Joachim Heintz
Live Electronics with Modern Csound (using CsoundQt)

Workshop at Seoul International Computer Music Festival 2018
28 - 30 September 2018

II. Interaction with GUI, MIDI, ASCII Keys and OSC

GUI (Graphical User Interface = Widgets Panel)

12 Show In/Out Signals in CsoundQt Widgets (outvalue)

A very simple way, only displaying raw amplitudes (create both, a controller widget and a display widget with channel name “input_amp”).

```
aMic inch 1
kShowTrig metro 10
kPeak max_k aMic, kShowTrig, 1
outvalue "input_amp", kPeak
```

Better to display decibels rather than raw amplitudes. For the following code, create another pair of controller plus display:

- **channel name** = “input_db”
- **set the range** from **-60** (minimum) to **0** (maximum)

```
outvalue "input_db", dbamp(kPeak)
```

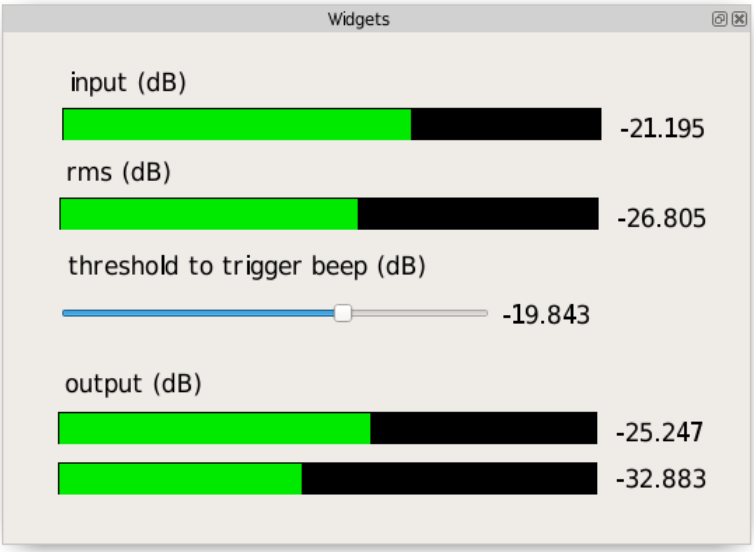
See (download and try) https://github.com/csudo/csudo/blob/master/csqt/CsQtMeter.csd for even nicer view ...

13 Set a flexible threshold with a widget (invalue)

Going back to example 09 but introducing an adjustable threshold in dB.

```
kThreshDb invalue "threshdb"
kThresh = ampdb(kThreshDb)
```

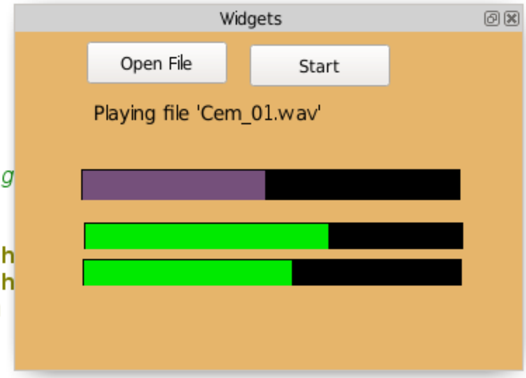
```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

/* use widgets to control csound
    (modification of 09) */

instr Analyze_and_trigger

 ;get input channel 1
 aMic inch 1

 ;get rms
 kRms rms aMic

 ;show peak and rms
 gkTrig metro 10
 kPeak max_k aMic, gkTrig, 1
 outvalue "input_db", dbamp(kPeak)
 outvalue "input_rms", dbamp(kRms)

 ;get value for threshold from slider
 kThreshDb invalue "threshdb"
 kThresh = ampdb(kThreshDb)

 ;look for thresh crossing and trigger beep
 kPause init 0 ;sec
 kPreviousRms init 0
 if kRms > kThresh && kPreviousRms <= kThresh && kPause <= 0 then
```



14 Browse a file name via GUI (chnget/chnset)

The *invalue/outvalue* opcodes are easier to use, but *chnget/chnset* are faster and able to work with strings. Only that the software channels must be **declared** once, at the header of the Csound program.

```
;first declare channels (1=input 2=output)
chn_S "_Browse", 1 ;reserved channel for browsing
chn_S "info", 2
chn_k "progress", 2
chn_k "out_L", 2
chn_k "out_L_over", 2
chn_k "out_R", 2
chn_k "out_R_over", 2

;user defined opcode to g
  opcode StrFiln, S, S
Spath       xin
ipos       strrindex Spath
Snam       strsub    Spath
           xout      Snam
  endop

;user defined opcode for signal display
opcode CsQtMeter, 0, SSak
```



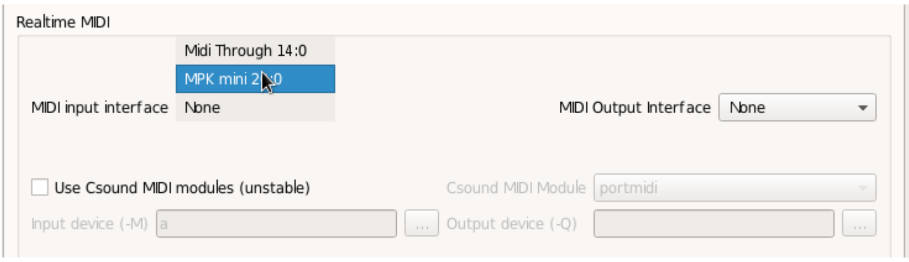
Playing file 'Cem_01.wav'

The button “Open File” has the reserved channel “_Browse” (see Examples > Widgets > Reserved Channels for more information). Once it is clicked, it opens a file browse dialog.
Sending strings to the GUI is done here via the channel “info” and a Display widget.

