Musical MIDI Accompaniment

Reference Manual

Bob van der Poel
Wynndel, BC, Canada

bvdp@uniserve.com

September 21, 2004

Table Of Contents

Chapter 1
Overview and Introduction

Musical MIDI Accompaniment, MMA[1], generates standard MIDI[2] files which can be used as a backup track for a soloist. It was written especially for me—I am an aspiring saxophonist and wanted something to practice my jazz solos. With MMA I can create a track based on the chords in a song, transpose it to the correct key for my instrument, and play my very bad improvisations until they get a bit better.

I also have a small combo group which is always missing at least one player. With MMA generated tracks we can practice and perform even if a rhythm player is missing. This all works much better than I expected when I started to write the program.

1.1 License, Version and Legalities

The program MMA was written and is copyright Robert van der Poel, 2002/2004.

This program, the accompanying documentation, and library files can be freely distributed according to the terms of the GNU General Public License (see the distributed file "COPYING").

If you enjoy the program, make enhancements, find bugs, etc. send a note to me at `bvdp@uniserve.com`; or a postcard (or even money) to PO Box 57, Wynndel, BC, Canada V0B 2N0.

The current version of this package is maintained at: `http://mypage.uniserve.com/~bvdp/mma/mma.html`.

This document reflects version 0.10d of MMA.

Warning: *This program is currently in a beta state. The commands used in the input files, the output, the overall logic and anything else you can think of might change in the future.*

This manual most likely has lots of errors. Spelling, grammar, and probably a number of the examples need fixing. Please give me a hand and report anything. . . it'll make it much easier for me to generate a really good product for all of us to enjoy.

[1]Musical MIDI Accompaniment and the short form MMA in the distinctive script are names for a program written by Bob van der Poel. The "MIDI Manufacturers Association, Inc." uses the acronym MMA, but there is no association between the two.

[2]MIDI is an acronym for Musical Instrument Digital Interface.

1.2 Installing MMA

MMA is a Python program developed with version 2.3 of Python. At the very least you will need this version (or later) of Python!

To play the MIDI files you'll need a MIDI player. Pmidi, tse3play, and many others are available for Linux systems. For Windows and Mac systems I'm sure there are many, many choices.

You'll need a text editor to create input files.

MMA consists of a variety of bits and pieces:

- The executable Python script, mma, must somewhere in your path. For users running a Windows system, please check our website for details on how to install on these systems.[3]
- A number of Python modules. These should all be installed under the directory `/usr/local/share/mma/modules`.
- A number of library files defining standard rhythms. These should all be installed under the directory `/usr/local/share/mma/lib/stdlib`.

The script "install" will (hopefully) install MMA properly for you. It assumes that main script is to be installed in `/usr/local/bin` and the support files in `/usr/local/share/mma`. If you want an alternate location, you can edit the paths in the script. The only supported alternate to use is `/usr/share/mma`.

In addition, you *can* run MMA from the directory created by the untar. This is not recommended, but will show some of MMA's stuff.

You should be "root" to run the install script.

1.3 Running MMA

For details on the command line operations in MMA please refer to chapter 2.

To create a MIDI file you need to:

1. Create a text file (also referred to as the "input file") with instructions which MMA understands. This includes the chord structure of the song, the rhythm to use, the tempo, etc. The file can be created with any suitable text editor.
2. Process the input file. From a command line the instruction:

   ```
   mma myfile <ENTER>
   ```

   will invoke MMA and, assuming no errors are found, create a MIDI file "myfile.mid".
3. Play the MIDI file with any suitable MIDI player.

[3]If someone using a Mac system could let me know how to install on this system I'd be glad to include those details on my website.

4. Edit the input file again and again until you get the perfect track.
5. Share any patterns, sequences and grooves with the author so they can be included in future releases!

An input file consists of the following information:

1. MMA directives. These include **Tempo**, **Time**, **Volume**, etc. See chapter 16. .
2. **Pattern**, **Sequence** and **Groove** See chapters 4, 5 and 6.
3. Music information. See chapter 8.
4. Comment lines and blank lines. See below.

Items 1 to 3 are detailed later in this manual. Please read them before you get too involved in this program.

1.4 Comments

We do believe that proper indentation, white space and comments are a *good thing*. But, in most cases MMA really doesn't care:

- Any leading space or tab characters are ignored,
- Multiple tabs and other white space are treated as single characters,
- Any blank lines in the input file are ignored.

Each line is initially parsed for comments. A comment is anything following a "//" (2 forward slashes).[4]

Comments are stripped from the input stream. Lines starting with the **Comment** directive are also ignored. See the *comment* discussion for details (see page 91).

1.5 Theory Of Operation

To understand how MMA works it's easiest to look at the initial development concept.Initially, a program was wanted which would take a file which looked something like:

```
Tempo 120
Fm
C7
...
```

and end up with a MIDI file which played the specified chords over a drum track.

Of course, after starting this "simple" project a lot of complexities developed.

[4]We wanted to use "#" for comments, but that sign is used for "sharps" in chord notation.

First, the chord/bar specifications. Just having a single chord per bar doesn't work—many songs have more than one chord per bar. Second, what is the rhythm of the chords? What about a bass line? Oh, and what drum track?

Well, things got more complex after that. At a bare minimum, we needed the ability to:

- ♫ Be able to specify multiple chords per bar,
- ♫ Be able to define different patterns for chords, bass lines and drum tracks,
- ♫ Make the input files easy to create and debug,
- ♫ Provide a reusable library that a user could simply plug in, or modify.

From these simple needs MMA was created.

The basic building blocks of MMA are **patterns**. A pattern is a specification which tells MMA what notes of a chord to play, the start point in a bar for the chord/notes, and the duration and the volume of the notes.

MMA patterns are combined into **sequences**. This lets you create multi-bar rhythms.

A collection of patterns can be saved and recalled as **grooves**. This makes it easy to pre-define complex rhythms in library files and incorporate them into your song with a simple two word command.

MMA is bar or measure based (we use the words interchangeably in this document). This means that MMA processes your song one bar at a time. The music specification lines all assume that you are specifying a single bar of music. The number of beats per bar can be adjusted; however, all chord changes must fall on a beat division (the playing of the chord or drum note can occur anywhere in the bar).

To make the input files look more musical, MMA supports **repeats** and **repeat endings**. However, complexities like D.S. and Coda are not internally supported (but can be created by using the *Goto* command).

1.6 Case Sensitivity

Just about everything in a MMA file is case insensitive.

This means that the command:

```
Tempo 120
```

could be entered in your file as:

```
TEMPO 120
```

or even

```
TeMpO 120
```

for the exact same results.

Names for patterns, and grooves are also case insensitive.

The only exceptions are the names for chords and filenames. In keeping with standard chord notation, chord names are in mixed case; this is detailed in Chapter 8. Filenames are covered in Chapter 19.

Chapter 2

Running MMA

MMA is a command line program. To run it, simply type the program name followed by the required options. For example,

```
mma test
```

processes the file “test”[1] and creates the MIDI file “test.mid”.

2.1 Command Line Options

The following command line options are available:

Option	*Description*
-v	Show program’s version number and exit.
-d	Enable LOTS of debugging messages. This option is mainly designed for program development and may not be useful to users.
-o	A debug subset. This option forces the display of complete filenames/paths as they are opened for reading. This can be quite helpful in determining which library files are being used.
-p	Display patterns as they are defined. The result of this output is not exactly a duplicate of your original definitions. Most notable are that the note duration is listed in MIDI ticks, and symbolic drum note names are listed with their numeric equivalents.
-s	Display sequence info during run. This shows the expanded lists used in sequences. Useful if you have used sequences shorter (or longer) than the current sequence length.

[1] Actually, the file “test” or “test.mma” is processed. Please read section 19.1 (see page 106).

-r	Display running progress. The bar numbers are displayed as they are created complete with the original input line. Don't be confused by multiple listing of "*" lines. For example the line
	`33 Cm * 2`
	would be displayed as:
	`88:  33 Cm *2`
	`89:  33 Cm *2`
	This makes perfect sense if you remember that the same line was used to create both bars 88 and 89.

-n	Disable generation of MIDI output. This is useful for doing a test run or to check for syntax errors in your script.
-e	Show parsed/expanded lines. Since MMA does some internal fiddling with input lines, you may find this option useful in finding mismatched **BEGIN** blocks, etc.
-c	Display the tracks allocated and the MIDI channel assignments after processing the input file. No output is generated.
-mBARS	Set the maximum number of bars which can be generated. The default setting is 500 bars (a long song![2]). This setting is needed since you can create infinite loops by improper use of the *goto* command. If your song really is longer than 500 bars use this option to increase the permitted size.
-g	Update the library database for the files in the *LibPath*. You should run this command after installing new library files or adding a new groove to an existing library file. If the database (stored in the file `MMADIR`) is not updated, MMA will not be able to auto-load an unknown groove. The current installation of MMA does not set directory permissions. It simply copies whatever is in the distribution. If you have trouble using this option, you will probably have to reset the permissions on the lib directory. MMA will update the groove database with all files in the current *LibPath*. All files *must* have a ".mma" extension. Any directory containing a file named `MMAIGNORE` will be ignored. Note, that `MMAIGNORE` consists of all uppercase letters and is usually an empty file.
-G	Same as the "-g" option (above), but the uppercase version forces the creation of a new database file—an update from scratch just in case something really goes wrong.
-fFILE	Set output to FILE. Normally the output is sent to a file with the name of the input file with the extension ".mid" appended to it. This option lets you set the output MIDI file to any filename.
-Mx	Generate type 0 or 1 MIDI files. The paramater "x" must be set to the single digit "0" or "1". For more details, see the *MidiSMF* section on see page 83.

The following commands are used to create the documentation. As a user you should probably never have a need for any of them.

[2] 500 bars with 4 beats per bar at 200 BPM is about 10 minutes.

-Dx	Expand and print **Doc** commands used to generate the standard library reference. No MIDI output is generated when this command is given. Doc strings in RC files are not processed. Files included in other files are processed.
-Dn	Create a table of the available chord types.
-Dda	Create a table of the MIDI drum note names, arranged alphabetically.
-Ddm	Create a table of the MIDI drum note names, arranged by MIDI value.
-Dia	Create a table of the MIDI instrument names, arranged alphabetically.
-Dim	Create a table of the MIDI instrument names, arranged by MIDI value.

A number of the debugging commands can also be set dynamically in a song. See the debug section (see page 92) for details.

2.2 Lines and Spaces

When MMA reads a file it processes the lines in various places. The first reading strips out blank lines and comments of the "//" type.

On the initial pass though the file any continuation lines are joined. A continuation line is any line ending with a single "\"—simply, the next line is concatenated to the current line to create a longer line.

Unless otherwise noted in this manual, the various parts of a line are delimited from each other by runs of whitespace. Whitespace can be tab characters or spaces. Other characters may work, but that is not recommended, and is really determined by Python's definitions.

2.3 Programming Comments

MMA is designed to read and write files; it is not a filter (this could be changed, but we're not sure why this would be needed).

As noted earlier in this manual, MMA has been written entirely in Python.There were some initial concerns about the speed of a "scripting language" when the project was started, but Python's speed appears to be entirely acceptable. On an AMD Athlon 1900+ system running Mandranke Linux 9.2, most of songs compile to MIDI in well under one second. If you need faster results, you're welcome to recode this program into C or C++, but it would be cheaper to buy a faster system, or spend a bit of time tweaking some of the more time intensive Python loops.

The manual has been prepared with the LaTeX typesetting system. Once life and the program settle down the source files may be released as well. Currently, there are two versions available: a PDF file intended for printing (generated with dvipdf) and a HTML version (transformed with LaTeX2HTML) for electronic viewing. If other formats are needed . . . please offer to volunteer.

Chapter 3

Tracks and Channels

This chapter discusses MMA tracks and MIDI channels. If you are reading this manual for the first time you might find some parts confusing. If you do just skip ahead—you can run MMA without knowing many of these details.

3.1 MMA Tracks

To create your accompaniment tracks, MMA divides output into several internal tracks. There are a total of 8 different types of tracks, and an unlimited number of sub-tracks.

When MMA is initialized there are no tracks assigned; however, as your library and song files are processed various tracks will be created Each created a unique name. The track types are discussed later in this chapter, but for now they are **Bass**, **Chord**, **Walk**, **Drum**, **Arpeggio**, **Scale**, **Melody** and **Solo**.

All tracks are named by appending a "-" and"name" to the type-name. This makes it very easy to remember the names, without any complicated rules. So, drum tracks can have names "Drum-1", "Drum-Loud" or even "Drum-a-long-name". The other tracks follow the same rule.

In addition to the hyphenated names described above, you can also name a track using the type-name. So, "DRUM" is a valid drum track name. In our library files we usually use the type-name to describe patterns.

All track names are case insensitive. This means that the names "Chord-Sus", "CHORD-SUS" and "CHORD-sus" all refer to the same track.

If you want to see the names defined in a song, just run MMA on the file with the "-c" command line option.

3.2 Track Channels

MIDI defines 16 distinct channels numbered 1 to 16.[1] There is nothing which says that "chording" should be sent to a specific channel, but the drum channel should always be channel 10.[2]

[1]We use the values 1 to 16 in this document. Internally they are stored as values 0 to 15.

[2]This is not a MIDI rule, but a convention established in the GM (General MIDI) standard. If you want to find out more about this, there are lots of books on MIDI available.

For MMA to produce any output, a MIDI channel must be assigned to a track. During initialization all of the DRUM tracks are assigned to special MIDI channel 10. As musical data is created other MIDI channels are assigned to various tracks as needed.

Channels are assigned from 16 down to 1. This means that the lower numbered channels will most likely not be used, and will be available for other programs or as a "keyboard" track on your synth.

In most cases this will work out just fine. However, there are a number of methods you can use to set the channels "manually." You might want to read the sections on *Channel* (see page 80), *ChShare* (see page 81), *On* (see page 96), and *Off* (see page 96).

Why bother with all these channels? It would be much easier to put all the information onto one channel, but this would not permit you to set special effects (like **Portamento** or **Pan**) for a specific track. It would also mean that all your tracks would need to use the same instrumentation.

3.3 Track Descriptions

You might want to come back to this section after reading more of the manual. But, somewhere we need to describe the different track types, and why they exist.

Musical accompaniment comes in a combination of the following:

- ♫ Chords played in a rhythmic or sustained manner,
- ♫ Single notes from chords played in a sustained manner,
- ♫ Bass notes. Usually played one at a time in a rhythmic manner,
- ♫ Scales, or parts of scales. Usually as an embellishment,
- ♫ Single notes from chords played one at time: arpeggios.
- ♫ Drums and other percussive instruments played rhythmically.

Of course, this leaves the melody … but that is up to you, not MMA… but, if you suspect that some power is missing here, read the brief description of *Solo* and *Melody* tracks (see page 17) and the complete "Solo and Melody Tracks" chapter (see page 46).

MMA comes with several types of tracks, each designed to fill different accompaniment roles. However, it's quite possible to use a track for different roles than originally envisioned. For example, the bass track can be used to generate a single, sustained treble note—or, by enabling *Harmony* multiple notes.

The following sections describe the tracks and give a few suggestions on their uses.

3.3.1 Drum

Drums are the first thing we usually think about when we hear the word "accompaniment". All MMA drum tracks share MIDI channel 10, which is a GM MIDI convention. Drum tracks play single notes determined by the **Tone** setting for a particular sequence.

3.3.2 Chord

If you are familiar with the sound of guitar strumming, then you're familiar with the sound of a chord. MMA chord tracks play a number of notes, all at the same time. The volume of the notes (and the number of notes) and the rhythm is determined by pattern definitions. The instrument used for the chord is determined by the **Voice** setting for a sequence.

3.3.3 Arpeggio

In musical terms an **arpeggio**[3] is the notes of a chord played one at a time. MMA arpeggio tracks take the current chord and, in accordance to the current pattern, play single notes from the chord. The choice of which note to play is mostly decided by MMA. You can help it along with the **Direction** modifier.

We use **Arpeggio** tracks quite often to highlight rhythms. Using the **RSkip** directive produces broken arpeggios.

Using different note length values in patterns helps to make interesting accompaniments.

3.3.4 Scale

Another embellishment. When MMA plays a scale, it first determines the current chord. Its scales are started on the first note of the chord (if the chord is a C7, the scale will be a C scale). Currently, three types of scales are supported: major, natural minor and chromatic.

The major scale is selected for all chords which are not of a minor flavor, or if the **ScaleType** is set to **Major**.

The natural minor scale is selected for all "minor" chords. This includes chords such as "Cm7", "G#m13", etc. If the **ScaleType** is set to **Minor** this scale is always used.

If the **ScaleType** is set to **Chromatic**, then a chromatic scale is used.

MMA plays successive notes of a scale. The timing and length of the notes is determined by the current pattern. Depending on the **Direction** setting, the notes are played up, down or up and down the scale.

3.3.5 Bass

Bass tracks are designed to play single notes for a chord for standard bass patterns. The note to be played, as well as its timing, is determined by the pattern definition. The pattern defines which note from the

[3]The term is derived from the Italian "to play like a harp".

current chord to play. For example, a standard bass pattern might alternate the playing of the root and fifth notes of a scale or chord. You could also use **Bass** tracks to play single, sustained treble notes.

3.3.6 Walk

The **Walk** tracks are designed to imitate "walking bass" lines. Traditionally, they are played on bass instruments like the upright bass, bass guitar or tuba.

A **Walk** track uses a pattern to define the note timing and volume. Which note is played is determined from the current chord and a simplistic algorithm. There is no user control over the note selection.

3.3.7 Solo and Melody

Solo and **Melody** tracks are used for arbitary note data. Most likely, this is a melody or counter-melody . . . but these tracks can also be used to create interesting ending, introductions or transitions.

3.4 Silencing a Track

There a number of ways to silence a track:

♫ Use the *Off* (page 96) command to stop the generation of MIDI data,

♫ Disable the sequence for the bar with an empty sequence (page 30).

♫ Delete the entire sequence with *SeqClear* (page 30).

♫ Disable the MIDI channel with a "Channel 0" (page 80).

Please refer to the appropiate sections on this manual for further details.

Chapter 4

Patterns

MMA builds its output based on **patterns** and **sequences** supplied by you. These can be defined in the same file as the rest of the song data, or can be included (see chapter 19) from a library file.

A pattern is a definition for a voice or track which describes what rhythm to play during the current bar. The actual notes selected for the rhythm are determined by the song bar data (Chapter 8).

4.1 Defining a Pattern

The formats for the different tracks vary, but are similar enough to confuse the unwary.

Each pattern definition consists of three parts:

- ♫ A unique label to identify the pattern. This is case-insensitive. Note that the same label names can be used in different tracks—for example, you could use the name “MyPattern” in both a Drum and Chord pattern…but this is probably not a good idea. Names can use punctuation characters, but must not begin with an underscore (“_”). The pattern names “z” or “Z” and “-” are also reserved.
- ♫ A series of note definitions. Each set in the series is delimited with a “;”.
- ♫ The end of the pattern definition is indicated by the end-of-line.

In the following sections we show the definitions in continuation lines; however, it is quite legal to mash all the information onto a single line.

The following concepts are used when defining a pattern:

Start When to start the note. This is expressed as a beat offset. For example, to start a note at the start of a bar you use “1”, the second beat would be “2”, the fourth “4”, etc. You can easily use off-beats as well: The “and” of 2 is “2.5”, the “and ahh” of the first beat is “1.75”, etc. Using a beat offset greater than the number of beats in a bar or less than “1” is not permitted. See **Time** (see page 60).

Duration The length of a note is somewhat standard musical notation. Since it is impractical to draw in graphical notes or even to use fractions like $\frac{1}{4}$ MMA uses a shorthand notation detailed in the following table:

Notation	*Description*
1	Whole note
2	Half
4	Quarter
8	Eighth
16	Sixteenth
32	Thirtysecond
64	Sixtyfourth
3	One note of an eighth-note triplet
0	A single MIDI tick

The last note length, “0” is a special value often used in drum tracks where the actual “ringing”length appears to be controlled by the MIDI synth, not the driving program. Internally, a “0” note length in converted to a single MIDI tick.

Lengths can have a single or double dot appended. For example, “2.” is a dotted half note and “4..” adds an eight and sixteenth value to a quarter note.

Note lengths can be combined using “+”. For example, to make a dotted eight note use the notation “8+16”, a dotted half “2+4”, and a quarter triplet “3+3”.

It is permissible to combine notes with “dots” and “+”s. The notation “2.+4” would be the same as a whole note.

The actual length of the note will be adjusted by the **Articulate** value (see page 90).

Volume The MIDI velocity[1] to use for the specified note. For a detailed explanation of how MMA calculates the volume of a note, see chapter 12.

MIDI velocities are limited to the range 0 to 127. However, MMA does not check the volumes specified in a pattern for validity. This is a feature. If you want to ensure that a note is always sounded use a very large value (eg. 1000) for the volume. That way, future adjustments will maintain a large value and this large value will be clipped to the maximum permitted MIDI velocity.

In most cases velocities in the range 50 to 100 are useful.

Offset The offset into the current chord. If you have, for example, a C minor chord (C, E♭, and G) has 3 offsets: 0, 1 and 2. Note that the offsets refer to the *chord* not the scale. For example, a musician might refer to the “fifth”—this means the fifth note of a scale . . . in a major chord this is the third note, which has an offset of 2 in MMA.

Patterns can be defined for **Bass**, **Walking**, **Chord**, **Arpeggio**, **Chord** and **Drum** tracks. All patterns are shared by the tracks of the same type—**Chord-Sus** and **Chord-Piano** share the patterns for **Chord**. As a convenience, MMA will permit you to define a pattern for a sub-track, but remember that it will be shared by all similar tracks. For example:

```
Drum Define S1 1 0 50
```

[1]MIDI “note on” events are declared with a “velocity” value. Think of this as the “striking pressure” on a piano.

and

```
    Drum-woof Define S1 1 0 50
```

Will generate identical outcomes.[2]

4.1.1 Bass

A bass pattern is defined with:

```
    Position Duration Offset Volume ; ...
```

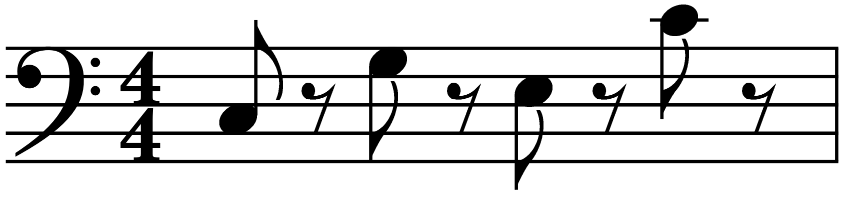
Each group consists of an beat offset for the start point, the note duration, the note offset and volume.

The note offset is one of the digits "1", "3" or "5", each representing the "root", "third" or "fifth" of the chord scale. Internally, MMA translates this into the values "0", "1" and "2" and plays the appropriate note from the current chord.

