lubyk

lua libraries for live arts

Gaspard Bucher (Buma)
artist, musician, coder
lubyk is not a framework
Why Lubyk?

- **Reuse** code from project to project
- **Accelerate** development (live coding)
- **Simple** APIs
- **Good** documentation
- Stability (unit **tests**)
History!

- **2006** First prototype in Ruby
  - Slow, inaccurate, *rubato* music
- **2008** Second pure C++ version, many threads, mutex. Lua scripting. Works when not crashing
2008 “Home” machine learning based movement recognition
History!

- **2006** First prototype in Ruby
  - Slow, inaccurate, *rubato* music
- **2008** Second pure C++ version, many threads, mutex. Lua scripting. Works when not crashing
- **2011** Third version, pure lua, Qt GUI, multi-process, network distribution (mdns, zeroMQ, Msgpack).
2011 “boats to nowhere”
2011 All written in lua (dub made Qt bindings)
What was wrong?

- Did not scale well to larger projects
- Confusing interface
- Hard to work on different parts (shaders, physics, control, music integration, etc).
- Too complex, hard to share modules
- **Complicated GUI takes too much dev time!!**
less is more

```lua
local midi = require 'lmidi'
local lens = require 'lens'

lens.run(function() lens.FileWatch() end)

midi_in = midi_in or midi.In(3)
function midi_in:receive(msg)
    print(msg.type)
end
```
History!

- **2006** First prototype in Ruby
  - Slow, inaccurate, **rubato** music
- **2008** Second pure C++ version, many threads, mutex. Lua scripting. Works when not crashing
- **2011** Third version, pure lua, Qt GUI, multi-process, network distribution (mdns, zeroMQ, Msgpack).
- **2014** Fourth version, modules, tests, doc
Modules

lut  |  Imidi  |  lui  |  media  |  bt  |  xml  |  yaml  |  osc  |  dub

lens  |  lug  |  xml  |  yaml  |  osc

lub
lub

- Class declaration
- Template \{{moustache}\}
- Directory traversal
- Algorithm (search)
- String, Array helpers
- etc
Documentation, testing

- Inline documentation
- Literate programming
- Testing framework
- Coverage
- Module setup/build
lut.Doc

Cross references

Functions

Functions neg, add, sub and smul are also accessible through operators:

```plaintext
local v = lug.V2(1, 2)
local nv = -v
local w = v + nv
local n = 4 * v
local m = v * 3
```

Latex Math

.neg (v)

Inverse vector $-v$.

.add (u, v)

Add two vectors: $u + v$.

.sub (u, v)

Subtract two vectors: $u - v$.

.mul (u, v)

Multiply two vectors: $u \cdot v$. 

```latex
\neg (v)
Inverse vector $-v$.
\add (u, v)
Add two vectors: $u + v$.
\sub (u, v)
Subtract two vectors: $u - v$.
\mul (u, v)
Multiply two vectors: $u \cdot v$.
```
# Automatic script reloading (live coding)

In this tutorial, we show how to use `lens FileWatch` to live code a lua script.

## Download source

`[LiveCoding.lua](example/lens/LiveCoding.lua)`

```lua
-- doc:lit

-- Preamble

-- Require lens library.
local lens = require 'lens'

-- Start scheduler and setup script reload hook with `lens FileWatch`. Starting
-- the scheduler at the top of the script and using file reloading is a nice
-- trick that ensures all the code after `lens run` is only executed within the
-- scope of the scheduler.

lens.run(function() lens.FileWatch() end)

-- This part of the script is executed on the initial `FileWatch` call. This
-- means we are actually within the scheduler loop and we can therefore create
-- threads and timers.

-- Do something

-- Here we create a timer for demonstration purpose but you could as well create
-- a window, a socket or whatever you need to do.

-- If the timer is not in a separate thread, `lentil` triggers the
-- timer's timer function only when `lens.run` is called again.
```
Automatic script reloading (live coding)

In this tutorial, we show how to use `lens.FileWatch` to live code a Lua script.

Download source
LiveCoding.lua

Preamble

Require lens library.

```lua
local lens = require 'lens'
```

Start scheduler and setup script reload hook with `lens.FileWatch`. Starting the scheduler at the top of the script and using file reloading is a nice trick that ensures all the code after `lens.run` is only executed within the scope of the scheduler.

```lua
lens.run(function() lens.FileWatch() end)
```

This part of the script is executed on the initial `FileWatch` call. This means we are actually within the scheduler loop and we can therefore create threads and timers.
Testing with \texttt{lut.Test}

```javascript
local lub = require 'lub'
local lut = require 'lut'
local should = lut.Test 'lub'

function should.readAll()
    local p = lub.path '|fixtures/io.txt'
    assertEquals('Hello Lubyk!\n', lub.content(p))
end

function should.absolutizePath()
    assertEquals(lufs.currentdir() .. '/foo/bar', lub.absolutizePath('foo/bar'))
    assertEquals('/foo/bar', lub.absolutizePath('/foo/bar'))
end

function should.merge()
    local base = {a = { b = {x=1}}, c = {d = 4}}
    lub.merge(base, {
        a = 'hello',
        d = 'boom',
    })
    assertEqualsEqual({a = 'hello', c = {d = 4}, d = 'boom'}, base)
end
```
Libraries using lut
lens

- Scheduling
- Socket (TCP, UDP)
- Thread
- Timer
- FileWatch
encode / decode
Core graphic types

- four
- lut
- l midi
- lui
- medi a
- bt
- lens
- lug
- xml
- yaml
- osc
- dub
- lub
bulletphysics (3D physics)

- lug integration
media (video, image)

- Native (AVFoundation)
- Fast, asynchronous
- Multi-threaded
OpenGL 4

- Shaders
- Geometry
- Etc
Native UI

- Provides OpenGL context
- Event loop integration with lens
- View (window/view)
- Mouse, keyboard
Midi in/out
Virtual ports, real ports
Note handling (off event)
C++ binding generator

- Advanced type management
- GC protection
- Fast callbacks from C
- Cast, operators, etc
- Customizable
- Uses doxygen
Libraries using **dub**

![Diagram showing various libraries using `dub`](image-url)
Windows

alpha
beta
released
Linux

alpha
beta
released

lut
Imidi
lui
media
bt
lens
lug
xml
yaml
osc
dlub
Mac OS X

- media = 10.9
Future plans

• Simple multi-machine, multi-process support (**mdns** + **zmq** + **dropbox**)  

• Parameter support (per effect, per object, etc). Makes code reuse and adaptation easy.  
• Eventually, use a simple web app for this.  
• Exploration needed, **ideas welcome** !
Immediate plans

• Finish Linux port, start Windows port, for fun RPi
• Extract **lug** from four and optimize data transfers
• Fix old lubyk libraries
  • **mdns** (Zeroconf plug&play network)
  • **zmq** (ZeroMQ messaging library)
  • **box2D** (2D physics)
• Parameter handling
• **Workshops** !
using lubyk

• Stable modules: luarocks install ...
• Licence: MIT
• Documentation: doc.lubyk.org
• Source code: github.com/lubyk
• Twitter: @lubyk_
спасибо

lots of projects to follow on

@bumagy