Liquidoap 2.0.x - A look back

...and forth!

Romain Beauxis, Jan. 23 2022

Liquidsoap 2.0.x

The look back

- FFmpeg integration
- Language features
- The book

FFmpeg integration

What FFmpeg provides

- De-facto reference implementation, library and API for multimedia programming
- Excellent quality and APIs
- All-encompassing project, codec, muxers, I/O protocols & devices, and filters
- 20+ years of existence, worldwide community of expert developers

FFmpeg integration

What Liquidsoap provides

- Programming language
- Specialized operators and variables (sources, filters, input/outputs)
- Programming abstractions (functions, abstract data structures etc)
- Static typing!

FFmpeg integration

A scripting language powered by FFmpeg

- Tight integration with FFmpeg APIs
- What the ffmpeg CLI can do, we should be able to do
- Support for all codecs, muxers, I/O protocols, etc.
- Programming language flexibility

FFmpeg integration

…but not only!

- There are limits to what FFmpeg provides vs. custom implementations
- HLS output: segment callbacks, name, etc.
- SRT I/O
- Devices support

FFmpeg integration

Example 1: Complex filter

FFmpeg integration

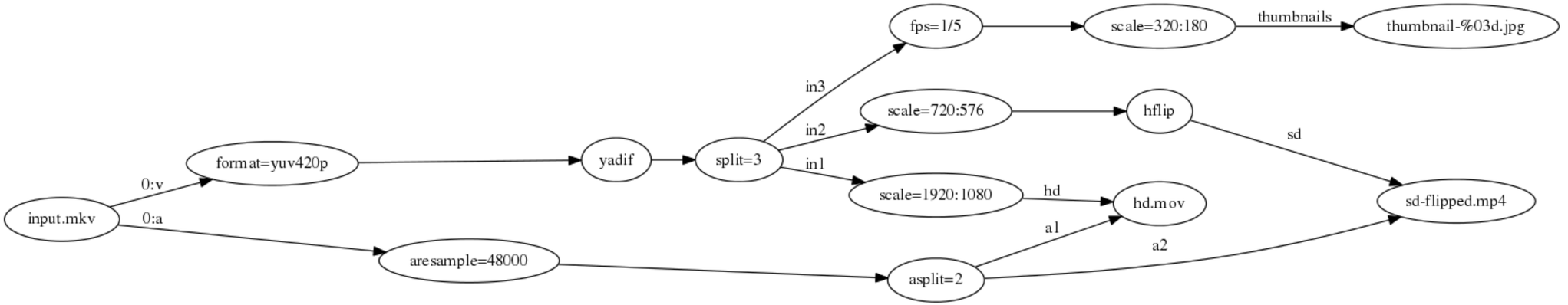
Example 1: Complex filter

```
ffmpeg -y -i input.mkv \
-filter_complex "[0:v]format=yuv420p,yadif,split=3[in1][in2][in3];
[in1]scale=1920:1080[hd];[in2]scale=720:576,hflip[sd];
[in3]fps=1/5,scale=320:180[thumbnails];
[0:a]aresample=48000,asplit=2[a1][a2]" \
-map [hd] -map [a1] hd.mov \
-map [sd] -map [a2] sd-flipped.mp4 \
-map [thumbnails] thumbnail-%03d.jpg
```

