Routing Open Sound Control messages via vanilla JACK to build low-latency event translator/filter chains and map unconventional controller data to musical events

INTRODUCTION

We have run continuously into situations where it would have been of advantage to have a means to route OSC directly via JACK. There would be multiple benefits:

- Sample-accuracy, low-latency, less context switches, same routing infrastructure and session management for OSC and MIDI.
- Support for more complex real-time event data, e.g. for continuous controllers and motion or gesture driven sound synthesis.
- Tighter coupling between OSC and MIDI, possibility for on-the-fly translation of the two.

As it turns out, we can have all of this with vanilla JACK already, by piggybacking on top of JACK MIDI, one has to be cautious though not to interfere with the latter.

METHODS

Route raw OSC messages via JACK MIDI ports

Make use of JACK MIDI infrastructure, simply replace raw MIDI with raw OSC messages. JACK is indifferent to what the raw bytes encode and will handle routing and multiplexing for you. OSC types are not network encoded while inside the JACK graph. Each message has a timestamp, there are no bundles.

C-Type	Description	MIDI	OSC
uint32_t	time	23	44
size_t	size	3	20
uint8_t *	buffer	`09 a0 7f`	`/hello□□,s□□world□□□`

OSC message ⟶ JACK event

Inject single OSC messages at the beginning of the next sample period.

```
OSC packet (UDP/TCP)
└─/hello s 'world'
JACK OSC events
└─period (n)
  └─00 /hello s 'world'
```

OSC nested bundles ⟶ JACK events

Unroll OSC bundles into single messages, translate between NTP timestamps and JACK frame times, add to sorting event queue for later dispatch into JACK graph.

```
OSC packet (UDP/TCP)
└─#bundle 00000000.00000001
  ├─#bundle d6ecd330.58399c7b
  │ ├─/tuio2/frm it 3459 d6ecd330.58000000
  │ ├─/tuio2/tok iiifff 129 0 0 0.89 0.23 0.0
  │ └─/tuio2/alv i 129
  └─#bundle 00000000.00000001
    ├─/s_new siiisisi 'base' 129 0 0 'out' 0 'gate' 0
    └─#bundle d6ecd330.583a1200
      └─/n_set iififsi 129 0 0.89 1 0.23 'gate' 1

JACK OSC events
├─period (n)
│ └─00 /s_new siiisisi 'base' 129 0 0 'out' 0 'gate' 0
├─period (n+2)
│ ├─33 /tuio2/frm it 3459 d6ecd330.58000000
│ ├─33 /tuio2/tok iiifff 129 0 0 0.89 0.23 0.0
│ └─33 /tuio2/alv i 129
└─period (n+3)
  └─12 /n_set iififsi 129 0 0.89 1 0.23 'gate' 1
```

