Making your app interact with Liquidsoap

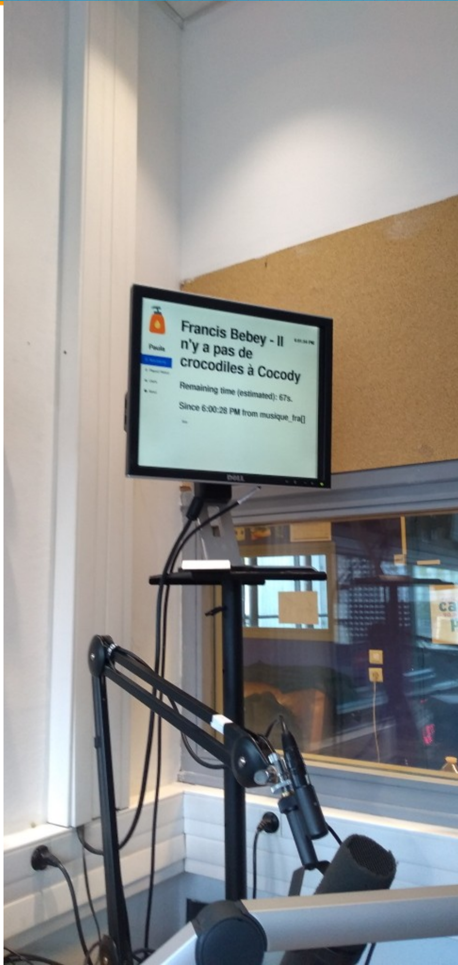
Martin Kirchgessner

Liquidshop 3

30 May 2023

Context

- Martin Kirchgessner, Web developer (Python)
- Benevolent tech at *Radio Campus Grenoble*
- Using Liquidsoap for 2 years
  … with Showergel, a Liquidsoap companion app


Francis Bebey - Il n'y a pas de crocodiles à Cocody
Remaining time (estimated): 67s.
Since 6:00:28 PM from musique_fra()

Liquidsoap needs companion apps

From https://www.liquidsoap.info/doc-2.1.4/

Non-Features

[...]

Liquidsoap itself doesn’t have a nice GUI or any graphical programming environment.

[...]

Liquidsoap doesn’t do any database or website stuff

Today's tutorial

- Make Liquidsoap call your app
- Send Liquidsoap commands from your app
- Useful commands
- External commands and source composition

Two strategies of interaction

1) Make Liquidsoap call your app
2) Send Liquidsoap commands from your app

You can use both !

Make Liquidsoap call your app

Command line call

- request.dynamic + process.read
- request.dynamic will call your program when trying to load something to play next
- great for music

Command line call - example

Using the beets music library:

```
$ beet random -f '$path' added:-1y.. genre:Reggae
/home/martin/beets/Non-Album/Yellowman/We Wish You a Reggae Christmas.mp3
```

Creating a Liquidsoap source:

```
recent_reggae = request.dynamic(id=“recent_reggae“, retry_delay=1., {
    request.create(string.trim(
          process.read("beet random -f '$path' added:-1y.. genre:Reggae")
    ))
})
```

HTTP calls

If your app has an HTTP server

- http.put
- http.post → with JSON data
- http.get
- http.head
- http.delete

Great for metadata !

Prepare metadata

- metadata.cover.remove
- metadata.export → removes LS metadata

```
def post_metadata(md)
  response = http.post("http://localhost:1234/metadata_log",
    headers=[("Content-Type", "application/json; charset=UTF-8")],
    data=metadata.json.stringify(metadata.cover.remove(md))
  )
  if response.status_code != 200
  then
    log(label="Warning", "#{response} #{response.status_code} #{response.status_message}")
  end
end

source.on_metadata(fun(m) -> thread.run(fast=false, {post_metadata(m)}))
```

Send Liquidsoap commands from your app

Choose your connection

Telnet or HTTP ?

Telnet

```
settings.server.telnet.set(true)
```

- Configure binding IP / port / socket
- Send a command, parse a response
- A legacy protocol… that works

