ　音のオクターブを指定するには o コマンドを使います。次の o コマンドが現れるまで指定したオクターブで演奏します。o コマンドには引数として 0 から 8 の数字を１つ与えます。数字が大きいほど高いオクターブを指定します。デフォルトのオクターブは 4 になっています。

例

```
"o2 cdefgab o3 cdefgab o4 cc o3 bagfedc"
```

上の例では o コマンドの位置を明確にするため、o コマンドの前後にスペースを入れていますが、なくてもかまいません。

オクターブは相対的に指定することもできます。今より n だけオクターブを上げたい場合は `o+`n、下げたい場合は `o-`n というコマンドをかきます。また、１オクターブの変化は頻繁に起こるので、`o+1` と `o-1` にはそれぞれ >、< という略記が用意されています。

例

```
"o2 cdefgab > cdefgab > cc < bagfedc"
```

■派生音　半音あげた音を出したいときは + を、半音下げた音を出したいときは-をそれぞれ音名の後につけます。例えば、ファのシャープは f+ とかけます。

例

```
"cdef+gab->c"
```

音名の後の + や-は複数個かくことができます。+ と-を混ぜて書いても構いません。この場合、+ の数だけ半音あがり、-の数だけ半音下がります。例えば、a++-+-は a+ と同じ音になります。

■調性　+ や-などを頻繁につける音がある場合、+ や-などをつけた後の音を {} の中で宣言しておくことによって、その後の + や-を省略することができます。例えば、{cf+b--}cdefgab とかくと、f と b はそれぞれ f+、b--と解釈されます。c はそのまま c です。{c+}c-は c+-と同じです。{ } の内部は演奏されません。{cf+b--} と記述した場合、この宣言自体がコマンド名となります。{ } の中にスペースや改行文字などが含まれているとエラーとなるので注意してください。

また、{} の中の宣言を一時的に無効にしたい場合 (楽譜でいうナチュラル) は音名の後に*をつけます。*をつけた後にも + や-をつけることができます。例えば {f+}f*-は f-と同じです。

{} による宣言は何度でも出来ます。新しい宣言は過去の宣言を上書きしますが、過去の宣言に含まれている音名で、新しい宣言に含まれていないものがあればその影響は引き継がれます。例えば、{c+d-}cd{c}cd は c+d-cd-と同じです。

例

```
"{d+}>edede<b>d*c<a"
```

2.4.2　音の長さ

■音の長さの指定　音の長さは音名の後に正の整数で指定します。整数 n が表す長さは、全音符が表す長さの n 分の 1 です。ですから、4 分音符の長さは 4、8 分音符の長さは 8 で表せます。例えば、高さがファのシャープの 8 分音符は f+8 とかけます。

■省略時の音の長さ　音の長さの引数を省略した場合、デフォルト値が適用されます。デフォルト値の初期値は 4 ですが、l(エル) コマンドによって変更できます。l コマンドは、引数として渡された整数を、音の長さのデフォルト値として設定します。

例

```
"cdefgab>c l8 c<bagfedc"
```

■休符　休符を表すには r コマンドを使います。r コマンドも引数に長さを表す整数をとります。省略時には音符コマンドと同じデフォルト値が適用されます。

■付点音符 (休符)　一般的な楽譜では、右側に小さな点が打たれている音符や休符は、点がない場合の 1.5 倍の長さになります。MML では長さを表す整数に続けて.(ドット) を書くことでこれを表現します。例えば、高さがファの付点 4 分音符は f4. と書けます。長さのデフォルト値が 4 なら f. とも書けます。

また、点が 2 つ打たれている場合 (複付点) は長さが 1.75 倍になります。MML では長さを表す整数の後

に..(ドット２つ) と書くことで表現します。高さがファの複付点４分音符は f4.. と書けます。

例

```
"c16.d.."
```

■連音符　連音符はある音符の長さをいくつかに分割し、各分割に対して音の高さを設定します。MML では連音符のための記述法はありませんが、音の長さとして指定する整数を n 倍することでもとの長さの n 分の 1 を表現できます。例えば、４分音符を３分割した長さは 12 と書けます。

■長さの加算　楽譜では同じ高さの音符を弧線 (タイ) で結ぶことで１つの音を表現することがあります。その場合、音の長さは、タイで結ばれたすべての音符の長さの和になります。MML では音の長さを加算するのに^ (サーカムフレックス) を使って以下の様に表します。

