# MML 実習マニュアル

第七版
2006 年 3 月 16 日（木）


## 目次

# 第 1 章　MML の基礎

## 1.1　MML とは何か

### 1.1.1　テキストによる音楽の記述

文字を並べることによって作られたデータは、「テキスト」(text) または「文字列」(string) と呼ばれます。人間は、テキストを作ることによって、さまざまなものを記述することができます。音楽の楽曲も、テキストを使って記述することのできるもののひとつです。

テキストを使って何かを記述するためには、何らかの言語を使う必要があります。音楽の楽曲をテキストで記述する場合は、普通、「音楽記述言語」(music description language) と呼ばれる専用の言語が使われます。

「音楽記述言語」というのは、ひとつの特定の言語の名前ではなくて、言語の種類の名前です。そして、音楽記述言語に分類できる言語は無数にあります。ABC、CMN、GUIDO、Lilypond、SMDL、MusicXML など、枚挙に遑がありません。

### 1.1.2　マクロ機能を持つ音楽記述言語

言語にはさまざまなものがあるわけですが、それらのうちには、「マクロ機能」(macro function) と呼ばれる機能を持っているものがあります。「マクロ機能」というのは、文字でできた記述に名前を付けておいて、その名前によってその記述を参照することができるという機能のことです。そして、名前によって参照することのできる記述は、「マクロ」(macro) と呼ばれます。

音楽記述言語のうちにも、マクロ機能を持っているものがあります。マクロ機能を持っている音楽記述言語は、「音楽マクロ言語」(music macro language)、略して「MML」と呼ばれます。そして、MML で書かれた文書は、「MML データ」(MML data) と呼ばれます。

MML は、ひとつの特定の言語の名前ではなくて、言語の種類の名前です。そして、MML に分類できる言語は無数にあります。ただし、MML のうちの多くは、自分を識別するための固有の名前を持っていません。そのような、自分に固有の名前を持っていない MML は、普通、その言語で書かれたデータを処理することのできるソフトの名前を冠して、「何々の MML」と呼ばれます。たとえば、SPICE というソフトによって処理することのできるデータを書くための MML は、「SPICE の MML」と呼ばれます。

ちなみに、この「MML 実習マニュアル」というチュートリアルは、mml2mid の MML、つまり mml2mid というソフトによって処理することのできるデータを書くための MML について解説することを目的として書かれたものです。

### 1.1.3　MIDI データ

楽曲の演奏情報を記録したデータは、「MIDI データ」(MIDI data) と呼ばれます。MIDI データは、「MIDI プレイヤー」と呼ばれるソフトを使うことによって、楽曲として再生することができます。

MIDI データの形式としては、普通、SMF(Standard MIDI File) と呼ばれる標準規格が使われます。MIDI データは、名前に `.mid` という拡張子が付いているファイルに格納されます。

MML で書かれた文書は楽曲を記述しているわけですから、それを MIDI データに変換することができます。MML の文書を MIDI データに変換するソフトは、「MML コンパイラ」(MML compiler) と呼ばれます。MML コンパイラには、SPICE、mml2mid、MUC、kfm、みかん、DualMusik など、さまざまなものがあります。

### 1.1.4　mml2mid

mml2mid は、門田暁人さん、藤井秀樹さん、黒田久泰さん、新出尚之さんによって開発された MML コンパイラです。これはフリーソフトですので、誰でも自由に利用することができます。また、C 言語のソースコードも公開されています。

mml2mid の公式サイトは、

mml2mid Home Page　`http://platz.jp/~mml2mid/`

です。ここに、mml2mid の本体や、支援ツールのリンク集などがあります。

## 1.2　mml2mid の使い方

### 1.2.1　MML データの入力

MML データというのはテキストですので、それを入力してファイルに保存したいときは、普通、テキストエディターを使います。

それでは、何らかのテキストエディターを使って、次の MML データを入力して、`cmajor.mml` という名前のファイルにそれを保存してください。

MML データの例　`cmajor.mml`

```
A cdefgab>c c<bagfedc
```

ちなみに、MML データを格納するファイルの名前には、このように `.mml` という拡張子を付けます。

### 1.2.2　MIDI データへの変換

mml2mid を起動するコマンドは、基本的には、

`mml2mid` パス名

と書きます。このコマンドをシェルに入力すると、パス名で指定されたファイルに格納されている MML データが処理されて、それを MIDI データに変換した結果がカレントディレクトリのファイルに出力されます。出力先のファイル名は、MML データのファイル名の拡張子を `.mid` に変更したものになります。

それでは、実際に試してみましょう。先ほど入力した MML データのファイルのあるディレクトリをカレントディレクトリにして、

```
mml2mid cmajor.mml
```

というコマンドをシェルに入力してください。すると、`cmajor.mml` に格納されている MML データが MIDI データに変換されて、その結果が `cmajor.mid` というファイルに出力されます。

ちなみに、`cmajor.mml` は、ハ長調の音階を記述したものですので、それを変換することによってできた `cmajor.mid` を再生すると、ハ長調の音階が聞こえてくるはずです。

## 1.3　MML データ

### 1.3.1　行の分類

末尾に改行 (newline) という文字がある文字列は、「行」(line) と呼ばれます。

MML データは、1 個以上の行から構成されます。MML データを構成する行は、次の 3 種類に分類されます。

- MML 行 (MML line)
- 定義行 (definition line)
- マクロ行 (macro line)

これらの行の種類は、先頭の 1 文字によって識別されます。先頭の文字が井桁 (number sign, `#`) ならば定義行、ドルマーク (dollar sign, `$`) ならばマクロ行、英字の大文字、1 から 9 までの数字、クエスチョンマーク (interrogation mark, `?`) のいずれかならば MML 行だと識別されます。ただし、行の先頭に空白の列がある場合は、空白ではない最初の文字が識別の対象となります。

ちなみに、先ほど入力していただいた、`cmajor.mml` という MML データは、1 個の行だけから構成されているわけですが、その行は、先頭が英字の大文字の `A` ですから、MML 行だということになります。

楽曲を MML で記述する場合、絶対に必要なのは MML 行だけです。定義行とマクロ行は、書く必要があるときだけ書けばいい行です。

### 1.3.2　MML 行

MML 行は、

トラック名 | 空白列 | コマンド列 | 改行

という構造を持つ行です。

トラック名というのは、「トラック」(track) と呼ばれるものの名前です。

トラックについては、このチュートリアルの、もっと先のところ（第 4.1 節）で説明する予定です。トラックについて説明するまでのあいだは、トラック名として A という名前を使っていくことにします。

「空白列」というところには、1 個以上の空白から構成される列を書きます。そして、その右側に「コマンド列」と呼ばれるものを書いて、最後に改行を書きます。

### 1.3.3 コマンド

MML では、楽曲の演奏者に対する指示をあらわす記述のことを「コマンド」(command) と呼びます。そして、0 個以上のコマンドを並べて書いたもののことを「コマンド列」(command sequence) と呼びます。

ひとつのコマンドは、

コマンド名 引数 , 引数 , ・・・

という構造になっています。2 個以上の引数を書く場合は、それらの引数をコンマ (comma, , ) で区切る必要があります。

コマンドは、「コマンド名」(command name) と「引数」(argument) から構成されます（引数を書かずにコマンド名だけを書く場合もあります)。コマンド名は、指示の種類をあらわす名前で、引数は、指示の具体的な内容をあらわす記述です。

基本的には、コマンド列の中に書かれたそれぞれのコマンドは、左から右へ向かって順番に実行されていきます。そして、MML データの中に書かれたそれぞれの MML 行は、上から下へ向かって順番に実行されていきます。

コマンドとコマンドとのあいだや、コマンド名と引数とのあいだや、引数とコンマとのあいだは、詰めて書いてもかまいませんし、空白やタブを挿入してもかまいません。また、それらの場所に縦棒 (vertical bar, | ) を挿入することも可能です。空白やタブや縦棒は、コマンドの意味に影響をまったく与えませんので、MML データを読みやすくするために使うことができます。縦棒は、たとえば小節の切れ目を示したいときなどに使うといいでしょう。

なお、MML にどのようなコマンドがあるのかということについては、次の章以降でくわしく説明していきたいと思います。ただし、このチュートリアルは、すべてのコマンドを網羅的に紹介しているわけではありません。このチュートリアルで紹介されていないコマンドについては、mml2mid に同梱されている文書を参照してください。

