MidiLang (TM)

1.31

USER'S GUIDE

CompuServe : 100417,2633
100417.2633@compuserve.com

WEB PAGE : http://ourworld.compuserve.com/homepages/sib

DISCLAIMER OF WARRANTY

THIS SOFTWARE AND MANUAL ARE SOLD "AS IS" AND WITHOUT WARRANTIES AS TO PERFORMANCE OF MERCHANTABILITY OR ANY OTHER WARRANTIES WHETHER EXPRESSED OR IMPLIED. BECAUSE OF THE VARIOUS HARDWARE AND SOFTWARE ENVIRONMENTS INTO WHICH THIS PROGRAM MAY BE PUT, NO WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE IS OFFERED. GOOD DATA PROCESSING PROCEDURE DICTATES THAT ANY PROGRAM BE THOROUGHLY TESTED WITH NON-CRITICAL DATA BEFORE RELYING ON IT. THE USER MUST ASSUME THE ENTIRE RISK OF USING THE PROGRAM. ANY LIABILITY OF THE SELLER WILL BE LIMITED EXCLUSIVELY TO PRODUCT REPLACEMENT OR REFUND OF PURCHASE PRICE.

Release nb	Date	Descript
1.31	01/30/96	Effects on MIDI files. Faster process
1.30b	01/27/96	Device config saved, savein and unsavein cmd. mpldebug 1.2 (registered version)
1.3	01/17/96	Midi Mapping, Hexa Int, Init label, More variables
1.2	11/20/95	New MPL effects, some small memory troubles removed
1.1	09/09/95	Doc errors removed MidiLang Chan=Midi
1.0	08/15/95	First Release

0. Introduction

MidiLang has been created to :

- allow effects such as **echo**, **delay**, **loop** in **live** (i.e. during your play) or on **Standard Midi Files** (.mid files)
- do any kind of **Midi Mapping** on any type of Midi Event (control changes..)
- calculate **best chords** for a given melody
- be able to **add automatically bass** based on your play in live.
- allow user defined **splits** (not only those specified by the manufacturer of your keyboard)
- save/load your play in **.MID file**
- get an **interactive link** between the PC and your synthe: you can change any parameters, or any options directly from your midi keyboard.
  There is no more need to run from the PC keyboard to the Midi keyb.
- allow anyone to create his own effect with a simple but powerful interpreter of **Midi Programming Language (MPL)**
- allow users to create their own **library of effects**, easy to update and to share.

There is no need to be a programmer to use MidiLang, you can just use already done mpl files. But MPL is a so easy to use language that you should try to define your own effect.

For a real programmer, this language is simple, a little bit like assembler. I'm working on a compiler of a high level MPL. but don't forget that the aim of the low level MPL is to get a easy way to define fast effects. This language is very fast (actually it's compiled in memory during the load process) so can do almost anything you may have dreamed to do with your MIDI instrument.

0.1 Installation

To install MidiLang

unzip the compressed file you have received (midilang.zip)
run (with the program manager) the program **setup.exe** from this package.

MidiLang is installed in its own directory
(c:/midilang by default)

a set of important files can be found in this directory :

midilang.wri : this file
midilang.exe : MidiLang exe
order.txt : the register form

0.2 Registration

MidiLang is distributed in two forms :

ShareWare Version:

All effects functionalities, and MPL capabilities are available with the non-registered version of MidiLang.

All examples work, all user made mpl files work. The only limitation is the maximum number of mpl instructions inside user made mpl files :

With the non registered version of MidiLang, you can't create mpl file with more than 10 MPL instructions.

(but you can have as much blank lines, Label, Keydef, Descript or end, you want)

A "please register" screen appears at each load effect command.

Registered version :

With the registered version of MidiLang, the maximum number of MPL instructions is set to 1000.

No more "please register" screen.

Included in the registered package of MidiLang :

MidiLoop : MidiLoop is a mini-MidiLang specialized in echos. Very easy to use.

MplDebug : a simple but useful debugger of MPL files to trace MPL behaviour.

for 35 US$ (+ Handling, Postage), you get the up-to-date registered version of MidiLang.

Please see : order.txt to get details about how to register.

MidiLang can also be registered via **CompuServe** :

To do so, on CompuServe, go SWREG
Use the registration item.
MidiLang registration ID is 7059
You will receive the last registered version of MidiLang directly via email.

0.3 The menu of MidiLang :

File :

Open Effect...

-to load your MPL file : load effects file

this option will load your MPL file, check its syntax and update the screen

New MIDI

-to erase your current play

Open MIDI,Save MIDI

-to save your play or load any midi files (.mid format)

the file saved can be used by any other standard midi tools (such as a sequencer..)

Exit

-to end MidiLang

Settings :

Set Tempo

- to change the internal tempo of MidiLang
  (nb beats / min) default : 120

Set Cmd Channel

- to change the midi channel of the command
  midi (default 2)

Input / Output Filters

- to filter the Midi IN and OUT

Record

- to Start or Stop the effect

Play

- to Start or Stop the play back your play (with effects)
- to Run an effect on a loaded Midi File.
  With MidiLang 1.31, you can run any type of effect on a Midi file.
  See chap. 9 for more information about it.

Input/Output Device

- to change the Midi IN/OUT device; your setting will be saved.
  (default SB (if exist))

0.4 MidiLang Quick Start

1) Check your Input/Output device
in the menu Input / Output Device
choose the proper device.

2) Load an effect file :
in the menu File
submenu : Open Effect...
choose the file : echo.mpl

3) run the effect :
in the menu Record
choose : Start

4) play on your keyboard

on the screen, the beat number is displayed.
Every note played is played back after a small delay, several time with decreasing volume.
That's an almost real echo.

5) Load an effect file :
in the menu File
submenu : Open Effect...
choose the file : echom.mpl

3) run the effect :
in the menu Record
choose : Start

4) play on your keyboard

on the screen the beat number is displayed
Every note played is played back after 1 beat delay several time or only one time

Split your Midi keyboard, and configure the lower part of it to Midi Channel 2

Switch off the lower part of your keyboard
(if you don't know how to do it, I can't help you, you should read your midi Keyb. User Guide.)

Play on the lower part, a C and then a C#
Notice that the delay value on the screen has changed

Play, on the upper part, some notes
the delay has increased

Play on the lower part, a D and then a D#

play on the upper part some notes
the number of repeat has increased.

ENJOY
---

1. Description of MPL files :

1.1 Label

A MPL (Midi Programming Language) file contains a set of commands to be executed at every Midi Events (i.e. every time you press or release a key of your Midi Keyboard, at every beats or every time you send a command request with your Midi Keyboard, or at all control changes (breath control, modulation..).

These commands can be : change to an another channel, calculate harmony based on the notes I have played, play some bass with my melody, add delay, echo, change the controller action, the curve of the controller..

MPL files are structured into groups, a group is a set of command. A group starts with the keyword : LABEL name (ex. LABEL TEST) and finishes with the keyword : END

NOTE : MidiLang is case-insensitive

This example does an echo of your play :

```
# start of the main group
LABEL MAIN
TIME+= 240
OUTMIDI
END
```

4 labels are defined by default :

The **label MAIN** is the one used at every MidiIn (every time you touch a key at your Midi Keyb. this note, with its time and volume, will be sent to your MPL file and MidiLang will execute the MAIN group.)

The **label INIT** is the one used at the start of the effect. This optional label can be used to initialize a set of variables

The optional **label BEATS** is the one executed at each beat.

The optional **label MAPPER** is executed at each Midi Events except the Note events. It can be used to do a user-defined Midi Mapping (to swap two controllers for example)

You can used up to 200 labels.

Note : you can use label inside a group :

```
Label Main
time+= 240
outmidi
#if time > 1000 go to next
time> 1000
goto next
```

```
time+= 240
outmidi

label next
end
```

1.2 goto, gosub

Anywhere in a MPL file you can do a goto, or a gosub

goto label_name

will jump to the label label_name and continue the execution from this point until a END command and stop.

gosub label_name

will jump to the label label_name, continue the execution from this point until a END command and resume the execution after the gosub command.

The gosub command is useful when, for example, you have created an effect that you want to reuse,
just copy you effect into a group (inside a LABEL ... END)
and run it (with a gosub) every time you need it
the program will run your effect and continue the execution.