長さ ^ 長さ

例えば、４分音符、８分音符、16 分音符を連結したものの長さは `4^8^16` と書けます。^ の右辺か左辺または両方を省略するとその時点でのデフォルトの長さで補完されます。例えば `l4f^^16` は `l4f4^4^16` と同じです。

■長さの減算　楽譜では装飾的な短い音を表すのに装飾音符という小さな音符が使われます。装飾音符を伴った音符は、通常よりも装飾音符の分だけ短く演奏される必要があります。MML ではこのような長さを-を用いて次のように表します。

長さ - 長さ

例えば、32 分音符の装飾を伴った４分音符の長さは `4-32` と書けます。^ と同様に、引数を省略した場合はデフォルトの長さで補完されますが、音名の直後の-は減算ではなくフラットとして解釈される点に注意してください。例えば `l4b-16` は `l4b4-16` という意味ではなく、デフォルトの長さを 4 に設定した後、シのフラットを 16 の長さ演奏するという意味になります。2 つ以上の-がある場合、右にあるものほど先に処理されます。例えば `4-8-32` はまず `8-32` が計算され、つぎにこの結果を 4 から減算するという意味になります。
^ と-は混ぜて使うこともできます。この場合も右にあるほど先に計算されます。

例

```
"f2-4^16"
```

■スタッカート　短く切るような演奏の仕方をスタッカートと言います。MML でスタッカートを表現するには q コマンドを使います。q コマンドは引数に正の整数をとり、本来の何パーセントの長さ音を出すかを設定します。(デフォルトは 100 パーセントです。) 例えば q50f4 の場合、ファの音が聞こえるのは４分音符の半分の長さになります。楽譜のスタッカートは q50 くらいで表現できると思います。

■レガート　異なる高さの音を途切れさせずに滑らかに演奏する方法をレガートと言います。MML では、q コマンドで引数を 100 より大きく指定することでレガートを表現できます。

■テンポ　MML で曲のテンポを設定するには t コマンドを使います。t コマンドには引数として 1 から 65535 までの整数を渡します。引数の単位は BPM で、１分間にいくつの４分音符が入るかを意味します。テンポの初期値は不定です (定義されていません)。現在のテンポに対して相対的に指定することもできます。今よりも n だけテンポを早くしたい場合には `t+`n、n だけ遅くしたい場合には `t-`n と書きます。

2.4.3　ベロシティー

ベロシティーとは音の強さのことです。音の強さは鍵盤を押し込む速さのことなのでこう呼ばれます。ベロシティーは 1 から 127 までの整数で表され、数値が大きいほど音は強くなります。

音符コマンドは第１引数に音の長さをとりましたが、実は２つ目の引数としてベロシティーをとることができます。例えば長さが 2、ベロシティーが 70 のドは c2,70 と書けます。長さの記述を省略する場合は c,70 などと書きます。

ベロシティーの指定を省略した場合はデフォルト値が適用されます。デフォルト値の初期値は 100 ですが、k コマンドで変更できます。例えばデフォルト値を 70 にしたければ、k70 と書きます。

ベロシティーのデフォルト値を絶対的に指定するのが k コマンドですが、現在のベロシティーに対する相対的な指定を行うコマンドもあります。左丸括弧「(」を使った「(コマンド」はデフォルト値を小さくする場合に、右丸括弧「)」を使った「) コマンド」はデフォルト値を大きくする場合に使います。例えば現在のデフォルト値が 100 であった場合、`(20` とするとデフォルト値は 80 になりますし、`)20` とするとデフォルト値は 120 になります。

また、これらのコマンドの引数は省略できます。省略した場合にはデフォルト値が適用されます。デフォルト値の初期値は 4 ですが、V コマンドで変更できます。例えば `V6c(d(e(f(g` と書くと各音は 6 ずつ弱く演奏されます。

2.4.4　和音

■ノートオンとノートオフ　和音について解説する前に、ノートオンとノートオフという２つの言葉を説明します。ノートオンとは音を出し始めるという意味で、ピアノでいえば鍵盤を押すことに当たります。ノートオフとは音を止めることで、ピアノでいえば鍵盤をはなすことに当たります。１つの音はノートオンとノートオフのペアで表されています。

