Main Help Menu

- MidiLang Introduction
- How to do
- Glossary
- MidiLang Menu
- Tutorial

How to do menu

- How to run an effect
- How to save a .mid file
- How to create your own MPL file
- How to register
- How to get contact

Glossary

Hexa

Hexadecimal value can be used (useful with event number) anywhere in the MPL files.

ex .

V= 1 16
is equivalent to
V= 1 0x10

Hexa integer must start with 0x or 0X

Savein / Unsavein

By default, all incoming Midi notes events are saved in memory. At the end of your play, you can play them back or save them into .mid files.

If you don't want these incoming notes to be saved, use the command : unsavein.
To go back to the default status, use the command : savein.

Status

Status is equivalent to Chan in the MPL files.
Status can be used in place of Chan anywhere in your MPL files.

Note that the Midi note event number is :

0X8n (with n = midi channel (0-15)) : Note On Event
0X9n : Note Off Event

But for simplification you can also use directly the channel number : 1-16

CHAN= 0X82
is equivalent to
STATUS= 0x82
or CHAN= 3
or STATUS= 3

See List of Midi Events for the complete list

Data1

Data1 is equivalent to Note in the MPL files.
Data1 can be used in place of Note anywhere in your MPL files.

It is the Midi Events first data

See List of Midi Events for the complete list

Data2

Data2 is equivalent to Vel in the MPL files.
Data2 can be used in place of Vel anywhere in your MPL files.

It is the Midi Event second data

See List of Midi Events for the complete list

Data3

Data3 is equivalent to Data in the MPL files.
Data3 can be used in place of Data anywhere in your MPL files.

It is the Midi Events third data

See List of Midi Events for the complete list

mapping

Description

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

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. Therefor 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

How to use the Midi Mapper
List of Midi Events
Examples ot Midi Mapper

How to use Midi Mapper

To define a midi mapper, 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)

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

Examples ot 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 breath 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== 2
GOTO GOTIT

DATA1== 11
GOTO GOTIT

GOTO ENDM


LABEL GOTIT

STATUS= 0XB4        switch to channel 5
DATA1= 2          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== 2
GOTO GOTIT

DATA1== 11
GOTO GOTIT

GOTO ENDM


LABEL GOTIT

STATUS= 0XB4        switch to channel 5
DATA1= 2          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
```

Main Menu

the Main menu item of Midilang are :

FILE	: load,save .mid files, load effect mpl file
SETTINGS	: set tempo and cmd channel
Run/Record	: record your play and run the effect
Stop/Play_back	: play back a loaded .mid file or your play
INPUT DEVICE:	change the input midi device
OUTPUT DEVICE	: change the output midi device

Settings Menu

The Settings Menu give access to

Tempo
Cmd Channel
Input & Output Filters

How to load and run an effect

To run an effect with MidiLang, you must :

Set the Midi Input & Output Device properly
Load a MPL file
Start it

and play on your midi keyboard.

Input Output Device

MidiLang will automatically search for the all Midi devices available in your PC.

Often, more than one device are available, but few of them are really linked to your Midi keyboard.

You must try, each of them, until MidiLang works properly.

P.S. If you have installed a Sound Blaster, MidiLang will automatically switch to its device.

To change the Input/Output device, use the Input and Output device menu, your changes will be saved.

Input Device

If you open the menu item : INPUT DEVICE

The list of available Midi Device will appear.

In this list, only one device can be used.

By default, the first one is taken or the one used by your Sound Blaster card (if you have one)

Just click on the device you want to use.

Output Device

If you open the menu item : OUTPUT DEVICE

The list of available Midi Device will appear.

In this list, only one device can be used.

By default, the first one is taken or the one used by your Sound Blaster card (if you have one)

Just click on the device you want to use.

How to Load an effect

An effect is defined into a text file, with MPL (Midi Programming Language) instructions.

This file must be loaded into MidiLang to be used.

In the menu item FILE ,open the sub-menu item LOAD EFFECT, a window will appear where you can choose the MPL file you want to use. (try for example echo.mpl)

Midilang will load this file, check its syntax, compile it in memory.

You can create your own MPL files or used already done one. The non-registered version of MidiLang limits the MPL instruction number of your files to 10. but the registered version limits it to 1000.

File Menu

If you open this menu Item, 5 sub menu items will appear :

NEW MIDI FILE : to start a new play
LOAD MIDI FILE : to load a saved .mid file
SAVE MIDI FILE : to save your play into a .mid file

LOAD EFFECT : to load a MPL file.

EXIT : to end MidiLang.

Example of Echo Effect

This mpl file will create an almost real echo effect