The goto command is useful for the test condition

if test is ok goto action1 else continue

ex.

```
TIME> 480
GOTO UPPER
END

LABEL UPPER
END
```

ex.

```
LABEL ECHO
TIME+= 240
OUTMIDI
END

LABEL MAIN
GOSUB ECHO
```

```
CHAN-= 1
TIME+= 240
OUTMIDI
END
```

The gosub ECHO is executed and the program resumes to the next command (CHAN-= 1 here)

NOTE 1: Gosub and Midi Value

Please note that any change done to the Midi Values (time, note/status, channel/data1,volume/data2) inside a gosub will be canceled at the end of this gosub.

In the previous example, the Main group run a gosub to the LABEL ECHO.

In this group, 240 is added to TIME but at the END of this group, TIME comes back to its initial value !!!

When, in LABEL MAIN, 240 is added to TIME, It has been added actually one time only.

This has been done to make sure that you can run a gosub without thinking about what this gosub will do to your midi values.

NOTE 2: OUTMIDI before a gosub

Please note that if in a gosub, the command OUTMIDI is used, this gosub will be ran for every OUTMIDI done before the gosub.

ex.

```
LABEL ECHO
TIME+= 240
END

LABEL MAIN
NOTE+= 1
OUTMIDI
NOTE+= 1
OUTMIDI
GOSUB ECHO
END
```

The gosub to the LABEL ECHO will be done two times, one for the first OUTMIDI, one for the second OUTMIDI.

This allows you to add, for example, an echo to all MidiOut you have done in the label MAIN, just by adding a gosub echo before the last END of your LABEL MAIN.

NOTE 3: gosub in a gosub

You can use as many gosubs you want, and you can have gosubs inside gosubs..

1.3 MIDI IN & OUT

Every time you play something on your midi keyboard, a MidiIn is received by MidiLang, and the Label MAIN is executed.

Every time you want MidiLang to play something on your Keyboard, you must use the command : OUTMIDI

The command OUTMIDI will play the note defined at the time defined.
The Midi Values are :
TIME : number of tics from start (480 tics = 1 beat)
CHAN or STATUS : Channel (Channel = Midi Channel)
NOTE or DATA1 : note number (C1=0, C2=12, D1=2..)
VEL or DATA2 : volume (0=OFF, 127=FULL)
DATA3 : for Control events

Please note that on the version 1.0 of MidiLang, Channel was set to Midi Channel + 143. (Midi Channel + 143 is in fact the real channel number for incoming events).
Because of that, both systems are available. (CHAN= 2 <-> CHAN= 145)

At each MidiIn these values are updated, you can change them and send them back to your synthe.

Ex.

```
LABEL MAIN
TIME+= 240
OUTMIDI
END
```

This file will add 240 (0.5 beat) to all In Midi and send it back.
It's a delay !! (sort of lexikon)

1.31 SAVEIN / UNSAVEIN :

By default, all notes played and all notes or events generated by the tool are saved in memory until the Record/Stop. You can then replay them or saved them into .mid files.

You can switch to save only the generated notes with the command : UNSAVEIN.The command : SAVEIN returns to the default status.

This can be used when you want to modify incoming midi events (swap two midi channel for example), only the generated notes should be saved in that case.

1.4 Tools

- RESCREEN which redraws the texts of the Window.
Useful especially to display the up-to-date values of the variables specified in keydef.
- Midi Mapping . see chap 7

2. List of Effect commands

The 'effects command' are used to get complex effects such as harmony calculation, or any effects too complex to be defined in mpl.
This section will be updated on each new release of MidiLang with new effects asked by users.

Midi Mapping :
Modify in live any Midi Event and change it the way you want

Harmony calculation :

This effect will calculate the chord you should use based on :
a tonality
a set of notes.

For example, if you choose a tonality of C
and give the notes : D# F G
This effect will give you the chord :
C-7-

Actually, you will have the scale of this chord : C D D# F G A A#

2.1 CALCHARM nb

The first time you use this command after the start or after a INITHARM or a OLDHARM, the current MidiIn Note defines the tonality (if you have played a C and after that you run CALCHARM, C will be your tonality)

After, the MidiIn Note will be taken to calculate the chord for this tonality.

The tonality will not change before a INITHARM or a OLDHARM.

CALCHARM will change two variables : V[nb] and V[nb+1]
V[nb] got the current tonality. You can change directly it by changing the value in V[nb]
V[nb+1] got a note number use to calculate the chord

2.2 GETHARM nb1 nb2 nb3

At any time, after at least one CALCHARM, you can have access to the scale of the current calculated chord.

You can, for example, ask for the second note of the scale of the current chord.

V= 10 2 set V[10] to 2
GETHARM nb1 10 nb3

GETHARM will calculate the Note number (1-12) of the second note of the current Chord. This Chord have been calculated by CALCHARM and saved in V[nb1] and V[nb1+1]
The Note number will be saved in V[nb3]

Ex.
CALCHARM 1 will save the chord in V[1],V[2]
V= 30 5
GETHARM 1 30 10 will calculate the note nb 5
(because V[30]=5)
of the chord saved in V[1],V[2]
and save it in V[10]

2.3 OLDHARM nb1

OLDHARM nb1 will reset the tonality.
The next CALCHARM will redefine it, and ask the Harmony calculator to decrease the weight of the note already used by the effect.
It allow changing to an other chord in a smooth way.

OLDHARM will work on the chord saved in V[nb1], V[nb2] by CALCHARM

2.4 INITHARM nb1

INITHARM nb1 will reset the tonality and start from zero the harmony calculation.
To be used only when you start a totally new play.

INITHARM will work on the chord saved in V[nb1], V[nb2] by CALCHARM

3. List of Description commands

The 'description commands' are used to describe
- the aim of your effect
- the link between the midi keyboard
  and your mpl file (midi command)

3.1 descript

the command "descript" adds a line of commentary at the top of the output screen of MidiLang.

you can use up to 10 lines of descriptions
Those lines are useful to explain what the effect is supposed to do.

syntax :

descript text_text_...
descript text_text_...
descript text_text_...

You can use this command anywhere in your mpl file, except at the first line.

ex.
descript this mpl file will create an echo effect
descript with delay, and nb loop modifiable directly
descript from your midi keyboard.

3.2 Commentary

Inside your Mpl file, you can use commentary

any line that starts with a # is a commentary
any text after a command and its parameters is a commentary

ex.

# this is a commentary
V= 1 2 this is too a commentary

3.3 Link between Midi Keyboard and Mpl File

MidiLang can use a midi channel to receive run request from the user.
For example, you can switch off the lower channel of your

keyboard, and send orders to MidiLang by just playing some notes in this channel.

Echom.mpl uses this feature to allow you to change the loop number, or the delay, just by playing notes.

MidiLang will wait for two keys (different keys) in the command channel, search for a keydef related to those key and run the label specified in the keydef command.

The command Channel number can be changed in the setting menu.
The command keys played are displayed at the top of MidiLang's window.

syntax:
keydef key1 key2 keydef_name label_name [var_name var_nb def_value.]

with key1 and key2 : key name (C,C#,D,D#,E,F,F#,G,G#,A,A#,B)
keydef_name : a name for this keydef (will appear in the window)
label_name : the label to be run if the user has played key1 and key2 in the command channel.

optional parameters :
var_name : a name for a variable (will appear in the window)
var_nb : the variable number (1-5000)
def_value : default value

you can use as many variables as you want in one keydef, the variable name and its value will be displayed in MidiLang window at each command run. (and at each rescreen command)

ex. keydef C C# to_slow_down slow delay 1 2.2

it defines a command called "to_slow_down", every time the user will play (in the command channel) the notes : C and C# (one after one), MidiLang will run the instruction starting at the "label slow" in the current mpl file (you must have this label in your file !).
But before running this label, the variable V[1] will be displayed with the name "delay"
you will have in your window :
C C# to_slow_down delay 2.200
because the default value of V[1] is 2.200.

If your mpl file changes V[1], next time you run a command (or do a rescreen) the new value will appear.

4. TUTORIAL

This tutorial will teach you how to do your own MPL files.

4.1 My first MPL file

With the Note-Pad tools of Windows, type :
(For MidiLang, Label = LABEL = label)

```
Label Main
time+= 240
outmidi
End
```

Save it to test1.mpl

Run MidiLang
Load this effect :
Menu : File
submenu : Open Effect...
choose test1.mpl
run the effect :
Menu : record
Submenu : Start
Play a note

You will notice that your note has been played back 0.5 beat after your play.
Why ?

When you press a key on your Midi Key, you send a note to MidiLang
Then MidiLang will run your mpl file from the label MAIN
The first instruction is (after the label)

```
time+= 240
```

which means : add 240 to time

time is the time of your midi event (the note you have played) , that's a number of impulses (of tic) since you have started the effect.
you have added 240 tics to the time of your note.
Because there is 480 tics per beat, adding 240 tics means adding 0.5 beats.

the next instruction is :

```
outmidi
```

outmidi will take your note (after your modif.)
and send it back to the synthe.