■和音　通常の音符コマンドで例えば c4 と書いた場合、c の音をノートオンし、長さ 4 だけ演奏した後にノートオフするという意味になります。しかし、音符コマンドの長さにゼロを直接[*1] 指定することでノートオンのみを指示ことができます。この場合、最も直後のノートオフされる音と一緒にノートオフされることになります。例えば c0e0g2 と書いた場合、c0e0 では c と e のノートオンのみを行い、g がノートオフされるとき一緒にノートオフされます。結果、c、e、g は同時に鳴り始めて同時に鳴り止みます。MML ではこのようにゼロの長さを使うことで和音を表現します。

■アルペジオ　和音を少しずつずらして演奏する方法をアルペジオと言います。MML にもアルペジオを表現するための記法があります。例えば和音 `c0e0g2` を、c、e、g の順で 32 ずつ早く鳴らすよう演奏したい場合、`c32&e32&g2-16` と書きます。こう書くと、c をノートオンして 32 後、e がノートオンされ、その 32 後、g がノートオンされます。そしてその `2-16` 後に c、e、g が同時にノートオフされます。このように&を使う

[*1] `c4-4` のように間接的に 0 を指定するとエラーになります。

ことでもノートオンのみを指示することができ、次の音がノートオンされるまでの長さも指定できます。&の前の長さがデフォルトの長さの場合は通常の音符コマンドと同じように省略できます。例えば先ほどの例を l32c&e&g2-16 のように書くこともできます。

注意点ですが、例からもわかるように、アルペジオ全体の長さは最初の音が鳴り始めてからすべての音がノートオフされるまでの長さです。よって、最後に書く音の長さは、アルペジオ全体の長さから既に書かれている長さの和を引いたものでなければなりません。また、&の前の長さは 0 より大きくなければなりません。

2.5 声部

2.5.1 チャンネル

チャンネルとは、シンセサイザがもつ音発生装置です。シンセサイザは 16 個のチャンネルを持っています。チャンネルには、それが発生する音の音色を指定することができます。この音色をプログラムと呼びます。1 つのチャンネルに対し同時に複数のプログラムを指定することはできません。ピアノとヴァイオリンからなる曲を演奏したい場合、チャンネル 0 番をピアノに、チャンネル 1 番をヴァイオリンに対応させ、ピアノの演奏に関するメッセージをチャンネル 0 番に、ヴァイオリンの演奏に関するメッセージをチャンネル 1 番に送る、という使い方をします。チャンネルは、音色が同じなら、異なる高さの音を同時に出すことができます。MML では 16 個のチャンネルを 0 から 15 の番号で指定します (1 から 16 ではないので気を付けてください)。

■チャンネルの設定　今まではチャンネルを特に意識することなく MML データを書いてきましたが、実はこれらのデータはデフォルトである 0 番のチャンネルに送られていました。自分で MML データの送り先を設定するには C コマンドを使います。C コマンドは引数にチャンネル番号をとります。

例　チャンネルを 5 番に設定

```
"C5"
```

■音色　現在指定されているチャンネルの音色を設定するには@コマンドを使います。@コマンドの引数は各音色に対応づけられた 0 から 127 までの整数です。これをプログラムナンバーと呼びます。音色とプログラムナンバーの対応は GM(General MIDI) という規格で規定されています。Wikipedia の記事に対応表が載っています。

```
General MIDI 「Melodic sounds」 - Wikipedia
  http://ja.wikipedia.org/wiki/General_MIDI#Melodic_sounds
```

いくつか注意点があります。まず、Wikipedia での表もそうですが、プログラムナンバーを 1 から表記している表が多くあります。そういう表を用いる場合は、プログラムナンバーから 1 を引いた値を@の引数にします。次に、各音色にはそれぞれふさわしい音程があります。音色を変更する場合にはこの点にも注意してください。また、ここでいう音色とは打楽器以外の音色です。打楽器については後で説明します。プログラムナンバーのデフォルト値は不定です。

例　聖者の行進（最初ピアノ、途中からトランペット）

```
"t100 l8@0cefg2rcefg2r@56cefl4geced2"
```

■パンポット　左右どの位置から音が聞こえてくるかをパンポットと言います。MML でチャンネルのパンポットを指定するには p コマンドを使います。p コマンドの引数は 0 から 127 までの整数で、値が小さいほど

左から、大きいほど右から聞こえます。中央は 64 です。パンポットのデフォルト値は 64 です。

例　聖者の行進（最初左から、途中から右から）

```
"t100 l8p0 cefg2rcefg2r p127cefl4geced2"
```

■ボリューム　MML でチャンネルのボリュームを設定するには v コマンドを使います。引数は 0 から 127 の整数で、値が大きいほどボリュームが大きくなります。ボリュームのデフォルト値は不定です。