FFmpeg integration

Example 1: Complex filter

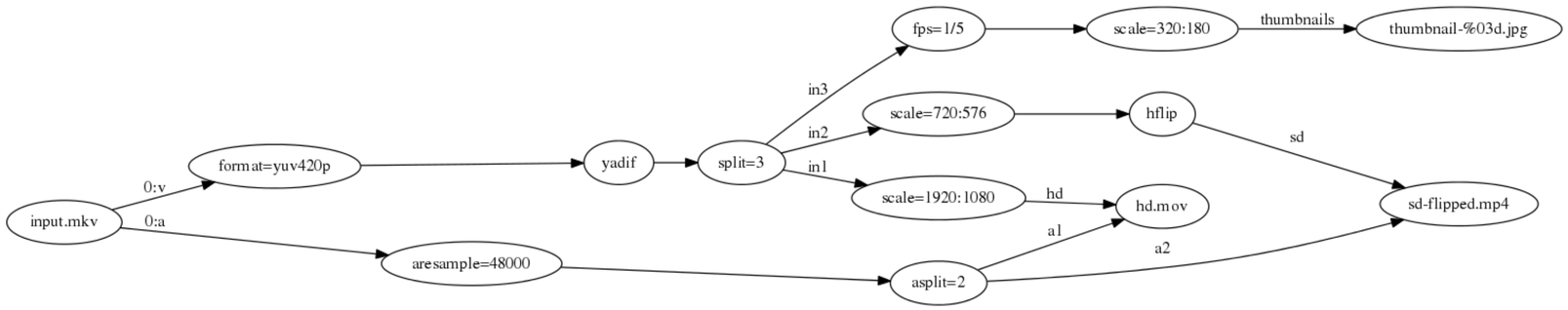
```
ffmpeg -y -i input.mkv \
-filter_complex "[0:v]format=yuv420p,yadif,split=3[in1][in2][in3];
[in1]scale=1920:1080[hd];[in2]scale=720:576,hflip[sd];
[in3]fps=1/5,scale=320:180[thumbnails];
[0:a]aresample=48000,asplit=2[a1][a2]" \
-map [hd] -map [a1] hd.mov \
-map [sd] -map [a2] sd-flipped.mp4 \
-map [thumbnails] thumbnail-%03d.jpg
```