### 1.3.4 注釈

MML データは、MML コンパイラによって処理できるように書かないといけないわけですが、それだけではなくて、さらに、人間にとって理解しやすいように書くということも大切です。

人間にとって理解しやすい MML データを書く上で効果のある技法のひとつは、人間による理解を助けるための記述を MML データの中に書いておくことです。そのような記述は、「注釈」(comment) と呼ばれます。

注釈は、MML コンパイラが、「この部分は注釈だ」ということを認識できるように書かないといけません。なぜなら、MML コンパイラが、自分にとって意味のある記述として注釈を解釈しようとすると、エラーが発生したり、正しくない結果が得られたりするからです。

MML では、注釈を書くための方法として、2 通りのものがあります。

そのうちのひとつは、セミコロン ( ; ) を使うという方法です。セミコロンを書くと、そこから改行までのあいだが注釈だとみなされます。

注釈を書くための方法の二つ目は、/* という文字列と */ という文字列を使うという方法です。ひとつの文字列があるとするとき、その左側に /* を書いて、その右側に */ を書くと、その文字列は注釈だとみなされます。改行を含んでいる文字列も、この方法を使って注釈にすることができます。

MML データの例　comment.mml

```
A cdefgab>c ; ascending scale of C major
/* This comment
   contains
   newlines. */
A c<bagfedc ; descending scale of C major
```

# 第 2 章　音

## 2.1　音の高さ

### 2.1.1　音名

楽曲を構成するひとつひとつの音 (note) を MML で記述したいときは、「音符コマンド」(note command) と呼ばれるコマンドを書きます。

音符コマンドのコマンド名は、基本的には、a から g までの英字の小文字です。これらの文字は、音楽で使われる音名 (pitch name) に対応していて、その音名によって示される高さ (pitch) の音を演奏することを、演奏者に対して指示します。たとえば、c という文字は、ハの高さの音を演奏してください、という指示をあらわすコマンド名です。

音符コマンドは、音の長さや強さを引数で指定するのですが、それらの引数は省略することができますので、コマンド名だけでも、文法的に正しいコマンドになります。

次の MML データは、ハニホヘトイロという音階を記述したものです。

MML データの例　pitch.mml

```
A cdefgab
```

### 2.1.2　オクターブ

ハからロまでの音の高さの範囲は、「オクターブ」(octave) と呼ばれます。MML では、オクターブには、低いものから順番に 0 から 8 までの番号が与えられています。ちなみに、中央のハがあるオクターブの番号は、4 番です。

MML データの中でオクターブを指定したいときは、o というコマンドを使います。オクターブを指定するコマンドは、o というコマンド名の右側に、引数としてオクターブ番号を書きます。そうすると、引数で指定されたオクターブが設定されて、それ以降の音符コマンドは、設定されたオクターブの中の音を指示することになります。たとえば、o3 というコマンドを書くと、それ以降の音符コマンドは、3 番のオクターブの中の音を指示します。

なお、オクターブのデフォルトは 4 番です。

次の MML データは、オクターブの 2 番から 6 番までの範囲で音階を演奏する、というものです。

MML データの例　octave.mml

```
A o2 cdefgab o3 cdefgab o4 cdefgab o5 cdefgab o6 cc
A o5 bagfedc o4 bagfedc o3 bagfedc o2 bagfedc
```

オクターブは、現在のオクターブに対して相対的に指定することもできます。オクターブを相対的に指定したいときは、

- o+$n$　相対的に $n$ だけ上のオクターブを指定する。
- o-$n$　相対的に $n$ だけ下のオクターブを指定する。

というコマンドを書きます。たとえば、o+3 というコマンドは、現在のオクターブよりも 3 オクターブだけ高いオクターブを指定します。

楽曲の途中では、オクターブをひとつ上に上げたり、ひとつ下に下げたりすることが、頻繁に必要になります。そのため、o+1 と o-1 というコマンドには、それぞれ、次のような略記法が準備されています。

- >　相対的にひとつ上のオクターブを指定する。
- <　相対的にひとつ下のオクターブを指定する。

曲の途中で別のオクターブを指定する場合は、普通、o3 とか o5 というように絶対的に指定するのではなく、相対的な指定を使います。なぜなら、そうしておけば、曲全体のオクターブを上下に移動させることが簡単にできるからです。

MML データの例　octave2.mml

```
A o2 cdefgab > cdefgab > cdefgab > cdefgab > cc
A  < bagfedc < bagfedc < bagfedc < bagfedc
```

この MML データも、さっきと同様に、2 番から 6 番までの範囲で音階を演奏するわけですが、曲の途中には絶対的なオクターブの指定がありませんので、曲の最初に指定されているオクターブ番号を修正するだけで、簡単に曲全体のオクターブを変更することができます。

### 2.1.3 宣言

コマンドのうちで、それを実行することによって何らかの属性が設定されるものは、「宣言」(declaration) と呼ばれます。たとえば、o コマンドは、オクターブという属性を設定するコマンドですので、「宣言」と呼ぶことができます。それに対して音符コマンドは、属性を設定するものではありませんので、「宣言」と呼ぶことはできません。

### 2.1.4 派生音

MML では、幹音 (natural tone)、つまりピアノの鍵盤の白鍵に相当する音の高さは、1 文字の英字の小文字をコマンド名で記述されるわけですが、派生音 (derived tone)、つまりピアノの黒鍵に相当する音の高さは、英字の右側にプラス (plus, +) またはマイナス (minus, -) を書いたものをコマンド名によって記述されます。

嬰音 (sharp) を記述したいときは、その音名の右側にプラス (+) という文字を書きます。つまり、音名の右側にプラス (+) を書いたコマンド名は、音名で指定された音の高さよりも半音だけ高い音を指示する、ということです。たとえば、f+ というコマンド名は、嬰ヘを指示します。

MML データの例 gmajor.mml

```
A o3 gab>cdef+g gf+edc<bag
```

この MML データを演奏させると、ト長調の音階が聞こえるはずです。ト長調では、シの音は、ヘではなくて嬰ヘになりますので、f+ と書く必要があります。

MML データの例 dmajor.mml

```
A o4 def+gab>c+d dc+<bagf+ed
```

これは、ニ長調の音階です。ニ長調では、このようにミの音とシの音が半音高くなります。すなわち、ミの音は嬰ヘで、シの音は嬰ハです。

変音 (flat) を記述したいときは、その音名の右側にマイナス (-) という文字を書きます。つまり、音名の右側にマイナス (-) を書いたコマンド名は、音名で指定された音の高さよりも半音だけ低い音を指示する、ということです。たとえば、b- というコマンド名は、変ロを指示します。

MML データの例 fmajor.mml

```
A o3 fgab->cdef fedc<b-agf
```

この MML データを演奏させると、ヘ長調の音階が聞こえてきます。ヘ長調では、ファの音は、ロではなくて変ロになりますので、b- と書く必要があります。

MML データの例 bflatma.mml

```
A o3 b->cde-fgab- b-agfe-dc<b-
```

これは変ロ長調の音階です。変ロ長調では、このようにドの音とファの音が半音低くなります。すなわち、ドの音は変ロで、ファの音は変ホです。

重嬰音 (double sharp) は、音名の右側にプラスプラス (++) を書いたコマンド名によってあらわされ、重変音 (double flat) は、音名の右側にマイナスマイナス (--) を書いたコマンド名によってあらわされます。たとえば、重嬰ハを指示するコマンド名は c++ と書き、重変ホを指示するコマンド名は e-- と書きます。

楽譜で楽曲を記述する場合、ひとつの音符に嬰記号または変記号が臨時記号として付けられているとすると、その音符よりも右側にある同じ高さの音符は、同じ臨時記号が付いているとみなされます（ただし、次の小節に移ると無効になります）。しかし、MML にはそのような規則はありませんので、音名の右側に書かれたプラスやマイナスが、それよりも右側の記述に影響を与える、ということはありません。

### 2.1.5 調性

楽譜で楽曲を記述する場合、その楽曲が、ト長調、ホ短調、ヘ長調、ニ短調、というような調性 (tonality) のはっきりしたものならば、譜表の左のほうに調号 (key signature) を書いておくことによって、個々の音符の臨時記号を省略する、という方法が使われます。それと同じように、MML でも、調性を設定する宣言を書いておけば、個々の音に付けるプラスやマイナスを省略することができるようになっています。