2.5.2　打楽器

先ほど紹介した GM という規格では、打楽器の音色は必ず 9 番のチャンネルから出力するよう定められています。MML もこれに従い、打楽器の MML データは 9 番のチャンネルに送らなければなりません。C コマンドを使って C9 とすることを忘れないでください。

打楽器以外の音色の指定は@コマンドで行うのでした。しかし、9 番のチャンネルにデータを送る場合は例外で、本来なら音の高さの指定となる記述が楽器の指定となります。(そして、@コマンドは無意味なコマンドとなります。) 音符コマンドは音の高さを指定してノートオンするのではなく、楽器を指定してノートオンするコマンドとなります。打楽器は出せる音の高さが 1 つだけなのでこれで十分なのです。

では、音の高さでどうやって楽器を指定するのでしょうか。

MIDI では各音の高さにノートナンバーと呼ばれる、0 から 127 の整数を低い音から順に割り振ってあります。1 増えるごとに半音ずつ高くなります。例えば、MML でオクターブ 4 の各音の番号は以下のようになっています。

…	c	d	e	f	g	a	b	…
…	60	62	64	65	67	69	71	…

また、各打楽器の音色にも GM によって 35 から 81 の番号が付けられています。

```
General MIDI 「Percussion notes」 - Wikipedia
  http://ja.wikipedia.org/wiki/General_MIDI#Percussion_notes
```

打楽器を指定する際はこの番号を o コマンドと音符コマンドを使って指定します。

例　バスドラム 1（36 番"o2c"）とスネアドラム 1（38 番"o2d"）

```
"C9o2cdcd8d8cdcd16d16d8"
```

ただ、楽器番号から対応するオクターブや音名を計算するのは面倒ですし、MML データを見ただけでは何番の音色を使っているのか分かりにくいです。また、上の例では 2 つの音色は同じオクターブでしたが、違った場合は毎回音名だけでなくオクターブまで記述しなければなりません。そこで、JAKLD-SICP では少しでも簡単に書けるよう、各番号にアルファベット 1 文字の名前をつけ、それらを使って MML データを記述できるような関数 rhythm を用意しています。この関数は第 1 引数に「名前 (アルファベット 1 文字) と番号のリスト」のリストをとり、第 2 引数以降にその名前を使った MML データをとります。名前の大文字と小文字は区別されます。返り値は、渡された MML データに含まれる名前を、o コマンドと音符コマンドの記述に置き換えたものです。

関数 rhythm を使った場合、先ほどの例は次のように書けます。

例　バスドラム１（36 番"o2c"）とスネアドラム１（38 番"o2d"）を使った演奏

```
"C9" (rhythm '((b 36) (s 38)) "bsbs8s8bsbs16s16s8")
```

まず、チャンネルの 9 番を使うことを宣言した後、rhythm の第 1 引数で、36 番に b という名前を、38 番に s という名前をつけています。そして第 2 引数で b と s を使った MML データを記述しています。

音色を表すアルファベットが、引数の MML データ中に現れる他のコマンド名とかぶらないように注意してください[*2]。例えば、(rhythm '((p 44)) "p60pppp8p8") と書いた場合、最初の p はパンポットのつもりでも、他の p と同じように o2g+ で置き換えられてしまいます。

2.6 その他

2.6.1 繰り返し

■繰り返しの基本的な記述　MML データの中に繰り返しがある場合、角括弧 ([ ]) を使うことで簡潔に記述できます。繰り返しを表すコマンドは次のような形をしています。

[コマンド列] 繰り返す回数 (正の整数)

例　cdefg を 24 回繰り返す

```
"[cdefg]24"
```

[コマンド列] 全体がコマンド名で、繰り返し回数が引数となっています。ですから、[コマンド列] と繰り返し回数は同じ文字列中に書く必要があります。繰り返し回数を Scheme で計算したい場合は、その数値を文字列に変換し、string-append などで [コマンド列] と結合することで実現できます。

例

```
(define x 3)
(play-mml (string-append "[cdefg]" (number->string (* x 8))))
```

{ } が内部にスペースや改行文字などを含んではならなかったのに対し、[ ] は内部の文字列を指定された回数展開するだけなので、中に含まれる文字に制限はありません。(もちろん、妥当な MML データである必要はあります。)