```
label main      Start of the Mpl
v== 1 5                 V[1] is set to 5
goto end
V+= 1 1
time+= 240
v=v 4 1
v/= 4 5
v+= 4 1.5

vel/=V 4
outmidi

goto main
label end
v= 1 0
end

descript an almost real echo
descript with decrease volume
```

Register

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 MPL, 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 appear 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.

Two more tools : MidiLoop and MplDebug

MidiLoop : a mini-MidiLang specialized in echos, very easy to use.
MplDebug : a simple debugger for MPL files. Very useful to trace mpl behaviour.

Registration fee :

35 US$ (+ Handling, Postage)

You get with it the up-to-date registered version of MidiLang.

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

Please note that you can also register via CompuServe :
GO SWREG
Select the : "register a shareware" item
The registration ID of MidiLang is 7059

You will then receive your registered version of MidiLang directly via email.

MPL files and programmation

MidiLang uses Effect Definition Files (with .mpl extension). These files explain what set of effects to do, and at what time.

These ".mpl" files are small text files, easy to share, to gather and to create.

To create your own effects, you need to explain to MidiLang what to do ,when ,and how..

For example, you can ask MidiLang to replay your play 0.5 beat later, and to add basses based on the harmony of your play.

The language used by MidiLang is called MPL (Midi Programming Language), an easy to used and fast effect definition language. Any one can use this language and get complex effects easily.

Depending on which version of MidiLang (Shareware or Register(register)), you can use up to 10 or 1000 mpl instructions in your file.

New Midi File

If you want to erase your play and to restart the record, use the NEW FILE item in the FILE menu

load_file
load,file,midi

MidiLang can load .mid files and play it.

You can even run an effect on this loaded file :
use the menu item : *Run an effect on loaded notes* of the *PLAY* Menu item.

Save Midi File

At the end of your play, after the Stop, you can ask MidiLang to save your play (with the effect) into a .mid file.

MidiLang will ask for a filename and save your play.

Note : the internal buffer of Midilang is only 16000 notes big.

Load Midi File

At any time, you can ask MidiLang to load a .mid file and be ready to play it.
MidiLang will ask for a filename and load it.
Note : the internal buffer of Midilang is only 16000 notes big.

Midi File and Format

The standard way to save music on PC is using the .mid format. All files with the extension .mid should be created with this format and can be used with any Midi Tool as sequencer.

For example, the file created by MidiLang can be used directly by CakeWalk.

Start a record and an Effect

To run an effect and be able to use it during your play, you must load it and run it.
To do so, inside the RECORD menu, select the start item.
You will notice that when the effect is running, the beat value is displayed in real time on the screen.

Stop a record and an Effect

At the end of your play, you must stop the recording and the effect run.
To do so you have to select the stop item in the RECORD.
The beat will stop.
You are ready to save your play and to play it back.

Record Menu

In this item, you will find the START item that starts the recording and the effect run, and the STOP item that stop the recording and the effect run.

Play Menu

After a record or after loading a .mid file, you can play it back
To do so, select the PLAY menu item. You can START the play back or STOP it.

Start a Playback

The PlayBack of your play (or of your loaded .mid file) is started by selecting the submenu item START from the PLAY menu item.

Run an effect on loaded notes

You can run any type of effect on a standard Midi File.
Load your effect, load you Midi file and run the effect on it.

Stop a Playback

The PlayBack of your play (or of your loaded .mid file) is stopped by selecting the submenu item STOP from the PLAY menu item.

How to save a .mid file

A .mid file can be saved from your play.

To do so, record a play.

If you want an effect, load it first.

At the end of your play, do not forget to stop the record

And save your play

How to create your own MPL file

MPL program are easy to create because there only a small set of MPL instructions.

A MPL file is a text file (use any text editor (as the NotePad) to create them.)

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.

Those commands can be : change to an another channel, calculate harmony based on the notes you have played, play some bass with your melody, add delay, echo. to your play...

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

There are 4 specialized LABELs in MPL :

LABEL MAIN
to be executed at each note (on or off) received

LABEL BEATS
to be executed at each beat

LABEL MAPPER
to be executed at each control event

LABEL INIT
to be executed once just at the RUN.

NOTE : MidiLand is case-insensitive

ex .

