Moony Workshop
Rapid prototyping of LV2 (MIDI) event filters in Lua

Hanspeter Portner
dev@open-music-kontrollers.ch
ventosus in #lv2, #lad on IRC

Linux Audio Conference, Université Jean Monnet, Saint-Etienne, France

May, 2017

Released under CC BY-SA 4.0
Goals

- Be able to write a (very) simple MIDI event filter in Lua for any LV2 Host
- And en route familiarize with:
  - LV2 and its atom event system
  - Lua scripting language
Overview and Tour

- Lua scripting language
- LV2 event system
- Inner workings of Moony
- Conducted tour through constructive example plugins

Hands-on experience by attendees
What is Moony?

- Programmable LV2 event plugin
- Event logic is scripted in Lua
- Lua is executed in real-time (not strictly necessary, but nice-to-have)
- Scripts can be updated on-the-fly
- Potentially runs in any LV2 host and on any platform
- Take your scripts to any host ...
What/whom is Moony meant for?

- Rapid prototyping
- Learning tool for LV2 atom event system
- Alternative event systems
- Algorithmic sequencing
- Algorithmic composition
- Advanced automation
- Live coding
Lua goals (architect’s view)

- Portability
- Simplicity
- Small size
- Scripting
Moony prerequisites

- Non-functional language
- Fast compilation
- Fast execution
- Small overhead
- Flexibility
- Full control over memory
- Modular sandboxing
Where is Lua used?

- Games
- AI Research (Google DeepMind)
- Window manager (Awesome)
- DAW (Renoise, Ardour)
Introduction

Lua language

Types

- Nil (aka undefined)
- Boolean
- Integer (5.3)
- Number
- String (character + byte string)
- Table
- Function
- Userdata (interfacing to C)
- Thread (coroutines)
Table

- Only data structure, but extremely versatile
- Associative array (any type as key and value: string, number, table, ...)
- Can implement various structures (set, array, sparse matrix, list, ...)
Function

- First-class value
- Anonymous functions
- Nested functions with full lexical scoping
Peculiarities

- All integer indexing starts at 1
- 0 evaluates to true in conditions
- and, or operators return operands on success
Port types

Audio port  sequence of audio samples
Control port  plugin parameters controlled by e.g. knobs and sliders
Atom port  sequence of events, e.g. MIDI ← todays topic
CV port  sequence of Control Voltage samples, rarely used)
## Port configurations

### Table: Stereo filter

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio In 1</td>
<td>Audio Out 1</td>
</tr>
<tr>
<td>Audio In 2</td>
<td>Audio Out 2</td>
</tr>
</tbody>
</table>

### Table: Event filter

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event In 1</td>
<td>Event Out 1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### Table: Stereo instrument

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event In</td>
<td>Audio Out 1</td>
</tr>
<tr>
<td>Audio In 1</td>
<td>Audio Out 1</td>
</tr>
<tr>
<td>Audio In 2</td>
<td>Audio Out 2</td>
</tr>
</tbody>
</table>
Why LV2?

- Support for any kind of event (not only MIDI)
- Sample accurate events
- Awesome session management
- Presets are portable and easily shareable
Atom prototype

LV2 atoms
Atom primitives (1)

- **Bool body**
  - size: int32
  - type: bool

- **32-bit Integer body**
  - size: int32
  - type: uint32

- **Single-precision body**
  - size: float
  - type: single

- **URID body**
  - size: uint32
  - type: urid

- **64-bit Integer body**
  - size: int64
  - type: int64

- **Double-precision body**
  - size: double
  - type: double
Atom primitives (2)

- **String body**
  - size
  - type
  - chars

- **Path body**
  - size
  - type
  - chars

- **URI body**
  - size
  - type
  - chars
  - datatype
  - lang

- **Literal body**
  - size
  - type
  - chars

- **Chunk body**
  - size
  - type
  - bytes

- **MIDI body**
  - size
  - type
  - bytes
Atom containers (1)

**Tuple body**
- size
- type
- atom 1
- atom 2
- atom N

**Vector body**
- size
- type
- child_size
- child_type
- body 1
- body 2
- body N
**Atom containers (2)**

- **Object body**
  - size
  - type
  - id
  - otype
  - property 1
  - property 2
  - property N

- **Property**
  - key
  - context
  - atom

- **Sequence body**
  - size
  - type
  - unit
  - pad
  - event 1
  - event 2
  - event N

- **Event**
  - frame/beat time
  - atom
A typical LV2 plugin and its threads

Worker → DSP → UI
Worker thread

start

wait_for_job

compile

mem_alloc

mem_free

compile_ok

mem_alloc_ok

mem_free_ok

end

error
DSP thread

- start
- is_first_time
- has_once
- once
- once_ok
- has_run
- run
- run_ok
- end
- error
UI thread

- start
- wait_for_event
- render
- port_event
- end
Moony code update cycle

Code replacement
Download / Install

- **download release (0.22.0) from**
  - [https://open-music-kontrollers.ch/lv2/moony](https://open-music-kontrollers.ch/lv2/moony)

- **install to**
  - `~/.lv2/moony.lv2`

- **work with presets**
  - study tutorials
  - modify templates

- **consult manuals**
  - `moony.lv2/manual.html`
  - [https://www.lua.org/manual/5.3](https://www.lua.org/manual/5.3)