The note offset can be modified by appending a single or multiple set of "+" or "-" signs. Each "+" will force the note up an octave; each "-" forces it down. This modifier is handy in creating bass patterns when you wish to alternate between the root note and the root up an octave …but we're sure users will find other interesting patterns. There is no limit to the number of "+"s or "-"s. You can even use both together if you're in a mood to obfuscate.

```
Bass Define Broken8 1 8 1 90 ; \
    2 8 5 80 ; \
    3 8 3 90 ; \
    4 8 1+ 80
```

Sheet Music Equivalent



Example 4.1: Bass Definition

Example 4.1 defines 4 bass notes (probably staccato eight notes) at beats 1, 2, 3 and 4 in a $\frac{4}{4}$ time bar. The first note is the root of the chord, the second is the fifth; the third note is the third; the last note is the root

[2]What really happens is that the definition is stored in a slot matching the track's type, not it's name.

up an octave. The volumes of the notes are set to a MIDI velocity of 90 for beats 1 and 3 and 80 for beats 2 and 4.

You should note that the application of chord modifications like *Invert* will change the notes selected since MMA just selects the first, second or third note in the chord. This is convenient if you are using a *Bass* pattern for a harmony line, etc. Generally speaking, if you are using a *Bass* pattern for a conventional bass line pattern, don't apply modifiers like *Invert* to it.

4.1.2 Chord

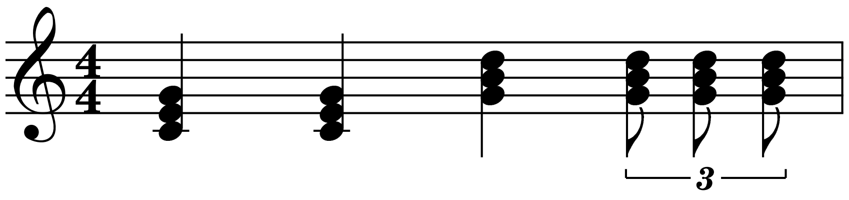
A Chord pattern is defined with:

```
Position Duration Volume1 Volume2 ..  ; ...
```

Each group consists of an beat offset for the start point, the note duration, and the volumes for each note in the chord. If you have fewer volumes than notes in a chord, the last volume will apply to the remaining notes.

```
Chord Define Straight4+3 1 4 100 ; \
    2 4 90 ; \
    3 4 100 ;\
    4 3 90 ; \
    4.3 3 80 ; \
    4.6 3 80
```

Sheet Music Equivalent



Example 4.2: Chord Definition

Example 4.2 defines a $\frac{4}{4}$ pattern in a quarter, quarter, quarter, triplet rhythm. The quarter notes sound on beats 1, 2 and 3; the triplet is played on beat 4. The example assumes that you have C major for beats 1 and 2, and G major for 3 and 4.

Using a volume of "0" will disable a note. So, you want only the root and third of a chord to sound, you could use something like:

```
Chord Define Dups 1 8 90 0 90 0; 3 8 90 0 90 0
```

4.1.3 Arpeggio

An Arpeggio pattern is defined with:

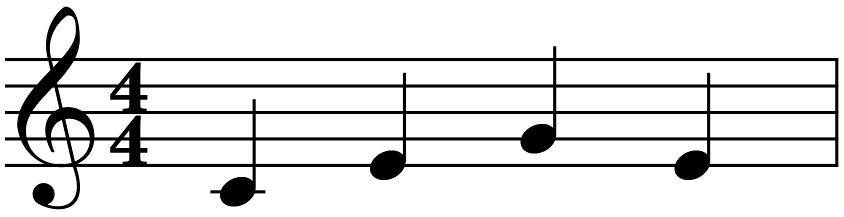
```
    Position Duration Volume ; ...
```

The arpeggio tracks play notes from a chord one at a time. This is quite different from chords where the notes are played all at once—refer to the **Strum** directive (see page 99).

Each group consists of an beat offset, the note duration, and the note volume. You have no choice as to which notes of a chord are played (however, they are played in alternating ascending/descending order.[3] Volumes are selected for the specific beat, not for the actual note.

```
Arpeggio Define 4s 1 4 100; \
    2 4 90; \
    3 4 100; \
    4 4 100
```

Sheet Music Equivalent



Example 4.3: Arpeggio Definition

Example 4.3 plays quarter note on beats 1, 2, 3 and 4 of a bar in $\frac{4}{4}$ time.

4.1.4 Walk

A Walking Bass pattern is defined with:

```
    Position Duration Volume ; ...
```

Walking bass tracks play up and down the first part of a scale, paying attention to the "color"[4] of the chord. Walking bass lines are very common in jazz and swing music. The appear quite often as an "emphasis" bar in marches.

[3] See the **Direction** command (see page 93).

[4] The color of a chord are items like "minor", "major", etc. The current walking bass algorithm generates acceptable (uninispired) lines. If you want something better there is nothing stopping you from using a *Riff* to over-ride the computer generated pattern for important bars.

Each group consists of an beat offset, the note duration, and the note volume. MMA selects the actual note pitches to play based on the current chord (you cannot change this).

```
Walk Define Walk4 1 4 100 ; \
    2 4 90; \
    3 4 90
```

Example 4.4: Walking Bass Definition

Example 4.4 plays a bass note on beats 1, 2 and 3 of a bar in $\frac{3}{4}$ time.

4.1.5 Scale

A scale pattern is defined with:

```
    Position Duration Volume ; ...
```

Each group consists of an beat offset for the start point, the note duration, and volume.

```
Scale Define S1 1 1 90
Scale Define S4 S1 * 4
Scale Define S8 S1 * 8
```

Example 4.5: Scale Definition

Example 4.5 defines three scale patterns: “S1” is just a single whole note, not that useful on its own, but it used as a base for “S4” and “S8”.

“S4” is 4 quarter notes and “S8” is 8 eight notes. All the volumes are set to a MIDI velocity of 90.

Scale patterns are quite useful in endings. More options for scales detailed in the *ScaleDirection* (see page 93) and *ScaleType* (see page 98) sections.

4.1.6 Drum

Drum tracks are a bit different from the other tracks discussed so far. Instead of having each track saved as a separate MIDI track, all the drum tracks are combined onto MIDI track 10.

A Drum pattern is defined with:

```
    Position Duration Volume; ...
```

```
Drum Define S2 1 0 100; \
    2 0 80 ; \
    3 0 100 ; \
    4 0 80
```

Example 4.6: Drum Definition

Example 4.6 plays a drum sound on beats 1, 2, 3 and 4 of a bar in $\frac{4}{4}$ time. The MIDI velocity (volume) of the drum is 100 on beats 1 and 3; 80 on beats 2 and 4.

In this example we have used the special duration of "0" which indicates 1 MIDI tick.

4.1.7 Drum Tone

Essential to drum definitions is the **Tone** directive.

When a drum pattern is defined, there is no drum tone or note specified in the pattern.. By default, all drum patterns use a snare drum sound. But, this can (and should) be changed using the **Tone** directive. This is normally issued at the same time as a sequence is set up (see chapter 5).

Tone is a list of drum sounds which match the sequence length. Here's a short, concocted example (see the library files for many more):

```
Drum Define S1 1 0 90
Drum Define S2 S1 * 2
Drum Define S4 S1 * 4
SeqClear
SeqSize 4
Drum Sequence S4 S2 S2 S4
Drum Tone SnareDrum1 SideKick LowTom1 Slap
```

Here we first define the drum patterns "S2" to sound a drum on beats 1 and 3 and "S4" to sound on beats 1, 2, 3 and 4 (see section 4.3 for details on the "*" option). Next we set a sequence size of 4 bars and set a drum sequence to use this pattern. Finally, we instruct MMA to use a SnareDrum1 sound in bar 1, a SideKick sound in bar 2, a LowTom1 in bar 3 and a Slap in bar 4. If the song has more than four bars, this sequence will be repeated.

In most cases you will probably use a single drum tone name for the entire sequence, but it can be useful to alternate the tone between bars.

To repeat the same "tone" in a sequence list, use a single "/".

The "tone" can be specified with a MIDI note value or with a symbolic name. For example, a snare drum could be specified as "38" or "SnareDrum1". Appendix A.3 lists all the defined symbolic names.

4.2 Including Existing Patterns in New Definitions

When defining a pattern, you can use an existing pattern name in place of a definition grouping. For example, if we have already defined a chord pattern (which is played on beats 1 and 3) as:

```
Chord Define M13 1 4 80; 3 4 80
```

We can create a new pattern which plays on same beats and adds a single push note just before the third beat:

```
Chord Define M1+3 M13; 2.5 16 80 0
```

A few points to note:

- ♫ the existing pattern must exist and belong to the same track,
- ♫ the existing pattern is expanded in place,
- ♫ it is perfectly acceptable to have several existing definitions, just be sure to delimit each with a ";",
- ♫ the order of items in a definition does not matter, each will be placed at the correct position in the bar.

This is a powerful shortcut in creating patterns. See the included library files for examples.

4.3 Multiplying and Shifting Patterns

Since most pattern definitions are, internally, repetitious, you can create complex rhythms by multiplying a copy of an existing pattern. For example, if you have defined a pattern to play a chord on beats 1 though 4 (a quarter note strum), you can easily create a similar pattern to play eighth note chords on beats 1, 1.5, etc. though 4.5 with a command like:

```
Track Define NewPattern OldPattern * N
```

where "Track" is a valid track name ("Chord", "Walk", "Bass", "Arpeggio" or "Drum", as well as "Chord2" or "DRUM3", etc.).

The "*" is absolutely required.

"N" can be any integer value between 2 and 100.

In example 4.7 we start by defining a Drum pattern which plays a drum tone on beat 1 (assuming $\frac{4}{4}$ time). We then derive a new pattern, "S13" which is the old "S1" multiplied by 2. This new pattern will play a tone on beats 1 and 3.

Next, "S1234" is created. This plays 4 notes on the each beat.

Note the definition for "S64". We could have multiplied "S32" by 2, but for illustrative purposes have used "S1" and multiplied it by 64.

```
Drum Define S1 1 1 100
Drum Define S13 S1 * 2
Drum Define S1234 S2 * 2
Drum Define S8 S1234 * 2
Drum Define S16 S8 * 2
Drum Define S32 S16 * 2
Drum Define S64 S1 * 64
```

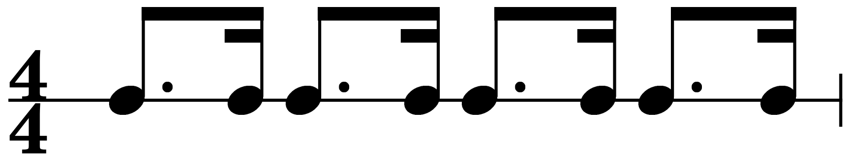
Example 4.7: Multiply Define

When MMA multiplies an existing pattern it will (usually) do what you expect. The start positions for all notes are adjusted to the new positions; the length of all the notes are adjusted (quarter notes become eighth notes, etc.). No changes are made to note offsets or volumes.

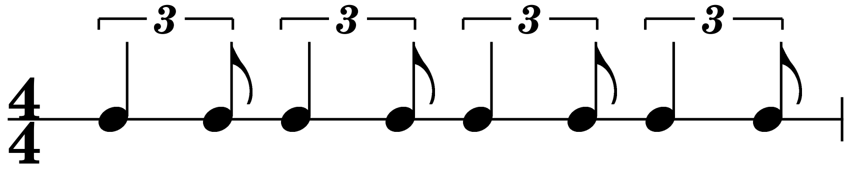
Example 4.8 shows how to get a swing pattern which might be useful on a snare drum.

```
Begin Drum Define
    SB8 1 2+16 0 90 ; 3.66 4+32 80
    SB8 SB8 * 4
End
```

Sheet Music Equivalent, Normal Notation



Sheet Music Equivalent, Actual Rhythm



Example 4.8: Swing Beat Drum Definition

To see the effects of multiplying patterns, create a simple test file and process it though MMA with the "-p" option.

Even cooler[5] is combining a multiplier, and existing pattern and a new pattern all in one statment. The following is quite legal (and useful):

```
    Drum Define D1234 1 0 90 * 4
```

which creates drum hits on beats 1, 2, 3 and 4.

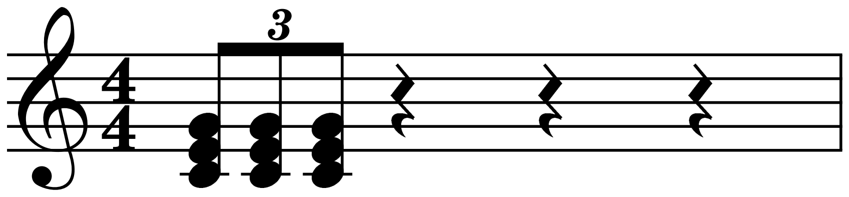
More contrived (but we need examples) is:

```
    Drum Define Dfunny D1234 * 2; 1.5 0 70 * 2
```

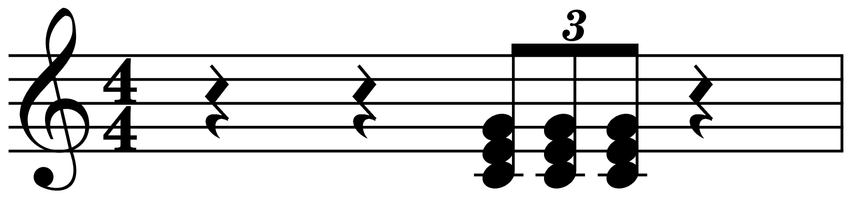
If you're really interested in the result, run MMA with the "-p" option with the above definition.

An existing pattern can be modified by *shifting* it a beat, or portion of a beat. This is done in a MMA definition with the **Shift** directive. Example 4.9 shows a triplet pattern created to play on beat 1, and then a second pattern played on beat 3.

```
Chord Define C1-3 1 3 90; \
    1.33 3 90; 1.66 3 90
```



```
Chord Define C3-3 C1-3 Shift 2
```



Example 4.9: Shift Pattern Definition

Note that the shift factor can be a negative or positive value. It can be fractional. Just be sure that the factor doesn't force the note placement to be less than 1 or greater than the **Time** setting.

And, just like the multiplier discussed earlier you can shift patterns as they are defined. And shifts and multipliers can be combined. So, to define a series of quarter notes on the offbeat you could use:

```
    Drum Define D1234' 1 0 90 * 4 Shift .5
```

which would create the same pattern as the longer:

[5] In this case the word "cool" substitutes for the more correct "useful".

```
Drum Define D1234’ 1.5 1 90; 2.5 1 90; 3.5 1 90; 4.5 1 90
```

Chapter 5

Sequences

Patterns by themselves don't do much good. They have to be combined into sequences to be of any use to you or to MMA.

A **sequence** command sets the pattern(s) used in creating each track in your song:

```
    Track Sequence Pattern1 Pattern2 ...
```

"Track" can be any valid track name: "Chord", "Walk", "Walk-Sus", "Arpeggio-88", etc.

All pattern names used when setting a sequence need to be defined when this command is issued; or you can use what appears to be a pattern definition right in the sequence command by enclosing the pattern definition in a set of curly brackets "{ }".

```
SeqClear
SeqSize 2
Begin Drum
    Sequence Snare4
    Tone Snaredrum1
End
Begin Drum-1
    Sequence Bass1 Bass2
    Tone KickDrum2
End
Chord Sequence Broken8
Bass Sequence Broken8
Arpeggio Sequence { 1 1 100 * 8 } { 1 1
   80 * 4 }
```

Example 5.1: Simple Sequence

Example 5.1 creates a 2 bar pattern. The Drum, Chord and Bass patterns repeat on every bar; the Drum-1 sequence repeats after 2 bars. Note how the Arpeggio pattern is defined at run-time. [1]

[1] If you run MMA with the "-s" option you'll see pattern names in the format "_1". The leading underscore indicates that the pattern was dynamically created in the sequence.

If there are fewer patterns than *SeqSize*, the sequence will be filled out to correct size. If the number of patterns used is greater than *SeqSize* (see Chapter 16) a warning message will be printed and the pattern list will be truncated.

When defining longer sequences, you can use the “repeat” symbol, a single “/”, to save typing. For example, the following two lines are equivalent:

```
Bass Sequence Bass1 Bass1 Bass2 Bass2
Bass Sequence Bass1 / Bass2 /
```

The special pattern name “-” (no quotes, just a single hyphen), or a single “z” can be used to turn a track off. For example, if we have set the sequences in example 5.1 and decide to delete the Bass halfway though the song we could:

```
Bass Sequence -
```

The special sequences, “-” or“z”, are also the equivalent of a rest or “tacet” sequence. For example, in defining a 4 bar sequence with a 1-5 bass pattern on the first 3 bars and a walking bass on bar 4 we might do something like:

```
Bass Sequence Bass4-13 / / z
Walk Sequence z / / Walk4-4
```

When a sequence is created a series of pointers to the existing patterns are created. If you change the definition of a particular pattern later in your file the new definition will have *no* effect on your exisiting sequences.

Sequences are the workhorse of MMA. With them you can set up many interesting patterns and variations. This chapter should certainly give more detail and many more examples.

The following commands help manipulate sequences in your creations:

5.1 SeqClear

This command clears all existing sequences from memory. It is useful when defining a new sequence and you want to be sure that no “leftover” sequences are active. The command:

```
SeqClear
```

deletes all sequence information.

Alternately, the command:

```
Drum SeqClear
```

deletes **all** drum sequences. This includes the track “Drum”, “Drum1”, etc.

If you use a sub-track:

```
Chord-Piano SeqClear
```

only the sequence for that track is cleared.[2]

In addition to clearing the sequence pattern, the following other settings are restored to a default condition:

- ♫ Track Invert setting,
- ♫ Track Sequence Rnd setting,
- ♫ Track MidiSeq setting,
- ♫ Track octave,
- ♫ Track voice,
- ♫ Track Rvolume,
- ♫ Track Volume,
- ♫ Track RTime,
- ♫ Track Strum.

CAUTION: It is not possible to clear only **Drum**, **Chord**, etc. using this command. Use the "-" option.

5.2 SeqRnd

Normally, the patterns used for each bar are selected in order. For example, if you had a sequence:

```
Drum-2 Sequence P1 P2 P3 z
```

bar 1 would use "P1", bar 2 "P2", etc. However, if you set **SeqRnd** for a specific track, the pattern used *for that track* will be selected at random from the sequence list. Note that 'Z' bars **are** included in the selection. Due to the nature of random selection, it is quite possible to get a several bars with the same (or in the above case, no) pattern.

You can only use this command in a track or in a global context:

```
Drum SeqRnd
```

or

```
SeqRnd
```

The latter example is interesting. Let us assume you have the following sequences defined (the contents of the patterns don't matter for the purpose of the example):

```
Chord C1 C2 C3 C4
Bass B1 B2 B3 z
```

[2]It is probably easier to use the command:

```
Chord-Piano Sequence -
```

if that is what you want to do. In this case **only** sequence pattern is cleared.

```
Walk z / / W1
```

The idea of the **Bass** and **Walk** sequences is to play *either* one of the patterns, never both. If you were to randomize the tracks you might get a bar with no bass at all, one of the two, or none. However, if you set **SeqRnd** outside the tracks, then you will have one of the following patterns:

C1	B1	z
C2	B2	z
C3	B3	z
C4	z	W1

A **SeqRnd** is cleared by a **SeqClear** or a **SeqNoRnd** directive.

If you have set **Invert** for a track, the inversions will follow the patterns. For example:

```
Chord Sequence C1 C2 C3 C4
Invert 0 1 2 3 SeqRnd
```

Whenever pattern "C1" is selected it will be used with inversion 0, "C2" will always be inversion 1, etc.

5.3 SeqNoRnd

This command sets the sequence order for the specified track to normal. It undoes the effect of the **SeqRnd** directive. Example:

```
Drum-3 SeqNoRnd
```

5.4 SeqSize

The number of bars in a sequence are set with the "SeqSize" command. For example:

```
SeqSize 4
```

sets it to 4 bars. The SeqSize applies to all tracks.

This command resets the **sequence counter** to 1.

If some sequences have already been defined, they will be truncated or expanded to the new size. Truncation is done by removing patterns from the end of the sequence; expansion is done by duplicating the sequence until it is long enough.

Chapter 6
Grooves

Grooves, in some ways, are MMA's answer to macros... but we think they are cooler, easier to use, and have a more musical name.

Really, though, a groove is just a simple mechanism for saving and restoring a set of patterns and sequences. Using grooves it is easy to create sequence libraries which can be incorporated into your songs with a single command.

6.1 Creating A Groove

A groove can be created at anytime in an input file with the command:

```
DefGroove SlowRhumba
```

Optionally, you can include a documentation string to the end of this command:

```
DefGroove SlowRumba A descriptive comment!
```

A groove name can include any character, including digits and punctuation. However, it cannot include a '/'[1].

In normal operation the documentation strings are ignored. However, when MMA is run with the -Dx command line option these strings are printed to the terminal screen in LaTeX format. The standard library document is generated from this data. The comments *must* be suitable for LaTeX: this means that special symbols like "#", "&", etc. must be "quoted" with a preceding "\".

At this point the following information is saved:

- Current Sequence size,
- The current sequence for each track,
- Time setting (quarter notes per bar),
- "Accent",
- "Articulation" settings for each track,
- "Compress",

[1]The '/' is reservered for future enhancements.

- “Direction”,
- “DupRoot”,
- “Duplicate”,
- “Harmony”
- “HarmonyOnly””,
- “Invert”,
- “Limit”,
- “MidiSeq”,
- “Octave”,
- “RSkip”,
- “Rtime”,
- “Rvolume”,
- “Scale”,
- “SeqRnd”, globally and for each track,
- “Strum”,
- “Tone” for drum tracks,
- “Voice”,
- “VoicingCenter”,
- “VoicingMode”,
- “VoicingMove”,
- “VoicingRange”,
- “Volume” for tracks and master.

6.2 Using A Groove

You can restore a previously defined groove a anytime in your song with:

```
Groove Name
```

At this point all of the previously saved information is restored.

A few cautions:

- ♫ Pattern definitions are *not* saved in grooves. Redefining a pattern results in a new pattern definition. Sequences use the pattern definition in effect when the sequence is declared.
- ♫ The "SeqSize" setting is restored with a groove. The sequence point is also reset to bar 1. If you have multi-bar sequences, restoring a groove may upset your idea of the sequence pattern.

To make the creation of variations easier, you can use **Groove** in a track setting:

```
Scale Groove Funny
```

In this case only the information saved in the corresponding **DefGroove Funny** for the **Scale** track will be restored. You might think of this as a "groove overlay". Have a look at the sample song "Yellow Bird" for an example.

When restoring track grooves, as in the above example, the **SeqSize** is not reset. The sequence size of the restored track is adjusted to fit the current sequence size setting.

If you are using a groove from a library file, you just need to do something like:

```
Groove Rhumba2
```

at the appropriate position in your input file.