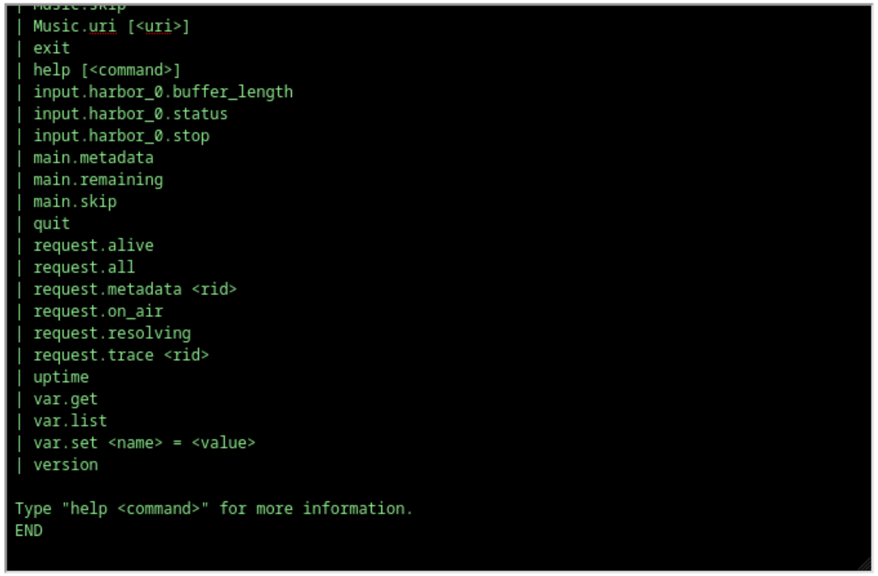
HTTP

`server.harbor()`

POST a command,

parse the response

Liquidsoap telnet server



```
| Music.skip
| Music.uri [<uri>]
| exit
| help [<command>]
| input.harbor_0.buffer_length
| input.harbor_0.status
| input.harbor_0.stop
| main.metadata
| main.remaining
| main.skip
| quit
| request.alive
| request.all
| request.metadata <rid>
| request.on_air
| request.resolving
| request.trace <rid>
| uptime
| var.get
| var.list
| var.set <name> = <value>
| version

Type "help <command>" for more information.
END
```

Type `help` if you are lost.

⚠There is only one harbor

- In my script

```
settings.harbor.bind_addrs.set(["192.168.1.14"])
server.harbor(port=8000)
live = input.harbor(port=8008, "live")
```

← **Always add auth/user/pass**

- In the log

```
[server.harbor:3] Website should be ready at <http://localhost:8000/telnet>.
```

- ⚠ But it's at http://192.168.1.14:8000/telnet
- Could be a public address 😱

I’m sticking to telnet…

… for now

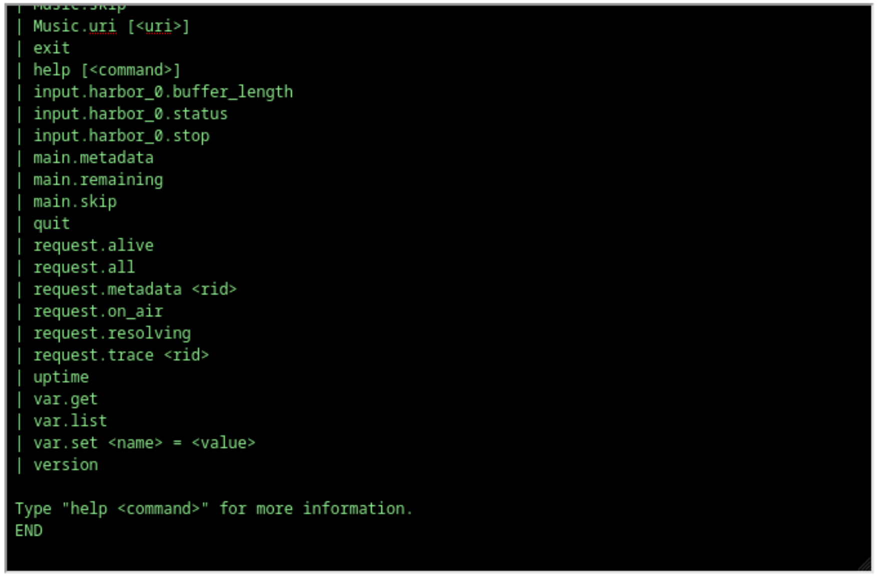
How about WebSockets ?

Useful commands

help

- Your best friend when developping !
- List available commands ; most start with the object ID

Liquidsoap telnet server



```
| Music.skip
| Music.uri [<uri>]
| exit
| help [<command>]
| input.harbor_0.buffer_length
| input.harbor_0.status
| input.harbor_0.stop
| main.metadata
| main.remaining
| main.skip
| quit
| request.alive
| request.all
| request.metadata <rid>
| request.on_air
| request.resolving
| request.trace <rid>
| uptime
| var.get
| var.list
| var.set <name> = <value>
| version

Type "help <command>" for more information.
END
```

Type `help` if you are lost.