Source: https://medium.com/craftsmenltd/ffmpeg-basic-filter-graphs-74f287dc104e

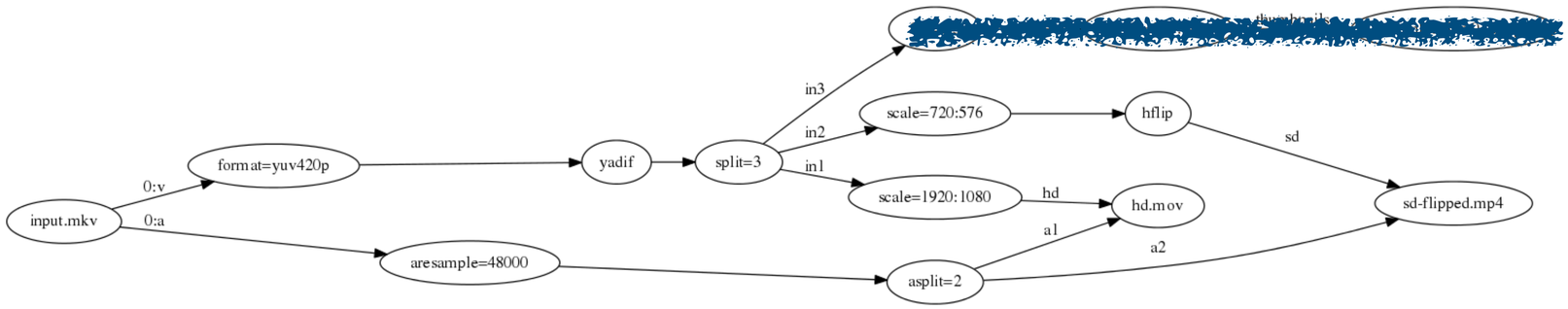
FFmpeg integration

Example 1: Complex filter



FFmpeg integration

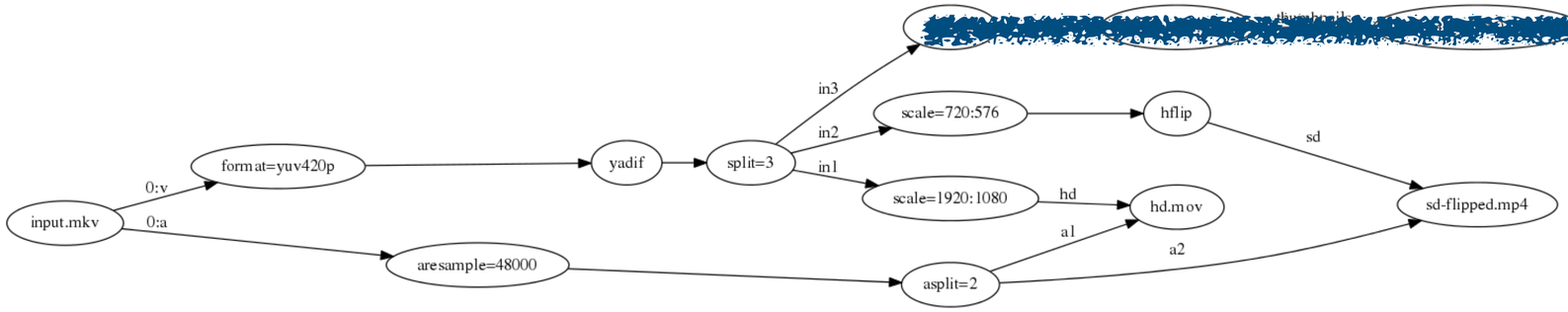
Example 1: Complex filter



FFmpeg integration

Example 1: Complex filter

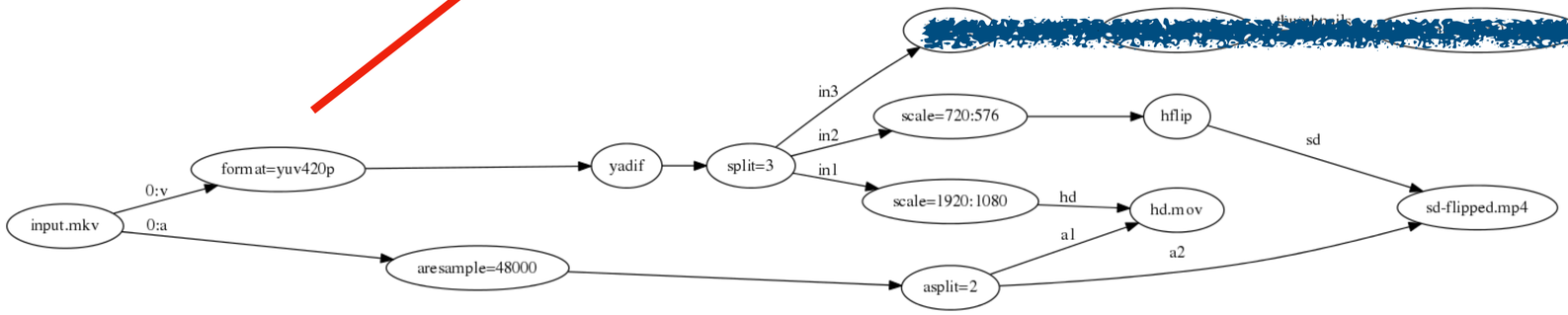
```
def complex_filter(s) =
  def mkfilter(graph) =
    a = ffmpeg.filter.audio.input(graph, drop_video(s))
    v = ffmpeg.filter.video.input(graph, drop_audio(s))

    v = ffmpeg.filter.format(pix_fmts="yuv420p", graph, v)
    v = ffmpeg.filter.yadif(graph, v)

    let (_, v) = ffmpeg.filter.split(outputs=2, graph, v)
    let [in1, in2] = v

    in1 = ffmpeg.filter.scale(size="1920:1080", graph, in1)

    in2 = ffmpeg.filter.scale(size="720:576", graph, in2)
    in2 = ffmpeg.filter.hflip(graph, in2)

    a = ffmpeg.filter.aresample(sample_rate=48000, graph, a)

    a   = ffmpeg.filter.audio.output(graph, a)
    in1 = ffmpeg.filter.video.output(graph, in1)
    in2 = ffmpeg.filter.video.output(graph, in2)

    {in_1 = mux_audio(audio=a, in1),
     in_2 = mux_audio(audio=a, in2)}
  end

  ffmpeg.filter.create(mkfilter)
end
```