MML で調性を設定したいときは、KJ または KI という宣言を書きます。KJ は長調 (major key) を設定する宣言で、KI は短調 (minor key) を設定する宣言です。K は key の頭文字で、J は maJor、I は mInor に由来する文字です。

調性を設定する宣言は、KJ または KI というコマンド名の右側に、引数として主音 (tonic) の音名を英字の小文字で書きます（主音が派生音の場合は、音名の右側にプラスまたはマイナスを書きます)。たとえば、ト長調は KJg、変ロ長調は KJb-、ホ短調は KIe、嬰ヘ短調は KIf+ という宣言によって設定することができます。

KI 宣言で短調を設定して、臨時記号を使わずに音階を作ると、それは自然短音階 (natural minor scale) になります。和声短音階 (harmonic minor scale) や旋律短音階 (melodic minor scale) を使う場合は、そのための臨時記号が必要になります。

なお、調性のデフォルトはハ長調またはイ短調です。

MML データの例 keysig.mml

```
A o3
A KJg  gab>cdefg gfedc<bag   ; G major
A KJf+ fgab>cdef fedc<bagf   ; F sharp major
A KJb- b>cdefgab bagfedc<b   ; B flat major
A KIe  efgab>cde edc<bagfe   ; E minor (natural minor scale)
A      efgab>cd+e ed+c<bagfe ; E minor (harmonic minor scale)
A      efgab>c+d+e edc<bagfe ; E minor (melodic minor scale)
```

### 2.1.6 ナチュラル

設定されている調性によって自動的に派生音になっている音を、本来の幹音に戻して演奏するという指示、つまり、楽譜ではナチュラル (natural) であらわされる指示は、MML では、音名の右側にアスタリスク (asterisk, *) を書いたコマンド名によって記述されます。たとえば、調性としてト長調が設定されているとすると、その設定のままでは f というコマンド名は嬰ヘを意味することになりますが、f* というコマンド名を書くことによって、への音を指示することができます。

MML データの例 natural.mml

```
A KJa- o3
A ab>cdefga agfedc<ba          ; A flat major
A >cd*e*fga*b*>c c<b*a*gfe*d*c ; C major
```

この MML データでは、変イ長調が調性として設定されていますので、変イ長調の音階にはプラスもマイナスも付ける必要はありません。しかし、この設定のままでハ長調の音階を記述するためには、レ、ミ、ラ、シ、のそれぞれを幹音に戻す必要があります。d*、e*、a*、b* というのは、そのためのコマンド名です。

## 2.2 音の長さ

### 2.2.1 音の長さをあらわす整数

MML では、音の長さ (note length) は、基本的には 1 個の整数によって記述されます。その整数というのは、1 小節の長さを 1 と考えて、音の長さを「$n$ 分の 1」という形の分数であらわしたときの分母です。たとえば、1 小節と同じ長さは、分数であらわすと 1 分の 1 ですから、その分母である 1 という整数で記述されることになります。同じように、1 小節の半分の長さは、2 分の 1 という分数の分母、つまり 2 と記述されます。言い換えれば、楽譜で使われている「$n$ 分音符」という音符があらわしている長さを、MML では $n$ という整数であらわす、ということです。

音符コマンドに、引数として音の長さを指定すると、そのコマンドは、指定された長さで音を演奏するという意味になります。たとえば、c8 という音符コマンドを書くことによって、ハの高さの音を、1 小節の 8 分の 1 の長さで演奏する、という指示を記述することができます。同じ

ように、高さが嬰イで、長さが 1 小節の 4 分の 1 であるような音は、a+4 と書きあらわすことができます。

MML データの例　length.mml

```
A o4 c1 f2a2 g4f4e4d4 e8f8g8a8g8f8g8a8
A d16f16g16e16d16f16g16e16d16e16f16g16f16e16f16g16
```

### 2.2.2　省略時の音の長さ

音符コマンドを書くとき、音の長さの引数は省略することができます。音の長さの引数を省略した場合、その音符コマンドは、そのときに省略時の長さとして設定されている長さで音を演奏するという意味だと解釈されます。

省略時の音の長さは、デフォルトでは 4 分音符の長さ、つまり 4 に設定されています。それ以外の長さを設定したいときは、l という宣言を使います。l 宣言は、引数として記述された長さを省略時の音の長さとして設定します。たとえば、l8 という宣言を実行することによって、省略時の音の長さとして 8 が設定されますので、その右側に書かれた c という音符コマンドは、8 の長さでハの音を演奏するという意味になります。

MML データの例　length2.mml

```
A o4 l1 c l2 fa l4 gfed l8 efgagfga l16 dfgedfgedefgfefg
```

この MML データは、さきほどの MML データを l 宣言を使って書き換えたものです。ですから、これをコンピュータに演奏させると、さきほどと同じ曲が聞こえるはずです。

### 2.2.3　音の休止

MML では、音の休止、つまり、音を出さずに時間だけを進行させるという指定は、r というコマンドによって記述されます。音の休止の長さは、r コマンドの引数として指定します。たとえば、r16 という記述は、1 小節の 16 分の 1 の長さの休止を意味しています。

MML データの例　rest.mml

```
A o4 c8r8c8d8e8r8e8f8 g8r8g8a8b8r4b8 >c1
```

音符コマンドと同じように、r コマンドも、長さの引数を省略することができます。長さの引数が省略された r コマンドは、l 宣言で設定されている省略時の長さの休止だと解釈されます。

MML データの例　rest2.mml

```
A o4 l4 cder efgr gabr >c1
```

### 2.2.4　付点音符

付点音符 (dotted note) と付点休符、つまり右側に小さな黒丸が書かれている音符と休符は、その長さが、本来の長さの 2 分の 1 だけ長くなります。MML の場合、付点音符（休符）の長さは、本来の長さをあらわす整数の右側にドット (dot, . ) を書くことによって記述します。たとえば、付点 4 分音符（休符）の長さは 4. 、付点 8 分音符（休符）の長さは 8. と書きます。

MML データの例　dotted.mml

```
A o4 c4.d8e8f8g4 f4.g8a8b8>c4
```

ちなみに、複付点音符（休符）の長さは、本来の長さをあらわす整数の右側にドットドット ( . . ) を書くことによって記述することができます。

### 2.2.5　連音符

ひとつの長さをいくつかの部分に等間隔に分割して演奏する、ということを楽譜で記述するときには、「連音符」(tuplet) と呼ばれる音符が使われます。

MML には、連音符に相当するものを記述するための特別なコマンドはありません。しかし、連音符を構成するそれぞれの音の長さとして適切な長さを指定することによって、連音符に相当するものを記述することができます。

全体の長さが $n$ であるような $m$ 連符を構成するそれぞれの音の長さを求めたいときは、$n \times m$ という掛け算をします（本当は割り算なのですが、何分の 1 という形の分数の分母だけを考えていますので、掛け算になるわけです）。たとえば、全体の長さが 8 であるような 3 連符を構成す

るそれぞれの音の長さは、8 × 3 で、24 になります。同じように、全体の長さが 4 であるような 5 連符を構成するそれぞれの音の長さは、4 × 5 で、20 になります。

MML データの例　tuplet.mml

```
A o4 c4 e4 l12gfe a4 d4 f4 l20efgfe c4
```

### 2.2.6　タイ

楽譜を使って音楽を記述する場合、同じ高さの音をあらわす 2 個以上の音符を連結することによって 1 個の音をあらわす、ということがしばしばあります。その場合、それらの音符があらわしているのが 1 個の音だということを示すために、それらの音符をつなぐ弧線を書きます。その場合に使われる弧線は、「タイ」(tie) と呼ばれます（楽譜で使われる弧線としては、「タイ」のほかに、「スラー」(slur) というのもあります。スラーは、高さの異なる 2 個以上の音をなめらかに演奏する、ということを意味する弧線です)。

タイによってあらわされている音を MML で記述したいときは、音の長さの加算をあらわす式を書きます。

MML では、音の長さを、加算や減算という計算をあらわす式の形で記述することができるようになっています。

MML では、加算をあらわす演算子として、サーカムフレックス (circumflex, ^ ) という文字を使います。二つの長さを加算する形で音の長さを記述したいときは、