One minor problem which *may* arise is that more than one library file has defined the same groove name. This might happen if you have a third-party library file. For the proposes of this example, lets assume that the standard library file "rhumba.mma" and a second file "xyz-rhumba.mma" both define the groove "Rhumba2". The auto-load routines (see page 109) which search the library database will load the first "Rhumba2" it finds, and the search order cannot be determined. To overcome this possible problem, do a explicit loading of the correct file. In this case, simply do:

```
Use xyz-rhumba
```

near the top of your file. And if you wish to switch to the groove defined in the standard file, you can always do:

```
Use rhumba
```

just before the groove call. The *Use* will read the specified file and overwrite the old definition of "Rhumba2" with its own.

Chapter 7

Riffs

In previous chapters we learned how to create a *Pattern* which becomes a part of a *Sequence*. And how to set a musical style by defining a *Groove*.

These predfined *Grooves* are wonderful things. And, yes, entire accompaniment tracks can be created with just some chords and few *Grooves*. But, often we want a bit of variety in the track.

The *Riff* command permits the setting of an alternate pattern for any track for a single bar–this overrides the current *Sequence* for that track.

The syntax for *Riff* is very similar to that of *Define*, with the execption that no pattern name is used. You might think of *Riff* as the setting of an *Sequence* with an anonymous pattern.

A *Riff* is set with the command:

```
Track Riff Pattern
```

where:

Track is any valid MMA track name,

Pattern is any existing pattern name defined for the specified track, or a pattern definition following the same syntax as a *Define*. In addition the pattern can be a single “z”, indicating no pattern for the specified track.

Following is a short example using *Riff* to change the Chord Pattern:

```
Groove Rhumba
1 Fm7
2 Bb7
3 EbM7
Chord Riff 1 4 100; 3 8 90; 3.666 8 80; 4.333 8 70
4 Eb6 / Eb
5 Fm7
```

In this case we have a Rhumba Groove for the song. But, in bar 4 we want to emphasize the melodic pattern by chording a quarter-note triplet over beats 3 and 4. In this case we have defined the pattern right in the *Riff* command.

Our next example shows that *Riff* patterns can be defined just like the patterns used in a sequence.

```
Begin Drum
```

```
  Define Emph1 1 0 128
  Define Emph8 Emph1 * 8
End

Groove Blues

1 C
2 G
Drum1 Riff Emph8
3 G
4 F
5 C
```

In this case we have defined the **Emph8** pattern as a series of eighth notes. We then apply this for the 3rd bar. If you compile and play this example you will hear a sporadic handclap on bar 3. The **Drum1** track is using a handclap tone with a random skip factor (previously defined in the Blues groove).

The special pattern "z" can be used to turn off a track for a single bar. This is similar to using a "z" in the *Sequence* directive.

A few things to keep in mind when using *Riffs*:

♫ A *Riff* is in effect for only one bar.

♫ *Riff* sequences are always enabled. Even if there is no sequence for a track, or if the "z" sequence is being used, the pattern specified in *Riff* will apply.

♫ The existing voicing, articulation, etc. for the track will apply to the *Riff*.

♫ It's quite possible to use a macro for repeated *Riff*s. In example 7.1 we have created a macro which sets the *Volume*, *Articulate*, etc. as well as the pattern. Note how the pattern is initially set as single whole note, but redfined in the *Riff* as a run controlled by another macro. In bar 2 an eight note run is played and in bar 5 this is changed to a run of triplets.

*Riff*s can also be used to specify a bar of music in a *Solo* or *Melody* track. Please see the "Solo and Melody" chapter (see page 46).

```
Mset CRiff
  Begin Scale
    Define Run 1 1 120
    Riff Run * $SSpeed
    Voice AltoSax
    Volume f
    Articulate 80
    Rskip 5
  End
MsetEnd
Groove Blues
1 C
Set SSpeed 8
$CRiff
2 G
3 G
Set SSpeed 12
$CRIFF
5 C
```

Example 7.1: Using Macros and Riffs

Chapter 8

Musical Data Format

Compared to patterns, sequences, grooves and the various directives used in MMA, the actual bar by bar chord notations are surprisingly simple.

Any line in your input file which is not a directive or comment is assumed to be a bar of chord data.

A line for chord data consists of the following parts:

- Optional line number,
- Chord or Rest data,
- Optional lyric data,
- Optional solo or melody data,
- Optional multiplier.

Formally, this becomes:

```
[num] Chord [Chord ...]  [lyric] [solo] [* Factor]
```

As you can see, all that is really needed is a single chord. So, the line:

```
Cm
```

is completely valid. As is:

```
10 Cm Dm Em Fm * 4
```

The optional solo or melody data is enclosed in "{ }". The complete format and use is detailed in the *Solo and Melody Tracks* chapter (see page 46).

8.1 Bar Numbers

The optional leading bar number is silently discarded by MMA. It is really just a specialized comment which helps you debug your music. Note that only a numeric item is permitted here.

Get in the habit of using bar numbers. You'll thank yourself when a song seems to be missing a bar, or appears to have an extra one. Without the leading bar numbers it can be quite frustrating to match your input file to a piece of sheet music.

You should note that it is perfectly acceptable to have only a bar number on a line. This is common when you are using bar repeat, for example:

```
1 Cm * 4
2
3
4
5 A
```

8.2 Bar Repeat

Quite often music has several sequential identical bars. Instead of typing these bars over and over again, MMA has an optional **multiplier** which can be placed at the end of a line of music data. The multiplier or factor can is specified as "* NN" This will cause the current bar to repeated the specified number of times. For example:

```
Cm / Dm / * 4
```

produces 4 bars of output with each the first 2 beats of each bar a Cm chord and the last 2 a Dm. (The "/" is explained below.)

8.3 Chords

The most important part of a musical data line is, of course, the chords. You can specify a different chord for each beat in your music. For example:

```
Cm Dm Em Fm
```

specifies four different chords in a bar. It should be obvious by now that in a piece in $\frac{4}{4}$ you'll end up with a "Cm" chord on beat 1, "Dm" on 2, etc.

If you have fewer chord names than beats, the bar will be filled automatically with the last chord name on the line. In other words:

```
Cm
```

and

```
Cm Cm Cm Cm
```

are equivalent (assuming 4 beats per bar). There must be one (or more) spaces between each chord.

One further shorthand is the "/". This simply means to repeat the last chord. So:

```
Cm / Dm /
```

is the same as

```
Cm Cm Dm Dm
```

It is perfectly okay to start a line with a "/". In this case the last chord from the previous line is used. If the first line of music data begins with a "/" you'll get an error—MMA tries to be smart, but it doesn't read minds.

MMA recognizes a wide variety of chords in standard notation. Refer to the complete table in the appendix for details (see page 117).

8.4 Rests

To disable a voice for a beat you can use a "z" for a chord name. If used by itself a "z" will disable all but the drum tracks for the given beat. However, you can disable "Chord", "Arpeggio", "Scale", "Walk" or "Bass" tracks as well by appending a track specifier to the "z". Track specifiers are the single letters "C", "A", "S", "W", "B" or 'D" and "!". Track specifiers are only valid if you also specify a chord. The track specifiers are:

D -	All drum tracks,
W -	All walking bass tracks,
B -	All bass tracks,
C -	All chord tracks,
A -	All arpeggio tracks,
S -	All scale tracks,
! -	All tracks (almost the same as DWBCA, see below).

Assuming the "C" is the chord and "AB" are the track specifiers:

CzAB -	mutes the "A" and "B" tracks,
z -	mutes all the tracks except for the drums,
Cz -	is not permitted,
zAB -	is not permitted.

Assuming that you have a drum, chord and bass pattern defined:

```
Fm z G7zC CmzD
```

would generate the following beats:

1 -	Drum pattern, Fm chord and bass,
2 -	Drum pattern only,
3 -	Drum pattern and G7 bass, no chord,
4 -	Cm chord and bass, no drum.

In addition, there is a super-z notation. "z!" forces all instruments to be silent for the given beats. "z!" is the same as "zABCDW", except that the later is not valid since it needs a prefixed chord.

The "z" notation is used when you have a "tacet" beat or beats. The alternate notations can be used to silence specific tracks for a beat or two, but this is used less frequently.

8.5 Case Sensitivity

In direct conflict with the rest of the rules for input files, all chord names *are* case sensitive. This means that you *can not* use notations like "cm"—use "Cm" instead.

The "z" and the associated track specifiers are also case sensitive. For example, the form "Zc" will *not* work!

8.6 Lyrics

MIDI files can include song lyrics. And some MIDI players or sequencers can display them as a file is played. Some, but not all.

We're not aware of any keyboards which display lyrics. And most Linux based do not display them. Exceptions to the rule are the programs **Kmid** which displays and highlights lyrics almost in a Karaoke manner, **xplaymidi** and **timidity** which display the lyrics in a secondary panel.

With this qualifier out of the way, there really is no reason for lyrics NOT to be useful in a program like MMA. Singers do not want a melody playing while they are vocalizing (really, they are no different in this than any other instrumentalist). And, it is our understanding that some platforms[1] other than Linux support lyric display in a more useful format.

The "Standard MIDI File" document describes a *Lyric* Meta-event:

> **FF 05 len text *Lyric.*** A lyric to be sung. Generally, each syllable will be a separate lyric event which begins at the event's time.[2]

Unfortunately, not all players and creators follow the specification—the most notable exception are ".kar" files. These files eschew the *Lyric* event and place their lyrics as a *Text Event*. There are programs strewn on the net which convert between the two formats, and this author doesn't really know if conversion is needed.

If you want to read the word from the source, refer to the official MIDI lyrics documentation at `http://www.midi.org/about-midi/smf/rp017.shtml`.

Just to be on the safe side, MMA supports both formats. The *Lyric* command is used to select the desired mode.

```
Lyric EVENT=LYRIC
```

selects the default *Lyric Event* mode.

```
Lyric EVENT=TEXT
```

[1]Pointers and reviews to other players would be would appreciated.

[2]I am quoting from "MIDI Documentation" distributed with the TSE Library. Pete Goodcliffe, Oct. 21, 1999. Page 41.

selects the *Text Event* mode. Use of this option also prints a warning message.

Another option controlled by the *Lyric* command is to determine the method used to split words. As mentioned earlier (and in various MIDI documents), the lyrics should be split into syllables. MMA does this by taking each word (ie. anything with whitespace surrounding it) and setting a MIDI event for that. However, depending on your player, you might want only one event per bar. You might even want to put the lyrics for several bars into one event. In this case simply set the "bar at a time" flag:

```
Lyric SPLIT=BAR
```

You can return to normal (syllable/word) mode at anytime with:

```
Lyric SPLIT=NORMAL
```

Adding a lyric to your song is a simple matter. Just place the text for the lyric for a bar in-between a pair of **[]**s somewhere in a data bar.[3] For example:

```
z [ Pardon ]
C [ me, If I'm ]
E7 [ sentimental, \r]
C [when we say good ]
```

The lyrics for each bar are separated into individual events, one for each word.

Note the difference in this example:

```
Lyric Split=Bar
z [ Pardon me, If I'm sentimental \r ]
C
E7
C [when we say good-bye ]
```

MMA recognizes two special characters in a *Lyric*:

- A **\r** is converted into an EOL character (hex value 0x0D). A **\r** should appear at the end of each lyrical line.
- A **\n** is converted into a LF character (hex value 0x0A). A **\n** should appear at the end of each verse or paragraph.

When a multi-verse section is created using a *Repeat* or *Goto*, different lyrics can be specified for different passes. In this case you simply specify two more sets of lyrics:

```
A / Am / [First verse] [Second Verse]
```

However, for this work properly you must set the internal counter *LyricVerse* for any verse other than 1. This counter is set with the command:

```
Lyric Verse=Value | INC | DEC
```

This means that you can directly set the value (the default value is 1) with a command like:

[3] Although the lyric can be placed anywhere in the bar, we recommend that you only place the lyric at the end of the bar. All the examples follow this style.

```
Lyric Verse=2
```

And you can increment or decrement the value with the *INC* and *DEC* options. This is handy at to use in repeat sections:

```
Lyric Verse=Inc
```

You cannot set the value to a value less than 1.

There are a couple of special cases:

- ♫ If there is only one set of lyrics in a line, it will be treated as text for verse 1, regardless of the value of *LyricVerse*.
- ♫ If the value of *LyricVerse* is greater than the number of verses found after splitting the line, then no lyrics are produced. In most cases this is probably not what you want.

At times you may wish to override MMA's method of determining the beat offsets for a lyric or a single syllable in a lyric. You can specify the beat in the bar by enclosing the value in "<>" brackets. For example, suppose that your song starts with a pickup bar and you'd like the lyrics for the first bar to start on beat 4:

```
z z z C [ <4>Hello ]
F [ Young lovers ]
```

Assuming $\frac{4}{4}$ the above would put the word "Hello" at beat 4 of the first bar; "Young" on the first beat of bar 2; and "lovers" on beat 3 of bar 2.

Note: there must not be a space inside the "<>", nor can there be a space between the bracket and the syllable it applies to.

If you really want to have "<>" in your lyric, you can include a dummy to keep MMA happy:

```
C [ <><Verse 1.>This is a Demo ]
```

Example 8.1 [4] shows a complete song with lyrics.

[4] Included in this distribution as `songs/twinkle.mma`.

```
Tempo 200
Groove Folk
Repeat
  1 G [Twinkle,] [When the]
  2 G [Twinkle] [blazing ]
  3 C [little] [sun is]
  4 G [star; \r] [gone, \r]
  5 Am [How I] [When he ]
  6 G [wonder] [nothing]
  7 D7 [what you] [shines u-]
  8 G [are.  \r] [pon.  \r]
  9 G [Up a-] [then you]
  10 D7 [bove the] [show your]
  11 G [world so] [little]
  12 D [high, \r] [light, \r]
  13 G [Like a] [Twinkle, ]
  14 D7 [diamond] [twinkle,]
  15 G [in the] [all the]
  16 D7 [sky!  \r] [night.  \r]
  17 G [Twinkle,]
  18 G [twinkle]
  19 C [Little]
  20 G [star, \r]
  21 Am [How I]
  22 G [wonder]
  23 D7 [what you]
  24 G [are.  \r \n]

  Lyric Verse=Inc
RepeatEnd
```

Example 8.1: Twinkle, Twinkle, Little Star

Chapter 9

Solo and Melody Tracks

So far we have discussed the creation of accompaniment tracks using drum and chord patterns. However, there are times when chording (and chord variations such as arpeggios) are not sufficient. Sometimes you might want a real melody line!

MMA has two internal track types reserved for melodic lines. They are the *Solo* and *Melody* tracks. These two track types are identical with two major exceptions:

- ♫ *Solo* tracks are only initialized once, at startup. Commands like *SeqClear* are ignored by *Solo* tracks.
- ♫ No settings in *Solo* tracks are saved or restored with *Groove* commands.

These differences mean that you can set parameters for a *Solo* track in a preamble in your music file and have those settings valid for the entire song. For example, you may want to set an instrument at the top of a song:

```
Solo Voice TenorSax
```

On the other hand, *Melody* tracks save and restore grooves just like all the other available tracks. If we have the following sequence in a song file:

```
Melody Voice TenorSax
Groove Blues
...  musical data
```

we should not be surprised to find that the *Melody* track playing with the default voice (Piano).

As a general rule, we have designed *Melody* tracks as a “voice” to accompany a predefined form defined in a *Groove*—it is a good idea to define *Melody* paramaters as part of a *Groove*. *Solo* tracks are thought to be specific to a certain song file, with their parameters defined in the song file.

Apart from the exceptions noted above, *Solo* and *Melody* tracks are identical.

Unlike the other available tracks, you do not define a sequence or pattern for a *Solo* or *Melody* track. Instead, you specify a series of notes as a *Riff* pattern. For example, consider the first two bars of “Bill Bailey” (the details of melody notation will be covered later in this chapter):

```
Solo Riff 4c;2d;4f;
F
Solo Riff 4.a;8g#;4a;4c+;
F
```

In this example we have added the melody to our song file.

Specifying a *Riff* for each bar of your song can get tedious, so there is a shortcut . . . any data surrounded by curly brackets "{ }" is interpeted as a *Riff* for a *Solo* or *Melody* track. This means that the above example could be rewritten as:

```
F {4c;2d;4f;}
F {4.a;8g#;4a;4c+;}
```

By default the note data is inserted into the *Solo* track. If more than one set of note data is present, it will be inserted into the next track set by the *AutoSoloTracks* command (see page 49).

9.1 Note Data Format

The notes is a *Solo* or *Melody* track are specified as a series of "chords". Each chord can be a single note, or several notes (all with the same duration). Each chord in the bar is delimited with a single semicolon.

Each chord can have several parts. All missing parts will default to the value in the previous chord. The various parts of a chord must be specified in the order given in the following table.

Duration The duration of the note. This is specified in the same manner as chord patterns. The following note values are permitted:

Notation	*Description*
1	Whole note
2	Half
4	Quarter
8	Eighth
16	Sixteenth
32	Thirtysecond
64	Sixtyfourth
3	One note of an eight note triplet
0	A single MIDI tick

A duration can be modified by appending a single "." which adds half the value to the note. For example, "2." would be three beats.

A duration can be modified by appending a two "."s which add three quarters of the value to the note. For example, "2.." would be three and one half beats.

Note lengths can be combined using "+". For example, to make a dotted eight note use the notation "8+16", a dotted half "2+4", and a quarter triplet "3+3".

It is permissible to combine notes with "dots" and "+"s. The notation "2.+4" would be the same as a whole note.

Pitch The note in standard musical notation. The lowercase letters “a” to “g” are recognized as well as “r” to specify a rest (please note the exception for *Drum Solo Tracks*, see page 49).

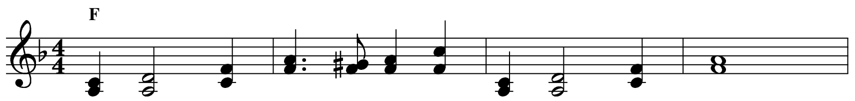
Accidental A pitch modifier consisting of a single “#” (sharp), “&” (flat) or “n” (natural). Please note that an accidental will override the current *Keysig* for the current bar (just like in real musical notation). Unlike standard musical notation the accidental **will** apply to similarly named notes in different octaves.

To avoid confusion between a flat sign and a “b” we have changed the flat notation to an “&” character.

Octave Without an octave modifier, the current octave specified by the *Octave* directive is used for the pitch(es). Any number of “-” or “+” signs can be appended to a note. Each “-” drops the note by an octave and each “+” will increase it. The base octave begins with “c” below the treble clef staff.

Volume A volume can be specified. The volume is a string like “ff” surrounded by “<>” brackets. For example, to set the volume of a chord to “very loud”, you could use the string <ffff> in the chord specification (see page 66) Of course, it is probably easier to set accented beats with the *Accent* directive (see page 67).

To make your note data more readable, you can include any number of space and tab characters (which are ignored by MMA).



```
KeySig 1b
F { 4ca-; 2da-; 4fd; }
F { 4.af; 8g#f; 4af; c+f; }
F { 4ca-; 2da-; 4fc; }
F { 1af; }
```

Example 9.1: Solo Notation

Example 9.1 shows a few bars of “Bill Bailey” with the MMA equivalent.

A few notes on duration:

♫ If you have a note tied into a new bar in your music score you can specify a note duration which creates a note ending past the current bar end. For example, if you have a bar with a 2 half notes, and the second one is tied to a half note in the next bar you might want something like:

```
Cm { 2a; 1b; }
F { 2r; 4a; b; }
```

Here we use a rest in the second bar to compensate for the extended duration of the preceeding note.

- ♫ Any notes which extend into the next bar will be reported in a warning message.
- ♫ Notes cannot start past the end of the of the current bar.

The use of default values can be a great timesaver, and lead to confusion! For example, the following all generate four quarter note "f"s:

```
Solo Riff 4f; 4f; 4f; 4f;
Solo Riff 4f; f; f; f;
Solo Riff 4f; 4; 4; 4;
Solo Riff 4f; ; ; ;
```

Most of the timing and volume commands available in other tracks also apply to *Solo* and *Melody* tracks. Important commands to consider include *Articulate*, *Voice* and *Octave*. Also note that *Transpose* is applied to your note data.

9.2 KeySig

If you are including *Solo* or *Melody* tracks you should set the key signature for the song:

```
KeySig 2b
```

The argument consists of a single digit "0" to "7" followed by a "b" or "&" for flats keys or a "#" for sharp keys.

Setting the key signature effects the notes used in *Solo* or *Melody* tracks and sets a MIDI Key Signature event.

9.3 AutoSoloTracks

When a "{ }" expression is found in a chord line, it is assumed to be note data and is treated as a *Riff*. You can have any number of "{ }" expressions in a chord line. They will be assigned to the tracks specified in the *AutoSoloTracks* directive.

By default, four tracks are assigned: **Solo**, **Solo-1**, **Solo-2**, and **Solo-3**. This order can be changed:

```
AutoSoloTracks Melody-Oboe Melody-Trumpet Melody-Horn
```

Any number of tracks can be specified in this command, but they must all be *Solo* or *Melody* tracks. You can reissue this command at any time to change the assignments.

9.4 Drum Solo Tracks

A solo or melody track can also be used to create drum solos. The first thing to do is to set a track as a drum solo type:

```
Solo-MyDrums DrumType
```

This will create a new *Solo* track with the name *Solo-MyDrums* and set its “Drum” flag. If the track already exists and has data in it, the command will fail. The MIDI channel 10 is automatically assigned to all tracks created in this manner. You cannot change a “drum” track back to a normal track.

These is no limit to the number of *Solo* or *Melody* tracks you can create . . . and it probably makes sense to have several different tracks if you are creating anything beyond a simple drum pattern.

Tracks with the “drum” setting ignore *Transpose* and *Harmony* settings.

The specification for pitches is different in these tracks. Instead of standard notation pitches, you must specify a series of drum tone names or MIDI values. If you want more than one tone to be sounded simultaneously, create a list of tones separated by commas.

Some examples:

```
Solo-MyDrums Riff 4 SnareDrum1; ; r ; SnareDrum1;
```

would create a snare hit on beats 1, 2 and 4 of a bar. Note how the second hit uses the default tone set in the first beat.

```
Solo-MyDrums Riff 8,38;;;;
```

creates 4 hits, starting on beat 1. Instead of “names” we have used MIDI values (in this case, 38 and “SnareDrum1” are identical. Note how we use a “,” to separate the initial length from the first tone.

```
Solo-MyDrums Riff 4 SnareDrum1,53,81; r; 4 SideKick ;
```

creates a “chord” of 3 tones on beat 1, a rest on beat 2, and a “SideKick” on beat 3.

Using MIDI values instead of names lets you use the full range of note values from 0 to 127. Not all will produce valid tones on all synths.

9.5 Mallet

Some instruments (Steel-drums, banjos, marimbas, etc.) are normally played with rapidly repeating notes. Instead of painfully inserting long lists of these notes, you can use the *Mallet* directive for a *Solo* or *Melody* track. The *Mallet* directive accepts a number of options, each a OPTION=VALUE pair. For example:

```
Solo-Marimba Mallet Rate=16 Decay=-5
```

The following options are supported:

9.5.1 Rate

The *Rate* must be a valid note length (ie. 8, 16, or even 16.+8).