FFmpeg integration

Example 1: Complex filter

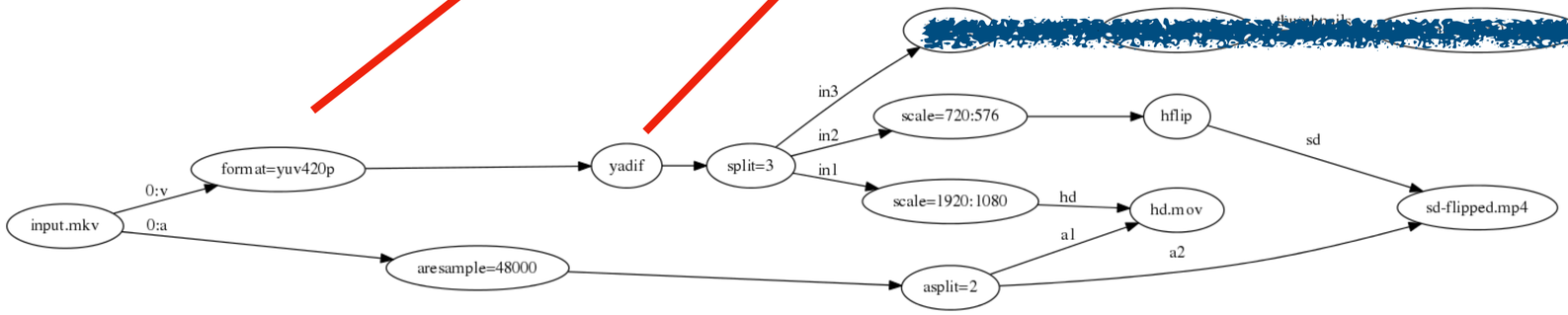
```
def complex_filter(s) =
  def mkfilter(graph) =
    a = ffmpeg.filter.audio.input(graph, drop_video(s))
    v = ffmpeg.filter.video.input(graph, drop_audio(s))

    v = ffmpeg.filter.format(pix_fmts="yuv420p", graph, v)
    v = ffmpeg.filter.yadif(graph, v)

    let (_, v) = ffmpeg.filter.split(outputs=2, graph, v)
    let [in1, in2] = v

    in1 = ffmpeg.filter.scale(size="1920:1080", graph, in1)

    in2 = ffmpeg.filter.scale(size="720:576", graph, in2)
    in2 = ffmpeg.filter.hflip(graph, in2)

    a = ffmpeg.filter.aresample(sample_rate=48000, graph, a)

    a   = ffmpeg.filter.audio.output(graph, a)
    in1 = ffmpeg.filter.video.output(graph, in1)
    in2 = ffmpeg.filter.video.output(graph, in2)

    {in_1 = mux_audio(audio=a, in1),
     in_2 = mux_audio(audio=a, in2)}
  end

  ffmpeg.filter.create(mkfilter)
end
```



FFmpeg integration

Example 1: Complex filter

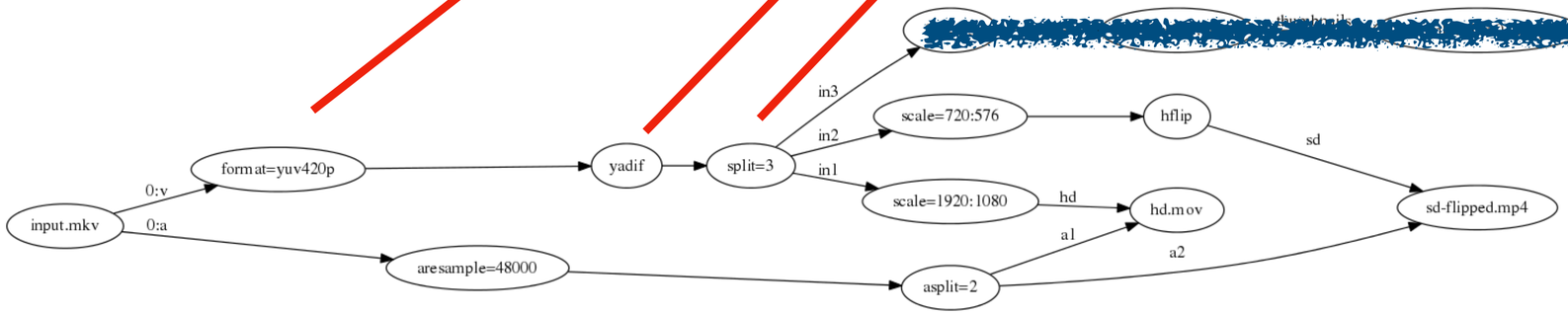
```
def complex_filter(s) =
  def mkfilter(graph) =
    a = ffmpeg.filter.audio.input(graph, drop_video(s))
    v = ffmpeg.filter.video.input(graph, drop_audio(s))

    v = ffmpeg.filter.format(pix_fmts="yuv420p", graph, v)
    v = ffmpeg.filter.yadif(graph, v)

    let (_, v) = ffmpeg.filter.split(outputs=2, graph, v)
    let [in1, in2] = v

    in1 = ffmpeg.filter.scale(size="1920:1080", graph, in1)

    in2 = ffmpeg.filter.scale(size="720:576", graph, in2)
    in2 = ffmpeg.filter.hflip(graph, in2)

    a = ffmpeg.filter.aresample(sample_rate=48000, graph, a)

    a   = ffmpeg.filter.audio.output(graph, a)
    in1 = ffmpeg.filter.video.output(graph, in1)
    in2 = ffmpeg.filter.video.output(graph, in2)

    {in_1 = mux_audio(audio=a, in1),
     in_2 = mux_audio(audio=a, in2)}
  end

  ffmpeg.filter.create(mkfilter)
end
```