長さ ^ 長さ

というように、サーカムフレックスの左右に音の長さの記述を書いた形の式を書きます。たとえば、2^1 という式は、2 の長さと 1 の長さを加算することによって得られる長さをあらわしています。

サーカムフレックスの左側または右側の長さを省略すると、省略時の長さがそこに指定されていると解釈されます。たとえば、省略時の長さとして 4 が設定されているとすると、^1 という式は、4^1 という式と同じ意味になります。

MML データの例　tie.mml

```
A o4 l4
A efgaa1 agfee1
A efga^1 agfe^1
```

### 2.2.7　装飾音符

楽譜では、装飾的な短い音をあらわしたい場合、「装飾音符」(grace note) と呼ばれる、通常の音符よりも小さな音符が使われます。通常の音符の前後に装飾音符が付加されている場合、その本体の音符の音は、装飾音符の長さだけ短く演奏される必要があります。

本来の長さよりも装飾音符の長さだけ短い長さを MML で記述したい、というときは、長さを減算する式の形で書くというのが、もっとも素直な書き方です。

MML では、減算をあらわす演算子として、マイナス (minus, - ) という文字を使います。音の長さを、ひとつの長さからひとつの長さを減算する形で記述したいときは、

長さ - 長さ

という形の式を書きます。そうすると、左辺の長さから右辺の長さを減算するという意味になります。たとえば、2-8 という式は、2 の長さから 8 の長さを減算することによって得られる長さをあらわしています。

長さを減算する式を使えば、装飾音符が付加された音符は、簡単に記述することができます。たとえば、装飾音符として 32 分音符が直前に付加された 4 分音符は、

```
e32 c4-32
```

というような記述によってあらわすことができます。

ひとつの式の中に演算子を 2 個以上書いた場合、それらの演算子は、右にあるものほど左右の長さと強く結合します。ですから、たとえば、

```
2-4^16
```

という式は、4 と 16 とを加算した長さを 2 から減算するという意味になります。

MML データの例 grace.mml

```
A o4 c2 e16 g2-16 b2 g16a16 e2-8 a2 l24ded c2^1-8
```

### 2.2.8 スタッカート

音符で指定されている長さよりも音を短くすることによって、次の音とのあいだのギャップを広くすることを、「スタッカート」(staccato) で演奏すると言います。

MML でスタッカートな演奏を記述したいときは、 q という宣言を使います。 q 宣言には、引数としてギャップの長さ（音符コマンドが指示する長さから、実際に演奏する音の長さを減算した長さ）を記述します。ただし、 q 宣言の引数として記述する音の長さは、1 小節の何分の 1 という分数の分母ではなくて、「ステップ」(step) と呼ばれる長さの単位を何倍するかということを示す整数です。

1 ステップは、1 小節の 192 分の 1 の長さです。ですから、2 分音符は 96 ステップ、4 分音符は 48 ステップ、8 分音符は 24 ステップということになります。

たとえば、 q12 という宣言で、ギャップの長さとして 12 ステップを設定したとしましょう。このとき、 c4d4 という二つのコマンドでハの音とニの音を演奏すると、どちらの音も、全体の長さは 48 ステップですが、実際の長さは 36 ステップになります。ですから、ハの音とニの音とのあいだに、12 ステップの長さのギャップができることになります。

音符コマンドが指示している音の長さが、 q 宣言で設定されているギャップの長さと同じか、またはそれよりも短い場合、音は、まったく演奏されなくなってしまいます。たとえば、 q24 という宣言で、ギャップの長さとして 24 ステップを設定しているとすると、 c8 や d16 という音符コマンドは、まったく音を出さなくなります。

q 宣言は、音符コマンドが音を出さなくなるという事態を避けるために、2 個目の引数を記述することができるように作られています。 q 宣言の 2 個目の引数は、音符コマンドが指示している長さがギャップの長さと同じかまたはそれよりも短い場合に演奏する音の長さです。

つまり、 q 宣言の 2 個目の引数として 1 ステップ以上の長さを記述しておけば、どんなに短い音を指示する音符コマンドでも、かならず音を出すことになるわけです。たとえば、 q24,1 という宣言で、実際に演奏する音の長さを設定したとすると、24 ステップ以下の長さの音を指示する音符コマンドも、かならず 1 ステップの長さの音を演奏することになります。

MML データの例 stacca.mml

```
A o4 l8
A       c4.defg4 f4.gab>c4<
A q12   c4.defg4 f4.gab>c4<
A q24,1 c4.defg4 f4.gab>c4<
A q48,1 c4.defg4 f4.gab>c4<
A q72,1 c4.defg4 f4.gab>c4<
```

### 2.2.9 レガート

スタッカートとは逆に、音と音とのあいだにギャップを入れないでなめらかに演奏することを、「レガート」(legato) で演奏すると言います。楽譜では、レガートは、「スラー」(slur) と呼ばれる弧線によってあらわされます。

MML でレガートな演奏を記述したいときは、スタッカートの場合と同じように、 q 宣言を使います。

q 宣言の引数として記述するステップ数の左側には、マイナス (minus, - という文字を書くことができます。この場合のマイナスは、ギャップの長さをマイナスにするということを意味します。マイナスの長さのギャップというのは、次の音が始まってから前の音を止めるということですから、音と音とがなめらかにつながることになります。

MML データの例 legato.mml

```
A o4 l8
A      c4.defg4 f4.gab>c4<
A q-12 c4.defg4 f4.gab>c4<
```

### 2.2.10 テンポ

MML では、音の長さを、1 小節を 1 としたときの分数の分母か、またはステップを単位とする整数で指定するわけですが、それらはいずれも相対的な長さの指定です。音の絶対的な長さを

決定するためには、楽曲を演奏する速さ、すなわちテンポ (tempo) を設定する必要があります。

MML では、テンポは、t という宣言を使うことによって設定することができます。

t 宣言には、1 から 65535 までの範囲の整数を引数として書く必要があります。この整数が意味しているのは、設定されたテンポで演奏した場合に、1 分間に演奏される 4 分音符が何個になるか、ということです。つまり、4 分音符を基準とするメトロノーム記号 (metronome mark) の中に書かれる整数と同じ意味です。たとえば、t120 という宣言を実行したとすると、1 分間に 120 個の 4 分音符を演奏するテンポが設定されることになります。

なお、MML は、t 宣言を実行していない状態でのテンポを定義していませんので、その場合のテンポは不定になります。

MML データの例 tempo.mml

```
A o4 l8
A t120 c4.defg4 f4.gab>c4<
A t60  c4.defg4 f4.gab>c4<
A t240 c4.defg4 f4.gab>c4<
```

テンポは、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。テンポを相対的に指定したいときは、

- t+$n$ 相対的に $n$ だけテンポを速くする。
- t-$n$ 相対的に $n$ だけテンポを遅くする。

という宣言を書きます。たとえば、t+7 という宣言は、テンポを 7 だけ速くします。

## 2.3 ベロシティー

### 2.3.1 ベロシティーとは何か

電子楽器と電子楽器とのあいだで演奏情報を交換するための標準規格は、MIDI(Musical Instrument Digital Interface) と呼ばれます。

MIDI では、音の強さのことを「ベロシティー」(velocity) と呼びます。

MIDI で表現された個々の音のデータは、かならず、そのベロシティーを示す数値を含んでいます。ベロシティーを示す数値というのは、1 から 127 までの整数で、数値が大きいほど音が強いことを意味します。

### 2.3.2 ベロシティーの指定を含む音符コマンド

音符コマンドは、1 個目の引数として、それが指示する音の長さを記述することができるわけですが、さらに 2 個目の引数として、それが指示する音のベロシティーを記述することもできるようになっています。たとえば、c8,70 という音符コマンドを書くことによって、長さが 8 でベロシティーが 70 であるようなハの音を指示することができます。

音符コマンドで、ベロシティーを指定したいけれども長さは省略したい、という場合は、コマンド名の右側にコンマとベロシティーを書きます。たとえば、d,60 というコマンドを書くことによって、ベロシティーが 60 で、省略時の長さを持つニの音を指示することができます。

MML データの例 velo.mml

```
A o4 l8
A c4.defg4 f4.gab>c4<
A c4.,100 d,90 e,80 f,70 g4,60 f4.,50 g,40 a,30 b,20 > c4,10
```