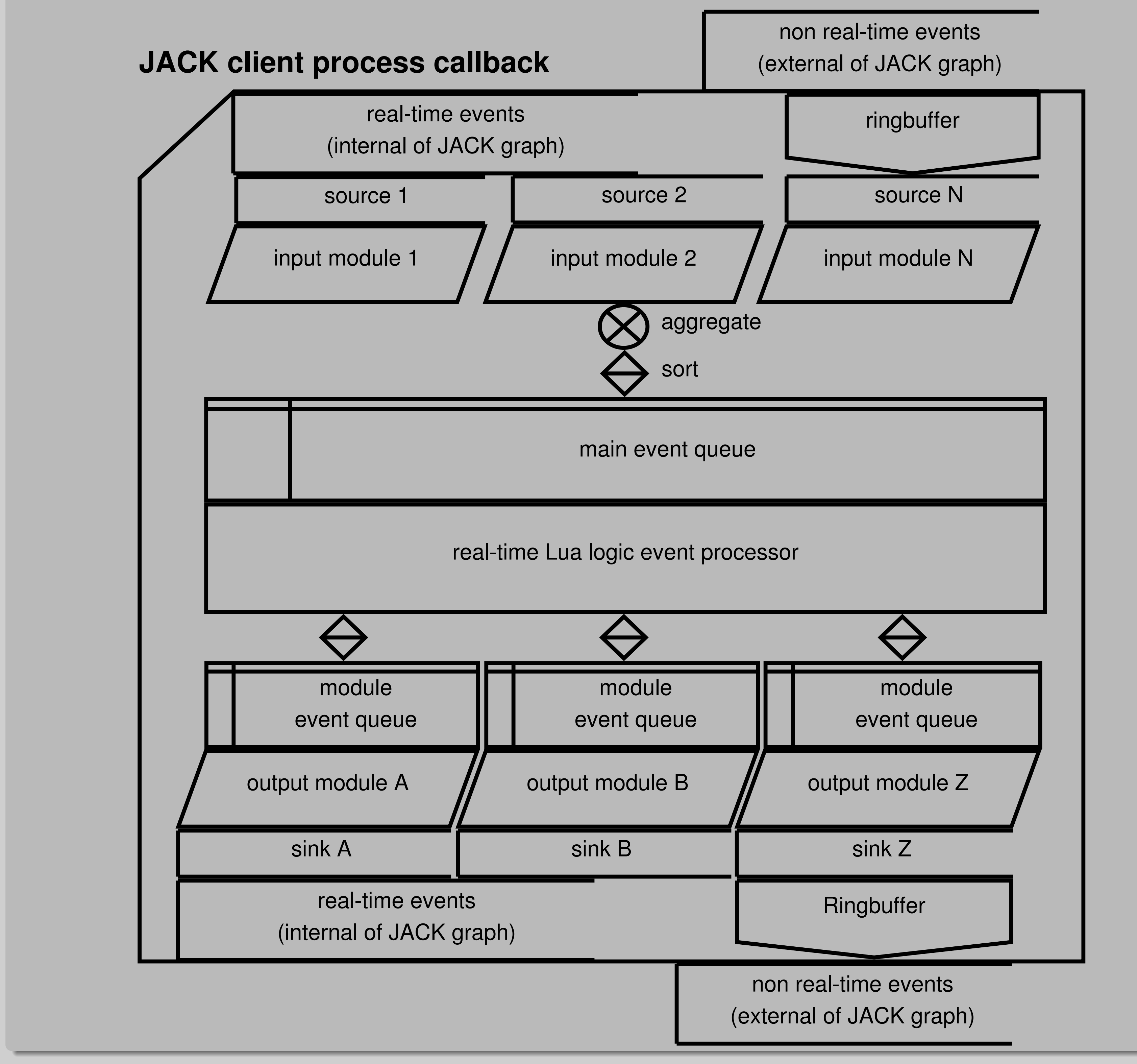
Discriminate between JACK MIDI and JACK OSC ports

Make use of the JACK metadata API to mark JACK MIDI ports used for OSC routing, makes JACK OSC clients aware of each other.

Key	Value
`'http://jackaudio.org/event/type'`	`'Open Sound Control'`
`'http://jackaudio.org/event/OSC/type'`	`'TUIO 2.0'`
`'http://jackaudio.org/event/OSC/type'`	`'SuperCollider'`