The note will be played at the time requested (0.5 beat after now, in our case)

The next instruction is END.
MidiLang go back to sleep until the next MidiEvent (your next note)

Note that playing a note creates 2 Midi Events :
one when you press it
one when you release it
Both will be executed by MidiLang.

4.2 My second MPL file : echo with decreasing volume

Now let's create a new file :

```
LABEL MAIN
TIME+= 240
VEL/= 2
OUTMIDI
END
```

Save it to test2.mpl

In this case two instructions are executed before the outmidi :
`time+= 240`
to add 0.5 beat to the time of my midi event

and `vel/= 2`
this instruction will divide by 2 the volume of the midiEvent

then when your note will be played back, it will be delayed and softer.

Try it and notice that the play back is softer.

4.3 My third MPL file : funny echo

Now let's have some fun :

```
LABEL MAIN
# this is my first variable
V+= 1 1
# this is my first test
V== 1 11
```

```
 V= 1 0
 TIME+= 240
 NOTE+=V 1
 OUTMIDI

 TIME+= 120
 VEL= 0
 OUTMIDI
 END
```

Save it to test3.mpl

In this example, you are using a new feature : variables and remarks.
Any line starting with a # is a remark, it will be simply ignored by MidiLang.

The first line (after LABEL MAIN and the remark) is V+= 1 1

It means : add 1 to the variable nb 1
the first one is the variable number
the second is the value to add

Note that at the start of MidiLang, all variables are set to 0

then the first time, MidiLang run this MPL,
the first variable will be set to 0+1 = 1

well, ok , why not ?

After, the next instruction is a little bit tricky !

```
 V>= 1 11
```

means : is the variable nb 1 equal or upper than 11 ?
This is a test, and if the test fails (if the answer of the question if NO) the next instruction will be ignored)

in our case (and for the first run) the variable 1 is equal to 1, then the answer is NO
the next instruction `V= 1 0` is then ignored
(This instruction was supposed to set the variable 1 to 0)

the next instruction is :

```
 time+= 240
```

well known...

the next is :

```
note+=V 1
```

this instruction means :

add to the note number the value of the variable nb 1

Note that the structure of the MidiEvent is:
Time, Chan or Status, Note or Data1, Vel or Data2, Data or Data3

in our case the variable 1 is (for the first run set to)1, then 1 is added to note.

Note is the note number.
C1 is equal to 0, C2 = 12, C3= 24 ....
D#1=3, D#3=27 ....

adding 1 to note means using the next note.

The Next instruction

```
Outmidi
```

Send, 0.5 beat after now, a higher note.

After the OutMidi
the instruction is :

```
TIME+= 240
```

we add once again 0.5 beat (we are 0.5+0.5=1 beat from now)
and

```
VEL= 0
```

we set the volume to 0.

And once again

```
OUTMIDI
```

This is just to be sure that the note we have send with the first OutMidi of the MPL file will be stopped.

Sending back a note, a little bit later, with a volume 0 stop it.

the next time MidiLang will run (at the next Midi Event), the Mpl will be run once again, and then the variable 1 will be set to 2, and the note will go higher and higher at each play until the test stops to fail :

when the variable 1 will be upper or equal to 11

then the instruction after the text will not be ignored and the variable 1 will go

back to 0.

and so on...

Try it, you will notice something strange, if you play a C you will have a C# (normal , C +1 = C#)
play once again a C, you will have a D# ???

this is because, every time you press a note, actually you press it twice :

one time when you press it
one time when you release your finger
the note is sent once again with a volume 0

but because the volume of the second Midi Event is 0 you won't listen it.

This is why we must send ourself the volume 0 of our note in the MPL

because the second MidiEvent with the volume 0 will be executed as the first one.

and its note number will be increased too.
then, if you don't play twice your note in the mpl file, the stop message will be send to an another note.

remove the second outmidi of your mpl and try it :

if you play C
two C will arrive to MidiLang
let say the volume of the first one is 100
first one : play back with volume 100
and note C + 1 = C#

second one : play back with volume 0
and note C + 2 = D

the stop message (volume 0) is sent to D
That means C will not stop, and D will stop
(D does not care)

This problem occur rarely only when you do special use of the tool.

4.4 Let's Loop

Try this example :

```
LABEL MAIN

  V= 1 0
  LABEL LOOP

  V+= 1 1
  TIME+= 240
  NOTE+= 1
  OUTMIDI
  V<= 1 10

  GOTO LOOP
END
```

This is your first loop ;-)

First the variable 1 is set to 0 (remember that you can use up to 5000 variables)
Then a second label (ignored for the moment)

1 is added to the variable 1
240 tic is added to time
1 is added to the note
the MidiEvent is sent

the test : is the variable 1 lower or equal to 10 ?
Yes, then the next instruction will be executed

the next instruction is
`GOTO LOOP`

This instruction force MidiLang to jump to the label specified and to continue from this location

Let's go back to LOOP
(the goto works even for a label located after the goto , not only before)

that's a loop

the variable 1 is increased at each loop
the time too
the note too
until the test fails (Variable upper than 10)

This mpl will output the 10 next notes
of everything you're playing.

5. Conclusion

MidiLang is a never finished tool, New effects will appear little by little.
Keep in touch with me and register to get the last release of this program and receive periodically new mpl files ready to be used.

Starting learning a new language (even as primitive as the MPL) is often frustrating, so please feel free to contact me if you have problems to report or suggestions for improvement. Your satisfaction is my goal!

6. List of basic commands

The 'basic commands' are the set of commands that allow calculations and basic modifications on the Midi values.

You have access to 5000 variables (float/integer), where you can save whatever you want.
You can use hexa integer (format : 0xNN (ex 16 <-> 0x10))

6.1 =

BEAT= value : the Beat number if set to value
ex. BEAT= 0
BEAT=V nb1 : the Beat number is set to the value of the variable nb1
ex. BEAT=V 1
means : Beat = V[1]
V=BEAT nb1 : the variable nb1 is set to Beat number
ex. V=BEAT 1
means : V[1] = Beat
V= nb1 value : the variable nb1 is set to value
ex. V= 10 1
means V[10]=1
V=V nb1 nb2 : the variable nb1 is set to the value of the variable nb2
ex. V=V 1 2
means V[1]=V[2]
V=VV nb1 nb2 : the variable nb1 is set to the value of the variable V[nb2]
ex. V=VV 1 2
means V[1]=V[V[2]]
if V[2] is set to 5, it means V[1]=V[5]
VV=V nb1 nb2 : the variable V[nb1] is set to the value of the variable nb2
ex. VV=V 1 2
means V[V[1]]=V[2]
if V[1] is set to 5, it means V[5]=V[2]
VV=VV nb1 nb2 : the variable V[V[nb1]] is set to the value of the variable V[V[nb2]]
ex. VV=VV 1 2
means V[V[1]]=V[V[2]]
if V[1] is set to 5 and V[2] to 6,
it means V[5]=V[6]
TIME= value : the MidiOut time is set to value
(value is a number of tic since start)

ex. TIME= 480
(Note: one Beat = 480 tics)

TIME=V nb1 : the MidiOut time is set to the value of the variable nb1
ex. TIME=V 1
means
time = V[1]

V=TIME nb1 : the variable nb1 is set to the value of the current MidiIn time
(number of tic since start)
(the current MidiIn is the last midi event received)
ex. V=TIME 1
means V[1]= time

CHAN= value : the MidiOut channel is set to value
ex. CHAN= 3
(Note: Channel = midi Chan)

CHAN=V nb1 : the MidiOut channel is set to the value of the variable nb1
ex. CHAN=V 1
means
Channel = V[1]

V=CHAN nb1 : the variable nb1 is set to the value of the current MidiIn channel
(the current MidiIn is the last midi event received)
ex. V=CHAN 1
means V[1]= Channel

VEL= value : the MidiOut volume/velocity is set to value
ex. VEL= 100
(Note: volume value : 0 to 127)

VEL=V nb1 : the MidiOut volume is set to the value of the variable nb1
ex. VEL=V 1
means
Volume = V[1]

V=VEL nb1 : the variable nb1 is set to the value of the current MidiIn volume
(number of tic since start)
(the current MidiIn is the last midi event received)
ex. V=VEL 1
means V[1]= Volume