### 2.3.3 省略時のベロシティー

音符コマンドを書くとき、ベロシティーの引数は省略することができます。ベロシティーの引数を省略した場合、その音符コマンドは、そのときに省略時のベロシティーとして設定されているベロシティーで音を演奏するという意味だと解釈されます。

省略時のベロシティーは、デフォルトでは 100 に設定されています。それ以外のベロシティーを設定したいときは、k という宣言を使います。k 宣言は、引数で指定されたベロシティーを省略時のベロシティーとして設定します。たとえば、k80 という宣言を書くことによって、省略時のベロシティーとして 80 が設定されますので、その右側に書かれた c という音符コマンドは、80 のベロシティーでハの音を演奏するという意味になります。

MML データの例 defvelo.mml

```
A o4 l8
A       c4.defg4 f4.gab>c4<
A k40   c4.defg4 f4.gab>c4<
A k127 c4.defg4 f4.gab>c4<
```

#### 2.3.4 省略時のベロシティーの相対的な指定

k 宣言は、省略時のベロシティーを、現在の設定とは無関係に設定するわけですが、現在の設定に対して相対的に省略時のベロシティーを設定する宣言もあります。それは、「ベロシティー宣言」(velocity declaration) と呼ばれる宣言です。ベロシティー宣言には、左丸括弧 (left parenthesis, ( ) と右丸括弧 (right parenthesis, ) ) という二つのものがあります。

左丸括弧 ( ( ) は、現在の省略時のベロシティーから、引数で指定された整数を減算した結果を、省略時のベロシティーとして設定する宣言です。たとえば、省略時のベロシティーとして 60 が設定されているときに (40 という宣言を実行すると、省略時のベロシティーは 20 に減少します。

右丸括弧 ( ) ) は、現在の省略時のベロシティーに、引数で指定された整数を加算した結果を、省略時のベロシティーとして設定する宣言です。たとえば、省略時のベロシティーとして 60 が設定されているときに )40 という宣言を実行すると、省略時のベロシティーは 100 に増加します。

MML データの例 relvelo.mml

```
A o4 l8
A c4.d (20 efg4 (20 f4.g (20 ab>c4<
A c4.d )20 efg4 )20 f4.g )20 ab>c4<
```

#### 2.3.5 ベロシティーの省略時の増減値

ベロシティー宣言を書くとき、その引数は省略することができます。引数を省略した場合、そのベロシティー宣言は、そのときにベロシティーの省略時の増減値として設定されている整数だけ、省略時のベロシティーを減少させたり増加させたりするという意味だと解釈されます。

ベロシティーの省略時の増減値は、デフォルトでは 4 に設定されています。それ以外の増減値を設定したいときは、大文字の V という宣言を使います。V 宣言は、引数で指定された整数をベロシティーの省略時の増減値として設定します。たとえば、V20 という宣言を実行することによって、ベロシティーの省略時の増減値として 20 が設定されますので、その右側に書かれた ( というベロシティー宣言は、省略時のベロシティーを 20 だけ減少させるという意味になります。

MML データの例 defrel.mml

```
A o4 l8 V10
A c4.(d(e(f(g4 (f4.(g(a(b>(c4<
A c4.)d)e)f)g4 )f4.)g)a)b>)c4<
```

### 2.4 和音

#### 2.4.1 ノートオンとノートオフ

MIDI に関する文献では、音の表現に関連して、「ノートオン」と「ノートオフ」という言葉がしばしば使われます。

「ノートオン」(note on) というのは、音を出し始めるという意味で、「ノートオフ」(note off) というのは、鳴っている音を止めるという意味です。

電子楽器から電子楽器へ演奏情報が送られる場合、ひとつの音は、ノートオンとノートオフというペアになったメッセージによってあらわされます。

#### 2.4.2 ノートオンだけを意味する音符コマンド

MML の音符コマンドは、基本的にはノートオンとノートオフのペアを意味しているわけですが、ノートオンだけを意味する音符コマンドを書くことも可能です。

音符コマンドの 1 個目の引数、つまり音の長さをあらわす整数としてゼロを書くと、その音符コマンドは、ノートオンだけを意味することになります。そして、その音符コマンドの次のコマンドは、その音符コマンドと同時に実行されます。長さがゼロの音符コマンドによってノートオンされた音は、別の音がノートオフしたときに、それと同時にノートオフされます。

たとえば、c0e4 というコマンド列を実行すると、ハの音のノートオンとホの音のノートオンが同時に実行されて、ホの音がノートオフするときに、それと同時にハの音もノートオフされます。つまり、ハの音とホの音とが同時に鳴り始めて同時に止まるわけです。

ですから、長さがゼロの音符コマンドを使うことによって、和音を記述することができます。和音を構成するそれぞれの音を、長さがゼロの音符コマンドで記述していって、最後に書く音符コマンドだけにゼロ以外の長さを書けばいいわけです。たとえば、c0e0g4 というコマンド列は、ハとホとトから構成される和音を 4 の長さで演奏するという意味になります。

MML データの例　chord.mml

```
A o4 l2
A c0e0g c0e0g c0e0g c0e0g
A <a0>c0e <a0>c0e <a0>c0e <a0>c0e
A d0f0a d0f0a d0f0a d0f0a
A <g0b0>d0f <g0b0>d0f <g0b0>d0f <g0b0>d0f
A c0e0g c0e0g c0e0g1
```

### 2.4.3　アルペジオ

ノートオフを実行しないでノートオンだけを実行するための方法としては、長さがゼロの音符コマンドを書くという方法のほかに、アンパサンド (ampersand, &) というコマンドを使うという方法もあります。

アンパサンドコマンドの左側に音符コマンドを書くと、その音符コマンドは、ノートオフを実行しなくなります。そして、そのアンパサンドコマンドの右側のコマンドは、左側の音符コマンドの引数で指定された長さの時間が経過したのちに実行されます。アンパサンドコマンドの左側の音符コマンドでノートオンされた音は、別の音がノートオフされるときに、それと同時にノートオフされます。

たとえば、c8&e4 というコマンド列を実行すると、まずハの音が鳴り始めます。そして、8 の時間が経過したのちにホの音が鳴り始めるのですが、その時点ではまだハの音は鳴りつづけています。そして、ホの音が鳴り始めてから 4 の時間が経過したのちに、ハの音とホの音とが同時に止まります。

ですから、アンパサンドコマンドを使うことによって、和音をアルペジオ (arpeggio) で演奏するという指示を記述することができます。たとえば、

```
c32&e32&g4-16
```

というコマンド列は、ハとホとトから構成される和音を、低い音から高い音へ向かってアルペジオで演奏する、という指示をあらわしています。

なお、アルペジオを記述する場合には、和音全体の長さが伸びてしまうことを防ぐために、最後の音の長さとして、和音全体の長さから、和音が鳴り始めてから自分が鳴り始めるまでの時間を減算したものを書く必要がある、という点に注意が必要です。

MML データの例　arpeggio.mml

```
A o4 l1
A c4&e4&g1-2 c4&e4&g1-2 <a8&>c8&e1-4 <a8&>c8&e1-4
A d16&f16&a1-8 d16&f16&a1-8
A <g32&b32&>d32&f1-16. <g32&b32&>d32&f1-16. c64&e64&g1^1-32
A g4&e4&c1-2 g4&e4&c1-2 e8&c8&<a1-4> e8&c8&<a1-4>
A a16&f16&d1-8 a16&f16&d1-8
A f32&d32&<b32&g1-16.> f32&d32&<b32&g1-16.> g64&e64&c1^1-32
```

# 第 3 章　繰り返し、マクロ、定義行

## 3.1　繰り返し

### 3.1.1　繰り返しの基礎

楽曲の中にはしばしば、同一の演奏を何回か繰り返すという部分が含まれていることがあります。MML で楽曲を記述するとき、繰り返しは、同一のコマンド列を何回も書くことによって記述することもできますが、角括弧 (bracket, [] ) というコマンドを使って記述するということもできて、それを使うことによって、MML データを簡潔なものにすることができます。