FFmpeg integration

Example 1: Complex filter

```
def complex_filter(s) =
  def mkfilter(graph) =
    a = ffmpeg.filter.audio.input(graph, drop_video(s))
    v = ffmpeg.filter.video.input(graph, drop_audio(s))

    v = ffmpeg.filter.format(pix_fmts="yuv420p", graph, v)
    v = ffmpeg.filter.yadif(graph, v)

    let (_, v) = ffmpeg.filter.split(outputs=2, graph, v)
    let [in1, in2] = v

    in1 = ffmpeg.filter.scale(size="1920:1080", graph, in1)

    in2 = ffmpeg.filter.scale(size="720:576", graph, in2)
    in2 = ffmpeg.filter.hflip(graph, in2)

    a = ffmpeg.filter.aresample(sample_rate=48000, graph, a)

    a   = ffmpeg.filter.audio.output(graph, a)
    in1 = ffmpeg.filter.video.output(graph, in1)
    in2 = ffmpeg.filter.video.output(graph, in2)

    {in_1 = mux_audio(audio=a, in1),
     in_2 = mux_audio(audio=a, in2)}
  end

  ffmpeg.filter.create(mkfilter)
end
```



FFmpeg integration

Example 2: Stream with no re-encoding

FFmpeg integration

Example 2: Stream with no re-encoding

```
static = single("/path/to/file")
source = playlist("list.m3u")

source = fallback(track_sensitive=false, [source, static])

enc=%ffmpeg(
  format="flv",
  %audio.copy,
  %video.copy)

output.url(
  url="rtmp://host:port/path/key",
  enc,
  source)
```

