

## MF2T/T2MF

Two programs to manipulate standard midifiles.

mf2t is a program that reads a standard midifile (format 0 or 1) and writes an ASCII representation of it that is both compact and easily parsable.

t2mf is the companion program that reparses the text representation into a midifile.

-----  
Please report any problems, suggestion for improvement, or actual improvements to:

Piet van Oostrum, Dept of Computer Science, Utrecht University,  
P.O. Box 80.089, 2508 TB Utrecht, The Netherlands  
email: piet@cs.ruu.nl

-----  
You can do with this program what you like, but if you think it is useful I would appreciate if you send me some of your midifiles. Not ones that I can find on the Internet. And only PD ones. Please send them uu- or btoa-encoded. Zoo and Arc archives (encoded) are also OK, Zip and Lharc may be problematic.  
-----

The text representation is chosen such that it is easily recognized and manipulated by programs like sed, awk or perl. Yet it is also humanly readable so that it can be manipulated with an ordinary text editor.

In this way you can make changes to your midi files using these powerful programs or even a Fortran program :=). Or you can write algorithmic compositions using a familiar programming language.

The programs use the midifile library written by Tim Thompson (tjt@blink.att.com) and updated by Michael Czeiszperger (mike@pan.com). However, there were some bugs in the write code and I added some features that I needed. I also changed some of the names to cope with the 7-character limit for external identifiers in the Sozobon compiler. I will make an updated version of the library available soon. I also anticipate to split the read and write portions.

Usage:

**mf2t [-mntv] [-f[n]] [midifile [textfile]]**

translate midifile to textfile.

When textfile is not given, the text is written to standard output.  
When midifile is not given it is read from standard input. The meaning of the options is:

- m merge partial sysex into a single sysex message
- n write notes in symbolic rather than numeric form. A-C optionally followed by # (sharp) followed by octave number.
- b or
- t event times are written as bar:beat:click rather than a click number
- v use a slightly more verbose output
- f<n> fold long text and hex entries into more lines <n>=line length (default 80).

## **t2mf [-r] [[textfile] midifile]**

translate textfile to midifile.

When textfile is not given, text is read from standard input, when midifile is not given it is written to standard output.

-r     Use running status

Note that if one file is given it is always the midifile. This is so that on systems like Unix you can write a pipeline:

```
mf2t x.mid | sed ... | t2mf y.mid
```

### **Format of the textfile:**

The following representation of the midi events is generated (between []) the form when -v is used:

File header:	Mfile <format> <ntrks> <division>
Start of track:	MTrk
End of track:	TrkEnd

Note On:	On <ch> <note> <vol>
Note Off:	Off <ch> <note> <vol>
Poly Pressure:	PoPr[PolyPr] <ch> <note> <val>
Channel Pressure:	ChPr[ChanPr] <ch> <val>
Controller parameter:	Par[Param] <ch> <con> <val>
Pitch bend:	Pb <ch> <val>
Program change:	PrCh[ProgCh] <ch> <prog>
Sysex message:	SysEx <hex>
Arbitrary midi bytes:	Arb <hex>

Sequence nr:	Seqnr <num>
Key signature:	KeySig <num> <manor>
Tempo:	Tempo <num>
Time signature:	TimeSig <num>/<num> <num> <num>
SMPTE event:	SMPTE <num> <num> <num> <num> <num>

Meta text events:	Meta <texttype> <string>
Meta end of track:	Meta TrkEnd
Sequencer specific:	SeqSpec <type> <hex>
Misc meta events:	Meta <type> <hex>

**The <> have the following meaning:**

<ch>	ch=<num>
<note>	n=<noteval> [note=<noteval>]
<vol>	v=<num> [vol=<num>]
<val>	v=<num> [val=<num>]
<con>	c=<num> [con=<num>]
<prog>	p=<num> [prog=<num>]
<manor>	minor or major
<noteval>	either a <num> or A-G optionally followed by #, followed by <num> without intermediate spaces.
<texttype>	Text Copyright SeqName TrkName InstrName Lyric Marker Cue or <type>
<type>	a hex number of the form 0xab
<hex>	a sequence of 2-digit hex numbers (without 0x) separated by space
<string>	a string between double quotes (like "text").

Channel numbers are 1-based, all other numbers are as they appear in the midifile.

