

MIDI INPUT CONTROL in CSOUND

It is possible to use MIDI input instead of the standard score to run csound instruments. Instruments will use midi input opcodes to take MIDI data off a device or a MIDI file.

MIDI CONTROLLER INPUT

If you have a MIDI input device you can connect MIDI controllers. Type something like:

```
csound -odac -d -m6 -M[dev_num] instr.csd
```

and the rest is up to you.

The `-M` flag indicate that you should accept input from a MIDI input port.

VIRTUAL MIDI DEVICE

Csound 5.04 also comes with a virtual MIDI device that can be used if you don't have a keyboard at hand. Here are the flags for it:

```
-M0 --rtmidi=virtual
```

The `--rtmidi=virtual` tells csound to use the virtual MIDI module; `-M0` is the only device available in it.

MIDI FILES

Use the `-F` flag on the command line to have Csound read a MIDI file:

```
csound -o sound.wav -W -FPrelude1.mid Prelude1.csd
```

This command line will generate a normal soundfile.

To have Csound read a MIDI file and send its output to the soundcard outputs:

```
csound -o dac -FPrelude1.mid Prelude1.orc Prelude1.sco
```

The file `Prelude1.csd` defines an instrument (called a 'program' in MIDI), and stored function tables it needs.

LISTENING FOR MIDI INPUT

Because with MIDI we do not usually use score events, we need to tell Csound to stay on, listening for MIDI events. You can set a time limit (for example 10 minutes) for listening on the midi input channels. **This is done using the `f 0` statement.**

```
f0 600 ; stay on for 600 seconds (10 minutes, 10 x 60 seconds)
```

The flag -F holds the midifile name. This flag will take either standard MIDI files or MPU401 files -- Csound will accept either format.

Use Control-C to stop the command-line Csound after a performance. This will however disrupt any files being written by Csound at the time of termination.

NOTE-OFF EVENTS

MIDI events have no determinate turnoff time. A Csound instrument started from a MIDI event will therefore be set to a special mode. This is indicated by a **negative p3** value, which is used to indicate that the instrument will continue performance indeterminately (a **held note**). So with MIDI, **DO NOT USE P3 AS THE TOTAL DURATION**. A MIDI noteoff event will terminate an instrument event which was triggered by MIDI. See *i* statements.

In Csound 3.47 a new feature was introduced for envelope signal generators: when the Csound instrument receives a MIDI note off event the instrument release section is used, extending the performance by a specified time, during which the value of the opcode changes to a selected value.

LINSEGR, EXPSEGR & LINENR

| | | |
|----|----------------|--|
| kr | linsegr | ia, idur1, ib[,idur2, ic[.]], irel, iz |
| ar | linsegr | ia, idur1, ib[,idur2, ic[.]], irel, iz |
| kr | expsegr | ia, idur1, ib[,idur2, ic[.]], irel, iz |
| ar | expsegr | ia, idur1, ib[,idur2, ic[.]], irel, iz |

linsegr and **expsegr** work like **linseg/expseg**, except that on a MIDI note off event the release section is used, extending the performance by *irel* seconds, during which the value of the opcode changes to *iz*.

| | | |
|----|---------------|---------------------------|
| kr | linenr | kamp, irise, idec, iatdec |
| ar | linenr | xamp, irise, idec, iatdec |

linenr works like **linen**, except that there is no fixed duration between the *irise* and *idec* periods. The latter will be reached after a NOTEOFF message is received.

The opcode **linenr** contains a note-off sensor and release time extender. When it senses either a score event termination or a MIDI noteoff, it will immediately extend the performance time of the current instrument by *idec* seconds, then execute an exponential decay towards the factor *iatdec*. For two or more units in an instrument, extension is by the greatest *idec*.

Note that **linenr**, **linsegr** and **expsegr** only work with events started by MIDI input. These are examples of the special Csound "r" units that contain a note-off sensor and release time extender. When each senses a score event termination or a MIDI noteoff, it will immediately extend the performance time of the current instrument by *idec* seconds, then begin a decay from wherever it was at the time.

CHANNEL ASSIGNMENTS

Instruments are assigned by channel no. when available (instr 1 to chnl 1, instr 2 to chnl 2, etc.), but if a requested instrument is absent then instr 1 is substituted. If you have a MIDI keyboard connected to your computer, you can invoke other instruments by sending MIDI program changes (1, 2, 3). You can also use *massign* to connect a MIDI channel with an instrument:

massign ichnl, insnum

This is a **header statement** (i.e. should be placed at the orchestra header) which assigns a MIDI channel *ichnl* to Csound instrument number *insnum*.