Language Features

Simple things should be simple…

Language Features

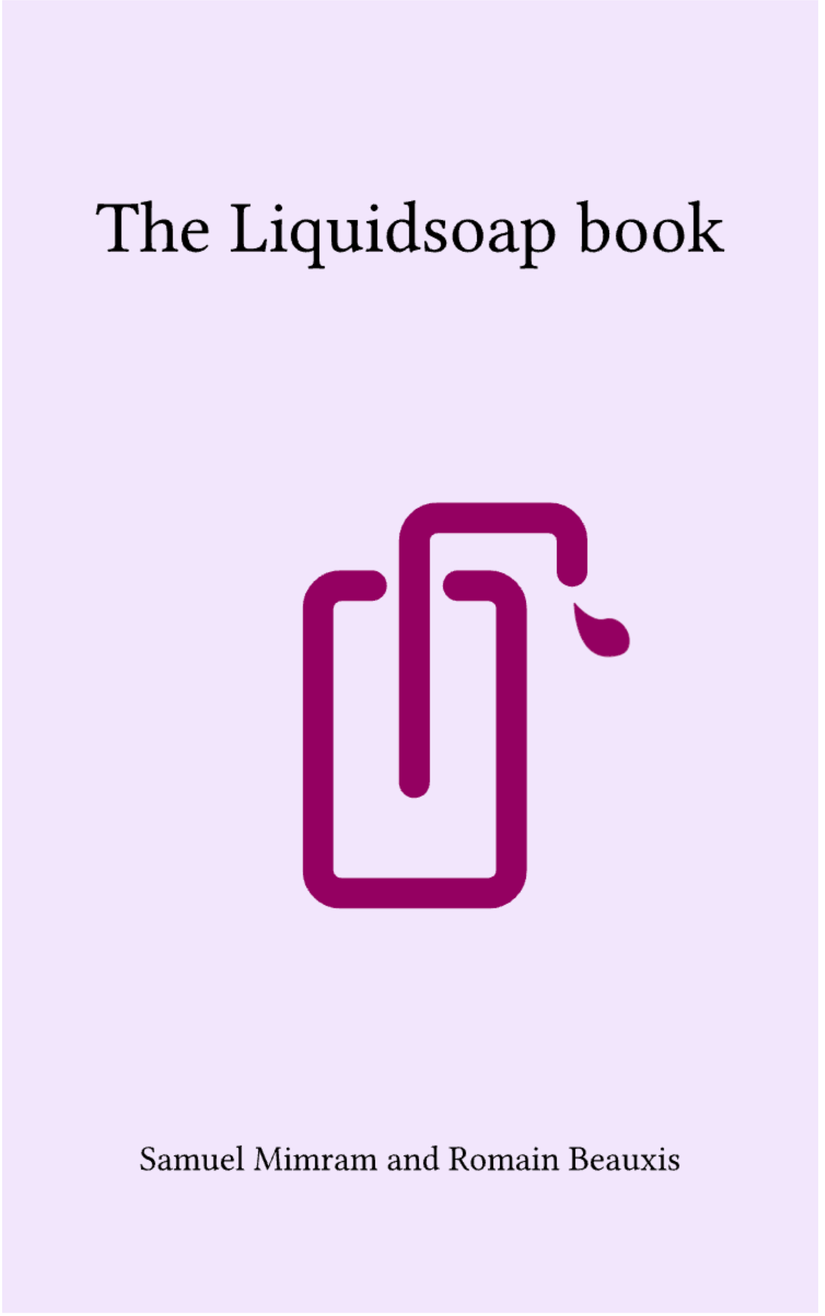
Simple things should be simple…

- Not all people interested in streaming are programmers!
- Programming language can and should help
- Complexity should arise only when needed
- Be aware of user requests and features
- Fix bugs!
- … but with limited resources

Language Features

Simple things should be simple…

- Not all people interested in streaming are programmers!
- Programming language can and should help
- Complexity should arise only when needed
- Be aware of user requests and features
- Fix bugs!
- … but with limited resources



Language Features

Simple things should be simple…

```
At /tmp/filter.liq, line 17, char 44-51:
    a = ffmpeg.filter.aresample(sample_rate="48000", graph, a)

Error 5: this value has type
  string
but it should be a subtype of
  int?
```

Language Features

Simple things should be simple…

Feature Request: Show Relevant Code in Error Messages #2063

Closed **SlvrEagle23** opened this issue on Nov 18, 2021 · 0 comments · Fixed by #2086

SlvrEagle23 commented on Nov 18, 2021

Is your feature request related to a problem? Please describe.

Due to the nature of Liquidsoap's error messages referring to line x and columns x-y of a given configuration file, we're often left referring back and forth to the actual generated configuration file to see what line is causing the problem. Given that the error printout already knows what line/columns are causing the problem, it seems like it would be helpful to actually print those lines themselves into the error log instead of just directing us to them in the file. When diagnosing issues, that would save a *huge* amount of time.

👍 2

Language Features

Simple things should be simple…

Liquidsoap scripting language reference

The **Source / …** categories contain all functions that return sources. The **Input** functions are those which build elementary sources (playing files, synthesizing sound, etc.). The **Output** functions are those which take a source and register it for being streamed to the outside (file, soundcard, audio server, etc.). The **Visualization** functions are experimental ones that let you visualize in real-time some aspects of the audio stream. The **Sound Processing** functions are those which basically work on the source as a continuous audio stream. They would typically be mixers of streams, audio effects or analysis. Finally, **Track Processing** functions are basically all others, often having a behaviour that depends on or affects the extra information that liquidsoap puts in streams: track limits and metadata.

Search:

- Source / Conversions
- Source / Input
- Source / Liquidsoap
- Source / MIDI Processing
- Source / Output
- Source / Sound Processing

Language Features

Simple things should be simple…

```
❯ liquidsoap -h request.queue

Play a queue of uris. Returns a function to push new uris in the queue as
well as the resulting source.

Type: (?id : string?, ?interactive : bool, ?prefetch : int,
 ?native : bool, ?queue : [request], ?timeout : float) ->
source(audio='a, video='b, midi='c)

Category: Source / Track Processing
```

Language Features

Simple things should be simple…

```
Parameters:

 * id : string? (default: null)
     Force the value of the source ID.

 * interactive : bool (default: true)
     Should the queue be controllable via telnet?

 * prefetch : int (default: 1)
     How many requests should be queued in advance.

 * native : bool (default: false)
     Use native implementation.

 * queue : [request] (default: [])
     Initial queue of requests.

 * timeout : float (default: 20.)
     Timeout (in sec.) for a single download.
```