NOTE= value : the MidiOut note is set to value
ex. NOTE= 60
(Note 0= C1, note 12=C2 ...)
NOTE=V nb1 : the MidiOut note is set to the value of the variable nb1
ex. NOTE=V 1
(Note 0=C1, note 12=C2 ...)
means Note = V[1]
V=NOTE nb1 : the variable nb1 is set to the value of the current MidiIn note
ex. V=NOTE 1
(the current MidiIn is the last midi event received)
(Note 0=C1, note 12=C2 ...)
means V[1] = Note

6.2 +=

V+= nb1 value : Value is added to the variable nb1
ex. V+= 10 1.5
means V[10] = V[10] + 1.5
V+=V nb1 nb2 : the value of the variable nb2 is added to the variable nb1
ex. V+=V 1 2
means V[1] = V[1] + V[2]
V+=VV nb1 nb2 : the value of the variable V[nb2] is added to the variable nb1
ex. V+=VV 1 2
means V[1] =V[1] + V[V[2]]
if V[2] is set to 5, it means V[1] =V[1]+V[5]
VV+=V nb1 nb2 : the value of the variable nb2 is added to the variable V[nb1]
ex. VV+=V 1 2
means V[V[1]] = V[nb1] + V[2]
if V[1] is set to 5, it means V[5] = V[5]+V[2]
VV+=VV nb1 nb2 : the value of the variable V[V[nb2]] is added to the variable V[V[nb1]]
ex. VV+=VV 1 2
means V[V[1]] = V[V[1]] + V[V[2]]
if V[1] is set to 5 and V[2] to 6,
it means V[5] = V[5] + V[6]
TIME+= value : Value is added to MidiOut time
(value is a number of tic)
ex. TIME+= 480
means time = time + 480
(Note: one Beat = 480 tics)
TIME+=V nb1 : the value of the variable nb1 is added to the MidiOut time
ex. TIME+=V 1
means
time = time + V[1]
V+=TIME nb1 : the current time is added to the variable nb1
(number of tic since start)
(the current MidiIn is the last midi event received)
ex. V+=TIME 1
means V[1] = V[1]+time
CHAN+= value : Value if added to the MidiOut channel
ex. CHAN+= 1
means Channel = Channel + 1

CHAN+=V nb1 : The value of the variable nb1 is added to the MidiOut channel

ex. CHAN+=V 1
means
Channel = Channel + V[1]

V+=CHAN nb1 : the current MidiIn channel is added to the variable nb1
(the current MidiIn is the last midi event received)
ex. V+=CHAN 1
means V[1] = V[1] + Channel

VEL+= value : Value is added to the MidiOut volume/velocity
ex. VEL+= 100
means volume = volume + 100
(Note: volume value : 0 to 127)

VEL+=V nb1 : The value of the variable nb1 is added to the MidiOut volume
ex. VEL+=V 1
means Volume = Volume + V[1]

V+=VEL nb1 : The current MidiIn volume is added to the variable nb1
(the current MidiIn is the last midi event
received)
ex. V+=VEL 1
means V[1] = V[1] + Volume

NOTE+= value : Value is added to the MidiOut note value
ex. NOTE+= 12
means Note = Note + 12
(Note 0=C1, note 12=C2 ...)
NOTE+=V nb1 : The value of the variable nb1 is added to the MidiOut note
ex. NOTE+=V 1
means Note = Note + V[1]
V+=NOTE nb1 : The current MidiIn note is added to the variable nb1
(the current MidiIn is the last midi event received)
ex. V+=NOTE 1
means V[1] = V[1] + Note

6.3 -=

V-= nb1 value : Value is subtracted to the variable nb1
ex. V-= 10 1.5
means V[10] = V[10] - 1.5
V-=V nb1 nb2 : the value of the variable nb2 is subtracted to the variable nb1
ex. V-=V 1 2
means V[1] = V[1] - V[2]
V-=VV nb1 nb2 : the value of the variable V[nb2] is subtracted to the variable nb1
ex. V-=VV 1 2
means V[1] =V[1] - V[V[2]]
if V[2] is set to 5, it means V[1] =V[1]-V[5]
VV-=V nb1 nb2 : the value of the variable nb2 is subtracted to the variable V[nb1]
ex. VV-=V 1 2
means V[V[1]] = V[nb1] - V[2]
if V[1] is set to 5, it means V[5] = V[5]-V[2]
VV-=VV nb1 nb2 : the value of the variable V[V[nb2]] is subtracted to the variable V[V[nb1]]
ex. VV-=VV 1 2
means V[V[1]] = V[V[1]] - V[V[2]]
if V[1] is set to 5 and V[2] to 6,
it means V[5] = V[5] - V[6]
TIME-= value : Value is subtracted to MidiOut time
(value is a number of tic)
ex. TIME-= 480
means time = time - 480
(Note: one Beat = 480 tics)
TIME-=V nb1 : the value of the variable nb1 is subtracted to the MidiOut time
ex. TIME-=V 1
means
time = time - V[1]
V-=TIME nb1 : the current time is subtracted to the variable nb1
(number of tic since start)
(the current MidiIn is the last midi event received)
ex. V-=TIME 1
means V[1] = V[1]-time
CHAN-= value : Value if subtracted to the MidiOut channel
ex. CHAN-= 1

means Channel = Channel - 1

CHAN-=V nb1 : The value of the variable nb1 is subtracted to the MidiOut channel

ex. CHAN-=V 1

means

Channel = Channel - V[1]

V-=CHAN nb1 : the current MidiIn channel is subtracted to the variable nb1

(the current MidiIn is the last midi event received)

ex. V-=CHAN 1

means V[1] = V[1] - Channel

VEL-= value : Value is subtracted to the MidiOut volume /velocity

ex. VEL-= 100

means volume = volume - 100

(Note: volume value : 0 to 127)

VEL-=V nb1 : The value of the variable nb1 is subtracted to the MidiOut volume

ex. VEL-=V 1

means Volume = Volume - V[1]

V-=VEL nb1 : The current MidiIn volume is subtracted to the variable nb1

(the current MidiIn is the last midi event

received)

ex. V-=VEL 1

means V[1] = V[1] - Volume

NOTE-= value : Value is subtracted to the MidiOut note value
ex. NOTE-= 12
means Note = Note - 12
(Note 0=C1, note 12=C2 ...)
NOTE-=V nb1 : The value of the variable nb1 is subtracted to the MidiOut note
ex. NOTE-=V 1
means Note = Note - V[1]
V-=NOTE nb1 : The current MidiIn note is subtracted to the variable nb1
(the current MidiIn is the last midi event received)
ex. V-=NOTE 1
means V[1] = V[1] - Note

6.4 *=

V*= nb1 value : the variable nb1 is multiplied by value
ex. V*= 10 2.4
means V[10] = V[10] * 2.4
V*=V nb1 nb2 : the variable nb1 is multiplied by the value of the variable nb2
ex. V*=V 1 2
means V[1] = V[1] * V[2]
V*=VV nb1 nb2 : the variable nb1 is multiplied by the value of the variable V[nb2]
ex. V*=VV 1 2
means V[1] = V[1] * V[V[2]]
if V[2] is multiplied by 5,
it means V[1] = V[1] * V[5]
VV*=V nb1 nb2 : the variable V[nb1] is multiplied by the value of the variable nb2
ex. VV*=V 1 2
means V[V[1]] = V[V[1]] * V[2]
if V[1] is multiplied by 5
, it means V[5]= V[5] * V[2]
VV*=VV nb1 nb2 : the variable V[V[nb1]] is multiplied by the value of the variable V[V[nb2]]
ex. VV*=VV 1 2
means V[V[1]] = V[V[1]] * V[V[2]]
if V[1] is multiplied by 5 and V[2] to 6,
it means V[5] = V[5] * V[6]
TIME*= value : the MidiOut time is multiplied by value
(value is a number of tic since start)
ex. TIME*= 2
means Time = Time * 2
TIME*=V nb1 : the MidiOut time is multiplied by the value of the variable nb1
ex. TIME*=V 1
means
time = time * V[1]
V*=TIME nb1 : the variable nb1 is multiplied by the value of the current MidiIn time
(number of tic since start)
(the current MidiIn is the last midi event received)
ex. V*=TIME 1
means V[1] = V[1]* time
CHAN*= value : the MidiOut channel is multiplied by value

ex. CHAN*= 2
CHAN*= 2 means midi channel = channel * 2

CHAN*=V nb1 : the MidiOut channel is multiplied by the value of the variable nb1

ex. CHAN*=V 1
means
Channel = Channel * V[1]

V*=CHAN nb1 : the variable nb1 is multiplied by the value of the current MidiIn channel

(the current MidiIn is the last midi event
received)
ex. V*=CHAN 1
means V[1] = V[1] * Channel

VEL*= value : the MidiOut volume/velocity is multiplied by value

ex. VEL*= 1.5
(Note: volume value : 0 to 127)
means Vel = Vel * 1.5

VEL*=V nb1 : the MidiOut volume is multiplied by the value of the variable nb1

ex. VEL*=V 1
means
Volume = Volume * V[1]

V*=VEL nb1 : the variable nb1 is multiplied by the value of the current MidiIn volume

(number of tic since start)
(the current MidiIn is the last midi event received)
ex. V*=VEL 1
means V[1] = V[1] * Volume

NOTE*= value : the MidiOut note is multiplied by value

ex. NOTE*= 2

means Note = Note * 2

(Note 0=C1, note 12=C2 ...)