Tjost is **JACK**ified **O**pen **S**ound Control **T**ransmission

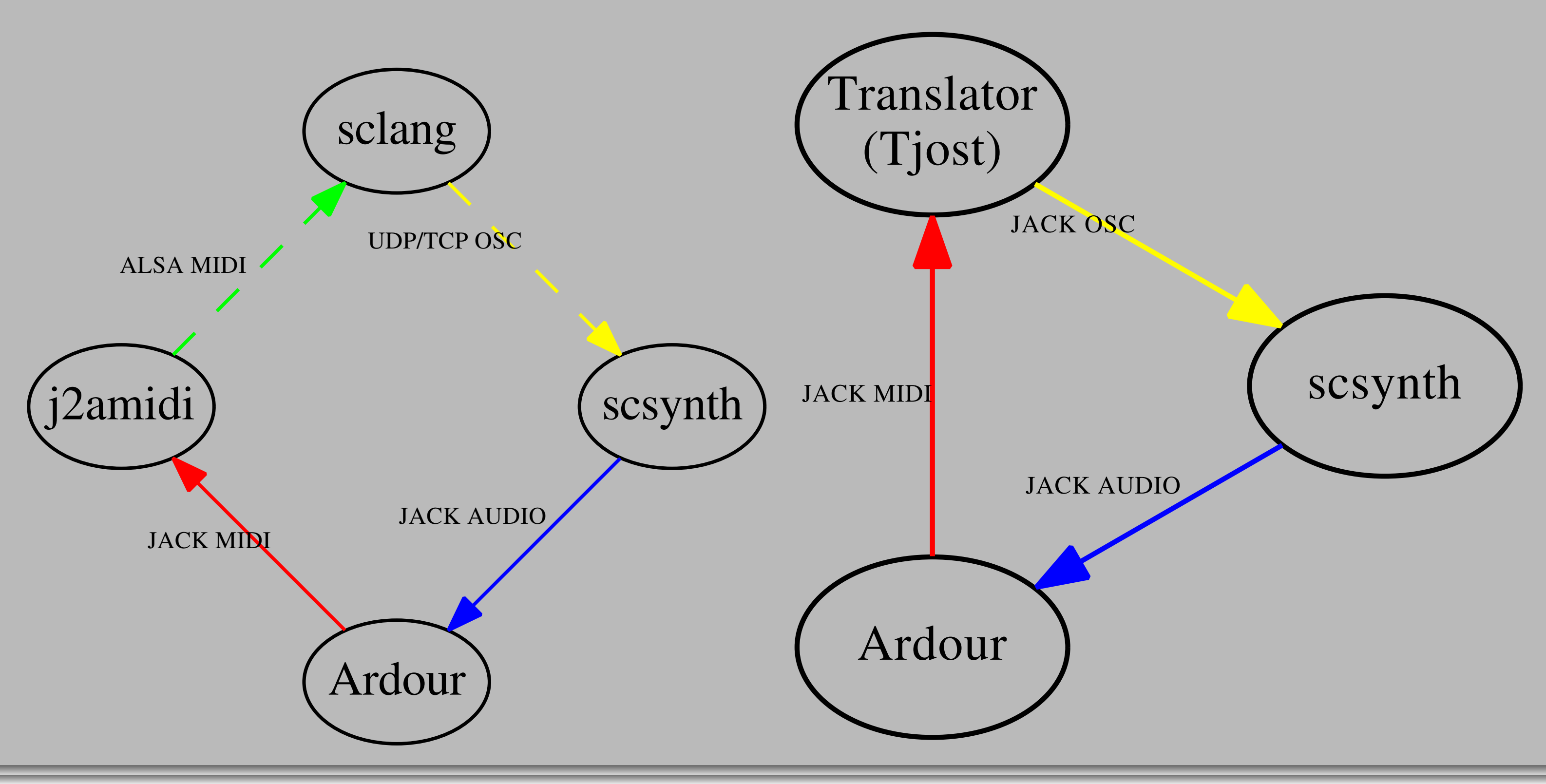
Tjost is a reference implementation of a scriptable event translator, handles OSC via UDP/TCP/JACK, JACK/ALSA MIDI, JACK AUDIO, embeds LuaJIT tuned for realtime execution in a JACK process thread.



JACK client process callback
non real-time events (external of JACK graph)
real-time events (internal of JACK graph)
ringbuffer
source 1
source 2
source N
input module 1
input module 2
input module N
aggregate
sort
main event queue
real-time Lua logic event processor
module event queue
module event queue
module event queue
output module A
output module B
output module Z
sink A
sink B
sink Z
real-time events (internal of JACK graph)
Ringbuffer
non real-time events (external of JACK graph)