For example:

```
    Solo-Marimba Mallet Rate=16
```

will set all the notes in the “Solo-Marimba” track to be sounded a series of 16th notes.

- ♫ Note duration modifiers such as articulate are applied to each resultent note,
- ♫ It is guaranteed that the note will sound at least once,
- ♫ The use of note lengths assures a consitant sound independent of the song tempo.

To disable this setting use a value of “0”.

9.5.2 Decay

You can adjust the volume (velocity) of the notes being repeated when *Mallet* is enabled:

```
    Solo-Mallet Mallet Decay=-15
```

The argument is a percentage of the current value to add to the note each time it is struck. In this example, assuming that the note length calls for 4 “strikes” and the initial velocity is 100, the note will be struck with a velocity of 100, 85, 73 and 63.

Important: a positive value will cause the notes to get louder, negative values cause the notes to get softer.

Note velocities will never go below 1 or above 255.

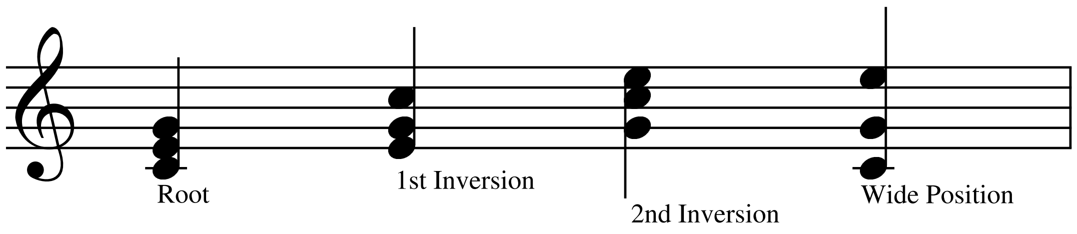
The decay option value must be in the range -20 to 20. The default value is 0 (no decay).

Chapter 10

Chord Voicing

In music, a chord is simply defined as two more notes played simultaneously. Now, this doesn't mean that you can play just any two or three notes and get a chord which sounds nice—but whatever you do get will be a chord of some type. And, to further confuse the unwary, different arrangements of the same notes sound better (or worse) in different musical situations.

As a simple example, consider a C major chord. Built on the first, third and fifth notes of a C major scale it can be manipulated into a variety of sounds:



These are all C major chords ... but they all have a different sound or color. The different forms a chord can take are called "voicings". Again, this manual is not intended to be a primer on musical theory—that's a subject way beyond our abilities, and (again) we really recommend your favorite music teacher and the study of basic music theory if you want to understand how and why MMA creates its tracks.

The different options in this chapter effect not only the way chords are constructed, but also the way bass lines and other tracks are formed.

There are generally two ways in MMA to take care of voicings.

1. use MMA's extensive *Voicing* options, most likely with the *"Optimal"* voicing algorithm,
2. do everything by yourself with the commands *Invert* and *Compress*.

The commands *Limit* and *DupRoot* may be used independently for both variants.

10.1 Voicing

The *Voicing* command is use to set the voicing mode and several other options relating to the selected mode. The command needs to have a *Chord* track specified and a series of Option=Value pairs. For example:

```
Chord-Piano Voicing Mode=Optimal Rmove=10 Range=9
```

In the following sections we will cover all the options available.

10.1.1 Voicing Mode

The easiest way to deal with chord voicings is to via the *Voicing Mode=XX* option.

When choosing the inversion of a chord to play an accompanist will take into consideration the style of the piece and the chord sequences. In a general sense, this is referred to as "voicing".

A large number of the library files have been written to take advantage of the following voicing commands. However, not all styles of music take well to the concept. And, don't forget about the other commands since they are useful in manipulating bass lines, as well as other chord tracks (eg. sustained strings).

MMA has a variety of sophisticated, intelligent algorithms[1] to deal with voicing.

As a general rule you should not use the *Invert* and *Compress* commands in conjunction with the *Voicing* command. If you do, you may create beautiful sounds. But, the results are more likely to be less-than-pleasing. Use of voicing and other combinations will display various warning messages.

The main command to enable voicings is:

```
Chord Voicing Mode=Type
```

As mentioned above, this command can only be applied to *Chord* tracks. Also note that this effects all bars in the sequence ... you cannot have different voicings for different bars in the sequence (attempting to do this would make no sense).

The following MODE types are available:

Optimal A basic algorithm which automatically chooses the best sounding voicing depending on the voicing played before. Always try this option before anything else. It might work just fine without further work.

The idea behind this algorithm is to keep voicings in a sequence close together. A pianist leaves his or her fingers where they are, if they still fit the next chord. Then, the notes closest to the fingers are selected for the next chord. This way characteristic notes are emphasized.

Root This Option may for example be used to turn off *Voicing* within a song. *Voicing Mode=Root* means nothing else than doing nothing, leaving all chords in root position.

None This is the same as the *Root* option.

Invert Rather than basing the inversion selection on an analysis of past chords, this method quite stupidly tries to keep chords around the base point of "C" by inverting "G" and "A" chords upward and "D", "E" and "F" downward. The chords are also compressed. Certainly not an ideal algorithm, but it can be used to add variety in a piece.

[1]Great thanks are due to Alain Brenzikofer who not only pressured me into including the *Voicing* options, but wrote a great deal of the actual code.

Compressed Does the same as the stand-alone *Compress* command. Like *Root*, it is only added to be used in some parts of a song where *Voicing Mode=Optimal* is used.

10.1.2 Voicing Range

To get wider or closer voicings, you may define a range for the voicings. This can be adjusted with the *Range* option:

```
Chord-Guitar Voicing Mode=Optimal Range=12
```

In most cases the default value of 12 should work just fine. But, you may want to fine tune . . . it's all up to you. This command only effects chords created with *Mode=Optimal*.

10.1.3 Voicing Center

Just minimizing the Euclidean distance between chords doesn't do the trick as there could be runaway progressions that let the voicings drift up or down infinitely.

When a chord is "voiced" or moved to a new position, a "center point" must be used as a base. By default, the fourth degree of the scale corresponding to the chord is a reasonable choice. However, you can change this with:

```
Chord-1 Voicing Center=<value>
```

The *value* in this command can be any number in the range 0 to 12. Try different values. The color of your whole song might change.

Note that the value is the note in the scale, not a chord-note position.

This command only effects chords created with *Mode=Optimal*.

10.1.4 Voicing Move

To intensify a chord progression you may want to have ascending or descending movement of voicings. This option, in conjunction with the *Dir* optional (see below) sets the number of bars over which a movment is done.

For the *Move* option to have any effect you must also set the direction to either -1 or 1. Be careful that you don't force the chord too high or low on the scale. Use of this command in a *Repeat* section can cause unexpected results. For this reason we suggest that you include an *Seq* command at the beginning of repeated sections of your songs.

In most cases the use of this command is limited to a section of a song, its use is not recommended in groove files. You might want to do something like this in a song:

```
..select groove with voicing
chords..
Chord-Piano Voicing Move=5 Dir=1
more chords..
Chord-Piano Voicing Move=5 Dir=-1
more chords..
```

10.1.5 Voicing Dir

This option is used in conjunction with the *Move* option to set the direction (-1 or 1) of the movement.

10.1.6 Voicing Rmove

As an alternate to movement in a specified direction, random movement can add some color and variety to your songs. The command option is quite useful (and safe to use) in groove files. The argument for this option is a percentage value specifying the frequency to apply a move in a random direction.

For example:

```
Chord-3 Voicing Mode=Optimal Rmove=20
```

would cause a movement (randomly up or down) in 20% of the bars. As noted earlier, using explicit movement instructions can move the chord into an undesirable range or even "off the keyboard"; however, the algorithm used in RMOVE has a sanity check to ensure that the chord center position remains, approximately, in a two octave range.

10.2 Compress

When MMA grabs the notes for a chord, the notes are spread out from the root position. This means that if you specify a "C13" you will have an "A" nearly 2 octaves above the root note as part of the chord. Depending on your instrumentation, pattern, and the chord structure of your piece, notes outside of the "normal" single octave range for a chord *may* sound strange.

```
Chord Compress 1
```

Forces MMA to put all chord notes in a single octave range.

The effects vary from track to track:

Drum, **Scale**, **Bass**, **Solo/Melody** and **Walking**: No effect.

Chord, **Arpeggio** and **Bass**: High notes in the chord are reduced by one octave (this means that the complete range of a chord will be limited to one octave).

Notes: **Compress** takes any value between 1 and 5 as arguments (however, some values will have no effect as detailed above). You can specify a different **Compress** for each bar in a sequence. Repeated values can be represented with a "/":

```
Chord Compress 1 / 0 /
```

To restore to its default (off) setting, use a "0" as the argument.

For a similar command, with different results, see the **Limit** command (see page 57).

10.3 DupRoot

To add a bit of fullness to chords, it is quite common of keyboard players to duplicate the root tone of a chord into a lower (or higher) octave. This is accomplished in MMA with the command:

```
DupRoot -1 1 -1 1
```

The command determines whether or not the root tone of a chord is duplicated in another octave. By default notes are not added. A value of -1 will add a note one octave lower than the root note, -2 will add the tone 2 octaves lower, etc. Similarly, the value of 1 will add a note one octave higher than the root tone, etc.

Only the values -9 to 9 are permitted.

Different values can be used in each bar of the sequence.

The option is reset to 0 after all *Sequence* or *SeqClear* commands.

The *DupRoot* command is only valid in *Chord* tracks. A similar command is *Duplicate* (see page 93).

10.4 Invert

By default MMA uses chords in the root position. By example, the notes of a C major chord are C, E and G. Chords can be inverted (something musicians do all the time). Sticking with our C major chord, the first inversion shifts the root note up an octave and the chord becomes E, G and C. The second inversion is G, C and E.

MMA extends the concept of inversion a bit by permitting the shift to be to the left or right, and the number of shifts is not limited. So, you could shift a chord up several octaves by using large invert values.[2]

Inversions apply to each bar of a sequence. So, the following is a good example:

```
SeqSize 4
Chord-1 Sequence STR1
```

[2]We've used the term "shift" here, but that's not quite what MMA does. The order of the notes in the internal buffer stays the same, just the octave for the notes is changed. So, if the chord notes are "C E G" with the MIDI values "0, 4, 7" an invert of 1 would change the notes to "C^2 E G" and the MIDI values to "12, 4, 7".

```
    Chord-1 Invert 0 1 0 1
```

Here we set the sequence pattern size to 4 bars and set the pattern for each bar in the Chord-1 track to "STR1". Without the next line, this would result in a rather boring, repeating pattern. But, the Invert command forces the chord to be in the root position for the first bar, the first inversion for the second, etc.

You can use a negative Invert value:

```
    Chord-1 Invert -1
```

In this case the C major chord becomes G, C and E.

Note that using fewer Invert arguments than the current sequence size is permitted. MMA simply expands the number of arguments to the current sequence size. You may use a "/" for a repeated value.

A **Sequence** or **ClearSeq** command resets **Invert** to 0.

Invert can be used in any track context, however, it does not effect **Drum** or **Walk** tracks.

Let's see what happens with a **Bass** track that's been inverted. First off the pattern and sequence definitions:

```
    Bass define X1 0 4 0 60; 25 4 2 60; 50 4 0 60; 75 4 2 60
    Bass Sequence X1
    Bass Invert 0 1 0 1
```

The define says to play quarter notes on each beat. On beats 1 and 3 we play the root (0 is the first note in the chord); on beats 2 and 4 we play the fifth (2 is the 2nd note in the chord and is the fifth of the scale). If you are expecting different notes to be played on the second and fourth bars you'll be (somewhat) disappointed. No matter what inversion is used the first note of the chord remains the root—only the octave of the note will be changed.

Frankly, **Arpeggios** sound a bit odd with inversions.

If you use a large value for **Invert** you can force the notes out of the normal MIDI range. In this case the lowest or highest possible MIDI note value will be used.

10.5 Limit

If you use "jazz" chords in your piece, some people might not like the results. To some folks, chords like 11th, 13th, and variations have a dissonant sound. And, sometimes they are in a chart, but don't really make sense. The **Limit** command can be used to set the number of notes of a chord used.

For example:

```
    Chord Limit 4
```

will limit any chords used in the **Chord** track to the first 4 notes of a chord. So, if you have a C11 chord which is C, E, G, B♭, D, and F, the chord will be truncated to C, E, G and B♭.

This command only applies to **Bass**, **Chord** and **Arpeggio** tracks. It can be set for other tracks, but the setting will have no effect.

Notes: **Limit** takes any value between 0 and 8 as an argument. The “0” argument will disable the command. This command applies to all chords in the sequence—only one value can be given in the command.

To restore to its default (off) setting, use a “0” as the argument.

For a similar command, with different results, see the **Compress** command (see page 55).

10.6 Range

For *Arpeggio* and *Scale* tracks you can specify the number of octave used. The effects of the *Range* command is slightly different between the two.

Scale: Scale tracks, by default, create three octave scales. The **Range** value will modify this to the number of octaves specified. For example:

```
    Scale Range 1
```

will force the scales to one octave. A value of 4 would create 4 octave scales, etc.

Arpeggio: Normally, arpeggios use a single octave (really, they use whatever notes are in the chord, which might exceed the octave). Using the *Range* command we specify the number of octaves to use. The values of “0” and ”1” have the same effect.

Chapter 11

Tempo and Timing

MMA has a rich set of commands to adjust and vary the timing of your song.

11.1 Tempo

The tempo of a piece is set in Beats per Minute with the “Tempo” directive.

```
    Tempo 120
```

sets the tempo to 120 beats/minute. You can also use the tempo command to increase or decrease the current rate by including a leading “+”, “-” or “*” in the rate. For example (assuming the current rate is 120):

```
    Tempo + 10
```

will increase the current rate to 130 beats/minute.

The tempo can be changed series of beats, much like a rit. or acc. in real music. Assuming that we are in $\frac{4}{4}$, the current tempo is 120, and there are 4 beats in a bar, the command:

```
    Tempo 100 1
```

will cause 4 tempo entries to be placed in the current bar (in the MIDI meta track). The start of the bar will be 115, the 2nd beat will be at 110, the 3rd at 105 and the last at 100.

You can also vary an existing rate using a “+”, ”-” or “*” in the rate.

You can vary the tempo over more than one bar. For example:

```
    Tempo + 20 5.5
```

tells MMA to increase the tempo by 20 beats per minute and to step the increase over the next five and a half bars. Assuming a start tempo of 100 and 4 beats/bar, the meta track will have a tempo settings of 101, 102, 103 . . . 120. This will occur over 22 beats (5.5 bars * 4 beats) of music.

Using the multiplier is handy if you are switching to “double time”:

```
    Tempo * 2
```

and to return:

```
Temp * .5
```

Note that for "+","-" or "*" the sign must be separated from the tempo value by at least one space. The value for **Tempo** can be any value, but will be converted to integer for the final setting.

11.2 Time

MMA doesn't really understand time signatures. It just cares about the number of beats in a bar. So, if you have a piece in $\frac{4}{4}$ time you would use:

```
Time 4
```

For $\frac{3}{4}$ use:

```
Time 3
```

For $\frac{6}{8}$ you'd probably want either "2" or "6".

Changing the time also cancels all existing sequences. So, after a time directive you'll need to set up your sequences or load a new groove[1].

11.3 TimeSig

Even though MMA doesn't really use Time Signatures, some MIDI programs do recognize and use them. So, here's a command which will let you insert a Time Signature in your MIDI output:

```
TimeSig NN DD
```

The NN parameter is the time signature numerator (the number of beats per bar). In $\frac{3}{4}$ you would set this to "3".

The DD parameter is the time signature denominator (the length of the note getting a single beat). In $\frac{3}{4}$ you would set this to "4".

The NN value must be an integer in the range of 1 to 126. The DD value must be one of 1, 2, 4, 8, 16, 32 or 64.

MMA assumes that all songs are in $\frac{4}{4}$ and places that MIDI event at offset 0 in the Meta track.

11.4 BeatAdjust

Internally, MMA tracks its position in a song according to beats. For example, in a $\frac{4}{4}$ piece the beat position is incremented by 4 after each bar is processed. For the most part, this works fine; however, there are some conditions when it would be nice to manually adjust the beat position:

[1]The time value is saved/restored with grooves so setting a time is redundant in this case.

♫ You may want to insert some extra (silent) beats at the end of bar to simulate a pause,

♫ You may want to delete some beats to handle a "short" bar.

Let us deal with both instances in turn. In example 11.1 we simulate a pause at the end of bar 10. One problem with this logic is that the inserted beat will be silent, but certain notes (percussive things like piano) often will continue to sound (this is related to the decay of the note, not that MMA has not turned off the note). Frankly, we've not been able to get this to work too well ... which is why the **fermata** (see page 62) was added.

```
Time 4
1 Cm / / /
...
10 Am / C /
BeatAdjust 1
...
```

Example 11.1: Adding Extra Beats

In example 11.2 we handle the problem of the "short bar". In this example, the sheet music has the majority of the song in $\frac{4}{4}$ time, but bar 4 is in $\frac{2}{4}$. We could handle this by setting the **Time** setting to 2 and creating some different patterns. Forcing silence on the last 2 beats and backing up the counter is a bit easier.

```
1 Cm / / /
...
4 Am / z!  /
BeatAdjust -2
...
```

Example 11.2: Short Bar Adjustment

Note that the adjustment factor can be a partial beat. For example:

```
BeatAdjust .5
```

will insert half of a beat between the current bars.

11.5 Fermata

A “fermata” or “pause” in written music tells the musician to hold a note for a longer period than the notation would otherwise indicate. In standard music notation it is represented by a “𝄐” above a note.

To indicate all this in MMA we use a command like:

```
Fermata 1 1 200
```

Note that there are three parts to the command:

1. The beat offset from the current point in the score to apply the “pause”. The offset can be positive or negative and is calculated from the current bar. Positive numbers will apply to the next bar; negative to the previous. For offsets into the next bar you use offsets starting at “0”; for offsets into the previous bar an offset of “-1” represents the last beat in that bar.

   For example, if you were in $\frac{4}{4}$ time and wanted the quarter note at the end of the next bar to be paused, you would use an offset of 3. The same effect can be achieved by putting the **Fermata** command after the bar and using an offset of -1.

2. The duration of the pause in beats. For example, if you have a quarter note to pause your duration would be 1, a half note (or 2 quarter notes) would be 2.

3. The adjustment. This represented as a percentage of the current value. For example, to force a note to be held for twice the normal time you would use 200 (two-hundred percent). You can use a value smaller than 100 to force a shorter note, but this is seldom done.

Example 11.3 shows how you can place a *Fermata* before or after the effected bar.

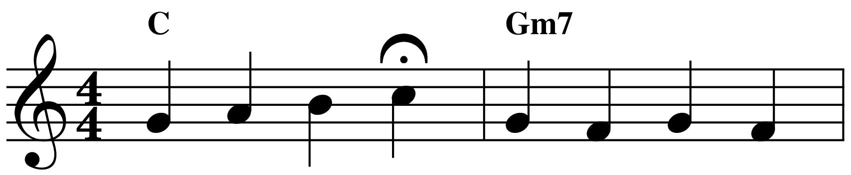
The second example, 11.4, shows the first four bars of a popular torch song. The problem with the piece is that we want the first beat of bar four to be paused, and then we want to switch the accompaniment in the middle of the bar. We have split the fourth bar with the first beat on one line and the balance on a second. The “z!”s are used to “fill in” the 4 beats skipped by the *BeatAdjust*.

The following conditions will generate warning messages:

- ♫ A beat offset greater than one bar,
- ♫ A duration greater than one bar,
- ♫ An adjustment value less than 100.

This command works by adjusting the global tempo in the MIDI meta track at the point of the fermata. In most cases you can put more than one **Fermata** command in the same bar, but they should be in beat order (no checks are done). If the *Fermata* command has a negative position argument, special code is invoked to remove any note-on events in the duration specified, after the start of the beat.[2] This means that extra rhythm notes will not be sounded—probably what you expect a held note to sound like.

[2]Technically speaking, MMA determines an interval starting 5% of a beat after the start of the fermata to a point 5% of a beat before the end. Any MIDI Note-On events in this range (in all tracks) are deleted.



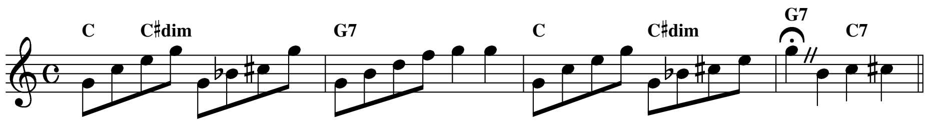
```
MMA Equivalent

Fermata 3 1 200
C
Gm7

Alternate

C
Fermata -1 1 200
Gm7
```

Example 11.3: Fermata



```
C C#dim
G7
C / C#dim
G7 z!
Fermata -4 1 200
Cut -3
BeatAdjust -3.5
Groove EasySwing
z!  G7 C7
```

Example 11.4: Fermata with Cut

11.6 Cut

This command was born of the need to simulate a "cut" or, more correctly, a "caesura". This is indicated in music by two parallel lines put at the top of a staff indicating the end of a musical thought. The symbol is also referred to as "railroad tracks".

The idea is to stop the music on all tracks, pause briefly, and resume.[3]

MMA provides the *cut* command to help deal with this situation. We have found it to be useful in other situations. But, before we describe the command in detail, a diversion: just how is a note or chord sustained in a MIDI file?

Let us assume that a MMA input file (and the associated library) files dictates that some notes are to be played from beat 2 to beat 4 in an arbitrary bar. What MMA does is:

- determine the position in the piece as a midi offset to the current bar,
- calculate the start and end times for the notes,
- adjust the times (if necessary) based on adjustable features such as *strum*, *articulate*, *rtime*, etc.,
- insert the required MIDI "note on" and "note off" commands at the appropriate point in the track.

You may think that a given note starts on beat 2 and ends (using *articulate 100*) right on beat 3—but you would most likely be wrong. So, if you want the note or chord to be "cut", what point do you use to instruct MMA correctly? Unfortunately, the simple answer is "it depends". Again, our answers will consist of some examples.

In this first case we wish to stop the track in the middle of the last bar. The simplest answer is:

```
1 C
...
36 C / z!  /
```

Unfortunately, this will "almost" work. But, any chords which are longer than one or two beats may continue to sound. This, often, gives a "dirty" sound to the end of the piece. The simple solution is to add to the end of the piece:

```
Cut -2
```

Depending on the rhythm you might have to fiddle a bit with the cut value. But, the example here puts a "all notes off" message in all the active tracks at the start of beat 3. The exact same result can be achieved by placing:

```
Cut 3
```

before the final bar.

In our second example we want a tiny bit of silence between bars 4 and 5. This might be the end of an introduction. The following bit should work:

```
1 C
2 G
3 G
4 C
```

[3]The answer to the music theory question of whether the "pause" takes time *from* the current beat or is treated as a "fermata" is not clear—but as far as MMA is concerned the command has no effect on timing.

```
Cut
BeatAdjust .2
5 G
...
```

In this case the “all notes off” is placed at the end of bar 4 and two-tenths of a beat is inserted at the same location. Bar 5 continues the track.