■繰り返しの入れ子　繰り返しを表す角括弧は入れ子にして記述することができます。

例　[cdcdefefefg]2 を [ ] の入れ子を使って表現

```
"[[cd]2 [ef]3 g]2"
```

■繰り返しからの抜けだし　[ ] の中のコマンド列中にコロン (:) を書くと、最後の繰り返しのとき、コロン以降のコマンドは実行されません。例えば、[gf:ed]4g と書くと、4 回目の繰り返しの際、ed は無視され代わりに g が実行されます。

[*2] 付録 A に MML コマンド・記号一覧を掲載しています。

2.6.2 コメント

MML データ中にコメントを書くことができます。コメントは/*と*/で囲みます。コメント部分は MML データを解析する前に取り除かれますので比較的自由な位置に書くことができます。例えば { } や [ ] の中や、コマンド名とその第 1 引数との間にも書けます。

例

```
"[gf:ed4,/*kome*/10]/*kome*/4g"
```

2.6.3 マクロについて

MML は Music Macro Language の略であると書きました。名前の通り一般的な MML にはマクロという機能があります。コマンド列に名前をつけ、その名前をつかって MML データを記述できるような機能です。例えば、[cd]8 に$a、[cdefgab>ab<]2 に$b という名前を付け、$a、$b を使って MML データを書いたり、そのデータにさらに名前をつける、などということができます。

しかし、JAKLD-SICP が提供している MML には、このマクロ機能が実装されていません。というのは、マクロでなされるような処理はわざわざ MML で書かなくとも、Scheme で書けばよいからです。

3 プリミティブ関数

JAKLD-SICP に実装されている MIDI 関連のプリミティブ関数を解説する前に、MIDI について簡単に触れておきます。

3.1 MIDI とは

MIDI(Musical Instrument Digital Interface) とは電子楽器同士や電子楽器とコンピュータの間で演奏情報をやり取りするための規格です。また、MIDI に従った演奏情報を MIDI データと呼びます。

MIDI における基本的な概念の多くは MML の解説の中で既に説明していますが、プリミティブ関数を扱う際に必要な「メッセージ」と「イベント」という言葉を以下で説明します。

■メッセージ　楽器や機器に対する指示をメッセージと呼びます。MIDI データはメッセージの集合です。メッセージには以下の 3 種類があります。

- ショートメッセージ — 最も基本的なメッセージ。ノートオン (音を出す)、ノートオフ (音を止める)、プログラムチェンジ (音色を変える)、など。
- システムエクスクルーシブメッセージ — MIDI 機器固有のメッセージ。各メーカが独自に定めているもの。
- メタメッセージ — 歌詞やテンポなどのメタ情報を格納するメッセージ。

■イベント　メッセージを楽器や機器に送ることをイベントと呼びます。メッセージと、それを送るタイミングを対にしたデータを MIDI イベントと呼びます。

3.2 プリミティブ関数

3.2.1 一覧

MIDI 関連のプリミティブ関数の一覧を掲載します。これらはデフォルトでインストールされています。set-mml、play-mml、rhythm 以外は、Java の javax.sound.midi パッケージで提供されている関数を使って実装してあります。Java の関数そのままのものが多いですが、多少手を加えているものもあります。手を加えたものは、簡単に扱えるようになっている反面、機能が制限されています。set-mml、play-mml、rhythm は Scheme で実装しています。

- (get-synthesizer)
  デフォルトの synthesizer を返す。
- (get-sequencer)
  デフォルトの sequencer を返す。
- (close-device *midi-device*)
  midi-device(synthesizer や sequencer) を close して解放し、ok を返す。
- (set-receiver *sequencer synthesizer*)
  sequencer を *synthesizer* につなぎ、ok を返す。
- (set-sequence *sequencer sequence*)
  sequence を *sequencer* にセットし、ok を返す。
- (start-sequencer *sequencer*)
  sequencer を再生する。再生が終わると done を返す。
- (sequence k)
  PPQ(4 分音符をいくつに分割するか) を k とする新たな sequence を生成して返す。
- (create-track *sequence*)
  sequence に新たな track をつくり、それを返す。
- (short-message)
  新たなショートメッセージを返す。
- (set-message *short-message symbol* $n1$ $n2$ $n3$)
  short-message にコマンド名 *symbol* とチャンネル $n1$, データ $n2$, $n3$ をセットして ok を返す。コマンド名としてとれるシンボルは、note-on、note-off、program-change、channel-pressure、pitch-bend、poly-pressure、control-change の 7 つ。
- (midi-event *midi-message* n)
  midi-message を時刻 n に実行するような midi-event を返す。
- (add-event *track midi-event*)
  midi-event を *track* に加え、ok を返す。
- (set-tempo-message n)
  テンポを BPM で n に設定するような midi-message を返す。BPM は 1 分間に 4 分音符がいくつ入るかを表す単位。テンポの設定にはメタメッセージを用いるが、BPM の指定が若干面倒になる。そこで、メタメッセージの役割をテンポ設定のみに限定することで BPM の指定が簡単にできるようにして