```
# start of the main group

LABEL MAIN
TIME+= 240
OUTMIDI
END
```

end of the main group

This example does an echo of every thing you play.

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.)

It is why you Must have a group MAIN in every MPL file.

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
```

The maximum number of MPL instructions in a MPL file is 10 for the non-registered version of MidiLang and 1000 for the registered version.

For more details about how to create MPL file, see the documentation included.
or look at those examples :

how to create a delay
how to create a split
how to create an echo
how to play on different channel
how to use harmony
how to get chord
how to get basses

Main

The Label Main must exist in any MPL file.
MidiLang will execute your MPL from this entry at each note played.

If you omit this Label, MidiLang will stop the loading of your effect file.

Beats

The Label Beats is an optionnal label in your MPL files.
MidiLang will execute your MPL from this entry (if exist) at each beat.

If you omit this Label, MidiLang will ignore the beat event.

Init

The Label Init is an optionnal label in your MPL files.
MidiLang will execute your MPL from this entry (if exist) at the RUN of your effect. (and initialize for example a set of variables).

REGISTRATION

Registration will give you update version of MidiLang, 1000 mpl instructions per file and possibilities of subscription.

MidiLoop : a simple way to get loop and echos.
MplDebug : a simple debugger for MPL files.

To Register, please use the form included in the package : order.txt

How to get a contact

If you need information or help, mail me your question :

100417,2633
INTERNET:100417.2633@COMPUSERVE.COM
WEB PAGE : http://ourworld.compuserve.com/homepages/sib

or call or write to

Pik a Program

see order.txt for phone number and address

How to create delay effect

This is one of the most simple effect to do !
Just ask MidiLang to play back your play a little bit later :

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

ALL MidiEvent will be sent back 0.5 beat later.
Save this file with .mpl extension, load it into MidiLang,
run it and play.

How to create your own split

To create a soft split :

```
LABEL MAIN          start
NOTE>= 60   if (note number > 60 (C6) upper )
GOTO UPPER          goto upper
NOTE>= 48   else if (note number > 48 (C5) lower)
GOTO LOWER          goto lower

CHAN= 145   else use channel 2 ( 143+2 )
OUTMIDI             and send it to the midi keyboard
GOTO END    and goto to the end

LABEL LOWER         lower part
CHAN= 146   use channel 3 (143+3)
OUTMIDI
GOTO END

LABEL UPPER
CHAN= 147   use channel 4
OUTMIDI

LABEL END
END                 end of the mpl
```

For each MidiIn, the note played will be send back at the same time, but on channel depending on the note number.

You should turn off your Keyboard MidiOut Volume.

Label command

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 : MidiLand is case-insensitive

ex .

```
# start of the main group

LABEL MAIN
TIME+= 240
OUTMIDI
END

# end of the main group
```

This example does an echo of every thing you play.

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.)

It is why you Must have a group MAIN in every MPL file.

You can used up to 100 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
```

goto, gosub commands

Any where 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 resume to the next command (CHAN-= 1 here)

NOTE 1 : Gosub and Midi Value

NOTE 2 : OUTMIDI before a gosub

NOTE 3 : gosub in a gosub

NOTE 1: Gosub and Midi Value

Please note that any changes done to the Midi Values (time, note, channel,volume) 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 run 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..

Outmidi and Midi Values

Every time you play something on your midi keyboard, a MidiIn is received by MidiLang, and the Label(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 you have defined at the time you have defined.

The Midi Values are :

TIME : number of tics from start (480 tics = 1 beat)
VEL : volume (0=OFF, 127=FULL)
CHAN : Channel (Channel = Midi Channel +143)
NOTE : Note Number (C1=0, C2=12; D1=2...)

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 lexicon)

Savein and Unsavein commands change the way incoming notes are saved.

Rescreen command

Tools available with the version 1.31 of MidiLang :

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

Midi Mapper : a new Label for control mapping (such as breath control..)

Initialization label : a new label to be used to initialize at the run of the effect

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#

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 (based on my own musical theory)
(if you want more data about this theory mail me your questions ..)

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
GETHARM nb1 10 nb3

V[10] is set to 2, Getharm will calculate the note number 2

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= 20 5 V[20] id set to 5
GETHARM 1 20 10 will calculate the note nb 5 (because V[20]=5)
of the chord saved in V[1],V[2]
and save it in V[10]

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

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

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)

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.
```