Our final example show how you might combine *cut* with *fermata*. In this case the sheet music shows a caesura after the first quarter note and fermatas over the quarter notes on beats 2, 3 and 4.

```
1 C C#dim
2 G7
3 C / C#dim
Fermata 1 3 120
Cut 1.9
Cut 2.9
Cut 3.9
4 G7 / C7 /
5 F6
```

A few tutorial notes on the above:

- ♫ The command

  ```
  Fermata 1 3 120
  ```

  applies a slow-down in tempo to the second beat for the following bar (an offset of 1), for 3 beats. These 3 beats will be played 20% slower than the set tempo.

- ♫ The three *cut* commands insert MIDI “all notes off” in all the active tracks just *before* beats 2, 3 and 4.

Finally, the proper syntax for the command:

```
[Voice] Cut [Offset]
```

If the voice is omitted, MIDI “all notes off” will be inserted into each active track.

If the offset is omitted, the current bar position will be used. This the same as using an offset value of 0.

Chapter 12 Volume and Dynamics

MMA is very versatile when it comes to the volumes or dynamics used in your song.

Each generated note goes though 4 volume adjustments:

1. The initial volume is set in the pattern definition, see chapter 4,
2. the initial volume is adjusted with the track volume,
3. this volume is further adjusted with the master volume,
4. if certain notes are to be accented, the volume is further adjusted,
5. and, finally, if the random volume is set, this is applied,

For the most part MMA uses conventional musical score notation for volumes. Internally, the dynamic name is converted to a percentage value. The note volume is adjusted by the percentage.

The following table shows the available volume settings and the adjustment values.

Symbolic Name	*Ratio Adjustment*
off	0
pppp	20
ppp	30
pp	45
p	55
mp	75
mf	90
f	100
ff	110
fff	120
ffff	150

The setting **Off** is useful for generating fades at the end of a piece. For example:

```
Volume ff
Decresc Off 5
G / Gm / * 5
```

will cause the last 5 bars of your music to fade from a “ff” to silence.

The initial volume (or velocity) is set in the pattern definition (see chapter 4). The following commands set the master volume, track volume and random volume adjustments.

In addition to the volumes (velocities) generated by MMA your MIDI device can also change the mix between channels. See the discussion for **ChannelVolume** (prefchannelvol).

12.1 Accent

"Real" musicians, in an almost automatic manner, emphasize notes on certain beats. In popular Western music written in $\frac{4}{4}$ time this is usually beats one and three. This emphasis sets the pulse or beat in a piece.

In MMA you can set the volumes in a pattern so that this emphasis is done. For example, when setting a walking bass line pattern you could use a pattern definition like:

```
    Define Walk W1234 1 4 100; 2 4 70; 3 4 80; 4 4 70
```

However, it is much easier to use a definition which has all the volumes the same:

```
    Define Walk W1234 1 1 90 * 4
```

and use the *Accent* command to increase or decrease the volume of notes on certain beats:

```
    Walk Accent 1 20 2 -10 4 -10
```

The above command will increase the volume for walking bass notes on beat 1 by 20%, and decrease the volumes of notes on beats 2 and 4 by 10%.

You can use this command for all tracks.

When specifying the accents, you must have matching pairs of data. The first item in the pair is the beat (which can be fractional), the second is the volume adjustment. This is a percentage of the current note volume that is added (or subtracted) to the volume. Adjustment factors must be in the range -100 to 100.

The *Accent*s apply to all bars in a track. You cannot set different accents for different bars. If you need to do this it's a simple matter to create duplicate tracks (which can even share the same MIDI channel). For example, you might want even bars to have beats 1 and 3 accented and odd bars to have only beat 1 accented. An abbreviated attempt might look like:

```
    Begin Chord-1
       Sequence C1234 z
       Voice Piano1
       Accent 1 20 3 30
    End
    Begin Chord-2
       Sequence z C1234
       Voice Piano1
       ChShare Chord-1
       Accent 1 20
    End
```

12.2 AdjustVolume

The ratios used to adjust the volume can be changed from the above table. For example, to change the percentage used for the "mf" setting:

```
AdjustVolume MF 95
```

If you want to adjust a number of settings:

```
Begin AdjustVolume
    PP 47
    ppp 50
End
```

All values must be positive integers. Any value over 180 will be reported as a warning.

You might want to do these adjustment in your MMArc file(s).

12.3 Volume

The volume for a track, or all tracks, is given the "Volume" command. Volumes can be specified much like standard sheet music with the conventional dynamic names. These volumes can be applied to a track or to the entire song. For example:

```
Arpeggio1 Volume p
```

sets the volume for Arpeggio1 track to something approximating *piano*.

```
Volume f
```

sets the master volume to *forte*.

In most cases the volume for a track will be set with the sequence definition; the master volume is used in the music file to adjust the overall feel of the piece.

12.4 Cresc and Decresc

If you wish to adjust over a series of bars use the **Cresc** or **Decresc** commands. These commands are only valid in the master context; they can not be applied to individual tracks.

For all practical purposes, the two commands are equivalent, expect for the warning. If the new volume in less than the current volume in a **Cresc** a warning will be displayed; the converse applies to a **Decresc**.

The command requires two arguments. The first is the new volume, the second is the number of bars to adjust it over.

For example:

```
    Cresc fff 5
```

will gradually vary the master volume from its current setting to a triple forte over the next 5 bars.

Similarly:

```
    Decresc mp 2
```

will decrease the master volume to mezzo piano over the next 2 bars.

A **SeqClear** command will reset all track volumes to the default **mf** (ie. no adjustment).

When using **Volume** for a specific track, you can use a different value for each bar in a sequence:

```
    Drum Volume mp ff / ppp
```

A “/” can be used to repeat values.

12.5 RVolume

Not even the best musician can play each note at the same volume. Nor would he or she want to—the result would be quite unmusical. The note volumes can be randomly adjusted with the **Rvolume** command.

The command can be applied to a specific track or (if you’re brave) to all tracks.[1] Examples:

```
    Chord RVolume 10
    RVolume 5
```

The **RVolume** argument is a percentage value by which a volume is adjusted. A setting of 0 disables the adjustment for a track (this is the default).

When set, the note velocity (after the track and master volume adjustments) is randomized up or down by the value. Again, using the above example, let us assume that a note in the current pattern gets a MIDI velocity of 88. The random factor of 10 will adjust this by 10% up or down—the new value can be from 78 to 98.

The idea behind this is to give the track a more human sounding effect. You can use large values, but it’s not recommended. Usually, values in the 5 to 10 range work well. You might want slightly larger values for drum tracks. Using a value greater than 30 will generate a warning message.

Notes:

- ♫ No generated value will be out of the valid MIDI velocity range.
- ♫ You may use **RVolume** without a leading track name. In this case it will effect all the tracks (probably not recommended).
- ♫ When using **RVolume** for a specific track, you can use a different value for each bar in a sequence:

```
        Scale RVolume 10 0 / 20
```

[1] The best use of using *RVolume* for all tracks is with a “0” argument to (temporarily) disable the setting for all tracks.

♫ A "/" can be used to repeat values.

12.6 Saving and Restoring Volumes

Dynamics can get quite complicated, especially when you are adjusting the volumes of a track inside a repeat or other complicated sections of music. In this section we will attempt to give some general guidelines and hints.

For the most part, the supplied groove files will have balanced volumes between the different instruments. In a future version of MMA a *volumeAdjust* command will let you fine tune differences between your synth and the standards in the library. This will be done before verison 1.0.

Remember that *Groove*s save all the current volume settings. This includes the master setting as well as individual track settings. So, if you are using the mythical groove "Wonderful" and think that the *Chord-Piano* volume should be louder in a particular song it's easy to do something like:

```
Groove Wonderful
Chord-Piano Volume ff
DefGroove Wonderful
```

Now, when you call this groove the new volume will be used. Note that you'll have to do this for each variation of the groove that you use in the song.

In most songs you will not need to do major changes. But, it is nice to use the same volume each time though a section. In most cases you'll want to do a explict setting at the start of a section. For example:

```
Repeat
Volume mf
....
Cresc ff 5
...
EndRepeat
```

Another useful technique is the use of the *$_LastVolume* macro. For example:

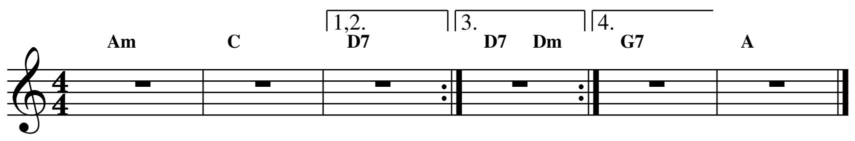
```
Volume pp
...
Cresc f 5
...
$_LastVolume // restores to pp
```

Chapter 13 Repeats

MMA attempts to be as comfortable to use as standard sheet music. This includes **repeats** and **endings**.

More complex structures like **D.S.**, **Coda**, etc. are *not* directly supported. But, they are easily simulated with by using some simple variables, conditionals and gotos. See chapter 14 for details. Often as not, it may be easier to use your editor to cut, paste and duplicate.

A section of music to be repeated is indicated with a “Repeat” and “Repeatend” or “EndRepeat”[1]. In addition, you can have “RepeatEndings”.



```
    Repeat
1     Am
2     C
    RepeatEnding 2
3     D7
    RepeatEnding
4     D7 / Dm
    RepeatEnd
5     G7
6     A
```

Example 13.1: Repeats

In example 13.1 MMA produces music with bars:

1, 2, 3,
1, 2, 3,

[1]The reason for both “EndRepeat” and “RepeatEnd” is that we have both “IfEnd” and “EndIf”.

1, 2, 4,
1, 2, 5, 6

This works just like standard sheet music. Note that **RepeatEnding** can take an optional argument indicating the number of times to use the ending.

MMA processes repeats by reading the input file and creating duplicates of the repeated material. This means that a directive in the repeated material would be processed multiple times. Unless you know what you are doing, directives should not be inserted in repeat sections. Be especially careful if you define a pattern inside a repeat. Using "Tempo" with a "+" or "-" will be problematic as well.

Repeats can be nested to any level.

There must be one "RepeatEnd" or "EndRepeat" for every "Repeat". Any number of "RepeatEnding"s can be included before the "RepeatEnd".

Chapter 14

Variables, Conditionals and Jumps

To make the processing of your music easier, MMA supports a very primitive set for variable manipulations along with some conditional testing and the oft-frowned-upon *goto* command.

14.1 Variables

MMA lets you set a variable, much like in other programming languages and to do some basic manipulations on them. Variables are most likely to be used for two reasons:

- ♫ For use in setting up conditional segments of your file,
- ♫ As a shortcut to entering complex chord sequences.

To begin, the following list shows the available commands to set and manipulate variables:

```
Set VariableName String
Mset VariableName ...  MsetEnd
UnSet VariableName
ShowVars
Inc Variablename [value]
Dec Variablename [value]
Vexpand ON/Off
```

All variable names are case-insensitive. Any characters can be used in a variable name. The only exceptions are that a variable name cannot start with a "$" or a "_" (an underscore—this is reserved for internal variables, see below).

Variables are set and manipulated by using their names. Variables are expanded when their name is prefaced by a space followed by single "$" sign. For example:

```
Set Silly Am / Bm /
1 $Silly
```

The first line creates the variable "Silly"; the second creates a bar of music with the chords "Am / Bm /".

Note that the "$" must be the first item on a line or follow a space character. For example, the following will NOT work:

```
Set Silly 4a;b;c;d;
1 Am {$Silly}
```

However:

```
1 Am { $Silly}
```

will work fine.

Following are details on all the available variable commands:

14.1.1 Set [string]

Set or create a variable. You can skip the *String* if you do want to assign an empty string to the variable. A valid example is:

```
Set PassCount 1
```

14.1.2 Mset [lines] MsetEnd/EndMset

This command is quite similar to *Set*, but *Mset* expects multiple lines. An example:

```
MSet LongVar
  1 Cm
  2 Gm
  3 G7
MsetEnd
```

It is quite possible to set a variable to hold an entire section of music (perhaps a chorus) and insert this via macro expansion at various places in your file.

Each *Mset* must be terminated by a *EndMset* or *MsetEnd* command (on its own separate line).

14.1.3 UnSet VariableName

Removes the variable. This can be useful if you have conditional tests which simply rely on a certain variable being “defined”.

14.1.4 ShowVars

Displays the names of the defined variables and their contents. Mainly used for debugging. The display will preface each variable name with a “$”. Note that internal MMA variable are also displayed with this command.

14.1.5 Inc and Dec

These commands increment or decrement a variable. If no argument is given, a value of 1 is used; otherwise, the value specified is used. The value can be an integer or a floating point number.

A short example:

```
Set PassCount 1
Set Foobar 4
Showvars
Inc FooBar 4
Inc PassCount
ShowVars
```

This command is quite useful for creating conditional tests for proper handling of codas or groove changes in repeats.

14.1.6 VExpand On or Off

Normally variable expansion is enabled. These two options will turn expansion on or off. Why would you want to do this? Well, here's a simple example:

```
Set LeftC Am Em
Set RightC G /
VExpand Off
Set Full $LeftC $RightC
VExpand On
```

In this case the actual contents of the variable "Full" is "$LeftC $RightC". If the *Off/On* option lines had not been used, the contents would be "Am Em G /". You can easily verify this with the *ShowVars* option.

When MMA processes a file it expands variables in a recursive manner. This means that, in the above example, the line:

```
1 $Full
```

will be changed to:

```
1 Am Em G /
```

However, if later in the file, you change the definition of one of the variables ... for example:

```
Set LeftC Am /
```

the same line will now be "1 Am / G /".

Most of MMA's internal commands *can* be redefined with variables. However, we really don't think you should use this feature. It's been left for two reasons: it might be useful, and, it's hard to disable.

However, not all commands can be redefined. The following is short list of things which will work (but, again, we're not suggesting you do this):

```
Set Rate Tempo 120
$Rate
Set R Repeat
$R
```

But, the following will *not* work:

```
Set B Begin
Set E End
$B Arpeggio Define
....
$E
```

This fails since the Begin/End constructs are expanded before variable expansion. However:

```
Set A Define Arpeggio
Begin $a ...  End
```

is quite alright.

Even though you can use a variable to substitute for the *Repeat* or *If* directives, using one for *Repeat-End/EndRepeat*, *RepeatEnding*. *Label* or *IfEnd/EndIf* will fail.

Variable expansion should usually not be a concern. In most normal files, MMA will expand variables as they are encountered. However, when reading the data in a *Repeat*, *If* or *Mset* section the expansion function is skipped—but, when the lines are processed, after being stored in an internal queue, variables are expanded.

14.2 Predefined Variables

For your convenience MMA tracks a number of internal settings and saves their values in variables you can access just like you would a user defined variable. All of these "internal" variables are prefaced with a single underscore. For example, the current tempo is saved in the variable *_TEMPO*; this can be accessed in your script with the notation *$_TEMPO*.

_Groove Name of the currently selected groove. May be empty if no groove has been selected.

_LastGroove Name of the groove selected *before* the currently selected groove.

_SeqSize Current *SeqSize* setting.

_Tempo Current *Tempo*. Note that if you have used the optional *bar count* in setting the tempo this will be the target tempo.

_Time The current *Time* (beats per bar) setting.

_Transpose Current *Transpose* setting.

_Volume Current global volume setting.

_LastVolume Previously set global volume setting.

_Debug Current debug settings.

_LastDebug Debug settings prior to last *Debug* command. This setting can be used to restore settings, ie:

```
Debug Warnings=off
...  stuff generating annoying warnings
Debug $_LastDebug
```

14.3 Conditionals

The most important reason we created variables in MMA was to use them in conditionals. In MMA a conditional consists of a line starting with an *If* directive, a test, a series of lines to process (depending upon the result of the test), and a closing *EndIf* or *IfEnd*[1] directive. An optional *Else* statement may be included.

The first set of tests are unary (they take no arguments):

Def VariableName Returns true if the variable has been defined.

Ndef VariableName Returns true if the variable has not been defined.

In the above tests you must supply the name of a variable—don't make the mistake of including a "$" which will invoke expansion and result in something you were not expecting.

A simple example:

```
If Def InCoda
  5 Cm
  6 /
Endif
```

The other tests are binary (they take two arguments):

LT Str1 Str2 Returns true if *Str1* is less than *Str2*. (Please see the discussion below on how the tests are done.)

LE Str1 Str2 Returns true if *str1* is less than or equal to *Str2*.

EQ Str1 Str2 Returns true if *str1* is equal to *Str2*.

NE Str1 Str2 Returns true if *str1* is not equal to *Str2*.

GT Str1 Str2 Returns true if *str1* is greater than *Str2*.

GE Str1 Str2 Returns true if *str1* is greater than or equal to *Str2*.

In the above tests you have several choices in specifying *Str1* and *Str2*. At some point, when MMA does the actual comparison, two strings or numeric values are expected. So, you really could do:

[1]We probably suffer from mild dyslexia and can't remember if the command is IFEND or ENDIF, so both are permitted. Use whichever is more comfortable for you.

```
If EQ abc ABC
```

and get a “true” result. The reason that “abc” equals “ABC” is that all the comparisons in MMA are case-insensitive.

You can also compare a variable to a string:

```
If GT $foo abc
```

will evaluate to “true” if the *contents* of the variable “foo” evaluates to something “greater than” “abc”. But, there is a bit of a “gotcha’ here. If you have set “foo” to a two word string, then MMA will choke on the command. In the following example:

```
Set Foo A B
If GT $Foo abc
```

the comparison is passed the line:

```
If GT A B abc
```

and MMA seeing three arguments generates an error. If you want the comparison done on a variable which might be more than one word, use the “$$” syntax. This delays the expansion of the variable until the *If* directive is entered. So:

```
If $$foo abc
```

would generate a comparison between “A B” and “ABC”.

Delayed expansion can be applied to either variable. It only works in an *If* directive.

Strings and numeric values can be confusing in comparisons. For example, if you have the strings “22” and ”3” and compare them as strings, “3” is greater than “22”; however, if you compare them as values then 3 is less than 22.

The rule in MMA is quite simple: If either string in a comparison is a numeric value, both strings are converted to values. Otherwise they are compared as strings. [2]

This lets you do consistent comparisons in situations like:

```
Set Count 1
If LE $$Count 4
  ....
IfEnd
```

Note that in the above example we could have used “$Count”, but you should probably always use the “$$” in tests.

Much like other programming languages, an optional *Else* condition may be used:

```
If Def Coda
  Groove Rhumba1
```

[2]An attempt is made to convert each string to a float. If conversion of both strings is successful, the comparison is made between two floats, otherwise two strings are used.

```
Else
  Groove Rhumba
Endif
```

The *Else* statement(s) are processed only if the test for the *If* test is false.

Nesting of *If*s is permitted:

```
If ndef Foo
  Print Foo has been defined.
Else
  If def bar
    Print bar has been defined.  Cool.
  Else
    Print no bar...go thristy.
  Endif
Endif
```

works just fine. We've used indentation in our examples to clearly show the nesting and conditions. We suggest you do the same.

14.4 Goto

The *Goto* command redirects the execution order of you script to the point at which a *Label* has been defined. There are really two parts to this:

1. A command defining a label, and,
2. The *Goto* command.

A label is set with the *Label* directive:

```
Label Point1
```

The string defining the label can be any sequence of characters. Labels are case-insensitive. You can not set two points in your file to the same label.

To cause execution to jump to a labeled point:

```
Goto Point1
```

This causes an immediate jump. Any remaining lines in a repeat or conditional segment are discarded.

MMA does not check to see if you are jumping into a repeat or conditional section of code—but doing so will usually cause an error. Jumping out of these sections is usually safe.

For an example of how to use some simple labels to simulate a "DS al Coda" examine the file "lullaby-of-Broadway" in the sample songs directory.

Chapter 15

Low Level MIDI Commands

The commands discussed in this chapter directly effect your MIDI output devices.

Not all MIDI devices are equal. Many of the effects in this chapter may be ignored by your devices. Sorry, but that's just the way MIDI is.

15.1 Channel

As noted in the Tracks and Channels chapter (see page 14), MMA assigns MIDI channels dynamically as it creates tracks. In most cases this works fine; however, you can if you wish force the assignment of a specific MIDI channel to a track with the **Channel** command.

You cannot assign a channel number to a track if it already defined (well, see the section **ChShare**, below, for the inevitable exception), nor can you change the channel assignments for any of the **Drum** tracks.

Let us assume that you want the **Bass** track assigned to MIDI channel 8. Simply use:

```
    Bass Channel 8
```

Caution: If the selected channel is already in use an error will be generated. Due to the way MMA allocates tracks, if you really need to manually assign track we recommend that you do this in a *MMArc* file.

You can disable a channel at any time by using a channel number of 0:

```
    Arpeggio-1 Channel 0
```

will disable the Arpeggio-1 channel, freeing it for use by other tracks. A warning message is generated. Disabling a track without a valid channel is fine. When you set a channel to 0 the track is also disabled. You can restart the track with the *On* command (see page 96).

You don't need to have a valid MIDI channel assigned to a track to do things like: **Pan**, **Portamento**, **ChannelVolume** or even the assignment of any music to a track. MIDI data is created in tracks and then sent out to the MIDI buffers. Channel assignment is checked and allocated at this point, and an error will be generated if no channels are available.

It's quite acceptable to do channel reassignments in the middle of a song. Just assign channel 0 to the unneeded track first.

MIDI channel settings are *not* saved in *Grooves*.

MMA inserts a MIDI “track name” meta event when the channel buffers are first assigned at a MIDI offset of 0. If the MIDI channel is reassigned, a new “track name” is inserted at the current song offset.

A more general method is to use *ChannelPref* detailed below.

15.2 ChannelPref

If you prefer to have certain tracks assigned to certain channels you can use the *ChannelPref* command to create a custom set of preferences. By default, MMA assigns channels starting at 16 and working down to 1 (with the expection of drum tracks which are all assigned channel 10). If, for example, you would like the *Bass* track to be on channel 9, sustained bass on channel 3, and *Arpeggio* on channel 5, you can have a command like:

```
ChannelPref Bass=9 Arpeggio=5 Bass-Sus=3
```

Most likely this will be in your *mmarc* file.

You can use multiple command lines, or have multiple assignments on a single line. Just make sure that each item consists of a trackname, an “=” and a channel number in the range 1 to 16.

15.3 ChShare

MMA is fairly conservative in its use of MIDI tracks. “Out of the box” it demands a separate MIDI channel for each of its tracks, but only as they are actually used. In most cases, this works just fine.

However, there are times when you might need more tracks than the available MIDI channels or you may want to free up some channels for other programs.

If you have different tracks with the same voicing, it’s quite simple. For example, you might have an arpeggio and scale track:

```
Arpeggio Sequence A16 z
Arpeggio Voice Piano1
Scale Sequence z S8
Scale Voice Piano1
```

In this example, MMA will use different MIDI channels for the **Arpeggio** and the **Scale**. Now, if you force channel sharing:

```
Scale ChShare Arpeggio
```

both tracks will use the same MIDI channel.