角括弧を使ったコマンドは、

[ コマンド列 ] プラスの整数

と書きます。このように、コマンド列を角括弧で囲んで、その右側にプラスの整数を書くと、角括弧で囲まれたコマンド列の実行が、右側の整数で指定された回数だけ繰り返されます。たとえば、

```
[cdefg]24
```

というコマンドは、cdefg というコマンド列の実行を 24 回だけ繰り返すという意味になります。

MML データの例　repeat.mml

```
A o4 l8
A [ c4.defg4 f4.gab>c4< ]4
```

なお、角括弧コマンドを書くとき、繰り返しの回数の引数を省略すると、2 回だけ繰り返すという意味だと解釈されます。

### 3.1.2　繰り返しの入れ子

角括弧コマンドは、入れ子にすることができます。つまり、角括弧の組の中に角括弧の組を書くことができて、そうすることによって、多重の繰り返しを記述することができるわけです。

MML データの例　nested.mml

```
A o4 l16
A [ [cd]8 [cdefgfed]2 ]4 c1
```

### 3.1.3　繰り返しの強制的な終了

楽曲に含まれる繰り返しは、しばしば、その最後の回だけ、末尾の部分がそれまでとは異なるということがあります。

最後の回だけ末尾が異なる繰り返しを MML で記述したいときは、コロン (colon, : ) というコマンドを使います。角括弧コマンドの中にコロンを書くておくと、コロンは、繰り返しの最後の回に自分が実行されたときに、その繰り返しを強制的に終了させます。ですから、角括弧コマンドの右側に、末尾の部分の変化形を書いておけば、繰り返しの最終回にはその変化形が演奏されることになります。

なお、角括弧コマンドが入れ子になっている場合、コロンが終了させるのは、自分を含んでいるもっとも内側の角括弧コマンドだけです。

MML データの例　break.mml

```
A o4 l8
A [ c4.defg4 f4.ga:b>c4< ]4 gf4 e4.dc2
```

## 3.2　マクロ

### 3.2.1　マクロの基礎

MML というのは、「マクロ機能」(macro function) と呼ばれる機能を持っている音楽記述言語のことです。「マクロ機能」というのは、コマンド列に名前を付けておいて、その名前によってそのコマンド列を参照することができるという機能のことです。そして、名前によって参照することのできるコマンド列は、「マクロ」(macro) と呼ばれます。

### 3.2.2　マクロ名

コマンド列に付けられた名前は、「マクロ名」(macro name) と呼ばれます。マクロ名は、1 文字の数字の右側に 1 文字の英字の小文字を並べることによって作られます。たとえば、4m、7x、8f、0a などをマクロ名として使うことができます。

また、1 文字の英字の小文字だけをマクロ名として使うことも可能です。ただし、英字だけのマクロ名は、ゼロの右側にその英字を並べたマクロ名によって参照されるマクロの別名だとみなされます。たとえば、a というマクロ名は、0a という名前によって参照されるマクロの別名とみなされることになります。

### 3.2.3　マクロの定義

コマンド列に名前を付けることを、マクロを「定義する」(define) と言います。

マクロを定義したいときは、「マクロ行」(macro line) と呼ばれる行を書きます。マクロ行というのは、

| $ | マクロ名 | コマンド列 | 改行 |
|---|---|---|---|

という構文を持つ文字列のことです。マクロ行を MML データの中に書いておくと、その中のコマンド列に対して、ドルマーク (dollar sign, `$`) の右側のマクロ名が与えられます。たとえば、

```
$a cdefg
```

というマクロ行を書くことによって、`cdefg` というコマンド列に対して `a` というマクロ名を与えることができます。

角括弧コマンドは、ひとつのマクロ行の中で完結している必要があります。言い換えれば、ひとつのマクロ行の中の角括弧は、左右の対応が取れていないといけないということです。ですから、

```
$4m cd[efg
```

というマクロ行はエラーになります。

### 3.2.4　マクロの展開

マクロ名を、それによって参照されるマクロに置き換えることを、マクロを「展開する」(unfold) と言います。

マクロを展開したいときは、ドルマーク (dollar sign, `$`) というコマンドを使います。このコマンドは、引数として記述されたマクロ名によって参照されるマクロを、その場所で展開して実行します。たとえば、`cdefg` というコマンド列に対して `a` というマクロ名が与えられているとすると、

```
gfed$afedc
```

というコマンド列を実行した結果は、

```
gfedcdefgfedc
```

というコマンド列を実行した結果と同じになります。

マクロ行と、それによって定義されたマクロを展開するコマンドを含んでいる MML 行とは、どちらを上に書いてもかまいません。

MML データの例　`macro.mml`

```
$a [cd]8
$b [cdefgfed]2
$c cdefgab>cdc<bagfed
A o4 l16
A $a $c $a $b $c $b $a $c $a c1
```

### 3.2.5　マクロの木

マクロは、マクロを展開するコマンドを含んでいてもかまいません。つまり、

```
$a $1d [$4f]2 $3m
```

というように、マクロを使ってマクロを定義してもかまわない、ということです。ただし、マクロの中で展開されるマクロを定義するマクロ行は、かならず、展開するマクロ行よりも上に書いておかないといけません。

マクロを使ってマクロを定義することによって、楽曲を階層的に記述することができます。つまり、マクロを部品とする木の形で楽曲を記述することができるということです。楽曲を木の形で記述してできた MML データは、そうでない MML データよりも、人間にとって読みやすいものになります。

MML データの例　`tree.mml`

```
$1a cgeg
$1b caea
$1c dafa
```

```
$1d <b>gfg
$a [$1a]2 [$1b]2 [$1c]2 [$1d]2
A o4 l8 [$a]8 c1
```

## 3.3 定義行

### 3.3.1 定義行の基礎

MML データの中に書くことができる行の種類としては、MML 行とマクロ行のほかに、「定義行」(definition line) と呼ばれるものがあります。定義行は、MML を処理するソフトが持っている何らかの機能を実行するという動作をあらわしています。

定義行というのは、

| # | 機能名 | 引数 | 改行 |
|---|---|---|---|

という構文を持つ文字列のことです。「機能名」のところには実行したい機能の名前を書いて、「引数」のところには、その機能に渡すデータを書きます。

### 3.3.2 ファイルのインクルード

ファイルの内容を読み込んで、それを文書の中に埋め込むことを、ファイルを「インクルードする」(include) と言います。

MML では、`include` という機能を実行する定義行を書くことによって、ファイルをインクルードすることができます。その定義行には、引数として、インクルードするファイルのパス名を二重引用符で囲んだものを書く必要があります。たとえば、

```
#include "/usr/local/mml/useful.mml"
```

という定義行を書くことによって、引数で指定したファイルを、この定義行の位置にインクルードすることができます。

なお、`include` の定義行に、引数として相対パス名を書いた場合、その相対パス名は、その定義行が書かれている MML データのファイルの位置を起点にしていると解釈されます。

MML データの例 `chords.mml`

```
$c c0e0g
$a <a0>c0e
$d d0f0a
$g <g0b0>d0f
```

MML データの例 `include.mml`

```
#include "chords.mml"
A o4 l2
A [$c]4 [$a]4 [$d]4 [$g]4 [$c]2 l1$c
```

### 3.3.3 メタイベント

MIDI データは、普通、SMF(Standard MIDI File) という標準規格にしたがって作成されます。SMF は、さまざまな演奏情報を表現する方法を規定しています。SMF で規定された方法で個々の演奏情報を表現したデータは、「MIDI イベント」(MIDI event) と呼ばれます。

SMF は、演奏情報だけではなくて、楽曲についての情報や、歌詞や、MIDI データを取り扱うソフトが利用する情報などを表現する方法についても規定しています。SMF で規定された方法で演奏情報ではない情報を表現したデータは、「メタイベント」(metaevent) と呼ばれます。

### 3.3.4 曲名

`title` という機能を実行する定義行を書くことによって、メタイベントとして曲名を MIDI データの中に書き込むことができます。

`title` を実行する定義行には、引数として、文字列を二重引用符で囲んだものを書きます。そうすると、その文字列が曲名のメタイベントとして MIDI データの中に書き込まれます。たとえば、

```
#title "Liebestraum No. 3 in A flat op. 62"
```

