Joachim Heintz
Live Electronics with Modern Csound (using CsoundQt)

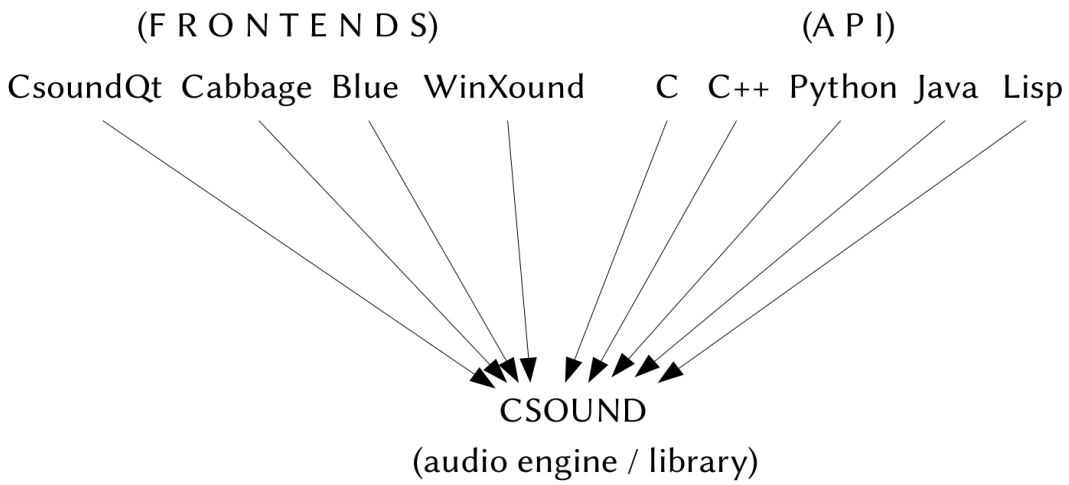
Workshop at Seoul International Computer Music Festival 2018
28 - 30 September 2018

I. Introduction to Modern Csound

01 Download and Help

- Download **Csound** (**CsoundQt** is **included** in the installers for Mac and Windows)
  https://csound.com/download.html
- **Latest CsoundQt** version with korean translation
  https://github.com/CsoundQt/CsoundQt/releases
- Free Introduction online (Csound **FLOSS Manual**)
  http://write.flossmanuals.net/csound
- CsoundQt Website and Documentation
  http://csoundqt.github.io
  http://csoundqt.github.io/pages/documentation.html
- *Getting Startet* is integrated in CsoundQt
  Examples > Getting Started
- New Csound Book (2016)
  Lazzarini et.al. — Csound, A Sound and Music Computing Language
  Springer Site Table of Contents

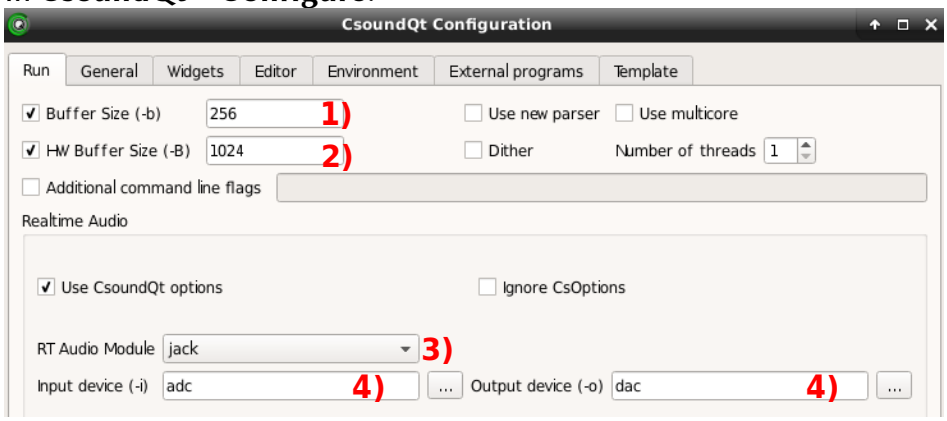
02 Csound and Frontends



Distinction between both layers in the case of CsoundQt
CsoundQt -> Editor, Widgets, Environment
Csound -> Syntax, Reference Manual, Audio Engine