This is really foolproof in the above example, especially since the same voice is being used for both. Now, what if we wanted to use a different voice for the tracks?

```
Arpeggio Sequence A16 z
Arpeggio Voice Piano1 Strings
Scale Sequence z S8
Scale ChShare Arpeggio
```

You might think that this would work, but it doesn't. MMA ignores voice changes for bars which don't have a sequence, so it will set "Piano1" for the first bar, then "Strings" for the second (so far, so good). But, when it does the third bar (an **Arpeggio**) it will not know that the voice has been changed to "Strings" by the **Scale** track.

So, the general rule for track channel sharing is to use only one voice.

One more example which doesn't work:

```
Arpeggio Sequence A8
Scale Sequence S4
Arpeggio Voice Piano1
Scale Voice Piano1
Scale ChShare Arpeggio
```

In this example we have an active scale and arpeggio sequence in each bar. Since both use the same voice, you may think that it will work just fine ... but it may not. The problem here is that MMA will generate MIDI on and off events which may overlap each other. One or the other will be truncated. If you are using a different octave, it will work much better. It may sound okay, but you should probably find a better way to do this.

When a *ChShare* directive is parsed the "shared" channel is first checked to ensure that it has been assigned. If not currently assigned, the assignment is first done. What this means is that you are subverting MMA's normal dynamic channel allocation scheme. This may cause is a depletion of avaiable channels.

Please note that we've never found it really necessary to use the **ChShare** command, so it might have more problems than outlined here. But, to do some testing we do use the command to share *Bass* and *Walk* channels in a few groove files.

This command will always display a warning message.

For another, simpler, way of reassigning MIDI tracks and letting MMA do most of the work for you, refer to the *Delete* command (see page 93).

15.4 MIDI

The complete set of MIDI commands is not limitless—but from this end it seems that adding commands to suit every possible configuration is never-ending. So, in an attempt to satisfy everyone, we've added a command which will place any arbitray MIDI stream in your tracks. In most cases this will be a MIDI "Sysex" or "Meta" event.

The data can be placed in the meta track or a specific voicing track.

For example, you might want to start a song off with a MIDI reset:

```
MIDI 0xF0 0x05 0x7e 0x7f 0x09 0x01 0xf7
```

The values passed to the MIDI command are normal integers; however, they must all be in the range of 0x00 to 0xff. In most cases it is easiest to use hexadecimal numbers by using the “0x” prefix. But, you can use plain decimal integers if you prefer.

In the above example:

0xF0 Designates a SYSEX message

0x05 The length of the message

0x7e ... The actual message

Another example places the key signature of F major (1 flat) in the meta track:

```
MIDI 0xff 0x59 0x02 0xff 0x00
```

Some **cautions:**

- MMA makes no attempt to verify the validity of the data!
- The “Length” field must be manually calculated.
- Malformed sequences can create unplayable MIDI files. In extreme situations, these might even damange your synth. You are on your own with this command ... be careful.
- The *Midi* directive always places data in the *Meta* track at the current time offset into the file. This should not be a problem.

Cautions aside, an include file which the author uses has been included in the main distribution as `includes/init.mma`. You might want to have the command:

```
MMAstart init
```

in your *mmarc* file. The file is pretty well commented and it sets a synth up to something reasonably sane.

If you need a brief delay after a raw MIDI command, it is possible to insert a silent beat with the *BeatAdjust* command (see page 60). See the file `includes/reset.mma` for an example.

15.5 MidiFile

This option controls some fine points of the generated MIDI file. The command is issued with a series of paramaters in the form “MODE=VALUE”. You can have mulitple settings in a single *MidiFile* command.

MMA can generate two types of SMF (Standard MIDI Files):

0. This file contains only one track into which the data for all the different channel tracks has been merged. A number of syths which accept SMF (Casio, Yamaha and others) only accept type 0 files.

1. This file has the data for each MIDI channel in its own track. This is the default file generated by MMA.

You can set the filetype in an RC file (or, for that matter, in any file processed by MMA) with the command:

```
MidiFile SMF=0
```

or

```
MidiFile SMF=1
```

You can also set it on the command line with the -M option. Using the command line option will override the *MidiSMF* command if it is in a RC file.

By default MMA uses "running status" when generating MIDI files. This can be disabled with the command:

```
MidiFile Running=0
```

or enabled (but this is the default) with:

```
MidiFile Running=1
```

Files generated without running status will be about 20 to 30% larger than their compressed counterparts. They may be useful for use with braindead sequencers and in debugging generated code. There is no command line equivalent for this option.

15.6 MIDISeq

It is possible to associate a set of MIDI controller messages with certain beats in a sequence. For example, you might want to have the Modulation Wheel set for the first beats in a bar, but not for the third. The following example shows how:

```
Seqsize 4
Begin Bass-2
 Voice NylonGuitar
 Octave 4
 Sequence { 1 4 1 90; 2 4 3 90; 3 4 5 90; 4 4 1+ 90}
 MIDIDef WheelStuff 1 1 0x7f ; 2 1 0x50; 3 1 0
 MidiSeq WheelStuff
 Articulate 90
End

C * 4
```

The *MidiSeq* command is specific to a track and is saved as part of the *Groove* definition. This lets style file writers use enhanced MIDI features to dress up their sounds.

The command has the following syntax:

```
TrackName MidiSeq <Beat> <Controller> <Datum> [ ; ...]
```

where:

Beat is the Beat in the bar. This can be an integer (1,2, etc.) or a floating point value (1.2, 2.25, etc.). It must be 1 or greater and less than the end of bar (in $\frac{4}{4}$ it must be less than 5).

Controller A valid MIDI controller. This can be a value in the range 0x00 to 0x7f or a symbolic name. See see page 125 for a list of defined names.

Datum All controller messages use a single byte "parameter" in the range 0x00 to 0x7f.

You can enter the values in either standard decimal notation or in hexadecimal with the prefixed "0x". In most cases, your code will be clearer if you use values like "0x7f" rather than the equivalent "127".

The MIDI sequences specified can take several forms:

1. A simple series like:

   ```
   MIDISeq 1 ReleaseTime 50; 3 ReleaseTime 0
   ```

   in this case the commands are applied to beats 1 and 3 in each bar of the sequence.

2. As a set of names predefined in an *MIDIDef* command:

   ```
   MIDIdef Rel1 1 ReleaseTime 50; 3 ReleaseTime 0
   MIDIdef Rel2 2 ReleaseTime 50; 4 ReleaseTime 0
   MIDISeq Rel1 Rel2
   ```

   Here, the commands defined in "Rel1" are applied to the first bar in the sequence, "Rel2" to the second. And, if there are more bars in the sequence than definitions in the line, the series will be repeated for each bar.

3. A set of series enclosed in { } braces. Each braced series is applied to a different bar in the sequence. The example above could have been does as:

   ```
   MIDISeq { 1 ReleaseTime 50; 3 ReleaseTime 0 } \
    { 2 ReleaseTime 50; 4 ReleaseTime 0 }
   ```

4. Finally, you can combine the above into different combinations. For example:

   ```
   MIDIDef Rel1 1 ReleaseTime 50
   MIDIDef Rel2 2 ReleaseTime 50
   MIDISeq { Rel1; 3 ReleaseTime 0 } { Rel2; 4 ReleaseTime 0 }
   ```

You can have specify different messages for different beats (or different messages/controllers for the same beat) by listing them on the same *MidiSeq* line separated by ";"s.

If you need to repeat a sequence for a measure in a sequence you can use the special notation "/" to force the use of the previous line. The special symbol "z" or "-" can be used to disable a bar (or number of bars). For example:

```
Bass-Dumb MIDISeq 1 ReleaseTime 20 z / FOOBAR
```

would set the "ReleaseTime" sequence for the first bar of the sequence, no MIDISeq events for the second and third, and the contents of "FOOBAR" for the fourth.

To disable the sending of messages just use a single “-”:

```
Bass-2 MidiSeq - // disable controllers
```

15.7 MIDIVoice

Similar to the *MIDISeq* command discussed in the previous section, the *MIDIVoice* command is used to insert MIDI controller messages into your files. Instead of sending the data for each bar as *MIDISeq* does, this command just sends the listed control events at the start of a track and then, if needed, at the start of each bar.

Again, a short example. Let us assume that you want to use the “Release Time” controller to sustain notes in a bass line:

```
Seqsize 4
Begin Bass-2
 Voice NylonGuitar
 MidiVoice 1 ReleaseTime 50
 Octave 4
 Sequence { 1 4 1 90; 2 4 3 90; 3 4 5 90; 4 4 1+ 90}
 Articulate 60
End

C * 4
```

should give an interesting effect.

The syntax for the command is:

```
Track MIDIVoice <beat> <controller> <Datum> [; ...]
```

This syntax is identical to that discussed in the section for *MIDISeq*, above. The >beat<value is required for the command, but it is ignored. Future versions of MMA may use this value.

By default MMA assumes that the MIDIVoice data is to be used only for the first bar in the sequence. But, it’s possible to have a different sequence for each bar in the sequence (just like you can have a different *Voice* for each bar). In this case, group the different data groups with {} brackets:

```
Bass-1 MIDIVoice {1 ReleaseTime 50} {1 ReleaseTime 20}
```

This list is stored with other *Groove* data, so is ideal for inclusion in a style file.

If you want to disable this command after it has been issued you can use the form:

```
Track MIDIVoice - // disable
```

Some technical notes:

- Since a common use of the command is to select the "Bank" for a voice MMA sends the controller data specified by the *MIDIVoice* setting before sending the MIDI Program Change event needed to switch the voice.
- MMA tracks the events sent for each bar and will not duplicate sequences.
- Be cautious in using this command to switch voice banks. If you don't switch the voice bank back to a sane value you'll be playing the wrong instruments!

15.8 MIDIClear

As noted earlier in this manual you should be very careful in programming MIDI sequences into your song and/or library files. Doing damage to a synthesizer is probably a remote possibility . . . but leaving it in a unexpected mode is likely. For this reason we have included the *MIDIClear* command as a companion to the *MIDIVoice* and *MIDISeq* commands.

Each time a MIDI track (not necessary the same as a MMA track) is ended or a new *Groove* is started, a check is done to see if any MIDI data has been inserted in the track with a *MIDIVoice* or *MIDISeq* command. If it has, a further check is done to see if there is an "undo" sequence defined via a *MIDIClear* command. That data is then sent; or, if data has not be defined for the track, a warning message is displayed.

The *MIDIClear* command uses the same syntax as *MIDIVoice* and *MIDISeq*; however, you can not specify different sequence for different bars in the sequence:

```
Bass-Funky MIDIClear 1 Modulation 0; 1 ReleaseTime 0
```

As in *MIDIVoice* and *MIDISeq* you can include sequences defined in a *MIDIDef*. The <beat>offsets are required, but ignored.

15.9 Pan

In MIDI-speak "pan" is the same as "balance" on a stereo. By adjusting the **Pan** for a track you can direct the output to the left, right or both speakers. Example:

```
Bass Pan 4
```

This command is only available in track mode. The data generated is not sent into the MIDI stream until musical data is created for the relevant MIDI channel.

The value specified must be in the range 0 to 127, and must be an integer.

Pan is not saved or restored by **Groove** commands, nor is it effected by **SeqClear**. A **Pan** is inserted directly into the MIDI track at the point at which it is encountered in the music file. This means that the effect of **Pan** will be in use until another **Pan** is encountered.

Pan can be used in MIDI compositions to emulate the sound of an orchestra. By assigning different values to different groups of instruments, you can get the feeling of strings, horns, etc. all placed in the "correct" position on the stage.

We use Pan for much cruder purposes. When creating accompaniment tracks for our jazz group. We set all the bass tracks (Bass, Walk, Bass-1, etc) to a Pan 0. Now, when practicing at home we can have a "full band"; and the bass player can practice without the generated bass lines simply by turning off the left speaker.

Because your MIDI keyboard most likely does not do a reset between tunes, you should probably undo any **Pan** effects at the end of your file. Example:

```
Include swing
Groove Swing
Bass Pan 0
Walk Pan 0
1 C
2 C
...
123 C
Bass Pan 64
Walk Pan 64
```

15.10 Portamento

This sets the MIDI portamento (in case you're new to all this, portamento is like glissando between notes—wonderful, if you like trombones! To enable portamento:

```
Arpeggio Portamento 30
```

The parameter can be any value between 1 and 127. To turn the sliding off:

```
Arpeggio Portamento 0
```

This command will work with any track (including drum tracks). However, the results may be somewhat "interesting" or "disappointing", and many MIDI devices don't support portamento at all. So, be cautious. The data generated is not sent into the MIDI stream until musical data is created for the relevant MIDI channel.

15.11 ChannelVolume

MIDI devices equipped with mixer settings can make use of the "Channel" or "Master" volume settings.[1]

[1] We discovered this on our keyboard after many frustrating hours attempting to balance the volumes in the library. Other programs would change the keyboard settings, and not being aware of the changes, we'd end up scratching our heads.

MMA doesn't set any channel volumes without your knowledge. If you want to use a set of reasonable defaults, look at the file `includes/init.mma` which sets all channels other than "1" to "100". Channel "1" is assumed to be a solo/keyboard track and is set to the maximum volume of "127".

You can set all or selected **ChannelVolume**s:

```
ChannelVolume 99
```

will set all channels to "99". And:

```
Chord ChannelVolume 55
```

will set only the Chord track channel. For most users, the use of this command is *not* recommended since it will upset the balance of the library grooves. If you need a track softer or louder you should use the volume setting for the track.

The data generated is not sent into the MIDI stream until musical data is created for the relevant MIDI channel.

Chapter 16

Other Commands and Directives

In addition to the “Pattern”, “Sequence”, “Groove” and “Repeat” and other directives discussed earlier, and chord data, MMA supports a number of directives which affect the flavor of your music.

The subjects presented in this chapter are ordered alphabetically.

16.1 Articulate

When MMA processes a music file, all the note lengths specified in a pattern are converted to MIDI lengths.

For example in:

```
Bass Define BB 1 4 0 100; 2 4 2 90; 3 4 0 80; 4 4 2 90
```

we define bass notes on beats 1, 2, 3 and 4. All these notes are defined as quarter notes. MMA, being quite literal about things, will make each note exactly 192 MIDI ticks long—which means that the note on beat 2 will start at the same time as the note on beat 1 ends.

MMA has an articulate setting for each voice. This value is applied to shorten the note length. By default, the setting is 90. Each generated note duration is taken to be a percentage of this setting, So, a quarter note with a MIDI tick duration of 192 will become 172 ticks long.

If articulate is applied to a short note, you are guaranteed that the note will never be less than 1 MIDI tick in length.

To set the value, use a line like:

```
Chord-1 Articulate 96
```

Articulate values must be greater than 0 and less than or equal to 100.

You can specify a different **Articulate** for each bar in a sequence. Repeated values can be represented with a “/”:

```
Chord Articulate 50 60 / 30
```

Notes: The full values for the notes are saved with the pattern definition. The articulate adjustment is applied at runtime. The articulate setting is saved with a **groove**.

16.2 Copy

Sometimes it is useful to duplicate the settings from one voice to another. The *Copy* command does just that:

```
    Bass-1 Copy Bass
```

will copy the settings from the *Bass* track to the *Bass-1* track.

The *Copy* command only works between tracks of the same type.

The following settings are copied:

- ♫ Volume (*see page 68*)
- ♫ RVolume (*see page 69*)
- ♫ RSkip (*see page 97*)
- ♫ RTime (*see page 98*)
- ♫ Strum (*see page 99*)
- ♫ Octave (*see page 96*)
- ♫ Harmony (*see page 94*)
- ♫ Direction (*see page 93*)
- ♫ ScaleType (*see page 98*)
- ♫ Voice or Tone (*see page 101 or 24*)
- ♫ Invert (*see page 56*)
- ♫ Articulate (*see page 90*)
- ♫ Compress (*see page 55*)

16.3 Comment

As previously discussed, a comment in MMA is anything following a "//" in a line. A second way of marking a comment is with the **Comment** directive. This is quite useful in combination the **Begin** and **End** directives. For example:

```
    Begin Comment
        This is a description spanning
            several lines which will be
            ignored by MMA.
    End
```

You could achieve the same with:

```
// This is a description spanning
// several lines which will be
// ignored by MMA.
```

or even:

```
Comment This is a description spanning
Comment several lines which will be
Comment ignored by MMA.
```

One minor difference between **//** and **Comment** is that the first is discarded when the input stream is read; the more verbose version is discarded during line processing.

We find that **Begin Comment/End** is handy to delete large sections of a song we are writing on a temporary basis.

16.4 Debug

To enable you to find problems in your song files (and, perhaps, even find problems with MMA itself) various debugging messages can be displayed. These are normally set from the command line (see page 11).

However, it is possible to enable various debugging messages dynamically in a song file using the *Debug* directive. In a debug statement you can enable or disable any of a variety of messages. A typical directive is:

```
Debug Debug=On Expand=Off Patterns=On
```

Each section of the debug directive consists of a *mode* and the command word *ON* or *Off*. The two parts must be joined by a single "=". You may use the values "0" for "Off" and "1" for "On" if desired.

The available modes with the equivalent command line switches are:

Mode	*Command Line Equivalent*	
Debug	-d	debugging messages
Filenames	-o	display filenames
Patterns	-p	pattern creation
Sequence	-s	sequence creation
Runtime	-r	running progress
Warnings	-w	warning messages
Expand	-e	display expanded lines

The modes and command are case-insensitive (although the command line switches are not).

The current state of the debug flags is saved in the variable $_Debug and the state prior to a change is saved in $_LastDebug.

16.5 Delete

If you are using a track in only one part of your song, especially if it is at the start, it may be wise to free that track's resources when you are done with it. The *Delete* command does just that:

```
Solo Delete
```

If a MIDI channel has been assigned to that track, it is marked as "available" and the track is deleted. Any data already saved in the MIDI track will be written when MMA is finished processing the song file.

16.6 Direction

In tracks using chords or scales you can change the direction in which they are applied:

```
Scale Direction UP
```

There are four direction options:

UP	Plays in upward direction only
DOWN	Plays in downward direction only
BOTH	Plays upward and downward (**default**)
RANDOM	Plays notes from the chord or scale randomly

This command is valid for any track, but is only used by **Scale**, **Arpeggio** and **Chord** tracks.

You can specify a different **Direction** for each bar in a sequence. Repeated values can be represented with a "/":

```
Arpeggio Direction Up Down / Both
```

The default value for **Scale** and **Arpeggio** tracks is **BOTH**.

In a **Walk** track only **Up** and **Down** are recognized. When set the pattern will walk up or down, skipping the random note selection. This is useful when using a walking bass line in an ending situation.

The setting is currently ignored by **Bass** tracks.

In a **Chord** track the command is only used when **Strum** is set. The default setting is **Up**; any setting other than **Down** is treated as **Up**).

16.7 Duplicate

Judicious use of the **Duplicate** directive can do much to make a composition sound "fuller". Essentially what it does is to duplicate all the notes played to a specified octave. For example:

```
Begin Bass
  Define B1234 0 4 0 90; 1 4 2 90; 2 4 0 90; 3 4 2 90
```

```
      Sequence B1234
      Octave 4
      Duplicate -1
    End
```

Creates a **Bass** line which plays a single note on beats 1, 2, 3 and 4 (the root and fifth of the chord). The **Duplicate** directive forces the notes to be played in the specified octave and one octave below that.

Notes: **Duplicate** takes any value between -9 and 9 as arguments—but, if the resulting note is forced out of the MIDI range, the note will not sound.

You can specify a different **Duplicate** for each bar in a sequence. Repeated values can be represented with a "/":

```
    Chord Duplicate -1 1 / 0
```

To restore to its normal (off) setting, use a "0" as the argument.

This command has no effect on a **Drum**, *Solo* and *Melody* tracks (no warnings or errors are generated). For a similar command see *DupRoot* (see page 56).

16.8 Harmony

MMA can generate harmony notes for you . . . just like hitting two or more keys on the piano! And you don't have to take lessons.

Automatic harmonies are available for the following track types: Bass, Walk, Apreggio, Scale, Solo and Melody. To enable harmony notes, use a command like:

```
    Solo Harmony 2
```

You can set a different harmony method for each bar in your sequence.

The following are valid harmony methods:

2 Two part harmony. The harmony note selected is lower (on the scale).

3 Three part harmony. The harmony notes selected are lower.

OPEN Two part harmony, however the gap between the two notes is larger than in "2".

2Above The same as "2", but the harmony note is raised an octave.

3Above The same as "3", but both notes are raised an octave.

OpenAbove The same as "Open", but the note is raised an octave.

All harmonies are created using the current chord.

To disable harmony use a "0" or a "-".

Be careful in using harmonies. They can make your song sound heavy, especially with *Bass* notes.

Just in case you are thinking that MMA is a wonderful musical creator when it comes to harmonies, don't be fooled. MMA's ideas of harmony are quite facile. It determines harmony notes by finding a note lower than the current note being sounded in the chord. And its notion of "open" is certainly not that of traditional music theory. But, the sound isn't too bad.

The command has no effect on *Drum* or *Chord* tracks.

16.9 HarmonyOnly

As a added feature to the automatic harmony generation discussed in the previous section, it is possible to set a track so that it *only* plays the harmony notes. For example, you might want to set up two arpeggio tracks with one playing quarter notes on a piano and a harmony track playing half notes on a violin. The following snippet is extracted from the song file "Cry Me A River" and sets up 2 different choir voices:

```
Begin Arpeggio
    Sequence A4
    Voice ChoirAahs
    Invert 0 1 2 3
    SeqRnd
    Octave 5
    RSkip 40
    Volume p
    Articulate 99
End

Begin Arpeggio-2
    Sequence A4
    Voice VoiceOohs
    Octave 5
    RSkip 40
    Volume p
    Articulate 99
    HarmonyOnly Open
End
```

Just like the *Harmony* command, above, you can have different settings for each bar in your sequence. Setting a bar (or the entire sequence) to '‘-" or "0" disables both the *Harmony* and *HarmonyOnly* settings.

The command has no effect on *Drum* or *Chord* tracks.

If you want to use this feature with *Solo* or *Melody* tracks you will need to duplicate the notes in your *Riff* or inline notation. This will be made automatic in a future release.

16.10 Octave

When MMA initializes and after the **SeqClear** command all track octaves are set to “4”. This will place most chord and bass notes in the region of middle C.

You can change the octave for any voice with **Octave** command. For example:

```
Bass-1 Octave 3
```

Sets the notes used in the “Bass-1” track one octave lower than normal.

The octave specification can be any value from 0 to 10. Various combinations of **Invert**, **Transpose** and **Octave** can force notes to be out of the valid MIDI range. In this case the lowest or highest available note will be used.

You can specify a different **Octave** for each bar in a sequence. Repeated values can be represented with a “/”:

```
Chord Octave 4 5 / 4
```

16.11 Off

To disable the generation of MIDI output on a specific track:

```
Bass Off
```

This can be used anywhere in a file. Use it to override the effect of a predefined groove, if you wish. This is simpler than resetting a voice in a groove. The only way to reset this command is with a **On** directive.