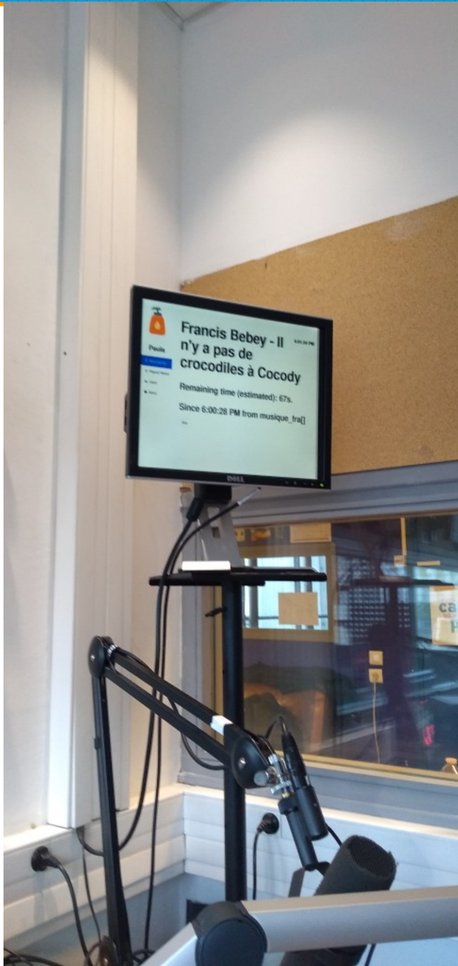
Always use „id“

- Every source/operator has an „id“ argument
- Give it a meaningful name
  - Clearer logs
  - Clearer commands

Output commands

When your main output's ID is `main_out`

- `main_out.skip` → skips current track
- `main_out.metadata` ⚠ can contain lyrics/image
- `main_out.remaining` → seconds in current track


Francis Bebey - Il n'y a pas de crocodiles à Cocody
Remaining time (estimated): 67s.
Since 6:00:28 PM from musique_fra[]

Pushing requests

If your script contains `request.queue(id="queuedfiles")`

- `queuedfiles.flush_and_skip`
- **`queuedfiles.push <uri>`**
- `queuedfiles.queue` → list of requests IDs (RID)
- `queuedfiles.skip`

Beware of RIDs

Those commands are always available:

- `request.alive`
- `request.all` → all RIDs in use by Liquidsoap (expect many)
- `request.metadata <rid>`
- `request.on_air` → can return „10 45“ 🤔
- `request.resolving`
- `request.trace <rid>`

HTTP Streams

If your script contains:

`input.http(id="external_stream", "https://...",` **`start=false, max_buffer=30.`**`)`

- `external_stream.buffer_length`
- `external_stream.start`
- `external_stream.status`
- `external_stream.stop`
- `external_stream.url [url]`

⬆

Won’t stream until the .start command

HTTP Streams

If your script contains `input.harbor(id="incoming", ...`

- `incoming.buffer_length`
- `incoming.status`
- `incoming.stop` → „kick“ current stream

External commands
and
source composition

Keep in mind

Everything can

Crash

Disappear

Stutter

Make noise

…

Commands lag

- Pushing a request starts its download
- Starting a stream starts its *buffering*
- Switching sources will leave the old one as it was
    - In the middle on a song/show
    - With a bit more content in the buffer

Track-sensitive or not ?

`Switch` / `fallback` operators have a `track_sensitive` option

- Enabling it (default) is useful to enqeue files
  - You can still flush push/after switch
- Track-insensitive is useful for streams

Staying on time is hard

The mixer

Liquidsoap script

```
v1 = interactive.float("volume1", 1.)
s1 = amplify(v1, request.queue(id="track1"))

v2 = interactive.float("volume2", 0.)
s2 = amplify(v2, request.queue(id="track2"))

source = add(s1, s2)
...
```

Commands

[track1 is on air]

```
track2.push /path/to/file.wav
var.set volume2 = 1.
var.set volume1 = 0.
track1.flush_and_skip
```

The mixer

⚠ Sources at `volume=0.` are still playing (if available)

- At least, it can flush HTTP sources’ buffers

… but is the source at `volume=1.` really playing ?

Conclusion

🤩So many options

(many others) Sources and operators!

- Active / passive app
- HTTP / telnet / CLI

So many dangers !

Everything can

Crash

Disappear

Stutter

Make noise

…

Showergel

- See „quick install“ on https://github.com/martinkirch/showergel

- quickstart.liq provides many examples

Showergel

- Plugging it on any Liquidsoap script is too hard
- Will progressively hide the Liquidsoap script
- Will change its name

  … yet another radio automaton !

Your app will be bound to Liquidsoap

- Your app and your script.liq are *heavily* coupled
  - call/connection strategy
  - mixing strategy
  - sources/operators identifiers
  - failure handling
- When upgrading LS, read the changelog (even bugfixes)

Documentation suggestion

- Add new / experimental / legacy / deprecated flags
- Show each page’s modification date
- Core/Extra API ?

Thanks !

github.com/martinkirch

@martin_kirch@piaille.fr