Language Features

Simple things should be simple…

```
Methods:

 * add : (request) -> bool
     Add a request to the queue. Requests are resolved before being added.
     Returns `true` if the request was successfully added.

 * current : () -> request?
     Get the request currently being played.

 * duration : () -> float
     Estimation of the duration of the current track.

 * elapsed : () -> float
     Elapsed time in the current track.

 * fallible : bool
     Indicate if a source may fail, i.e. may not be ready to stream.
```

Language Features

…but complex things should be possible

```
def replaces request.dynamic(%argsof(request.dynamic), fn) =
  s = request.dynamic(%argsof(request.dynamic), fn)

  s.on_get_ready(memoize({
    server.register(namespace=s.id(), description="Flush the queue and skip the current track",
                    "flush_and_skip", fun (_) -> try
                      s.set_queue([])
                      s.skip()
                      "Done."
                    catch err do
                      "Error while flushing and skipping source: #{err}"
                    end)
  }))

  s
end
```

Language Features

Yet, sometimes, complexity is inevitable

Language Features

Yet, sometimes, complexity is inevitable

```
def playlist_fallback(a)
  fallback([a, playlist("list.m3u")])
end
```

Language Features

Yet, sometimes, complexity is inevitable

```
❯ liquidsoap /tmp/record.liq -h playlist_fallback

No documentation available.

Type: (source(audio='a, video='b, midi='c)
 .{
   time : () -> float,
   shutdown : () -> unit,
   fallible : bool,
   skip : () -> unit,
   seek : (float) -> float,
   is_active : () -> bool,
   is_up : () -> bool,
   log :
   {level : (() -> int?).{set : ((int) -> unit)}
   },
   self_sync : () -> bool,
   duration : () -> float,
   elapsed : () -> float,
   remaining : () -> float,
   on_track : ((([string * string]) -> unit)) -> unit,
```

```
   on_leave : ((() -> unit)) -> unit,
   on_get_ready : ((() -> unit)) -> unit,
   on_shutdown : ((() -> unit)) -> unit,
   on_metadata : ((([string * string]) -> unit)) -> unit,
   last_metadata : () -> [string * string]?,
   is_ready : () -> bool,
   id : () -> string,
   current : () -> request?,
   set_queue : ([request]) -> unit,
   add : (request) -> bool,
   queue : () -> [request],
   fetch : () -> bool,
   reload : (?uri : string?) -> unit,
   length : (() -> int)
 }) -> source(audio='a, video='b, midi='c)
```

Language Features

Yet, sometimes, complexity is inevitable

```
def playlist_fallback(a)
  fallback([a, playlist("list.m3u")])
end

s = playlist_fallback(input.harbor("mount"))
```

Language Features

Yet, sometimes, complexity is inevitable

```
def playlist_fallback(a)
  fallback([a, playlist("list.m3u")])
end

s = playlist_fallback(input.harbor("mount"))
```

```
At /tmp/record.liq, line 5, char 22-43:
s = playlist_fallback(input.harbor("mount"))

Error 5: this value has no method length
```

Language Features

Yet, sometimes, complexity is inevitable

```
def playlist_fallback(a)
  fallback([a, (playlist("list.m3u"):source)])
end

s = playlist_fallback(input.harbor("mount"))
```

Language Features

Yet, sometimes, complexity is inevitable

```
def playlist_fallback(a)
  fallback([a, (playlist("list.m3u"):source)])
end

s = playlist_fallback(input.harbor("mount"))
```

```
› liquidsoap /tmp/record.liq -h playlist_fallback

No documentation available.

Type: (source(audio='a, video='b, midi='c)) -> source(audio='a,
video='b, midi='c)
```

A look forward

More simply simple…

A look forward

More simply simple…

```
let json.parse ({
  name,
  version,
  scripts = {
    test
  }
} : {
  name: string,
  version: string,
  scripts: {
    test: string
  }
}) = file.contents("/path/to/package.json")
```

A look forward

…with revisited complexity

- Continue moving functionalities out of the OCaml core
- Core release vs. standard library release?
- Improve type checker whenever possible
- Refactor, modernize internal implementations
- Frames: breaks vs. track marks, immutable content vs. content copy
- Streaming model: code complexity (clocks, source readiness)
- Prepare for multicore OCaml!