NOTE*=V nb1 : the MidiOut note is multiplied by the value of the variable nb1

ex. NOTE*=V 1

(Note 0=C1, note 12=C2 ...)

means Note = Note * V[1]

V*=NOTE nb1 : the variable nb1 is multiplied by the value of the current MidiIn note

ex. V*=NOTE 1

(the current MidiIn is the last midi event received)

(Note 0=C1, note 12=C2 ...)

means V[1] = V[1] * Note

6.5 /=

V/= nb1 value : the variable nb1 is divided by value
ex. V/= 10 2.4
means V[10] = V[10] / 2.4
V/=V nb1 nb2 : the variable nb1 is divided by the value of the variable nb2
ex. V/=V 1 2
means V[1] = V[1] / V[2]
V/=VV nb1 nb2 : the variable nb1 is divided by the value of the variable V[nb2]
ex. V/=VV 1 2
means V[1] = V[1] / V[V[2]]
if V[2] is divided by 5,
it means V[1] = V[1] / V[5]
VV/=V nb1 nb2 : the variable V[nb1] is divided by the value of the variable nb2
ex. VV/=V 1 2
means V[V[1]] = V[V[1]] / V[2]
if V[1] is divided by 5
, it means V[5]= V[5] / V[2]
VV/=VV nb1 nb2 : the variable V[V[nb1]] is divided by the value of the variable V[V[nb2]]
ex. VV/=VV 1 2
means V[V[1]] = V[V[1]] / V[V[2]]
if V[1] is divided by 5 and V[2] to 6,
it means V[5] = V[5] / V[6]
TIME/= value : the MidiOut time is divided by value
(value is a number of tic since start)
ex. TIME/= 2
means Time = Time / 2
TIME/=V nb1 : the MidiOut time is divided by the value of the variable nb1
ex. TIME/=V 1
means
time = time / V[1]
V/=TIME nb1 : the variable nb1 is divided by the value of the current MidiIn time
(number of tic since start)
(the current MidiIn is the last midi event received)
ex. V/=TIME 1
means V[1] = V[1]/ time
CHAN/= value : the MidiOut channel is divided by value

ex. CHAN/= 2
CHAN/= 2 means midi channel = channel / 2

CHAN/=V nb1 : the MidiOut channel is divided by the value of the variable nb1
ex. CHAN/=V 1
means
Channel = Channel / V[1]

V/=CHAN nb1 : the variable nb1 is divided by the value of the current MidiIn channel
(the current MidiIn is the last midi event received)
ex. V/=CHAN 1
means V[1] = V[1] / Channel

VEL/= value : the MidiOut volume/velocity is divided by value
ex. VEL/= 1.5
(Note: volume value : 0 to 127)
means Vel = Vel / 1.5

VEL/=V nb1 : the MidiOut volume is divided by the value of the variable nb1
ex. VEL/=V 1
means
Volume = Volume / V[1]

V/=VEL nb1 : the variable nb1 is divided by the value of the current MidiIn volume
(number of tic since start)
(the current MidiIn is the last midi event received)
ex. V/=VEL 1
means V[1] = V[1] / Volume

NOTE/= value : the MidiOut note is divided by value
ex. NOTE/= 2
means Note = Note / 2
(Note 0=C1, note 12=C2 ...)
NOTE/=V nb1 : the MidiOut note is divided by the value of the variable nb1
ex. NOTE/=V 1
(Note 0=C1, note 12=C2 ...)
means Note = Note / V[1]
V/=NOTE nb1 : the variable nb1 is divided by the value of the current MidiIn note
ex. V/=NOTE 1
(the current MidiIn is the last midi event received)
(Note 0=C1, note 12=C2 ...)
means V[1] = V[1] / Note

6.6 ==

6.6.1 How to do a test

== : the equal test
!= : the unequal test
> : the upper test
< : the lower test
>= : the upper or equal test
<= : the lower or equal test

work the same way :

The next instruction, just after the test will be done only if the test did not failed.

Otherwise, the run will ignore this instruction.

ex.

```
V= 1 5        V[1] is set to 5
V= 2 4        V[2] is set to 4

V== 1 3       if V[1] equal to 3 do
V= 2 3        set V[2] to 3

V= 3 4        set V[3] to 4
```

Because V[1] is not equal to 3, the instruction V= 2 3 just after the test have been ignored.

But the instruction V= 3 4 is done whatever the result of the test (only the instruction just after the test is touched).

V== nb1 value : the variable nb1 is tested to value
ex. V== 10 1
means : is V[10] equal to 1 ?
V==V nb1 nb2 : the variable nb1 is tested to the value of the variable nb2
ex. V==V 1 2
means : is V[1] equal to V[2] ?
V==VV nb1 nb2 : the variable nb1 is tested to the value of the variable V[nb2]
ex. V==VV 1 2
means : is V[1] equal to V[V[2]] ?
if V[2] is set to 5,
it means : is V[1] equal to V[5] ?
VV==V nb1 nb2 : the variable V[nb1] is tested to the value of the variable nb2
ex. VV==V 1 2
means : is V[V[1]] equal to V[2] ?
if V[1] is set to 5,
it means : is V[5] equal to V[2] ?
VV==VV nb1 nb2 : the variable V[V[nb1]] is tested to the value of the variable V[V[nb2]]
ex. VV==VV 1 2
means : is V[V[1]] equal to V[V[2]] ?
if V[1] is set to 5 and V[2] to 6,
it means : is V[5] equal to V[6] ?
TIME== value : the MidiOut time is tested to value
(value is a number of tic since start)
ex. TIME== 480
(Note: one Beat == 480 tics)
it means : is time equal to 480 ?
TIME==V nb1 : the MidiOut time is tested to the value of the variable nb1
ex. TIME==V 1
it means : is time equal to V[1] ?
V==TIME nb1 : the variable nb1 is tested to the value of the current MidiIn time
(number of tic since start)
(the current MidiIn is the last midi event
received)
ex. V==TIME 1
it means : is V[1] equal to time ?
CHAN== value : the MidiOut channel is tested to value
ex. CHAN== 2
CHAN== 2 means
if the Midi Channel equal to 2 ?

CHAN==V nb1 : the MidiOut channel is tested to the value of the variable nb1

ex. CHAN==V 1

it means : is Channel equal to V[1] ?

V==CHAN nb1 : the variable nb1 is tested to the value of the current MidiIn channel

(the current MidiIn is the last midi event received)

ex. V==CHAN 1

it means : is V[1] equal to Channel ?

VEL== value : the MidiOut volume/velocity is tested to value

ex. VEL== 100

(Note: volume value : 0 to 127)

it means : is Vel equal to 100 ?

VEL==V nb1 : the MidiOut volume is tested to the value of the variable nb1

ex. VEL==V 1

it means : is Volume equal to V[1] ?

V==VEL nb1 : the variable nb1 is tested to the value of the current MidiIn volume

(number of tic since start)

(the current MidiIn is the last midi event received)

ex. V==VEL 1

it means : is V[1] equal to Volume ?

NOTE== value : the MidiOut note is tested to value
ex. NOTE== 60
(Note 0=C1, note 12=C2 ...)
it means : is Note equal to 60 ?
NOTE==V nb1 : the MidiOut note is tested to the value of the variable nb1
ex. NOTE==V 1
(Note 0=C1, note 12=C2 ...)
it means : is Note equal to V[1] ?
V==NOTE nb1 : the variable nb1 is tested to the value of the current MidiIn note
ex. V==NOTE 1
(the current MidiIn is the last midi event received)
(Note 0=C1, note 12=C2 ...)
it means : is V[1] equal to Note ?