MIDI CONVERTERS

ival **notnum**
ival **veloc** [imin, imax]

notnum - get the MIDI byte value (0 - 127) denoting the note number of the current event.

veloc - get the MIDI byte value (0 - 127) denoting the velocity of the current event, scaled to the range imin, imax.

icps **cpsmidi**
kcps **cpsmidib** [irange]
ioct **octmidi**
koct **octmidib** [irange]
ipch **pchmidi**
kpch **pchmidib** [irange]

cpsmidi, **octmidi**, **pchmidi** - get the note number of the current MIDI event, expressed in Herz (cycles per second), oct, or pch units for local processing.

cpsmidib, **octmidib**, **pchmidib** - get the note number of the current MIDI event, modify it by the current pitch-bend value, and express the result in Herz, oct, or pch units. Available as an I-time value or as a continuous **ksig** value. The parameter *irange* is the pitch bend range in semitones.

ival **cpstmid** ifn

cpstmid is similar to cpsmidi above, but allows fully customized micro-tuning scales. It requires five or more parameters for the tuning ratios. For convenience, these should be stored in a function table ifn, which is sent as the only argument to the opcode. This function table ifn should be generated by the GEN02. The first four values stored in the function are: numgrades (the number of grades of the micro-tuning scale), interval (the frequency range covered before repeating the grade ratios, for example 2 for one octave, 1.5 for a fifth etcetera), basefreq (the base frequency of the scale in Herz), basekeymidi (the midi-note-number to which to assign the basefreq unmodified). After these four values, the user can begin to insert the tuning ratios.

Examples: for a standard 12-grade scale with the base-frequency of 261 Hz assigned to the key-number 60, the corresponding f-statement in the score to generate the table should be:

```
f1 0 64 -2 12 2 261 60 1 1.059463 1.122462 1.18920 ..etc...
```

Another example with a 24-grade scale with a base frequency of 440 assigned to the key-number 48, and a repetition interval of 1.5:

```
f1 0 64 -2 24 1.5 440 48 1 1.01 1.02 1.03 ..etc...
```

`iamp` **ampmidi** `iscal` [,ifn]

ampmidi - get the velocity of the current MIDI event, optionally pass it through a normalized translation table, and return an amplitude value in the range 0 – `iscal`. The optional parameter *ifn* is function table number of a normalized translation table, by which the incoming value is first interpreted. The default value is 0, denoting no translation. This translation table will have size 128, with values for velocities 0-127.

`kaft` **aftouch** `[imin [,imax]]`

`xbend` **pchbend** `[imin, imax]`

`kval` **midictrl** `inum [,imin [,imax]]`

aftouch, chpress, pchbend - get the current after-touch, channel pressure, or pitch-bend value for this channel, and maps it to the specified range. Note that this access to pitch-bend data is independent of the MIDI pitch, enabling the value here to be used for any arbitrary purpose. The initial value of `aftouch` defaults to 127. The parameters *imin*, *imax* map midi values to this range. The default values are 0 and 127, respectively .

midictrl - get the current value (0 - 127) of a specified MIDI controller. The parameter *inum* is the MIDI controller number.

There are a number of other specialised MIDI opcodes. Please consult the `csound` manual to get more information on them.

A COMPLETE EXAMPLE

The orchestra + score pair shown below use MIDI note on messages to trigger the instrument, control its amplitude and initial frequency. The pitch is also controlled via pitchbend wheel. The filter frequency is set by the modulation wheel, whose values (between 0 and 127) are converted to frequency on an equal-interval (logarithmic) basis, between *imin* and *imax*, according to the following relationship:

$$\text{freq} = \text{imin} + (\text{imax} - \text{imin}) \times \left(2^{\frac{\text{midi_value}}{127}} - 1 \right)$$

The code includes the `powoftwo` opcode, which does exactly what it says: two to the power of whatever is under the parenthesis () :

```
kmod = ifmin + (ifmax - ifmin)*(powoftwo(kmod/127) - 1)
```

Above the value of `kmod` on the right is the value out of the modulation wheel (0-127). The whole expression assigns a new value for `kmod` in Hz (between `ifmin` and `ifmax`). Notice that the score does not have any note statements (i-statements), because those are going to be triggered by the MIDI notes.

<CsInstruments>

```

; MIDI subtractive synthesizer
; Victor Lazzarini, 1998
;
; takes the oscillator frequency from
; MIDI note + pitch bend messages,
; amplitude from MIDI velocity and
; filter frequency from modulation wheel

    sr = 44100
    kr = 441
    ksmps = 100
    nchnls = 1

    instr 1

iq = 50          ; filter Q
ifmin = 400      ; min filter freq
ifmax = 4000     ; max filter freq

iamp ampmidi 0dbfs ; amplitude (scaled from 0 - 0dbfs, which is set
;                          to 32767 by default)
kcps cpsmidib 2   ; cps + pitch bend (range = 2 semitones)
kmod midictrl 1   ; modulation control

; scale MIDI modulation values (between ifmin and ifmax)
; using a logarithmic (equal-interval) scale
kmod = ifmin + (ifmax - ifmin)*(powoftwo(kmod/127) - 1)

kmp  linear  iamp, .05, .5, .01 ; envelope
a1  oscili  kmp, kcps, 1        ; oscillator
aout reson  a1, kmod, kmod/iq, 1 ; filter

    out      aout

    endin
</CsInstruments>

```

<csScore>

```

; midisynth.sco
; function table initialisation

; oscillator wave
f1 0 1024 10 1 .8 .7 .6 .5 .4 .3 .2 .1 .08 .07 .06 .05 .04 .03 .02

; listen to MIDI for 600 secs (10 minutes)
f0 600

e
</CsScore>

```