<division> is either a positive number (giving the time resolution in clicks per quarter note) or a negative number followed by a positive number (giving SMPTE timing).  
<format> <ntrks> <num> are decimal numbers.  
The <num> in the Pb is the real value (two midibytes combined)  
In Tempo it is a long (32 bits) value. Others are in the interval 0-127  
The SysEx sequence contains the leading F0 and the trailing F7.

In a string certain characters are escaped:

" and \ are escaped with a \

a zero byte is written as \0

CR and LF are written as \r and \n respectively

other non-printable characters are written as \x<2 hex digits>

When -f is given long strings and long hex sequences are folded by inserting \<newline><tab>. If in a string the next character would be a space or tab it will be escaped by \

This facility is for those programs that have a limited buffer length.

Of course parsing is more difficult with this option (see below).

### **Input:**

t2mf will accept all formats that mf2t can produce, plus a number of others.

Input is case insensitive (except in strings) and extra tabs and

spaces are allowed. Newlines are required but empty lines are allowed.

Comment starts with # at the beginning of a lexical item and continues to the end of the line. The only other places where a # is legal are insides strings and as a sharp symbol in a symbolic note.

In symbolic notes + and # are allowed for sharp, b and - for flat.

In bar:beat:click time the : may also be /

On input a string may also contain \t for a tab, and in a folded string any whitespace at the beginning of a continuation line is skipped.

Hex sequences may be input without intervening spaces if each byte is given as exactly 2 hex digits.

Hex sequences may be given where a string is required and vice versa.

Hex numbers of the form 0xaaa and decimal numbers are equivalent.

Also allowed as numbers are "bank numbers" of the form '123. In fact this is equivalent to the octal number 012 (subtract 1 from each digit, digits 1-8 allowed). The letters a-h may also be used for 1-8.

The input is checked for correctness but not extensively. An error message will generally imply that the resulting midifile is illegal.

### **Implementation:**

I have compiled the programs on an Atari ST with the Sozobon compiler and the dlibs library. The scanner is generated using flex 2.3. The output of flex (t2mflex.c) is included for those that do not have flex. The module yyread.c is a flex library module that you need on TOS (and on MSDOS). The corresponding makefile is makefile.st. For Unix use makefile.unx. For Borland C on MSDOS use makefile.bcc. The makefiles may need minor changes for other systems.

### Useful hints:

channel number can be recognized by the regular expression /ch=/.  
note numbers by /n=/ or /note=/, program numbers by /p=/ or /prog=/.  
Meta events by /<sup>^</sup>Meta/ or /<sup>^</sup>SeqSpec/.  
Text events by /"/, continued lines by /\\$/, continuation lines by /\\$\\t/  
(that was a TAB character).

In awk each parameter is a field, in perl you can use split to get the parameters (except for strings).

The following perl script changes note off messages to note on with vol=0, deletes controller 3 changes, makes some note reassignments on channel 10, and changes the channel numbers from channel 1 depending on the track number.

```
----- test.pl -----
%drum = (62, 36, 63, 47, 65, 61, 67, 40, 68, 54);

while (<>) {
    next if /c=3/;
    s/Off(.*)v=[0-9]*/On\\1v=0/;
    if (/ch=10/ && /n=([0-9]*)/ && $drum{$1}) {
        s/n=$1/"n=".$drum{$1}/e;
    }
    if (/^MTrk/) {++$trknr ; print "track $trknr\\n";}
    if ($trknr > 2) { s/ch=1\\b/ch=3/; }
    else { s/ch=1\\b/ch=4/; }
    print || die "Error: $!\\n";
}
-----
```

and this is the corresponding awk script.

```
----- test.awk -----
BEGIN { drum[62] = 36; drum[63] = 47; drum[65] = 61; \
        drum[67] = 40; drum[68] = 54 }
/c=3/ { next }
($2 == "Off") { $2 = "On"; $5 = "v=0" }
/ch=10/ { n = substr($4, 3); if (n in drum) $4 = "n=" drum[n] }
/^MTrk/ { trknr++ }
/ch=1 / { if (trknr > 2) { $3 = "ch=2" } else { $3 = "ch=3" } }
{ print }
-----
```

Good luck!

\$Id: readme.,v 1.5 1991/11/16 20:26:48 piet Rel \$