6.7 !=

V!= nb1 value : the variable nb1 is tested to value
ex. V!= 10 1
means : is V[10] unequal to 1 ?
V!=V nb1 nb2 : the variable nb1 is tested to the value of the variable nb2
ex. V!=V 1 2
means : is V[1] unequal to V[2] ?
V!=VV nb1 nb2 : the variable nb1 is tested to the value of the variable V[nb2]
ex. V!=VV 1 2
means : is V[1] unequal to V[V[2]] ?
if V[2] is set to 5,
it means : is V[1] unequal to V[5] ?
VV!=V nb1 nb2 : the variable V[nb1] is tested to the value of the variable nb2
ex. VV!=V 1 2
means : is V[V[1]] unequal to V[2] ?
if V[1] is set to 5,
it means : is V[5] unequal to V[2] ?
VV!=VV nb1 nb2 : the variable V[V[nb1]] is tested to the value of the variable V[V[nb2]]
ex. VV!=VV 1 2
means : is V[V[1]] unequal to V[V[2]] ?
if V[1] is set to 5 and V[2] to 6,
it means : is V[5] unequal to V[6] ?
TIME!= value : the MidiOut time is tested to value
(value is a number of tic since start)
ex. TIME!= 480
(Note: one Beat != 480 tics)
it means : is time unequal to 480 ?
TIME!=V nb1 : the MidiOut time is tested to the value of the variable nb1
ex. TIME!=V 1
it means : is time unequal to V[1] ?
V!=TIME nb1 : the variable nb1 is tested to the value of the current MidiIn time
(number of tic since start)
(the current MidiIn is the last midi event received)
ex. V!=TIME 1
it means : is V[1] unequal to time ?
CHAN!= value : the MidiOut channel is tested to value

ex. CHAN!= 2
CHAN!= 2 means
is midi channel unequal to 2 ?

CHAN!=V nb1 : the MidiOut channel is tested to the value of the variable nb1
ex. CHAN!=V 1
it means : is Channel unequal to V[1] ?

V!=CHAN nb1 : the variable nb1 is tested to the value of the current MidiIn channel
(the current MidiIn is the last midi event received)
ex. V!=CHAN 1
it means : is V[1] unequal to Channel ?

VEL!= value : the MidiOut volume/velocity is tested to value
ex. VEL!= 100
(Note: volume value : 0 to 127)
it means : is Vel unequal to 100 ?

VEL!=V nb1 : the MidiOut volume is tested to the value of the variable nb1
ex. VEL!=V 1
it means : is Volume unequal to V[1] ?

V!=VEL nb1 : the variable nb1 is tested to the value of the current MidiIn volume
(number of tic since start)
(the current MidiIn is the last midi event received)
ex. V!=VEL 1
it means : is V[1] unequal to Volume ?

NOTE!= value : the MidiOut note is tested to value
ex. NOTE!= 60
(Note 0=C1, note 12=C2 ...)
it means : is Note unequal to 60 ?
NOTE!=V nb1 : the MidiOut note is tested to the value of the variable nb1
ex. NOTE!=V 1
(Note 0=C1, note 12=C2 ...)
it means : is Note unequal to V[1] ?
V!=NOTE nb1 : the variable nb1 is tested to the value of the current MidiIn note
ex. V!=NOTE 1
(the current MidiIn is the last midi event received)
(Note 0=C1, note 12=C2 ...)
it means : is V[1] unequal to Note ?

6.8 <

V< nb1 value : the variable nb1 is tested to value
ex. V< 10 1
means : is V[10] lower than 1 ?
V<V nb1 nb2 : the variable nb1 is tested to the value of the variable nb2
ex. V<V 1 2
means : is V[1] lower than V[2] ?
V<VV nb1 nb2 : the variable nb1 is tested to the value of the variable V[nb2]
ex. V<VV 1 2
means : is V[1] lower than V[V[2]] ?
if V[2] is set to 5,
it means : is V[1] lower than V[5] ?
VV<V nb1 nb2 : the variable V[nb1] is tested to the value of the variable nb2
ex. VV<V 1 2
means : is V[V[1]] lower than V[2] ?
if V[1] is set to 5,
it means : is V[5] lower than V[2] ?
VV<VV nb1 nb2 : the variable V[V[nb1]] is tested to the value of the variable V[V[nb2]]
ex. VV<VV 1 2
means : is V[V[1]] lower than V[V[2]] ?
if V[1] is set to 5 and V[2] to 6,
it means : is V[5] lower than V[6] ?
TIME< value : the MidiOut time is tested to value
(value is a number of tic since start)
ex. TIME< 480
(Note: one Beat < 480 tics)
it means : is time lower than 480 ?
TIME<V nb1 : the MidiOut time is tested to the value of the variable nb1
ex. TIME<V 1
it means : is time lower than V[1] ?
V<TIME nb1 : the variable nb1 is tested to the value of the current MidiIn time
(number of tic since start)
(the current MidiIn is the last midi event received)
ex. V<TIME 1
it means : is V[1] lower than time ?
CHAN< value : the MidiOut channel is tested to value

ex. CHAN< 2
CHAN< 2 means
is midi channel lower than 2 ?

CHAN<V nb1 : the MidiOut channel is tested to the value of the variable nb1
ex. CHAN<V 1
it means : is Channel lower than V[1] ?

V<CHAN nb1 : the variable nb1 is tested to the value of the current MidiIn channel
(the current MidiIn is the last midi event received)
ex. V<CHAN 1
it means : is V[1] lower than Channel ?

VEL< value : the MidiOut volume/velocity is tested to value
ex. VEL< 100
(Note: volume value : 0 to 127)
it means : is Vel lower than 100 ?

VEL<V nb1 : the MidiOut volume is tested to the value of the variable nb1
ex. VEL<V 1
it means : is Volume lower than V[1] ?

V<VEL nb1 : the variable nb1 is tested to the value of the current MidiIn volume
(number of tic since start)
(the current MidiIn is the last midi event received)
ex. V<VEL 1
it means : is V[1] lower than Volume ?

NOTE< value : the MidiOut note is tested to value
ex. NOTE< 60
(Note 0=C1, note 12=C2 ...)
it means : is Note lower than 60 ?
NOTE<V nb1 : the MidiOut note is tested to the value of the variable nb1
ex. NOTE<V 1
(Note 0=C1, note 12=C2 ...)
it means : is Note lower than V[1] ?
V<NOTE nb1 : the variable nb1 is tested to the value of the current MidiIn note
ex. V<NOTE 1
(the current MidiIn is the last midi event received)
(Note 0=C1, note 12=C2 ...)
it means : is V[1] lower than Note ?

6.9 >

V> nb1 value : the variable nb1 is tested to value
ex. V> 10 1
means : is V[10] upper than 1 ?
V>V nb1 nb2 : the variable nb1 is tested to the value of the variable nb2
ex. V>V 1 2
means : is V[1] upper than V[2] ?
V>VV nb1 nb2 : the variable nb1 is tested to the value of the variable V[nb2]
ex. V>VV 1 2
means : is V[1] upper than V[V[2]] ?
if V[2] is set to 5,
it means : is V[1] upper than V[5] ?
VV>V nb1 nb2 : the variable V[nb1] is tested to the value of the variable nb2
ex. VV>V 1 2
means : is V[V[1]] upper than V[2] ?
if V[1] is set to 5,
it means : is V[5] upper than V[2] ?
VV>VV nb1 nb2 : the variable V[V[nb1]] is tested to the value of the variable V[V[nb2]]
ex. VV>VV 1 2
means : is V[V[1]] upper than V[V[2]] ?
if V[1] is set to 5 and V[2] to 6,
it means : is V[5] upper than V[6] ?
TIME> value : the MidiOut time is tested to value
(value is a number of tic since start)
ex. TIME> 480
(Note: one Beat > 480 tics)
it means : is time upper than 480 ?
TIME>V nb1 : the MidiOut time is tested to the value of the variable nb1
ex. TIME>V 1
it means : is time upper than V[1] ?
V>TIME nb1 : the variable nb1 is tested to the value of the current MidiIn time
(number of tic since start)
(the current MidiIn is the last midi event received)
ex. V>TIME 1
it means : is V[1] upper than time ?
CHAN> value : the MidiOut channel is tested to value

ex. CHAN> 4
CHAN> 4 means
is midi channel upper than 4 ?

CHAN>V nb1 : the MidiOut channel is tested to the value of the variable nb1
ex. CHAN>V 1
it means : is Channel upper than V[1] ?

V>CHAN nb1 : the variable nb1 is tested to the value of the current MidiIn channel
(the current MidiIn is the last midi event received)
ex. V>CHAN 1
it means : is V[1] upper than Channel ?