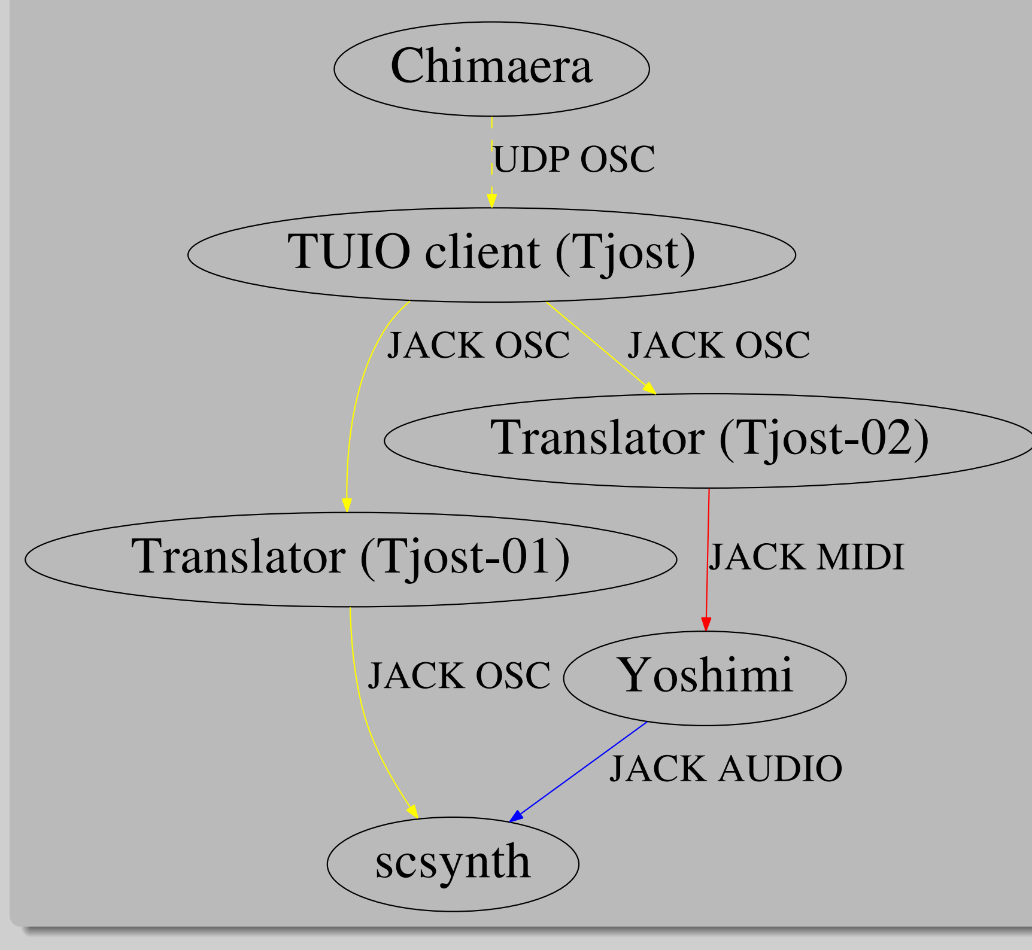
Case Study 1: Ardour Seq ⟶ SuperCollider ⟶ Ardour

From Ardour sequencer to scsynth in the same JACK process period.



Case Study 2: TUIO ⟶ MIDI + SuperCollider

Inject TUIO from UDP into JACK graph, translate to general on/off/set events, route to downstream translators to steer SuperCollider or some JACK MIDI app.



```
UDP OSC packet (Chimaera)
└─#bundle d6ecd332.5a3cf000
  ├─/tuio2/frm it 3460 d6ecd332.58000000
  ├─/tuio2/tok iiifff 234 1 128 0.89 0.23 0.0
  └─/tuio2/alv i 234
JACK OSC events (Tjost)
├─33 /on iii 234 1 128
└─33 /set iff 234 0.89 0.23
JACK OSC events (Tjost-01)
├─00 /s_new siiisi 'base' 234 0 1 'out' 0
└─33 /n_set iififsi 234 0 0.89 1 0.23 'gate 1
JACK MIDI events (Tjost-02)
├─33 91 4a 7f // note on
├─33 e1 0a 3f // pitch bend
├─33 b1 21 1a // controller fine
└─33 b1 01 7f // controller coarse
```

DISCUSSION

Problems!

- Legacy clients cannot distinguish between JACK MIDI and JACK OSC.
- Only JACK clients which route OSC set the event type via the metadata API and thus are aware of each other.
- JACK MIDI and JACK OSC can be connected by `jack_connect`.
- As long as OSC stays in the JACK graph we are fine, but when one tries to export the event, e.g. to ALSA (a2jmidid) or NetJACK2, clients will expect MIDI data.

Fixes?

- Generalize JACK MIDI for arbitrary unidirectional events (MIDI, OSC, ...)?
- Define `jack_event_*` and alias to `jack_midi_*` (downwards compatibility)?
- Use metadata API to set event type of a given port (MIDI, OSC, ...)?
- By default an event port would have type MIDI (downwards compatibility)?
- User can overwrite event port type, e.g. to route OSC?
- Modify `jack_connect` to only connect event ports with matching event types?
- Only allow MIDI to exit JACK graph, e.g. to/from ALSA, NetJACK2, ...?
- Include `libjack_osc` into JACK, or provide it as an extension?