03 Some Basic Settings

In **CsoundQt > Configure:**



1) Set software buffer size to 256 samples
2) Set hardware buffer size to 1024 samples
3) Choose Real Time Audio Module according to you operating system (usually portaudio)
4) Choose **adc** as input device and **dac** as output device. **Select your audio card in the operating system.**

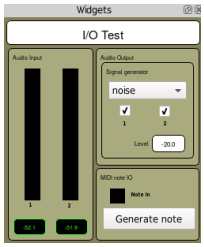
In your **Csound program (.csd file):**

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

sr = 44100 5)
ksmps = 32  6)
nchnls = 2  7)
0dbfs = 1  8)

instr 1
endin

</CsInstruments>
<CsScore>

</CsScore>
</CsoundSynthesizer>
```

5) Set sample rate (usually 44100 or 48000)
6) Set number of samples per block or control-cycle (usually 32)
7) Set number of channels (usually 2)
8) Set the number which represents 0 dB full scale (always 1)

Save this in Configure > Template. It will then be opened for each new file in CsoundQt.

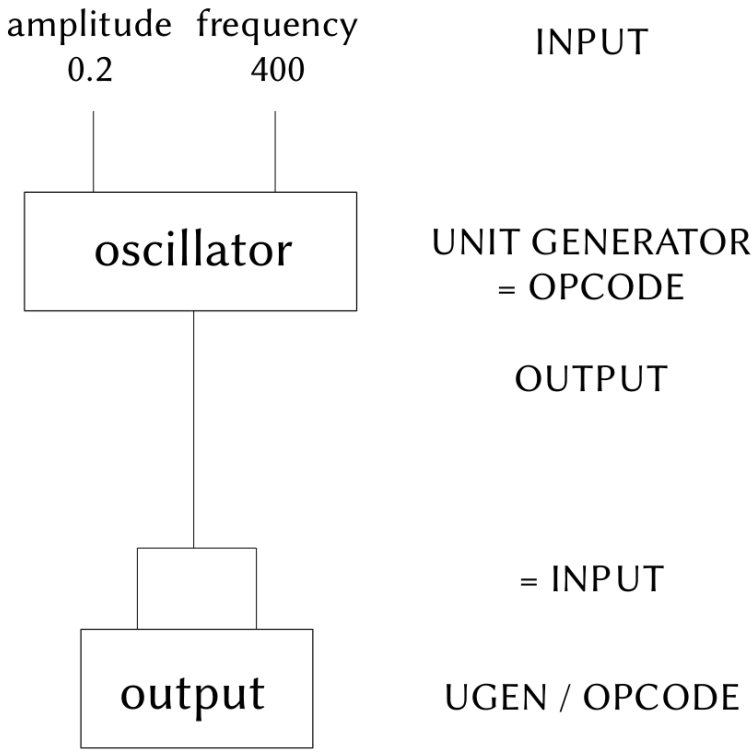
Now run **CsoundQt > Examples > Useful > IO Test**:
1. Klick on the “Run” button top left.
2. Klick on the “Widgets” button. You should see this panel:



3. Klick on the checkboxes to hear noise test signal.

04 Start Coding

A typical flow chart



How to **translate** this signal flow **into Csound code**

- Know the opcode names (here: oscillator = *poscil*, output = *out*)
- For each Input-Opcode-Output chain, write one line in the order
  OUTPUT OPCODE INPUT
- Know and specify the signal types:
  - **i**Bla -> only updated at start of an instrument instance
  - **a**Bla -> updated every sample
  - **k**Bla -> updated every ksmps block

So in this case:

```
instr Sine_Test
 aBla poscil 0.2, 400
 out aBla, aBla
endin
```

Executable code in Csound is embedded in "instruments". Each instrument starts with the keyword **instr** (following a number or a name: here Sine_Test) and ends with the keyword **endin**.

There are several methods to **call an instrument**.
1. Write this line in the *CsScore* section of your .csd file:

```
i "Sine_Test" 0 3
```

2. Click on the "Widgets" button. Right-click in the Widgets panel. Select "Create Button". In the properties dialog which open now, paste the same score as "Event":

```
i "Sine_Test" 0 3
```

Now run Csound **with empty score section** and click on the button to start the instrument.

3. Run the instrument by putting this code under the *endin* line:

```
schedule "Sine_Test", 0, 3
```

You can also write in functional style (see below item 10):

```
schedule("Sine_Test",0,3)
```

05 Live Input

Only specify the channel number for the *inch* opcode to get your mic signal into Csound.
(Be careful with the following example; can lead to feedback.)

```
aBla inch 1
out aBla, aBla
```

06 Modifications

Rather than directly feeding the mic input into the output, we usually want to modify the live input.
As a simple example, ring modulation is applied.

```
aMic inch 1
aSine poscil 1, 400
aRM = aMic * aSine
out aRM, aRM
```

07 Query Live Input

```
aMic inch 1
kRms rms aMic
```

The second line analyzes the energy of the signal via the root mean squared formula.
If you put this line after it, you will see the rms level every 0.1 seconds printed in the console:

```
printk 0.1, kRms
```

This is pure Csound, but if you insert this line instead

```
outvalue "mic_rms", kRms
```

you can create a widget with the channel name *mic_rms* and will see the value displayed.
Possible widgets are here: Knob, Slider, Controller; Display, Spinbox, Scroll Number.

08 More Analyzing

When does the RMS cross a threshhold? Get this point:

```
aMic inch 1
kRms rms aMic
kThresh init 0.1
kPreviousRms init 0
if kRms > kThresh && kPreviousRms <= kThresh then
 printks "Crossed!\n", 0
endif
kPreviousRms = kRms
```

It is good to wait after one detection for a reasonable time (~ 0.1 seconds). How to implement this?
An instrument in Csound is updated every *ksmps/sr* seconds. For instance at sr=44100 and ksmps=32, one control cycle equals 32/44100=0.00726 seconds (less than 1 ms). So we can count the time like this:

```
aMic inch 1
kRms rms aMic
kThresh init 0.1 ;amp
kPause init 0 ;sec
kPreviousRms init 0
if kRms > kThresh && kPreviousRms <= kThresh && kPause <= 0 then
 printks "Crossed!\n", 0
 kPause = 0.1 ;reset pause
endif
kPause -= ksmps/sr ;count backwards
kPreviousRms = kRms
```

09 Trigger another instrument

Inside the if-condition (instead of the printks), call another instrument:

```
event "i", "Beep", 0, 0.1
```

And the *Beep* instrument can be something like:

```
instr Beep
 aBeep poscil 0.2, 500
 aEnv linen aBeep, 0, p3, p3
 out aEnv, aEnv
endin
```

10 If you prefer ...

... you can use **functional style**. Always specify the variable type when necessary.
This would be the functional version of our last instrument:

```
aBeep = poscil:a(0.2, 500)
aEnv = linen:a(aBeep, 0, p3, p3)
out(aEnv, aEnv)
```

Or even:

```
aEnv = linen:a(poscil:a(0.2, 500), 0, p3, p3)
out(aEnv, aEnv)
```

11 Exercises

- Apply *transeg* curve instead of *linen* in 09 (10). Compare a-rate and k-rate version.
- Set the amplitude and frequency input in 04 in the GUI, for instance with a scroll box.
  Use the opcode *invalue* to receive the widget‘s value in your Csound code.
- After having implemented this, use dB input rather than amplitude, and MIDI key input rather than frequencies.
  Use ampdb() to convert decibel in amplitude, and cpsmidinn() to convert MIDI key to frequency (Hertz).
- Any idea what could be used instead of the Beep instrument in 09? Try to write it down.