VEL> value : the MidiOut volume/velocity is tested to value
ex. VEL> 100
(Note: volume value : 0 to 127)
it means : is Vel upper than 100 ?

VEL>V nb1 : the MidiOut volume is tested to the value of the variable nb1
ex. VEL>V 1
it means : is Volume upper than V[1] ?

V>VEL nb1 : the variable nb1 is tested to the value of the current MidiIn volume
(number of tic since start)
(the current MidiIn is the last midi event received)
ex. V>VEL 1
it means : is V[1] upper than Volume ?

NOTE> value : the MidiOut note is tested to value
ex. NOTE> 60
(Note 0=C1, note 12=C2 ...)
it means : is Note upper than 60 ?
NOTE>V nb1 : the MidiOut note is tested to the value of the variable nb1
ex. NOTE>V 1
(Note 0=C1, note 12=C2 ...)
it means : is Note upper than V[1] ?
V>NOTE nb1 : the variable nb1 is tested to the value of the current MidiIn note
ex. V>NOTE 1
(the current MidiIn is the last midi event received)
(Note 0=C1, note 12=C2 ...)
it means : is V[1] upper than Note ?

6.10 <=

V<= nb1 value : the variable nb1 is tested to value
ex. V<= 10 1
means : is V[10] equal or lower than 1 ?
V<=V nb1 nb2 : the variable nb1 is tested to the value of the variable nb2
ex. V<=V 1 2
means : is V[1] equal or lower than V[2] ?
V<=VV nb1 nb2 : the variable nb1 is tested to the value of the variable V[nb2]
ex. V<=VV 1 2
means : is V[1] equal or lower than V[V[2]] ?
if V[2] is set to 5,
it means : is V[1] equal or lower than V[5] ?
VV<=V nb1 nb2 : the variable V[nb1] is tested to the value of the variable nb2
ex. VV<=V 1 2
means : is V[V[1]] equal or lower than V[2] ?
if V[1] is set to 5,
it means : is V[5] equal or lower than V[2] ?
VV<=VV nb1 nb2 : the variable V[V[nb1]] is tested to the value of the variable V[V[nb2]]
ex. VV<=VV 1 2
means : is V[V[1]] equal or lower than V[V[2]] ?
if V[1] is set to 5 and V[2] to 6,
it means : is V[5] equal or lower than V[6] ?
TIME<= value : the MidiOut time is tested to value
(value is a number of tic since start)
ex. TIME<= 480
(Note: one Beat <= 480 tics)
it means : is time equal or lower than 480 ?
TIME<=V nb1 : the MidiOut time is tested to the value of the variable nb1
ex. TIME<=V 1
it means : is time equal or lower than V[1] ?
V<=TIME nb1 : the variable nb1 is tested to the value of the current MidiIn time
(number of tic since start)
(the current MidiIn is the last midi event received)
ex. V<=TIME 1
it means : is V[1] equal or lower than time ?
CHAN<= value : the MidiOut channel is tested to value

ex. CHAN<= 4
CHAN<= 4 means
is midi channel equal or lower than 4 ?

CHAN<=V nb1 : the MidiOut channel is tested to the value of the variable nb1

ex. CHAN<=V 1
it means : is Channel equal or lower than V[1] ?

V<=CHAN nb1 : the variable nb1 is tested to the value of the current MidiIn channel

(the current MidiIn is the last midi event received)
ex. V<=CHAN 1
it means : is V[1] equal or lower than Channel ?

VEL<= value : the MidiOut volume/velocity is tested to value

ex. VEL<= 100
(Note: volume value : 0 to 127)
it means : is Vel equal or lower than 100 ?

VEL<=V nb1 : the MidiOut volume is tested to the value of the variable nb1

ex. VEL<=V 1
it means : is Volume equal or lower than V[1] ?

V<=VEL nb1 : the variable nb1 is tested to the value of the current MidiIn volume

(number of tic since start)
(the current MidiIn is the last midi event received)
ex. V<=VEL 1
it means : is V[1] equal or lower than Volume ?

NOTE<= value : the MidiOut note is tested to value
ex. NOTE<= 60
(Note 0=C1, note 12=C2 ...)
it means : is Note equal or lower than 60 ?
NOTE<=V nb1 : the MidiOut note is tested to the value of the variable nb1
ex. NOTE<=V 1
(Note 0=C1, note 12=C2 ...)
it means : is Note equal or lower than V[1] ?
V<=NOTE nb1 : the variable nb1 is tested to the value of the current MidiIn note
ex. V<=NOTE 1
(the current MidiIn is the last midi event received)
(Note 0=C1, note 12=C2 ...)
it means : is V[1] equal or lower than Note ?

6.11 >=

V>= nb1 value : the variable nb1 is tested to value
ex. V>= 10 1
means : is V[10] equal or upper than 1 ?
V>=V nb1 nb2 : the variable nb1 is tested to the value of the variable nb2
ex. V>=V 1 2
means : is V[1] equal or upper than V[2] ?
V>=VV nb1 nb2 : the variable nb1 is tested to the value of the variable V[nb2]
ex. V>=VV 1 2
means : is V[1] equal or upper than V[V[2]] ?
if V[2] is set to 5,
it means : is V[1] equal or upper than V[5] ?
VV>=V nb1 nb2 : the variable V[nb1] is tested to the value of the variable nb2
ex. VV>=V 1 2
means : is V[V[1]] equal or upper than V[2] ?
if V[1] is set to 5,
it means : is V[5] equal or upper than V[2] ?
VV>=VV nb1 nb2 : the variable V[V[nb1]] is tested to the value of the variable V[V[nb2]]
ex. VV>=VV 1 2
means : is V[V[1]] equal or upper than V[V[2]] ?
if V[1] is set to 5 and V[2] to 6,
it means : is V[5] equal or upper than V[6] ?
TIME>= value : the MidiOut time is tested to value
(value is a number of tic since start)
ex. TIME>= 480
(Note: one Beat >= 480 tics)
it means : is time equal or upper than 480 ?
TIME>=V nb1 : the MidiOut time is tested to the value of the variable nb1
ex. TIME>=V 1
it means : is time equal or upper than V[1] ?
V>=TIME nb1 : the variable nb1 is tested to the value of the current MidiIn time
(number of tic since start)
(the current MidiIn is the last midi event received)
ex. V>=TIME 1
it means : is V[1] equal or upper than time ?
CHAN>= value : the MidiOut channel is tested to value

ex. CHAN>= 4
CHAN>= 4 means
is midi channel equal or upper than 4 ?

CHAN>=V nb1 : the MidiOut channel is tested to the value of the variable nb1

ex. CHAN>=V 1
it means : is Channel equal or upper than V[1] ?

V>=CHAN nb1 : the variable nb1 is tested to the value of the current MidiIn channel

(the current MidiIn is the last midi event received)
ex. V>=CHAN 1
it means : is V[1] equal or upper than Channel ?

VEL>= value : the MidiOut volume/velocity is tested to value

ex. VEL>= 100
(Note: volume value : 0 to 127)
it means : is Vel equal or upper than 100 ?

VEL>=V nb1 : the MidiOut volume is tested to the value of the variable nb1

ex. VEL>=V 1
it means : is Volume equal or upper than V[1] ?

V>=VEL nb1 : the variable nb1 is tested to the value of the current MidiIn volume

(number of tic since start)
(the current MidiIn is the last midi event received)
ex. V>=VEL 1
it means : is V[1] equal or upper than Volume ?

NOTE>= value : the MidiOut note is tested to value

ex. NOTE>= 60

(Note 0=C1, note 12=C2 ...)

it means : is Note equal or upper than 60 ?

NOTE>=V nb1 : the MidiOut note is tested to the value of the variable nb1

ex. NOTE>=V 1

(Note 0=C1, note 12=C2 ...)

it means : is Note equal or upper than V[1] ?

V>=NOTE nb1 : the variable nb1 is tested to the value of the current MidiIn note

ex. V>=NOTE 1

(the current MidiIn is the last midi event received)

(Note 0=C1, note 12=C2 ...)

it means : is V[1] equal or upper than Note ?

7. MAPPING

7.1 Description

An useful feature has been added to the version 1.3 of MidiLang : a midi mapping.

Midi Mapping is used to control in detail all Midi Events generated by your Midi instrument, and to modify them the way you want.

For example, the breath control used by all wind instruments (such as the digital sax WX11) is very difficult to find in most of the expanders of the market. This control can be replaced by the volume control which is more likely to be find in any expander. Therefore the mapping will replace any breath control event by a volume change event which can be sent to the expander.