MIDI

15 MIDI Keyboard (note on/off messages)

First **select your device** in CsoundQt’s Configure > Run



Realtime MIDI

MIDI input interface: None (Midi Through 14:0, MPK mini 2:0)

MIDI Output Interface: None

☐ Use Csound MIDI modules (unstable)

Csound MIDI Module: portmidi

Input device (-M): a

Output device (-Q):

Klick “Apply” and then “Ok”. Then hack this simple instrument:

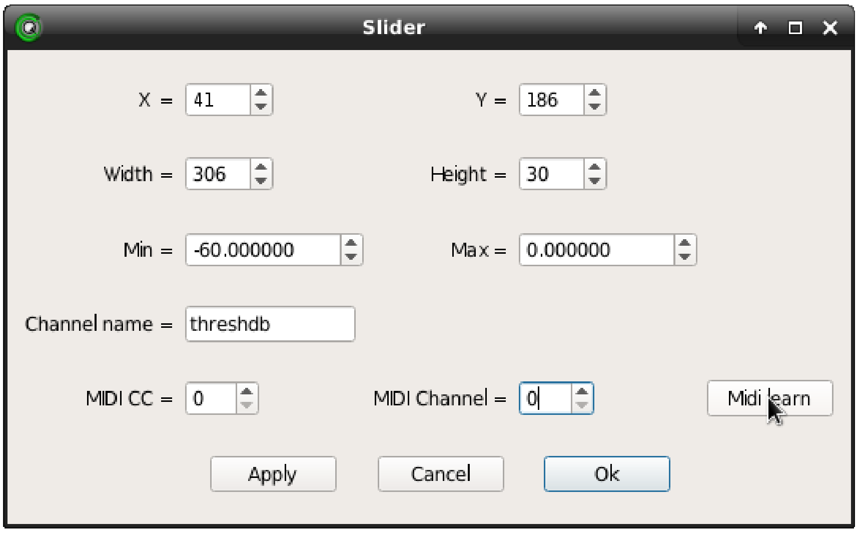
```
instr OneMIDINote
 aNote poscil .2, 400
 out aNote, aNote
endin
```

Pushing any key of the MIDI keyboard triggers this instrument. If the key is released, the instance of this instrument will terminate. The rest of the code only applies reading of note number, velocity and fades.

```
 iCps cpsmidi
 iVel veloc
 aNote poscil ampdb(-64+iVel/2), iCps
 aEnv linenr aNote, 1/20, 1/5, 1/100
 out aEnv, aEnv
```

16 MIDI Control Change

In CsoundQt, the best way is to connect a MIDI controller directly to a widget. Right-click on a widget (for example the slider in 13), open the Properties, click on “Midi learn”, move the controller, and MIDI channel plus CC are recognized and connected to the widget.



Slider

X = 41

Y = 186

Width = 306

Height = 30

Min = -60.000000

Max = 0.000000

Channel name = threshdb

MIDI CC = 0

MIDI Channel = 0

ASCII Keys

17 Trigger events by hitting any key

General functionality (create a scroll number widget with channel “key”)

```
 gkKey, gkDown sensekey
 if gkDown == 1 then
  outvalue "key", gkKey
 endif
```

Now it is easy to trigger events by a certain key (here ASCII 32 = space bar)

```
 if gkDown == 1 && gkKey == 32 then
  event "i", "Beep", 0, random:k(0.2,1)
 endif
```

See also Examples > Useful > SF Snippets Player

OSC (Open Sound Control)

18 Receive OSC from Processing (Max, PD, ...)

What must be specified to receive OSC messages from another application:

- the port on which OSC is sent by the other application
- the string containing the OSC address
- the type of values received (i=integer, f=float, s=string etc.)

This is the basic code to receive four variables, each of them a floating point number, from processing at the address “/Proc/xpos”, and show the incoming values in four controller widgets:

```
giPort OSCinit 12002

instr Receive

 kx1, kx2, kx3, kx4 init 0
 kPing OSClisten giPort, "/Proc/xpos", "ffff",  kx1, kx2, kx3, kx4
 outvalue "x1", kx1
 outvalue "x2", kx2
 outvalue "x3", kx3
 outvalue "x4", kx4

endin
```

16_a shows a simple “translation” of the moving lines to sliding pitches.

19 Send OSC to INScore

What must be specified to send OSC messages to another application:

- the IP address of the receiver, or “localhost” for this computer
- the port to send the OSC messages
- the OSC address
- the types to be sent via OSC

This is a minimal example to send the message “Hello Inscore” from Csound to Inscore (http://inscore.sourceforge.net and https://github.com/grame-cncm/inscore):

```
giPort = 7000
instr Hello
OSCsend  1,"", giPort, "/ITL/scene/text", "sss", "set", "txt", "Hello Inscore!"
endin
```

17_a generates a random structure in Csound and displays shapes accordingly in Inscore, continuously communicating via OSC.

20 Exercises

- Implement in 12 a controller which holds for two seconds a red light whenever the signal is more than 0 dB.
- Change the code in 13 so that higher thresholds result in higher pitches.
- Apply a volume slider in 14.
- Change the fade-out time for a note in 15 so that a random value between 1 and 2 seconds is applied for each note.
- Implement different pitches for different ASCII keys in 17.
- Change 18_a so that every crossing of a line to one of the edges results in a pitch change.