いる。

- (get-sequence *string*)
  string で指定されたパスの MIDI ファイルから sequence を抽出して返す。既存の MIDI ファイルを再生するには、この関数を使って抽出した sequence を sequencer で再生すればよい。
- (write-midi *sequence string*)
  sequence を、*string* で指定されたパスに MIDI ファイルとして書きだし、done を返す。例えば、seq という名のシーケンスを現在のディレクトリに foo.mid として保存したければ、

```
(write-midi seq "./foo.mid")
```

  とする。
- (set-mml *track string* ...)
  MML データ *string* ... を左から順に midi-event に変換し、*track* に加える。set-mml は内部に時刻 (初期値 0) を保持しており、音符コマンドや休符コマンドなど、時刻の経過を伴うコマンドを読むごとに、その内部時刻を進める。(四分音符は時間 384 に相当。) midi-event を生成する際は、この内部時刻を用いている。よって、

```
(set-mml track string1 string2)
```

  と

```
(set-mml track string1)
(set-mml track string2)
```

  は異なる操作なので注意。前者は string1 の解析によって経過した時刻を保持したまま string2 を解析するので、string1 が再生された後、string2 が再生されることになる。後者は string1 を解析した後、再び時刻 0 の状態から string2 を解析する。よって、string1 と string2 が同時に再生されることになる。なお、

```
(set-mml track (string-append string1 string2))
```

  は、前者と同じ操作。
- (play-mml *string* ...)
  内部で synthesizer、sequencer、sequence を定義し、sequence に track を 1 つ作成する。その track と *string* ... に set-mml を適用し、sequencer を再生する。再生が終わると synthesizer と sequencer を解放し、done を返す。単に MML を鳴らしてみたいときに便利。
- (rhythm *list string* ...)
  list は、「アルファベット 1 文字からなるシンボルと打楽器のノートナンバーのリスト」のリスト。例えば ((b 36) (s 38))。これで各ノートナンバーに名前を付けたことになる。*string* ... はこれらの名前を音符コマンドのように使った MML データ。rhythm はまず、*string* ... を 1 つの文字列につなげる。そして、それに含まれる名前を、o コマンドと音符コマンドを使った表現に置き換えた文字列を返す。

3.2.2 MML を使わずに音を鳴らす

MML を使わずに音を鳴らす例を以下に示します。

```
(define synth (get-synthesizer))
(define seqr  (get-sequencer))
(set-receiver seqr synth)

(define seq (sequence 384))
(define trk (create-track seq))

;;;;;;;MML を使う場合と異なる部分;;;;;;;
(define msg (short-message))
(define msg (set-tempo-message 80)) ; テンポを 80BPM に設定。
(add-event trk (midi-event msg 0))  ; このメッセージを時刻 0 で送る。

(define msg (short-message))
(set-message msg 'program-change 0 0 0)
                        ; チャンネル 0 番の program を 0 番（アコースティックピアノ）に指定。
(add-event trk (midi-event msg 0))   ; このメッセージを時刻 0 で送る。

(define msg (short-message))
(set-message msg 'note-on 0 60 100)
                        ; チャンネル 0 番のノートナンバー 60 をベロシティー 100 でノートオンする。
(add-event trk (midi-event msg 0))   ; このメッセージを時刻 0 で送る。

(define msg (short-message))
(set-message msg 'note-off 0 60 100)
                        ; チャンネル 0 番のノートナンバー 60 をベロシティー 100 でノートオフする。
(add-event trk (midi-event msg 384)) ; このメッセージを時刻 384 で送る。

(define msg (short-message))
(set-message msg 'control-change 0 10 0)
                                     ; チャンネル 0 番のパンポットを 0 に設定
(add-event trk (midi-event msg 192)) ; このメッセージを時刻 192 で送る。
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(set-sequence seqr seq)
(start-sequencer seqr)

(close-device synth)
(close-device seqr)
```

set-message を用いる際、最後の 2 つの引数 ($n2$、$n3$) にはコマンド名に応じた値を渡す必要があります。これも MIDI で定められています (Wikipedia「MIDI」3.1.1.1 チャンネルボイスメッセージ を参照)。ここでは代表的なものをいくつか説明します。