The pitch bender action can be modified too (it is just a controller as the volume or expression controller).
For example, you can link it to the expression controller.

The effect of the controller can be modified too :

- you can change its curve
- you can inverse its action

7.2 How to use the Midi Mapper

To define a midi mapping, you have to create a function in your mpl file.
This function name is : MAPPER

```
LABEL MAPPER

END
```

This function will be executed every time a controller event is received.
In the MAIN function or the BEATS function, the only Midi Event received is the note Event :
A channel number, a Note number and a velocity.
Here, in the MAPPER function, the Events used are all the other Events (Control change, Program change, Pitch Bender..)

Note that even if you can't received any control event in the MAIN or BEATS function, you can generate them inside these functions

There is one main difference between the MAIN or BEATS functions and the MAPPER function :

in the MAIN/BEATS functions, all received Notes are saved in memory (as those generated by MidiLang).

in the MAPPER functions only the Events generated by MidiLang are saved.

The saved Notes and Events can then be saved to disk (.mid files)

7.3 List of Midi Events

This section describes MIDI events as documented by the MIDI 1.0 specification.

More information may be obtained from:

The International MIDI Association
5316 W. 57th St.
Los Angeles, CA 90056
(310) 649-6434

Midi Messages

There are two types of MIDI messages:

Channel Messages
System Messages

Channel Messages

Channel Messages communicate performance information.
These messages are assigned to one of 16 channels.

Channel messages are as coded in the table below.
'n' stand for the channel number, with 0-15 corresponding to the channels 1-16

Status (hexa)	Data1	Data2	Message Type
0x8n	Note #	n/a	Note Off
0x9n	Note #	Velocity	Note On
0xAn	Note #	Pressure	Polyphonic Key Pressure
0xBn	Control #	Value	Control Change
0xCn	Program #	n/a	Program Change
0xDn	Pressure	n/a	Channel Pressure
0xEn	Pitch LSB	Pitch MSB	Pitch Bend

Note : The note off function is represented by either the 0x8n message (Note Off) or by the 0x9n message (Note on) with zero velocity.

Controller Type

Control change message apply to a given controller, as given below, The value associated to the controller varies from 0 to 127.

Controller type	Number (hexa)
Modulation Wheel	0x01
Breath Controller	0x02
Foot Controller	0x04
Portamento time	0x05
Data Entry MSB	0x06
Main Volume	0x07
Balance	0x08
Pan	0x0A
Expression Controller	0x0B
General Purpose	0x10-0x13,0x50-0x53
LSB for value 0-31	0x20-0x3F
Sustain Pedal	0x40
Portamento	0x41
Sostenuto	0x42
Soft Pedal	0x43
Hold 2	0x45
External Effects Depth	0x5B
Tremelo Depth	0x5C
Chorus Depth	0x5D
Detune Depth	0x5E
Phaser Depth	0x5F
Data Increment	0x60
Data Decrement	0x61
Nonregistered Parameter LSB	0x62
Nonregistered Parameter MSB	0x63
Registered Parameter LSB	0x64
Registered Parameter MSB	0x65
Channel Mode Messages	0x79-0x7F

Note : Check if your Midi Keyboard support a controller before sending a Control Change.

For example, not all synthetisers support breath control change.

If your Midi intrument receives a control change for a unknown controller, It will simply ignore this message.

System Messages

System Messages apply to all devices on the Midi Network.

Status (hexa)	Data1	Data2	Message Type
0xF1	Value	n/a	Midi Time Code
0xF2	LSB	MSB	Song Position Pointer
0xF3	Song #	n/a	Song Select
0xF6	n/a	n/a	Tune Request
0xF8	n/a	n/a	Timing Clock
0xFA	n/a	n/a	Start
0xFB	n/a	n/a	Continue
0xFC	n/a	n/a	Stop
0xFE	n/a	n/a	Active Sensing
0xFF	n/a	n/a	System Reset

7.4 Example of Midi Mapper

A digital sax is generating notes and control events in channel 4.
We want all the notes to be sent to channel 5 without modifications
We want to replace the Expression change and the Modulation change of the channel 4 with the volume change in the channel 5.
No modification of the control curve is done now.

```
LABEL MAPPER   Mapper ( if exist ) is executed at each control,program..  event
               received
STATUS!= 0XB3     if the event a control event in the channel 4 ?
GOTO ENDM

STATUS= 0XB4
DATA1== 1
GOTO GOTIT

DATA1== 11
GOTO GOTIT

GOTO ENDM


LABEL GOTIT

STATUS= 0XB4      switch to channel 5 ( control change status )
DATA1= 7       and to volume control
OUTMIDI       and send it

LABEL ENDM
END



LABEL MAIN     Main is executed at each note received

CHAN!= 4      is the note in the channel 4
GOTO END      if not, forget it
CHAN= 5      if yes change it to channel 5
OUTMIDI          and regenerated it

LABEL END
END
```

Same example but with inverse effect

```
LABEL MAPPER   Mapper ( if exist ) is executed at each control,program..
                        event received
STATUS!= 0XB3     if the event a control event in the channel 4 ?
GOTO ENDM

STATUS= 0XB4
DATA1== 1
GOTO GOTIT

DATA1== 11
GOTO GOTIT

GOTO ENDM


LABEL GOTIT

STATUS= 0XB4      switch to channel 5
DATA1= 7       and to volume control

V=DATA2 1      V[1]=DATA2
V= 2 127
V-=V 2 1       V[2]=127-V[1]
DATA2=V 2      DATA2=V[2]

OUTMIDI       and send it

LABEL ENDM
END


LABEL MAIN     Main is executed at each note received

CHAN!= 4      is the note in the channel 4
GOTO END      if not, forget it
CHAN= 5      if yes change it to channel 5
OUTMIDI          and regenerated it

LABEL END
END
```

8. Questions / Answers

- What kind of files are saved by MidiLang ? Are they "real" Midi files ?
Midilang saves Midi events into .mid files. These files are Standard MIDI Files (SMF). This file interchange format is similar to the RIFF file formats used in Windows and is related to the Electronic Arts' IFF format. Most commercial sequencer programs can read and write SMFs.

An SMF is one of three formats. Format 0 is the simplest format, with all sequence data contained in a single track. All MIDI application should be able to read and write format 0 files. Format 1 files can have any number of tracks. Certain data, such as tempo changes, can appear only in the first track, while MIDI events can be written to any number of tracks. Format 2 file consist of multiple independent multitrack sequences or patterns.
Because of the aim of MidiLang, the files are saved as format 0 files, but format 1 or 2 can be read.

In conclusion, the answer of the question is : The saved files are Standard Midi files, and Yes they are "real" Midi files.

Why short notes are badly played by MidiLang, How can I solve it ?.
MidiLang process speed is linked with the internal tempo of your Midi Keyboard, the higher your tempo is, the more accurate MidiLang will be.

Please note that we are speaking about the internal tempo of your Midi Keyboard not the one used by MidiLang (found in the setting menu).
A good value is 120 (the accuracy of MidiLang is then less than 20 milisec.)
Short notes should then be played without any troubles by MidiLang.

Why a Midi Programming Language ? most of the users care only about already made effects files.
Well, at first perhaps, but because each user has is own view of useful effects, and because soon you will need a set of specialized and personnalized effects, you will be happy to be able to create your own effect with this language.

You are typically not English-Native, are you ?
No, that's true, and I know that my english can be strange and difficult to decrypt sometime. I am sorry about that and I am working hard to improve this document and make it easy to read, but as all language user manuals, you must practice and try to create your own MPL files.

Feel free to ask me any questions regarding any part of this document or MPL files. (100417.2633@compuserve.com)

9. RUNNING AN EFFECT ON MIDI FILES

With the version 1.31 of MidiLang, you can run any type of effect on MIDI files (.mid files).

This feature can be used on any PC, even without a MIDI link to a keyboard.

All types of Midi files can be used , those saved by MidiLang or any standard one. The only limitation is that MidiLang will read only the track one of the given MIDI file.

9.1 Example of adding echo.mpl effect on a MIDI file

First you must load the effect :
Menu FILE / Open Effect / file : ECHO.MPL

Then load your MIDI file :
Menu FILE / Open Midi

At this time you can listen to the original Midi File :
Menu Play / Start

And/Or run the effect on it :
Menu Play / Run Effect on loaded notes

Wait until the end of the process

MidiLang will display the number of notes loaded and generated

Listen to the result :
Menu Play / Start

Save the result to a Standard Midi File (.mid) :
Menu File / Save Midi

Note that even if you are using an effect in live (Record / Start) , you can save your record and then use the saved file with another effect.

INDEX