16.12 On

To enable the generation of MIDI output on a specific track which has been disabled with an **Off** directive:

```
Bass On
```

16.13 Print

The **Print** directive will display its argument to the screen when it is encountered. For example, if you want to print the filename of the input file while processing, you could insert:

```
Print Making beautiful music for MY SONG
```

No control characters are supported.

This can be useful in debugging input files.

16.14 PrintActive

The **PrintActive** directive will the currently active *Groove* and the active tracks. This can be quite useful when writing groove files and you want to modify and existing groove.

Any parameters given are printed as single comment at the end of the header line.

This is strictly a debugging tool. No *PrintActive* statements should appear in finalized grooves or song files.

16.15 RSkip

To aid in creating syncopated sounding patterns, you can use the **RSkip** directive to randomly silence or skip notes. The command takes a value in the range 0 to 99. The “0” argument disables skipping. For example:

```
Begin Drum
    Define D1 1 0 90
    Define D8 D1 * 8
    Sequence D8
    Tone OpenHiHat
    RSkip 40
End
```

In this case we have defined a drum pattern to hit short notes 8 per bar and have set up a sequence to play this with “OpenHiHat”. The **RSkip** argument of “40” will cause the note to be NOT sounded (randomly) only 40% of the time.

Using a value of “10” will cause notes to be skipped 10% for the time (they are played 90% of the time), “90” means to skip the notes 90% of the time, etc.

You can specify a different **RSkip** for each bar in a sequence. Repeated values can be represented with a “/”:

```
Scale RSkip 40 90 / 40
```

If you use the **RSkip** in a chord track, the entire chord *will not* be silenced. The option will be applied to the individual notes of each chord. This may or may not be what you are after. You cannot use this option to generate entire chords randomly. For this effect you need to create several chord patterns and select them with **SeqRnd**.

You can use *RSkip* without a track argument. This is useful when used with an argument of “0” to (temporarily) disable the setting for all tracks.

16.16 RTime

One of the biggest problem with computer generated drum and rhythm tracks is that, unlike real musicians, the beats are precise and “on the beat”. The **RTime** directive attempts to solve this.

The command can be applied to all tracksfootnote:The best use of using *RTime* for all tracks is with a “0” argument to (temporarily) disable the setting for all tracks.

```
RTime 5
```

or a specified one:

```
Drum4 Rtime 4
```

The value passed to the RTime directive are the number of MIDI ticks with which to vary the start time of the notes. For example, if you specify “5” the start times will vary from -5 to +5 ticks) on each note for the specified track. There are 192 MIDI ticks in each quarter note.

Any value from 0 to 100 can be used; however values in the range 0 to 10 are most commonly used. Exercise caution in using large values!

You can specify a different **RTime** for each bar in a sequence. Repeated values can be represented with a “/”:

```
Chord RTime 4 10 / 4
```

16.17 ScaleType

This option is only used by **Scale** tracks. It can be set for other tracks, but the setting is not used.

By default, the **ScaleType** is set to **Auto**. The settings permissible are:

MAJOR	Forces use of a major scale
MINOR]	Forces use of a natural minor scale
CHROMATIC	Forces use of a chromatic scale
AUTO	Uses major or minor scale depending on chord (default)

16.18 Seq

If your sequence, or groove, has more than one pattern (ie. you have set SeqSize to a value other than 1), you can use this directive to force a particular pattern point to be used. The directive:

```
Seq
```

resets the **sequence counter** to 1. This means that the next bar will use the first pattern in the current sequence. You can force a specific pattern point by using an optional value after the directive. For example:

```
Seq 8
```

forces the use of pattern point 8 for the next bar. This can be quite useful if you have a multibar sequence and, perhaps, the eight bar is variation which you want used every eight bars, but also for a transition bar, or the final bar. Just put a **seq 8** at those points. You might also want to put a **seq** at the start of sections to force the restart of the count.

This command will also disable the effects of **SeqRnd**. One difference between **SeqNoRnd** and **Seq** is that the current sequence point is set with the latter; with **SeqNoRnd** it is left at a random point.

Note: Using a value greater than the current **SeqSize** is not permitted.

This is a very useful command! For example, look at the four bar introduction of the song "Exactly Like You":

```
Groove BossanovaEnd
seq 3
1 C
seq 2
2 Am7
seq 1
3 Dm7
seq 3
4 G7 / G7#5
```

Here we have used the four bar ending groove to create an interesting introduction.

16.19 Strum

By default MMA plays all the notes in a chord at the same time. To make the chord more like something a guitar or banjo might play, use the **Strum** directive. For example:

```
Chord-1 Strum 5
```

sets the strumming factor to 5 for track Chord-1.

Setting the **Strum** in any track other than a **Chord** track will generate a warning message and the command will be ignored.

The strum factor is specified in MIDI ticks. Usually values around 10 to 15 work just fine. The valid range for **Strum** is 0 to 100.

You can specify a different **Strum** for each bar in a sequence. Repeated values can be represented with a "/":

```
Chord Strum 20 5 / 10
```

Note: When chords have both a *strum* and *invert* applied, the order of the notes played will not necessarily be root, third, etc. The notes are sorted into ascending order, so for a C major scale with and *invert* of 1 the notes played would be "E G C". That is, unless the *Direction* (see page 93) has been set to "DOWN" in which case the order would be reversed (but the notes would be the same).

16.20 Transpose

You can change the key of a piece with the “Transpose” command. For example, if you have a piece notated in the key of “C” and you want it played back in the key of “D”:

```
Transpose 2
```

will raise the playback by 2 semi-tones. Since I play tenor saxophone, I quite often do:

```
Transpose -2
```

which puts the MIDI keyboard into the same key as my horn.

You can use any value between -12 and 12. All tracks (with the logical exception of the drum tracks) are effected by this command.

16.21 Unify

The *Unify* command is used to force multiple notes of the same voice and pitch to be combined into a single, long, tone. This is very useful when creating a sustained voice track. For example, consider the following which might be used in real groove file:

```
Begin Bass-Sus
 Sequence  1 1 1 90  4
 Articulate 100
 Unify On
 Voice TremoloStrings
End
```

Without the *Unify On* command the strings would be sounded (or hit) four times during each bar; with it enabled the four hits are combined into one long tone. This tone can span several bars if the note(s) remain the same.

The use of this command depends on a number of items:

♫ The *Voice* being used. It makes sense to use enable the setting if using a sustained tone like “Strings”; it probably doesn’t make sense if using a tone like “Piano1”.

♫ For tones to be combined you will need to have *Articulate* set to a value of 100. Otherwise the on/off events will have small gaps in them which will cancel the effects of *Unify*.

♫ Ensure that *Rtime* is not set for *Unify* tracks since the start times may cause gaps.

♫ If your pattern or sequence has different volumes in different beats (or bars) the effect of a *Unify* will be to igore volumes other than the first. Only the first *Note On* and the last *Note Off* events will appear in the MIDI file.

You can specify a different *Unify* for each bar in a sequence. Repeated values can be represented with a “/”:

```
Chord Unify On / / Off
```

But, we're not sure why you'd want to.

Valid arguments are "On" or "1" to enable; "Off" or "0" to disable.

16.22 Voice

The MIDI instrument or voice used for a track is set with:

```
Chord-2 Voice Piano1
```

Voices apply only to the specified track. The actual instrument can be specified via the MIDI instrument number, or with the symbolic name. See the tables in the MIDI voicing section (see page 120) for lists of the recognized names.

You can create interesting effects by varying the voice used with drum tracks. By default "Voice 0" is used. However, you can change the drum voices. Our library files do not change the voices since this appears to be highly dependent on the MIDI synth you are using.

You can specify a different **Voice** for each bar in a sequence. Repeated values can be represented with a "/":

```
Chord Voice Piano1 / / Piano2
```

16.23 VoiceTr

In previous section we saw how to set a voice for a track by using its standard MIDI name. The *VoiceTr* command sets up a translation table that can be used in two different situations:

- ♫ It permits creation of your own names for voices (perhaps for a foreign language),
- ♫ It lets you override or change voices used in standard library files.

VoiceTr works by setting up a simple translation table of "name" and "alias" pairs. Whenever MMA encounters a voice name in a track command it first attempts to translate this name though the alias table.

To set a translation (or series of translations):

```
VoiceTr Piano1=Clavinet Hmmm=18
```

Note that you additional *VoiceTr* commands will add entries to the existing table. To clear the table use the command with no arguments:

```
VoiceTr // Empty table
```

Assuming the first command, the following will occur:

```
Chord-Main Voice Hmmm
```

The *Voice* for the *Chord-Main* track will be set to “18” or “Organ3”.

```
Chord-2 Voice Piano1
```

The *Voice* for the *Chord-2* track will be set to “Clavinet”.

If your synth does not follow standard GM-MIDI voice naming conventions you can create a translation table which can be included in all your MMA song files via an RC file. But, do note that the resulting files will not play properly on a synth conforming to the GM-MIDI specification.

Following is an abbreviated and untested example for using an obsolete and unnamed synth:

```
VoiceTr Piano1=3 \
  Piano2=4 \
  Piano3=5 \
  ...  \
  Strings=55 \
  ...
```

Notes: the translation is only done one time and no verification is done when the table is created.

Chapter 17

Begin/End Blocks

Entering a series of directives for a specific track can get quite tedious. To make the creation of library files a bit easier, you can create a block. For example, the following:

```
Drum Define X 0 2 100; 50 2 90
Drum Define Y 0 2 100
Drum Sequence X Y
```

Can be replaced with:

```
Drum Begin
    Define X 0 2 100; 50 2 90
    Define Y 0 2 100 End
Drum Sequence X Y
```

Or, even more simply, with:

```
Drum Begin Define
    X 0 2 100; 50 2 90
    Y 0 2 100
End
```

If you examine some of the library files you will see that we use this shortcut a lot.

17.1 Begin

The **Begin** command requires any number of arguments. Valid examples include:

```
Begin Drum
Begin Chord2
Begin Walk Define
```

Once a **Begin** block has been entered, all subsequent lines have the words from the **Begin** command prepended to each line of data. There is not much magic here—**Begin/End** is really just some syntactic sugar.

17.2 End

To finish off a **Begin** block, use a single **End** on a line by itself.

Defining musical data, repeats, or other **Begin**s inside a block (other than COMMENT blocks) will not work.

Nesting is permitted. Eg:

```
Scale Begin
    Begin Define
        stuff
    End
    Sequence stuff
End
```

A **Begin** must be competed with a **End** before the end of a file, otherwise an error will be generated. The **Use** and **Include** commands are not permitted inside a block.

Chapter 18

Documentation Strings

We've mentioned a few times already the importance of clearly documenting your files and library files. For the most part, you can use comments in your files; but in library files we suggest you use the **Doc** directive.

In addition to the commands listed in this chapter, you should also note the *DefGroove* section (see page 33).

For some real-life examples of how to document your library files, look at any of the library files supplied with this distribution.

18.1 Doc

A *Doc* command is pretty simple:

```
Doc This is a documentation string!
```

In most cases, **Doc**s are treated as **Comments**. However, if the **-Dx**[1] option is given on the command line, **Doc**s are processed and printed to standard output.

For producing the MMA **Standard Library Reference** a trivial Python program is used to collate the output generated with a command like:

```
mma -Dx -w /usr/local/lib/mma/swing
```

Note, we added the '-w' option to suppress the printing of warning messages.

18.2 Author

As part of the documentation package, there is a *Author* command:

```
Author Bob van der Poel
```

Currently *Author* lines are processed and the data is saved, but never used. It may be used in a future library documentation procedures, so you should use it in any library files your write.

[1]See the command summary (see page 11).

Chapter 19

Paths, Files and Libraries

This chapter covers MMA filenames, extensions and a variety of commands and/or directives which effect the way in which files are read and processed.

But, first a few comments on the location of the MMA Python modules.

The Python language (which was used to write MMA) has a very useful feature: it can include other files and refer to functions and data defined in these files. A large number of these files or modules are included in every Python distribution. The program MMA consists of a short "main" program and several "module" files. Without these additional modules MMA will not work.

The only sticky problem in a program intended for a wider audience is where to place these modules. We've decided that they should be in one of three locations:

- `/usr/local/share/mma/modules`
- `/usr/share/mma/modules`
- `./modules`

If, when initializing itself, MMA cannot find one of the above directories, it will terminate with an error message.

19.1 File Extensions

For most files the use of a the filename extension ".mma" is optional. However, we suggest that most files (with the exceptions listed below) have the extension present. It makes it much easier to identify MMA song and library files and to do selective processing on these files.

In processing an input song file MMA can encounter several different types of input files. For all files, the initial search is done by adding the filename extension ".mma" to filename (unless it is already present), then a search for the file as given is done.

For files included with the *Use* directive, the directory set with *setLibPath* is first checked, followed by the current directory.

For files included with the *Include* directive, the directory set with *setIncPath* is first checked, followed by the current directory.

Following is a summary of the different files supported:

Song Files The input file specified on the command line should always be named with the ".mma" extension. When MMA searches for the file it will automatically add the extension if the file name specified does not exist and doesn't have the extension.

Library Files Library files *really should* all be named with the extension. MMA will find non-extension names when used in a *Use* or *Include* directive. However, it will not process these files when creating indexes with the "-g" command line option—these index files are used by the *Groove* commands to automatically find and include libraries.

RC Files As noted in the RC-File discussion (see page 111) MMA will automatically include a variety of "RC" files. You can use the extension on these files, but common usage suggests that these files are probably better without.

MMAstart and MMAend MMA will automatically include a file at the beginning or end of processing (see page 111). Typically these files are named *MMAstart* and *MMAend*. Common usage is to *not* use the extension if the file is in the current directory; use the file if it is in an "includes" directory.

One further point to remember is that filenames specified on the command line are subject to wildcard expansion via the shell you are using.

19.2 Eof

Normally, a file is processed until its end. However, you can short-circuit this behavior with the **Eof** directive. If MMA finds a line starting with **EOF** no further processing will be done on that file … it's just as if the real end of file was encountered. Anything on the same line, after the **Eof** is also discarded.

You may find this handy if you want to test process only a part of a file, or if you making large edits to a library file. It is often used to quit when using the *Label* and *Goto* directives to simulate constructs like **D.C. al Coda**, etc.

19.3 LibPath

The search for library files can be set with the LibPath variable. To set **LibPath**:

```
SetLibPath PATH
```

You can have only one path in the **SetLibPath** directive.

When MMA starts up it sets the library path to the first valid directory in the list:

- ♫ `/usr/local/share/mma/lib`
- ♫ `/usr/share/mma/lib`
- ♫ `./lib`

The last choice lets you run MMA directly from the distribution directory.

You are free to change this to any other location in a RCFile (see page 111).

The LibPath is used by the routine which auto-loads grooves from the library, and the *Use* directive. The -g command line option is used to maintain the library database (see page 12).

You can include a leading "~/" in the path. In this case the path will be expanded to a complete pathname.

19.4 OutPath

MIDI file generation is to an automatically generated filename (see page 11). If the **OutPath** variable is set, that value will be prepended to the output filename. To set the value:

```
SetOutPath PATH
```

Just make sure that "PATH" is a simple pathname with **no** spaces in it. The variable is case sensitive (assuming that your operating system supports case sensitive filenames). This is a common directive in a RC file (see page 111). By default, it has no value.

You can disable the **OutPath** variable by not using an argument in the **SetOutPath** directive.

The PATH used in this command is processed though the Python *os.path.expanduser()* library routine, so it is permissible to include a leading "~" in the name (which expands, on Unix and Linux systems, to the name of the user's home directory).

If the name set by this command begins with a ".", "/" or "\" it is prepended to the complete filename specified on the command line. For example, if you have the input filename `test.mma` and the output path is `~/mids`—the output file will be `/home/bob/mids/test.mid`.

If the name doesn't start with the special characters noted in the preceeding paragraph the contents of the path will be inserted before the filename portion of the input filename. Again, an example: the input filename is `mma/rock/crying` and the output path is "midi"—the output file will be `mma/rock/midi/crying.mid`.

19.5 Include

Other files with sequence, pattern or music data can be included at any point in your input file. There is no limit to the level of includes.

```
Include Filename
```

A search for the file is done in the **IncPath** directory (see below) and the current directory. The ".mma" filename extension is optional.

The use of this command should be quite rare in user files. We use it extensively in our library files to include standard patterns.

19.6 IncPath

The search for include files can be set with the **IncPath** variable. To set **IncPath**:

```
SetIncPath PATH
```

You can have only one path in the **SetIncPath** directive.

When MMA initializes it sets the include path to first found directory in:

♫ `/usr/local/share/mma/includes`

♫ `/usr/share/mma/includes`

♫ `./includes`

The last location lets you run MMA from the distribution directory.

If this value is not appropriate for your system, you are free to change it in a RC File. You can include a leading "~/" in the path. In this case the path will be expanded to a complete pathname.

19.7 Use

Similar to **Include**, but a bit more useful. The **Use** command is used to include library files and their predefined grooves.

Compared to **Include**, **Use** has important features:

♫ The search for the file is done in the paths specified by the LibPath variable,

♫ The current state of the program is saved before the library file is read and restored when the operation is complete.

Let's examine each feature in a bit more detail.

When a **Use** directive is issued, eg:

```
use stdlib/swing
```

MMA first attempts to locate the file "stdlib/swing" in the directory specified by *LibPath* or the current directory. As mentioned above, MMA automatically added the ".mma" extension to the file and checks for the non-extension filename if that can't be found.

If things aren't working out quite right, check to see if the filename is correct. Problems you can encounter include:

♫ Search order: you might be expecting the file in the current directory to be used, but the same filename exists in the *LibPath*, in which case that file is used.

♫ Not using extensions: Remember that files *with* the extension added are first checked.

- ♫ Case: The filename is **case sensitive**. The files “Swing” and “swing” are not the same. Since most things in MMA are case insensitive, this can be an easy mistake to make.
- ♫ The file is in a subdirectory of the *LibPath*. In a standard distribution the actual library files are in `/usr/local/share/mma/lib/stdlib`, but the libpath is set to `/usr/local/share/mma/lib`. In this case you must name the file to be used as *stdlib/rhumba* **not** *rhumba*.

As mentioned above, the current state of the compiler is saved during a **Use**. MMA accomplishes this by issuing a slightly modified **DefGroove** and **Groove** command before and after the reading of the file. Please note that **Include** doesn’t do this. But, don’t let this feature fool you—since the effects of defining grooves are cumulative you *really should* have *SeqClear* statements at the top of all your library files. If you don’t you’ll end up with unwanted tracks in the grooves you are defining.

In most cases you will not need to use the *Use* directive in your music files. If you have properly installed MMA and keep the MMADIR files up-to-date by using the command:

```
mma -g
```

grooves from library files will be automatically found and loaded. Internally, the *Use* directive is used, so existing states are saved.

If you are developing new or alternate library files you will find the *Use* directive handy.

19.8 MmaStart

If you wish to process a certain file or files before your main input file, set the **MmaStart** filename in an RCFile. For example, we have a number of files in a directory which we wish certain **Pan** settings. In that directory, we have a file `mmarc` which contains the following command:

```
MmaStart setpan
```

The actual file `setpan` has the following directives:

```
Bass Pan 0
Bass1 Pan 0
Bass2 Pan 0
Walk Pan 0
Walk1 Pan 0
Walk2 Pan 0
```

So, before each file in that directory is processed, the **Pan** for the bass and walking bass voices are set to the left channel.

If the file specified by a **MmaStart** directive does not exist a warning message will be printed (this is not an error).

Also useful is the ability to include a generic file with all the MIDI files you create. For example, we like to have a MIDI reset at the start of our files, so we have the following in our `mmarc` file:

```
MMAstart reset
```

This includes the file `reset.mma` located in the "includes" directory (see page 109).

Because it is not uncommon to have multiple **mmarc** files, each with a different *MMAstart* directive, the files are appended to the existing list. Each file will be processed in the order it is declared. You can have multiple filenames on a *MMAstart* line.

19.9 MmaEnd

Just the opposite of **MmaStart**, this command specifies a file to be included at the end of a main input file. See our comments above for more details.

To continue our example, in our `mmarc` file we have:

```
MmaEnd nopan
```

and in the file `nopan` we have:

```
Bass Pan 64
Bass1 Pan 64
Bass2 Pan 64
Walk Pan 64
Walk1 Pan 64
Walk2 Pan 64
```

If the file specified by a **MmaEnd** directive does not exist a warning message will be printed (this is not an error).

Because it is not uncommon to have multiple **mmarc** files, each with a different *MMAend* directive, the files are appended to the existing list. Each file will be processed in the order it is declared. You can have multiple filenames on a *MMAend* line.

19.10 RC Files

When MMA starts it checks for initialization files. Only the first found file is processed.

The following files are checked (in order):

1. mmarc
2. `~/.mmarc`
3. `/usr/local/etc/mmarc`
4. `/etc/mmarc`

All found files will be processed.

Note that the second file is an “invisible” file due to the leading “.” in the filename.

By default, no rc files are installed.

The rc file is processed as a MMA input file. As such, it can contain anything a normal input file can, including music commands. However, we suggest you limit the contents of your RC files to things like:

```
SetOutPath
SetLibPath
MMAStart
MMAEnd
```

A useful setup is to have your source files in one directory and MIDI files saved into a different directory. Having the file `mmarc` in the directory with the source files permits setting **OutPath** to the MIDI path.

19.11 Library Files

Included in this distribution are a number of predefined patterns, sequences and grooves. They are in different files in the “lib” directory.

The library files should be self-documenting. A list of standard file and the grooves they define is included in the separate document, supplied in this distribution as “**mma-lib.ps**”.

Chapter 20

Creating Effects

It's really quite amazing how easy and effective it is to create different patterns, sequences and special effects. As we develop the program we try lots of silly things…this chapter is an attempt to display and preserve some of them.

The examples don't show any music to apply the patterns or sequences to. We assume that if you've gotten this far in the manual you'll know that you should have something like:

```
1 C
2 G
3 G
4 C
```

as a simple test piece to apply tests to.

20.1 Overlapping Notes

We've mentioned earlier that you should create patterns so that notes don't overlap. However, here's an interesting effect which relies on ignoring that advice:

```
Begin Scale
    define S1 1 1+1+1+1 90
    define S32 S1 * 32
    Sequence S32
    ScaleType
    Direction Both
    Voice Accordion
    Octave 5
End
```

We define "S1" with a note length of 4 whole notes (1+1+1+1) so that when we multiply it for S32 we end up with a pattern of 32 8th notes. Of course, the notes overlap. Running this up and down a chromatic scale is "interesting." You might want to play with this a bit and try changing "S1" to:

```
define S1 1 1 90
```

to see what the effect is of the notes overlapping.