Remarks

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
```

Link between Midi Keyboard and Mpl File : keydef

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 use 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 play 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 variable 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 by 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.

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 or integer).
You can use hexadecimal integers
Where you can save whatever you want.

The Basic command keyword, you are looking for, contains :

-> =
-> +=
-> -=
-> *=
-> /=
-> ==
-> !=
-> <
-> >
-> <=
-> >=

Command =

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= 147
(Note: Channel = midi chan + 143)
CHAN= 147 means midi channel = 4 (143+4)

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
(Note: Channel = midi chan + 143)

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

Command +=

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]] = VV[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
(Note: Channel = midi chan + 143)

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
(Note: Channel = midi chan + 143)

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

Command -=

V-= nb1 value : Value is substracted 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 substracted 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 substracted 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 substracted to the variable V[nb1]
ex. VV-=V 1 2
means V[V[1]] = VV[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 substracted 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 substracted 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 substracted to the MidiOut time
ex. TIME-=V 1
means
time = time - V[1]

V-=TIME nb1 : the current time is substracted 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 substracted to the MidiOut channel
ex. CHAN-= 1
means Channel = Channel - 1
(Note: Channel = midi chan - 143)

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

V-=CHAN nb1 : the current MidiIn channel is substracted to the variable nb1
(the current MidiIn is the last midi event received)
ex. V-=CHAN 1
means V[1] = V[1] - Channel
(Note: Channel = midi chan - 143)

VEL-= value : Value is substracted 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 substracted to the MidiOut volume
ex. VEL-=V 1
means Volume = Volume - V[1]

V-=VEL nb1 : The current MidiIn volume is substracted 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 substracted 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 substracted to the MidiOut note
ex. NOTE-=V 1
means Note = Note - V[1]

V-=NOTE nb1 : The current MidiIn note is substracted to the variable nb1
(the current MidiIn is the last midi event received)
ex. V-=NOTE 1
means V[1] = V[1] - Note

Command *=

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*= 147
(Note: Channel = midi chan + 143)
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

Command /=

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/= 147
(Note: Channel = midi chan + 143)
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

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).

Command ==

that's a test

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== 147
(Note: Channel = midi chan + 143)
CHAN== 147 means
is midi channel equal to 4 ? (143+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 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 ?
(Note: Channel = midi chan + 143)

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 ?

Command !=

that's a test

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!= 147
(Note: Channel = midi chan + 143)
CHAN!= 147 means
is midi channel unequal to 4 ? (143+4)

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 ?
(Note: Channel = midi chan + 143)

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 ?

Command <

that's a test

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< 147
(Note: Channel = midi chan + 143)
CHAN< 147 means
is midi channel lower than 4 ? (143+4)

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 ?
(Note: Channel = midi chan + 143)

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 ?

Command >

that's a test

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> 147
(Note: Channel = midi chan + 143)
CHAN> 147 means
is midi channel upper than 4 ? (143+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 ?
(Note: Channel = midi chan + 143)

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 ?

Command <=

that's a test

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<= 147
(Note: Channel = midi chan + 143)
CHAN<= 147 means
is midi channel equal or lower than 4 ? (143+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 ?
(Note: Channel = midi chan + 143)

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 ?

Command >=

that's a test

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>= 147
(Note: Channel = midi chan + 143)
CHAN>= 147 means
is midi channel equal or upper than 4 ? (143+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 ?
(Note: Channel = midi chan + 143)

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 ?

Tutorial

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

My first MPL file
My second MPL file
My third MPL file
Let's Loop MPL FILE

My first MPL file

With the Note-Pad tools of Windows, type :

For MidiLang, Label = LABEL = label, then use Uppercase where you want !