という定義行を書くことによって、その引数を、曲名のメタイベントとして MIDI データの中に書き込むことができます。

#### 3.3.5　著作権表示

copyright という機能を実行する定義行を書くことによって、メタイベントとして著作権表示を MIDI データの中に書き込むことができます。

copyright を実行する定義行には、引数として、文字列を二重引用符で囲んだものを書きます。そうすると、その文字列が著作権表示のメタイベントとして MIDI データの中に書き込まれます。たとえば、

```
#copyright "Copyright (C) 2199 Kodai Mamoru"
```

という定義行を書くことによって、その引数を、著作権表示のメタイベントとして MIDI データの中に書き込むことができます。

# 第 4 章　声部

## 4.1　トラック

### 4.1.1　トラックとは何か

MML データの中に書かれたコマンドは、それをプログラムが処理するとき、「トラック」(track) と呼ばれる容器に格納されて管理されます。

ひとつのトラックに格納されているコマンドは、それらが並んでいる順番のとおりに実行されていきます。それに対して、複数のトラックのそれぞれに格納されているコマンドは、時間的に並列に実行されます。ですから、複数のトラックのそれぞれにコマンドを入れておくことによって、複数の声部から構成される楽曲を MML で記述することができます。

### 4.1.2　主トラックと従属トラック

トラックは、「主トラック」(primary track) と「従属トラック」(dependent track) という 2 種類のものに分類されます。

主トラックというのは、普通のトラックのことです。通常、ひとつの主トラックはひとつの声部に対応します。主トラックは、最大 26 個まで作ることができます。

従属トラックというのは、主トラックに従属するトラックのことです。楽曲の先頭や先頭付近ではなくて、楽曲の途中から始まる声部を記述するときは、主トラックを使うよりも従属トラックを使うほうが便利です。従属トラックは、それぞれの主トラックごとに最大 9 個まで作ることができます。

### 4.1.3　トラック名

個々のトラックは、「トラック名」(track name) と呼ばれる名前によって識別されます。主トラックの名前は、A、B、C のような、1 文字の英字の大文字です。

従属トラックの名前は、1 から 9 までの 1 文字の数字と、1 文字の英字の大文字とを組み合わせたものです。たとえば、1A、2M、9B などは、従属トラックの名前になります。

従属トラックの名前の中の英字は、その従属トラックが従属している主トラックを意味しています。たとえば、6M という従属トラックの名前の中の M は、その従属トラックが M という主トラックに従属しているということをあらわしています。

従属トラックの名前の中の数字は、同じ主トラックに従属している従属トラックの中のひとつを識別するためのものです。異なる主トラックに従属している従属トラックのそれぞれは、たとえその名前の中の数字が同一だったとしても、互いに無関係です。たとえば、3A と 3B という二つの従属トラックのあいだには、何の関係もありません。

### 4.1.4　トラックを指定する方法

MML のコマンド列は、トラックの中に格納されていなければ、実行することができません。ですから、コマンド列を書くときは、そのコマンド列をどのトラックに入れるのかということを指定する必要があります。

コマンド列は、MML 行と呼ばれる行の中に書くわけですが、MML 行の先頭には、かならず何らかのトラック名を書く必要があります。MML 行の先頭に書かれたトラック名は、その MML 行の中のコマンド列が格納されるトラックを意味しています。たとえば、

```
F cdefgab>c
```

という MML 行を書いたとすると、その中のコマンド列は、F という主トラックに格納されます。同じように、

```
8G cdefgab>c
```

という MML 行を書いたとすると、その中のコマンド列は、8G という従属トラックに格納されます。

MML 行の先頭には、2 個以上のトラック名から構成される列を書くことも可能です。2 個以上のトラック名から構成される列が MML 行の先頭に書かれている場合、その MML 行のコマンド列は、それらのトラック名で指定されたすべてのトラックに格納されます。たとえば、

```
BMX cdefgab>c
```

という MML 行を書いたとすると、その中のコマンド列は、B、M、X、という三つの主トラックのそれぞれに格納されます。

### 4.1.5 属性の分類

「宣言」(declaration) と呼ばれるコマンドを実行すると、それによって何らかの属性が設定されます。

属性は、何が持っている属性なのかということによって、トラックが持っているものや、楽曲が持っているものなどに分類することができます。

トラックが持っている属性としては、オクターブ、調性、省略時の音の長さ、ギャップの長さ、省略時のベロシティー、ベロシティーの省略時の増減値などがあります。それらの属性を設定する宣言は、その宣言が格納されているトラックの属性だけを設定します。

それに対して、テンポは、楽曲が持っている属性です。テンポを設定する t 宣言は、それが格納されているトラックとは無関係に、楽曲のテンポを設定します。

### 4.1.6 テンポトラック

テンポを設定する t 宣言は、テンポの設定のみを目的とするトラックに格納するのが普通です。そのような、テンポの設定のみを目的とするトラックは、「テンポトラック」(tempo track) と呼ばれます。

MML データの例　track.mml

```
$d dafa
$g dbgb
$a caga

A l1 t120
B KJd- o4 l8 k80 V10 q0
C KJd- o3 l8 k80 V10 q0
D KJd- o2 l1 k80 V10 q0

A r     r
B r1    r1
C [$d]2 [$d]2
D d     d

A r      r      r          r
B f4.ga2 abagf2 b4fgabf4 ( d4e4f2
C [$d]2  [$d]2  [$g]2    ( [$d]2
D d      d      g        ( d

A   r      r            t-20 r         t-20
B ) f4.ga2 ) ag>c<ba2 ( b4.a>d<ba>d< ( f4e4d2
C ) [$d]2  ) [$g]2    ( $d$a         ( $dd2
D ) d      ) g        ( d2a2         ( d
```

#### 4.1.7　従属トラックの使い方

楽曲の先頭ではなくて、楽曲の途中から始まる声部を記述するとき、もしも主トラックを使うとすると、楽曲の先頭からその声部が始まるまでの休止を実行するための r コマンドを書く必要があります。しかし、従属トラックを使えば、そのための r コマンドは必要ではなくなります。

従属トラックを指定した MML 行が実行されるタイミングは、その従属トラックが従属している主トラックの MML 行のうちで、もっとも直前に書かれているものと同期が取られます。たとえば、

```
A cdefg2
A fgab>c2
1A defga2
```

という MML データを実行すると、 fgab>c2 と defga2 は、同じタイミングで実行が開始されます。

なお、従属トラックも、主トラックと同じように、オクターブ、調性、省略時の音の長さなどのトラックの属性は、そのための宣言が実行されていなければ、デフォルトの設定になっています。つまり、従属トラックの属性は、主トラックから自動的に継承されるわけではない、ということです。

MML データの例　depend.mml

```
A o4 l8
A c4.defg4 f4.gab>c4<
A c4.defg4 f4.gab>c4<
1A o3 l8
1A g4.ab>cd4 c4.defg4<
```

### 4.2　チャンネル

#### 4.2.1　チャンネルとは何か

MIDI では、電子楽器から電子楽器へ送られる、演奏情報をあらわすデータのことを、「MIDI メッセージ」(MIDI message) と呼びます。MIDI メッセージの多くは、それを受信する電子楽器のうちのひとつを特定するための、「チャンネル」(channel) と呼ばれる番号を持っています。MIDI メッセージを受信する電子楽器は、送られてきた MIDI メッセージの中のチャンネルを調べることによって、それが自分に宛てて送られたものかどうかを識別することができます。

チャンネルは、1 から 16 までの整数です。

#### 4.2.2　トラックのチャンネルの設定

MML データを MIDI データに変換すると、MML データの中にあるコマンドの多くは、それに対応する MIDI メッセージに変換されます。

個々のトラックは、自分に格納されているコマンドを変換することによって作られた MIDI メッセージに持たせるチャンネル、という属性を持っています。

トラックのチャンネルを設定をしたいときは、そのトラックに、大文字の C という宣言を入れます。

C 宣言には、引数としてチャンネル（1 から 16 までの整数）を書きます。そうすると、そのチャンネルがトラックに設定されます。たとえば、

```
B C5
```

という MML 行を書くことによって、トラック B に対して 5 番というチャンネルを設定することができます。

なお、2 個以上のトラックに対して同一のチャンネルを設定してもかまいませんし、ひとつのトラックの途中でチャンネルを変更してもかまいません。

#### 4.2.3　チャンネルが持っている属性

属性は、何が持っている属性なのかということによって分類することができます。たとえば、オクターブはトラックが持っている属性で、テンポは楽曲が持っている属性です。

チャンネルというのも、さまざまな属性を持っています。たとえば、音色やパンポットやボリュームなどは、チャンネルが持っている属性です。

チャンネルが持っている属性も、何らかの宣言を書くことによって設定することができます。チャンネルの属性を設定する宣言は、その宣言が格納されているトラックに設定されているチャンネルに対して属性を設定します。