20.2 Jungle Birds

Here's another use for **Scale**s. We decided that some jungle sounds would be perfect as an introduction to "Yellow Bird".

```
groove Rhumba
Begin Scale
    define S1 1 1 90
    define S32 S1 * 32
    Sequence S32
    ScaleType Chromatic
    Direction Random
    Voice BirdTweet
    Octave 5 6 4 5
    RVolume 30
    Rtime 2 3 4 5
    Volume pp pp ppp ppp
End
DefGroove BirdRumba
```

The above is an extract from the MMA score. The entire song is included in the "songs" directory of this distribution.

A neat trick is to create the bird sound track and then add it to the existing Rhumba groove. Then we define a new groove. Now we can select either the library "rhumba" or our enhanced "BirdRhumba" with a simple **Groove** directive.

Chapter 21

Frequency Asked Questions

This chapter will serve as a container for questions asked by some enthusiastic MMA users. It may make some sense in the future to distribute this information as a separate file.

21.1 AABA Song Forms

How can one define parts as part "A", part "B" . . . and arrange them at the end of the file? An option to repeat a "solo" section a number of times would be nice as well.

Using MMA variables and some simple looping, one might try something like:

```
Groove Swing
// Set the music into a
// series of macros
mset A
  Print Section A
  C
  G
endmset
mset B
  print Section B
  Dm
  Em
endmset
mset Solo
  Print Solo Section $Count
  Am / B7 Cdim
endmset
// Use the macros for an
// "A, A, B, Solo * 8, A"
// form
$A
$A
$B
set Count 1
label a
  $solo
  inc COUNT
  if le $count 8
    goto A
  endif
$A
```

Note that the "Print" lines are used for debugging purposes. We have mixed the case of the variable names just to illustrate the fact that "Solo" is the same as "SOLO" which is the same as "solo".

Now, if you don't like things that look like old BASIC program code, you could just as easily duplicate the above with:

```
Groove Swing
repeat
  repeat
    Print Section A
    C
    G
    If Def count
      eof
    Endif
    Endrepeat
    Print Section B
    Dm
    Em
    Set Count 1
    Repeat
      Print Solo $Count
      Am
      Inc Count
    Repeatending 7
  Repeatend
Repeatend
```

The choice is up to you.

21.2 Where's the GUI?

I really think that MMA *is a cool program. But, it needs a* GUI. *Are you planning on writing one? Will you help me if I start to write one?*

Well, we appreciate the fact that you like MMA. We like it too.

We've actually started to write a number of *GUI*s for MMA. But, nothing seemed to be much more useful than the existing text interface. So, we figured that it just wasn't worth the bother.

Now, we are not against graphical programming interfaces. We just don't see it in this case.

But, we may well be wrong. If you think it'd be better with a *GUI* ... well, this is open source and you are more than welcome to write one. If you do, we'd suggest that you make your program a front-end which lets a user compile standard MMA files. If you find that more error reporting, etc. is required to interact properly with your code, let us know and we'll probably be quite willing to make those kind of changes.

21.3 Where's the manual index?

We agree that this manual needs an index. We just don't have the time to go though and do all the necessary work. Is there a volunteer?

Appendix A

Symbols and Constants

This appendix is a reference to the chords that MMA recognizes and name/value tables for drum and instrument names. The tables have been auto-generated by MMA using the -D options.

A.1 Chord Names

MMA recognizes standard cord names as listed below. The names are case sensitive and must be entered in uppercase letters as shown:

A	C♯	E♭
A♯	C♭	F
A♭	D	F♯
B	D♯	F♭
B♯	D♭	G
B♭	E	G♯
C	E♯	G♭

Please note that in your input files you must use a lowercase "b" to represent a ♭ and a "#" for a ♯.

The following types of chords are recognized (these are case sensitive and must be in the mixed upper and lowercase shown):

+	See "aug".
11	9th chord plus 11th.
11b9	9th chord plus flat 11th.
13	Dominant 7th (including 5th) plus 13th.
6	Major tiad with added 6th.
7	Dominant 7th.
7#11	See "9#11".
7#5	7th, sharp 5.
7#5#9	Dominant 7th with sharp 5th and sharp 9th.
7#5b9	Dominant 7th with sharp 5th and flat 9th.

7#9	Dominant 7th with sharp 9th.
7#9#11	Dominant 7th plus sharp 9th and sharp 11th.
7+	See "aug7".
7+5	See "7#5".
7+9	See "7#9".
7-5	See "7b5".
7-9	See "7b9".
7b5	7th, flat 5.
7b5#9	Dominant 7th with flat 5th and sharp 9th.
7b5b9	Dominant 7th with flat 5th and flat 9th.
7b9	Dominant 7th with flat 9th.
7sus	7th with suspended 4th, dominant 7th with 3rd raised half tone.
7sus2	A sus2 with dominant 7th added.
7sus4	See "sus4".
9	Dominant 7th plus 9th.
9#11	Dominant 7th plus 9th and sharp 11th.
9#5	Dominant 7th plus 9th with sharp 5th.
9b5	Dominant 7th plus 9th with flat 5th.
M	Major triad. This is the default and is used in the absense of any other chord type specification.
M13	Major 7th (including 5th) plus 13th.
M7	Major 7th.
M7#11	Major 7th plus 9th and sharp 11th.
M7b5	Major 7th with a flatted 5th.
M9	Major 7th plus 9th.
aug	Augmented triad.
aug7	An augmented chord (raised 5th) with a dominant 7th.
aug7b9	Augmented 7th with flat 5th and sharp 9th.
dim	Diminished. MMA assumes a diminished 7th.
dim7	See "dim".
m	Minor triad.
m#5	Major triad with augmented 5th.
m(maj7)	See "mM7".
m(sus9)	Minor triad plus 9th (no 7th).
m+5	See "m#5".
m+7	See "mM7".
m11	9th with minor 3rd, plus 11th.
m6	Minor 6th.
m7	Minor 7th.
m7-5	See "m7b5".
m7b5	Minor 7th, flat 5 (aka 1/2 diminished).
m7b9	Minor 7th with added flat 9th.
m9	Minor triad plus 7th and 9th.

mM7	Minor Triad plus Major 7th. You will also see this printed as “m(maj7)”, “m+7”, “min(maj7)” and “min#7” (which MMA accepts); as well as the MMA *invalid* forms: “-(Δ7)”, and “min♮7”.
maj7	See “M7”.
mb5	Minor triad with flat 5th.
min#7	See “mM7”.
min(maj7)	See “mM7”.
sus	See “sus4”.
sus2	Suspended 2nd, major triad with major 2nd above root substituted for 3rd.
sus4	Suspended 4th, major triad with 3rd raised half tone.
sus9	Dominant 7th plus 9th, omit 7th.

In modern pop charts the “M” in a major 7th chord (and other major chords) is often represented by a “Δ”. When entering these chords, just replace the “Δ” with an “M”. For example, change “GΔ7” to “GM7”.

Modern pop charts sometimes use “slash” chords in the form “Am/E”. MMA is not capable of correctly interpreting this notation. If you encounter it just leave the “slash” part off and all should work fine. See your favorite music theory book or teacher for an explanation!

A chord name without a type is interpreted as a major chord (or triad). For example, the chord “C” is identical to “CM”.

A.2 MIDI Voices

When setting a voice for a track (ie Bass Voice NN), you can specify the patch to use with a symbolic constant. Any combination of upper and lower case is permitted. The following are the names with the equivalent voice numbers:

A.2.1 Voices, Alphabetically

5thSawWave	86	EnglishHorn	69	Organ2	17
Accordion	21	Fantasia	88	Organ3	18
AcousticBass	32	Fiddle	110	OverDriveGuitar	29
AgogoBells	113	FingeredBass	33	PanFlute	75
AltoSax	65	Flute	73	Piano1	0
Applause/Noise	126	FrenchHorn	60	Piano2	1
Atmosphere	99	FretlessBass	35	Piano3	2
BagPipe	109	Glockenspiel	9	Piccolo	72
Bandoneon	23	Goblins	101	PickedBass	34
Banjo	105	GuitarFretNoise	120	PizzicatoString	45
BaritoneSax	67	GuitarHarmonics	31	PolySynth	90
Bass&Lead	87	GunShot	127	Recorder	74
Bassoon	70	HaloPad	94	ReedOrgan	20
BirdTweet	123	Harmonica	22	ReverseCymbal	119
BottleBlow	76	HarpsiChord	6	RhodesPiano	4
BowedGlass	92	HelicopterBlade	125	Santur	15
BrassSection	61	Honky-TonkPiano	3	SawWave	81
BreathNoise	121	IceRain	96	SeaShore	122
Brightness	100	JazzGuitar	26	Shakuhachi	77
Celesta	8	Kalimba	108	Shamisen	106
Cello	42	Koto	107	Shanai	111
Charang	84	Marimba	12	Sitar	104
ChifferLead	83	MelodicTom1	117	SlapBass1	36
ChoirAahs	52	MetalPad	93	SlapBass2	37
ChurchOrgan	19	MusicBox	10	SlowStrings	49
Clarinet	71	MutedGuitar	28	SoloVoice	85
Clavinet	7	MutedTrumpet	59	SopranoSax	64
CleanGuitar	27	NylonGuitar	24	SoundTrack	97
ContraBass	43	Oboe	68	SpaceVoice	91
Crystal	98	Ocarina	79	SquareWave	80
DistortonGuitar	30	OrchestraHit	55	StarTheme	103
EPiano	5	OrchestralHarp	46	SteelDrums	114
EchoDrops	102	Organ1	16	SteelGuitar	25

Strings	48	SynthVox	54	TubularBells	14
SweepPad	95	TaikoDrum	116	Vibraphone	11
SynCalliope	82	TelephoneRing	124	Viola	41
SynthBass1	38	TenorSax	66	Violin	40
SynthBass2	39	Timpani	47	VoiceOohs	53
SynthBrass1	62	TinkleBell	112	WarmPad	89
SynthBrass2	63	TremoloStrings	44	Whistle	78
SynthDrum	118	Trombone	57	WoodBlock	115
SynthStrings1	50	Trumpet	56	Xylophone	13
SynthStrings2	51	Tuba	58		

A.2.2 Voices, By MIDI Value

0	Piano1	28	MutedGuitar	56	Trumpet
1	Piano2	29	OverDriveGuitar	57	Trombone
2	Piano3	30	DistortonGuitar	58	Tuba
3	Honky-TonkPiano	31	GuitarHarmonics	59	MutedTrumpet
4	RhodesPiano	32	AcousticBass	60	FrenchHorn
5	EPiano	33	FingeredBass	61	BrassSection
6	HarpsiChord	34	PickedBass	62	SynthBrass1
7	Clavinet	35	FretlessBass	63	SynthBrass2
8	Celesta	36	SlapBass1	64	SopranoSax
9	Glockenspiel	37	SlapBass2	65	AltoSax
10	MusicBox	38	SynthBass1	66	TenorSax
11	Vibraphone	39	SynthBass2	67	BaritoneSax
12	Marimba	40	Violin	68	Oboe
13	Xylophone	41	Viola	69	EnglishHorn
14	TubularBells	42	Cello	70	Bassoon
15	Santur	43	ContraBass	71	Clarinet
16	Organ1	44	TremoloStrings	72	Piccolo
17	Organ2	45	PizzicatoString	73	Flute
18	Organ3	46	OrchestralHarp	74	Recorder
19	ChurchOrgan	47	Timpani	75	PanFlute
20	ReedOrgan	48	Strings	76	BottleBlow
21	Accordion	49	SlowStrings	77	Shakuhachi
22	Harmonica	50	SynthStrings1	78	Whistle
23	Bandoneon	51	SynthStrings2	79	Ocarina
24	NylonGuitar	52	ChoirAahs	80	SquareWave
25	SteelGuitar	53	VoiceOohs	81	SawWave
26	JazzGuitar	54	SynthVox	82	SynCalliope
27	CleanGuitar	55	OrchestraHit	83	ChifferLead

84	Charang
85	SoloVoice
86	5thSawWave
87	Bass&Lead
88	Fantasia
89	WarmPad
90	PolySynth
91	SpaceVoice
92	BowedGlass
93	MetalPad
94	HaloPad
95	SweepPad
96	IceRain
97	SoundTrack
98	Crystal
99	Atmosphere
100	Brightness
101	Goblins
102	EchoDrops
103	StarTheme
104	Sitar
105	Banjo
106	Shamisen
107	Koto
108	Kalimba
109	BagPipe
110	Fiddle
111	Shanai
112	TinkleBell
113	AgogoBells
114	SteelDrums
115	WoodBlock
116	TaikoDrum
117	MelodicTom1
118	SynthDrum
119	ReverseCymbal
120	GuitarFretNoise
121	BreathNoise
122	SeaShore
123	BirdTweet
124	TelephoneRing
125	HelicopterBlade
126	Applause/Noise
127	GunShot

A.3 Drum Notes

When defining a drum tone, you can specify the patch to use with a symbolic constant. Any combination of upper and lower case is permitted. The following are the names with the equivalent note numbers:

A.3.1 Drum Notes, Alphabetically

Cabasa	69	LongLowWhistle	72	OpenSudro	86
Castanets	84	LowAgogo	68	OpenTriangle	81
ChineseCymbal	52	LowBongo	61	PedalHiHat	44
Claves	75	LowConga	64	RideBell	53
ClosedHiHat	42	LowTimbale	66	RideCymbal1	51
CowBell	56	LowTom1	43	RideCymbal2	59
CrashCymbal1	49	LowTom2	41	ScratchPull	30
CrashCymbal2	57	LowWoodBlock	77	ScratchPush	29
HandClap	39	Maracas	70	Shaker	82
HighAgogo	67	MetronomeBell	34	ShortGuiro	73
HighBongo	60	MetronomeClick	33	ShortHiWhistle	71
HighQ	27	MidTom1	47	SideKick	37
HighTimbale	65	MidTom2	45	Slap	28
HighTom1	50	MuteCuica	78	SnareDrum1	38
HighTom2	48	MuteHighConga	62	SnareDrum2	40
HighWoodBlock	76	MuteSudro	85	SplashCymbal	55
JingleBell	83	MuteTriangle	80	SquareClick	32
KickDrum1	36	OpenCuica	79	Sticks	31
KickDrum2	35	OpenHiHat	46	Tambourine	54
LongGuiro	74	OpenHighConga	63	VibraSlap	58

A.3.2 Drum Notes, by MIDI Value

27	HighQ	38	SnareDrum1	49	CrashCymbal1
28	Slap	39	HandClap	50	HighTom1
29	ScratchPush	40	SnareDrum2	51	RideCymbal1
30	ScratchPull	41	LowTom2	52	ChineseCymbal
31	Sticks	42	ClosedHiHat	53	RideBell
32	SquareClick	43	LowTom1	54	Tambourine
33	MetronomeClick	44	PedalHiHat	55	SplashCymbal
34	MetronomeBell	45	MidTom2	56	CowBell
35	KickDrum2	46	OpenHiHat	57	CrashCymbal2
36	KickDrum1	47	MidTom1	58	VibraSlap
37	SideKick	48	HighTom2	59	RideCymbal2

60	HighBongo	69	Cabasa	78	MuteCuica
61	LowBongo	70	Maracas	79	OpenCuica
62	MuteHighConga	71	ShortHiWhistle	80	MuteTriangle
63	OpenHighConga	72	LongLowWhistle	81	OpenTriangle
64	LowConga	73	ShortGuiro	82	Shaker
65	HighTimbale	74	LongGuiro	83	JingleBell
66	LowTimbale	75	Claves	84	Castanets
67	HighAgogo	76	HighWoodBlock	85	MuteSudro
68	LowAgogo	77	LowWoodBlock	86	OpenSudro

A.4 MIDI Controllers

When specifying a MIDI Controller in a *MidiSeq* or *MidiVoice* command you can use the absolute value in (either as a decimal number or in hexadecimal by prefixing the value with a "0x"), or the symbolic name in the following tables. The tables have been extracted from information at `http://www.midi.org/about-midi/table3.shtml`. Note that all the values in these tables are in hexadecimal notation.

Complete reference for this is not a part of MмA. Please refer to a detailed text on MIDI or the manaul for your synthesizer.

A.4.1 Controllers, Alphabetically

AllNotesOff	7b	Ctrl15	0f	Ctrl79	4f
AllSoundsOff	78	Ctrl20	14	Ctrl85	55
AttackTime	49	Ctrl21	15	Ctrl86	56
Balance	08	Ctrl22	16	Ctrl87	57
BalanceLSB	28	Ctrl23	17	Ctrl88	58
Bank	00	Ctrl24	18	Ctrl89	59
BankLSB	20	Ctrl25	19	Ctrl9	09
Breath	02	Ctrl26	1a	Ctrl90	5a
BreathLSB	22	Ctrl27	1b	Data	06
Brightness	4a	Ctrl28	1c	DataDec	61
Chorus	5d	Ctrl29	1d	DataInc	60
Ctrl102	66	Ctrl3	03	DataLSB	26
Ctrl103	67	Ctrl30	1e	DecayTime	4b
Ctrl104	68	Ctrl31	1f	Detune	5e
Ctrl105	69	Ctrl35	23	Effect1	0c
Ctrl106	6a	Ctrl41	29	Effect1LSB	2c
Ctrl107	6b	Ctrl46	2e	Effect2	0d
Ctrl108	6c	Ctrl47	2f	Effect2LSB	2d
Ctrl109	6d	Ctrl52	34	Expression	0b
Ctrl110	6e	Ctrl53	35	ExpressionLSB	2b
Ctrl111	6f	Ctrl54	36	Foot	04
Ctrl112	70	Ctrl55	37	FootLSB	24
Ctrl113	71	Ctrl56	38	General1	10
Ctrl114	72	Ctrl57	39	General1LSB	30
Ctrl115	73	Ctrl58	3a	General2	11
Ctrl116	74	Ctrl59	3b	General2LSB	31
Ctrl117	75	Ctrl60	3c	General3	12
Ctrl118	76	Ctrl61	3d	General3LSB	32
Ctrl119	77	Ctrl62	3e	General4	13
Ctrl14	0e	Ctrl63	3f	General4LSB	33

General5	50	Pan	0a	Resonance	47
General6	51	PanLSB	2a	Reverb	5b
General7	52	Phaser	5f	SoftPedal	43
General8	53	PolyOff	7e	Sostenuto	42
Hold2	45	PolyOn	7f	Sustain	40
Legato	44	Portamento	05	Tremolo	5c
LocalCtrl	7a	Portamento	41	Variation	46
Modulation	01	PortamentoCtrl	54	VibratoDelay	4e
ModulationLSB	21	PortamentoLSB	25	VibratoDepth	4d
NonRegLSB	62	RegParLSB	64	VibratoRate	4c
NonRegMSB	63	RegParMSB	65	Volume	07
OmniOff	7c	ReleaseTime	48	VolumeLSB	27
OmniOn	7d	ResetAll	79		

A.4.2 Controllers, by Value

00	Bank	19	Ctrl25	32	General3LSB
01	Modulation	1a	Ctrl26	33	General4LSB
02	Breath	1b	Ctrl27	34	Ctrl52
03	Ctrl3	1c	Ctrl28	35	Ctrl53
04	Foot	1d	Ctrl29	36	Ctrl54
05	Portamento	1e	Ctrl30	37	Ctrl55
06	Data	1f	Ctrl31	38	Ctrl56
07	Volume	20	BankLSB	39	Ctrl57
08	Balance	21	ModulationLSB	3a	Ctrl58
09	Ctrl9	22	BreathLSB	3b	Ctrl59
0a	Pan	23	Ctrl35	3c	Ctrl60
0b	Expression	24	FootLSB	3d	Ctrl61
0c	Effect1	25	PortamentoLSB	3e	Ctrl62
0d	Effect2	26	DataLSB	3f	Ctrl63
0e	Ctrl14	27	VolumeLSB	40	Sustain
0f	Ctrl15	28	BalanceLSB	41	Portamento
10	General1	29	Ctrl41	42	Sostenuto
11	General2	2a	PanLSB	43	SoftPedal
12	General3	2b	ExpressionLSB	44	Legato
13	General4	2c	Effect1LSB	45	Hold2
14	Ctrl20	2d	Effect2LSB	46	Variation
15	Ctrl21	2e	Ctrl46	47	Resonance
16	Ctrl22	2f	Ctrl47	48	ReleaseTime
17	Ctrl23	30	General1LSB	49	AttackTime
18	Ctrl24	31	General2LSB	4a	Brightness

4b	DecayTime	5d	Chorus	6f	Ctrl111
4c	VibratoRate	5e	Detune	70	Ctrl112
4d	VibratoDepth	5f	Phaser	71	Ctrl113
4e	VibratoDelay	60	DataInc	72	Ctrl114
4f	Ctrl79	61	DataDec	73	Ctrl115
50	General5	62	NonRegLSB	74	Ctrl116
51	General6	63	NonRegMSB	75	Ctrl117
52	General7	64	RegParLSB	76	Ctrl118
53	General8	65	RegParMSB	77	Ctrl119
54	PortamentoCtrl	66	Ctrl102	78	AllSoundsOff
55	Ctrl85	67	Ctrl103	79	ResetAll
56	Ctrl86	68	Ctrl104	7a	LocalCtrl
57	Ctrl87	69	Ctrl105	7b	AllNotesOff
58	Ctrl88	6a	Ctrl106	7c	OmniOff
59	Ctrl89	6b	Ctrl107	7d	OmniOn
5a	Ctrl90	6c	Ctrl108	7e	PolyOff
5b	Reverb	6d	Ctrl109	7f	PolyOn
5c	Tremolo	6e	Ctrl110		

Appendix B

Command Summary

Commands Requiring a Leading Track Specification

Accent	67	**Groove**	34	**Pan**	87
Articulate	90	**HarmonyOnly**	95	**Portamento**	88
ChShare	81	**Harmony**	94	**Range**	58
Channel	80	**Invert**	56	**Riff**	36
Compress	55	**Limit**	57	**ScaleType**	98
Copy	91	**Lyric**	42	**Sequence**	29
Debug	92	**MIDIClear**	87	**Strum**	99
Define	18	**MIDISeq**	84	**Tone**	24
Delete	93	**MIDIVoice**	86	**Unify**	100
Direction	93	**Mallet**	50	**Voice**	101
DrumType	49	**Octave**	96	**Voicing**	52
DupRoot	56	**Off**	96		
Duplicate	93	**On**	96		

Commands With an Optional Leading Track Specification

ChannelVolume	88	**RVolume**	69	**Volume**	68
Cut	63	**SeqClear**	30		
RSkip	97	**SeqNoRnd**	32		
RTime	98	**SeqRnd**	31		

Non-track Commands

AdjustVolume	68	**Decresc**	68	**Goto**	79
Author	105	**Dec**	75	**IfEnd**	77
AutoSoloTracks	49	**DefGroove**	33	**If**	77
Bar Numbers	39	**Doc**	105	**Include**	108
Bar Repeat	40	**EndIf**	77	**Inc**	75
BeatAdjust	60	**EndMset**	74	**KeySig**	49
Begin	103	**EndRepeat**	71	**Label**	79
ChannelPref	81	**End**	103	**MIDI**	82
Comment	91	**Eof**	107	**MidiFile**	83
Cresc	68	**Fermata**	62	**MmaEnd**	111