```
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 the note played to MidiLang.
MidiLang will run your mpl file from the label MAIN
The first instruction is (after the label)
    time+= 240
that 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)

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.

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.

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
at the start of MidiLang, all variable 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 (the blank lines are always 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

note is the third (and last) data of the MidiEvent
(the data of the MidiEvent are :
Time
Vel
Note
Chan
)

this instruction means :
add to the note number the value of the variable nb 1

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 ask for 0 volume

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.

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 increase at each loop
the time too
the note too
until the test failed (Variable upper than 10)

This mpl will output the 10 next notes of everything you play.

Introduction to MidiLang

MidiLang has been created to :

- allow effects such as echo, delay, loop in live (i.e. during your play) and on Midi Files (.mid files)
- calculate best chords for a given melody in live
- be able to add automatically bass based on your play in live.
- do any kind of Midi mapping on any type of controller
- 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 keyb. 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 file. But MPL is a so easy to use language that you should try to define your own effect.

How to set the Midi Command Channel Number

In the Menu : Settings select the item cmd Channel.

You can change the Midi channel used to get command from your Midi Keyboard. (Default is 3)

See example : echos.mlp

How to change input / output filters

In the Menu : Settings select the item input filters or output filters.

The list of the available type of MidiEvent will appear, those flaged won't be sent or received.

How to set the tempo

In the Menu : Settings selects the item set Tempo.

You can change the internal tempo (i.e. number of beats per minutes) used by MidiLang (no link with the tempo of your Midi Keyboard).

(Default is 120 beats/min.)

Exit Menu

In the file menu the exit item will stop MidiLang and purge all MidiEvent ready to be sent.

What is a beat , a tic

MidiLang calculates time in number of tics.

There is 480 tics in one beat.

Then if you want a time one half beat later, you need to add 0.5*480 = 240 tic to the current time.

MidiLang will run the Label beats (if exist in the current mpl file) at each beat.

See example

How to get chord, bass or harmony effects

This mpl shows how to get harmony data about your play, and how to use it.

```
label beats             This label will be run at each beat
# to be run at each beat
v+= 10 1

v< 10 9
goto suite
v= 10 1
beat= 0

v== 9 0
goto end_beat

oldharm 1               Decrease the weight of the old notes
calcharm 1              reset the tonality to the current note
label suite

v== 9 0
goto end_beat

getharm 1 10 5          get the note nb v(10) and put it in v(5)

chan= 145               change to channel 2 ( 143+2 ) ( basses )
vel= 100
note= 5
note+= 24                       add 24 to the note number and play it now
outmidi
time+= 479
vel= 0                  and turn it off 1 beat later
outmidi
label end_beat
end


label main
calcharm 1
v= 9 1
end

descript Automatic basse
descript based on your melody
descript
```

descript reset of the harmony every 8 beats

How to use Midi Command Channel

This mpl example shows you how to use your Midi Keyboard to send command to MidiLang directly.

A Midi Command Channel must be used for it (By default 3). The commands will be run after playing in this channel two defined keys (for example, C and C# will run the label run_command...)

The definition of those command and the link with the Midi keyb is done with the Keydef command.

```
label faster            this label will be executed only by the keydef
v*= 1 1.41214
end

label slower
v/= 1 1.41214
end

label softer
v*= 3 1.141214
end

label harder
v/= 3 1.141214
end

label chan-
v-= 2 1
end

label chan+
v+= 2 1
end

label main
v=v 4 1
V*= 4 480
time+=V 4
vel/= V 3
v= 4 2
v+= 4 143
chan=v 4
outmidi
end
```

descript simple echo
descript with any delay, and out channel
descript all parameters chosen and modified
descript directly from the midi keyboard

keydef c c# faster faster delay 1 1 this defines a link between the keys C and C# (in the cmd channel) and the label faster This label will be executed every time those keys will be played.

keydef c b slower slower

keydef d d# softer softer volume_divide 3 1.5
keydef d C# harder harder

keydef F E chan- chan- out_channel 2 2
keydef F F# chan+ chan+

What About MidiLang

MidiLang (TM)

1.31

Compuserve 100417,2633

Web Page : http://ourworld.compuserve.com/homepages/sib

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. Note that only the first track is used by MidiLang.

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.