- program-change — プログラムの変更
  $n2$ にはプログラムナンバーを指定します。指定する値の範囲は MML の場合と一緒で 0 から 127 の整数です。$n3$ のデータは使われませんので任意の整数を入れます。上の例では 0 を入れています。

- note-on — ノートオン
  $n2$ にはノートナンバーを指定します。これも値の範囲は MML と同様です。$n3$ にはベロシティーを指定します。MML では 1 から 127 でしたが、ここでは 0 を指定することもできます。0 を指定した場合は、ノートオフと同じ効果になります。
- note-off — ノートオフ
  note-on の場合と同様です。ノートオフする際のベロシティーというのは分かりにくいかもしれません。ノートオンする際のベロシティーが鍵盤を押し込む速さであったように、ノートオフのベロシティーは鍵盤をはなす速さなのですが、実はこのパラメータに対応している MIDI 機器はあまりありません。なので note-off コマンドの効果は、note-on でベロシティーを 0 にした場合と実質同じです。
- control-change — 音量、音質などの制御
  $n2$ にはコントロールナンバー (どのパラメータを制御するか)、$n3$ は設定するパラメータ値です。コントロールナンバーは以下に一覧があります。

  Wikipedia - **コントロールチェンジ**

  http://ja.wikipedia.org/wiki/**コントロールチェンジ**

  これを見ると、ボリュームのコントロールナンバーは 7、パンポットのコントロールナンバーは 10 であることが分かります。

付録 A　MML コマンド・記号一覧

コマンド・記号	説明
>	オクターブを 1 上げる
<	オクターブを 1 下げる
)	デフォルトのベロシティーを相対的に大きくする
(	デフォルトのベロシティーを相対的に小さくする
@	プログラムの指定
+	半音上げる
-	半音下げる、または、音の長さの減算
^	音の長さの加算
*	ナチュラル
&	アルペジオの表現に使用
.	付点音符を表現
..	複付点音符を表現
{ }	調性
[ ]	繰り返し
:	繰り返しからの抜けだし
/* */	コメント
a	音符コマンド
b	音符コマンド
c	音符コマンド
C	チャンネルの指定
d	音符コマンド
e	音符コマンド
f	音符コマンド
g	音符コマンド
k	ベロシティーのデフォルト値の指定
l	音の長さのデフォルト値の指定
o	オクターブのデフォルト値の絶対指定
o+	オクターブのデフォルト値の相対指定
o-	オクターブのデフォルト値の相対指定
p	パンポットの指定
q	音を本来の長さの何パーセント鳴らすかを指定
r	休符コマンド
t	テンポのデフォルト値の絶対指定
t+	テンポのデフォルト値の相対指定
t-	テンポのデフォルト値の相対指定
v	ボリュームの指定
V	「(」や「)」のデフォルトの引数を指定

付録 B　判明している問題

JAKLD-SICP の MIDI 機能のうち、判明している問題点を記しておきます。

■play-mml を何度も実行すると音が鳴らなくなる　play-mml を使って何度も MML データを再生していると音が鳴らなくなる場合があります。こうなった場合は JAKLD-SICP を再起動してください。音が出るようになります。

■シンセサイザーを定義しなくても音が鳴る　これまでの説明では音を鳴らすにはシンセサイザーが必要であるように書いてきました。しかし実際は、シンセサイザーをつながなくともシーケンサを起動すれば音は出ます。デフォルトのシンセサイザーとしてハードウェアを指定している場合には、２つをつなぐ操作が必要なのかもしれませんが、PC のみで音を出す場合のシンセサイザーはソフトウェアなので必要ないのではないかと考えています。

参考文献

[1] 筑波大 久野靖先生の、MIDI と javax.sound.midi パッケージに関する講義資料
http://lecture.ecc.u-tokyo.ac.jp/ kuno/cp05/siryou/r10.pdf

[2] MML 実習マニュアル
http://tutorial.jp/music/mml/mmlman.pdf