たとえば、トラックBにチャンネル5が設定されているとするとき、トラックBに格納されるコマンド列の中に、チャンネルの属性を設定する宣言を書いたとすると、その属性は、チャンネル5に対して設定されることになります。

2個以上のトラックに対して同一のチャンネルを設定している場合、チャンネルの属性を設定する宣言は、それらのトラックのうちのどれかひとつに書かれていれば充分です。

### 4.2.4 音色

音が持っている質感のことを、その音の「音色（おんしょく）」(timbre) と言います。

MIDI メッセージを音声の信号に変換する装置は、「音源」(tone generator) と呼ばれます。1台の音源は、さまざまな種類の音色を持っていて、それらのうちのどれかの音色を使って音を出力します。

音源が音を出力するために使う音色を設定したいときは、その音源に対して、「プログラムチェンジ」(program change) と呼ばれる MIDI メッセージを送信します。プログラムチェンジは、音色を識別するための番号を含んでいて、音源は、その番号にしたがって音色を設定します。

プログラムチェンジが持っている、音色を識別するための番号は、「プログラムナンバー」(program number) と呼ばれます。プログラムナンバーは、0 から 127 までのあいだの整数です。

GM(General MIDI) という規格は、さまざまな音色に対して「音色番号」(patch number) と呼ばれる番号を与えています。たいていの音源は GM に準拠して設計されていますので、GM の音色番号をプログラムナンバーとして音源に送ることによって、その番号を持つ音色を音源に設定することができます。ただし、GM の音色番号は、0 から 127 までではなくて、1 から 128 までですので、それをプログラムナンバーとして送信するためには、それから 1 を減算する必要があります。

MML では、アットマーク (at sign, `@`) という宣言を書くことによって、チャンネルに対して音色を設定することができます。`@` 宣言には、引数としてプログラムナンバーを記述します。そうすると、それに 1 を加算したものを GM の音色番号とする音色がチャンネルに対して設定されます。たとえば、トランペットの音色は 57 という GM の音色番号を持っていますので、`@56` という宣言を書くことによって、チャンネルに対してトランペットの音色を設定することができます。

MML データの例　`timbre.mml`

```
$m c4.d8e8f8g4 f4.g8a8b8>c4<

A C1 @11 o4 ; vibraphone
B C2 @19 o3 ; church organ

A $m   r1r1 $m
B r1r1 $m   $m
```

なお、プログラムチェンジという MIDI メッセージによって設定される音色というのは、基本的にはリズム楽器以外の音色です。リズム楽器の音色を設定する方法については、次の節で説明したいと思います。

### 4.2.5 パンポット

左と右のあいだのどの方向から音が聞こえてくるのかということを、その音の「パンポット」(panpot) と言います。

MIDI では、パンポットは 0 から 127 までのあいだの整数で指定されます。0 がもっとも左、64 が中央、127 がもっとも右をあらわします。

MML では、小文字の p という宣言を書くことによって、チャンネルに対してパンポットを設定することができます。p 宣言には、パンポットをあらわす整数を引数として記述します。たとえば、p80 という宣言を書くことによって、80 というパンポット（中央よりもやや右寄り）をチャンネルに対して設定することができます。

MML データの例　`panpot.mml`

```
$m c4.d8e8f8g4 f4.g8a8b8>c4<
```

```
A C1 p0   o3 ; left
B C2 p127 o4 ; right

A $m   r1r1 $m
B r1r1 $m   $m
```

### 4.2.6　ボリューム

音の大きさのことを、その音の「ボリューム」(volume) と言います。

MIDI では、ボリュームは 0 から 127 までのあいだの整数で指定されます。0 がもっとも小さなボリュームで、127 がもっとも大きなボリュームをあらわします。

MML では、小文字の v という宣言を書くことによって、チャンネルに対してボリュームを設定することができます。v 宣言には、ボリュームをあらわす整数を引数として記述します。たとえば、v70 という宣言を書くことによって、70 というボリュームをチャンネルに対して設定することができます。

MML データの例　volume.mml

```
$m c4.d8e8f8g4 f4.g8a8b8>c4<

A C1 @19 p0   v30  o3 ; small
B C2 @19 p127 v120 o4 ; large

A $m   r1r1 $m
B r1r1 $m   $m
```

次の MML データは、前の節で登場した track.mml に、C 宣言、@ 宣言、p 宣言、v 宣言を追加したものです。

MML データの例　channel.mml

```
$d dafa
$g dbgb
$a caga

A l1 t120
B C1 @82 p64  v100 KJd- o4 l8 k80 V10 q4 ; calliope
C C2 @12 p117 v70  KJd- o3 l8 k80 V10 q0 ; marimba
D C3 @11 p10  v110 KJd- o3 l1 k80 V10 q0 ; vibraphone

A r     r
B r1    r1
C [$d]2 [$d]2
D d     d

A r      r      r          r
B f4.ga2 abagf2 b4fgabf4 ( d4e4f2
C [$d]2  [$d]2  [$g]2    ( [$d]2
D d      d      g        ( d

A   r      r            t-20 r         t-20
B ) f4.ga2 ) ag>c<ba2 ( b4.a>d<ba>d< ( f4e4d2
C ) [$d]2  ) [$g]2    ( $d$a         ( $dd2
D ) d      ) g        ( d2a2         ( d
```

## 4.3　リズム楽器

### 4.3.1　リズム楽器のチャンネル

GM という規格は、すでに説明したように、音色に番号を与えているわけですが、それだけではなくて、さらに、「リズム楽器はかならずチャンネル 10 を使わなければならない」ということも規定しています。ですから、GM に準拠している音源は、チャンネルが 10 のノートオンを受信した場合、リズム楽器の音を出力することになります。

そのような理由で、MML で楽曲を記述する場合、リズム楽器の声部を格納するトラックには、

チャンネルとしてかならず 10 を設定する必要があります。

### 4.3.2 リズムトラック

ノートオンの MIDI メッセージが送られてからノートオフの MIDI メッセージが送られるまでの時間の長さは、「ゲートタイム」(gate time) と呼ばれます。

リズム楽器は、ゲートタイムが長くても短くても、まったく同じ音を出します。ですから、MIDI データを作るとき、リズム楽器のゲートタイムはどんな長さでもかまわないわけですが、普通は、ほとんどゼロに近い長さにします。

リズム楽器を鳴らすためのコマンド列は、通常、「リズムトラック」(rhythm track) と呼ばれる、リズム楽器専用のトラックに格納されます。リズムトラックに格納された音符コマンドは、ゲートタイムのきわめて短い音のデータに変換されます。

MML のトラックは、「リズムトラックであるかどうか」という属性を持っていて、「リズムトラックではない」という状態がデフォルトで設定されています。トラックをリズムトラックにしたいときは、`RT` という宣言を書きます。この宣言は、引数を何も書かないで、ただ単に `RT` と書くだけです。

### 4.3.3 リズム楽器の音色

すでに説明したように、音源が出力する楽器の音色は、プログラムチェンジという MIDI メッセージによって設定されます。しかし、リズム楽器の音色を指定する MIDI メッセージは、プログラムチェンジではありません。

リズム楽器の音色は、ノートオンの MIDI メッセージによって指定されます。ノートオンの MIDI メッセージは、音の高さをあらわすデータを含んでいるのですが、10 番のチャンネルに限り、本来ならば音の高さをあらわすはずのデータが、楽器の音色をあらわすことになるのです。

どんな音の高さがどんな音色を意味するかということは、GM で規定されています。たとえば、「平仮名に」の高さは、スネアドラムの音色をあらわしています。

MML データの例 `rhythm.mml`

```
$h [f+8]16        ; closed hi-hat
$s [r4d4]4        ; snare drum
$b [c4r4]3c8c8r4  ; bass drum

A RT C10 o2 k60  [$h]4
B RT C10 o2 k50  [$s]4
C RT C10 o2 k120 [$b]4
```

# 索引