Showergel

Release 0.3.0

Martin Kirchgessner

Aug 07, 2023

CONTENTS:

Showergel is made to live aside Liquidsoap: while a Liquidsoap script creates a radio stream, Showergel provides complementary features like playlist logging or occasional scheduling, with a Web interface. It is made to run on a Linux box (with systemd) dedicated to your radio stream.

Note: With Showergel you still have to write/tune the Liquidsoap script that will fit your radio. That is, the set of sources (playlist, input.harbor, …) and operators (random, switch, fallback, …) that fits your programs and schedule. This documentation provides a *Showergel's quickstart.liq* script, covering Liquidsoap's basics and Showergel's integration.

CHAPTER

ONE

WHAT IS SHOWERGEL ?

Showergel is made for community and benevolent radios, so it is meant to stay small, simple, and as self-contained as possible. Showergel is made for Liquidsoap users, programming their own radio scripts. If you need a Web interface to schedule your programs without writing a line of code, you might consider other free softwares[1].

This page discusses Showergel's design and components.

1.1 What's inside Showergel - and what's not

The REST/Web interface is served by the Bottle framework, because it's enough and allows keeping everything in a single process.

Showergel's data is stored in SQLite because it lets us store everything in a single, local file.

Scheduling is delegated to APScheduler, who also needs SQLAlchemy to access SQLite, so we use SQLAlchemy too (also because it's a great ORM).

Showergel does not hold your music and shows collection. For that matter we suggest Beets. You can find examples of its integration with Liquidsoap in Liquidsoap's documentation.

Showergel is here to schedule what cannot be implemented with Liquidsoap's scheduling functions, ie. *occasional* actions/programs. Liquidsoap's switch is a better fit for regular programs.

1.2 Predicting Liquidsoap is hard

Showergel is meant to let you do anything allowed by Liquidsoap. This implies that Showergel can't predict Liquidsoap's playlist.

Note: This property of the Showergel-Liquidsoap couple might be the most surprising if you've ever worked with radio automation software: *none of these two can predict what will be played at some point in time*. Liquidsoap does not pre-compute its playlist, and neither does Showergel.

A more theoretical explanation is that Liquidsoap lets you write a stream generation function. Predicting what will play out would require reverse-engineering users' scripts - automatically! On its side, Showergel has a scheduler so it can predict what it will do, but that's not enough. For example, even if you have scheduled remote_radio.start at some time, the remote stream might not be available right away: what will play instead depends on what's in the Liquidsoap script. As a user you're responsible not only for preparing fallback content for such case, but also its conditions of appearance in the stream (using fallback, typically). Some pieces of Liquidsoap scripts may also block Showergel: for example, putting a *buffer* before your outputs might block Showergel's ability to display the current track's remaining time.

[1] among Liquidsoap-based solutions, we can cite LibreTime, AzuraCast, CrazyArms, or AirTime. See also Rivendell.

Most automation softwares relying on Liquidsoap try to hide this. Indeed it is much simpler to let the user click on a weekly calendar. But, in that setting, choosing an automation software forces you to abide by its scheduling logic. Instead, we embrace Liquidsoap's DIY possibilities and make all sources/outputs *you* created visible in Showergel.

1.3 Time

Showergel needs time zones to prevent a potential mismatch between the front and back ends: Javascript's `Date` is always bound to a time zone. Internally, time is represented on the UTC time zone because SQLAlchemy's datetime for SQLite does not stores the time zone. We use Arrow for data parsing and time zones conversions.

Showergel always expects and outputs times with time zone information (using the ISO 8601 format), But Liquidsoap is not timezone-aware: Liquidsoap endpoints or commands are converting time to the local time zone (guessed from the system's environment).

1.4 Events

Showergel proposes events to be set-up at milisecond precision (although the GUI currently forces the milisecond term at zero). Events are unrolled according to the schedule's order, _never_ in parallel. If an event might take a significant time (typically because it triggers a download) note that it might delay following events. This order is enforced because Showergel can't decide if events are dependent or not.

CHAPTER

TWO

INSTALLING SHOWERGEL

Installing Showergel requires

- a Linux box relying on systemd (ie. a recent mainstream distribution),
- Python, at least version 3.7,
- pip, check it is available by calling `pip --version` or `pip3 --version`
- a running Liquidsoap radio - version 2.x is Showergel's best friend.

Note: We assume you already know the basics of Liquidsoap and its script language. If you have never played with only Liquidsoap, we advise you read at least its quick start guide. Then you might use our *Showergel's quickstart.liq* script.

Install Showergel by running `pip install showergel` (maybe replace `pip` by `pip3`).

2.1 Create an instance with the interactive installer

Run the interactive installer by calling `showergel_install`. It will explain on the terminal what is happening and what to do from here. If you stick to defaults, the instance's basename will be `radio`, so the installer will:

- create a database (`radio.db`) and a configuration file (`radio.toml`) in the current directory,
- create a systemd user service,
- enable the service and systemd's lingering so Showergel will start automatically at boot time.

The installer allows you to create another systemd user service, for your Liquidsoap script. This is recommended, because systemd will automatically launch both Showergel and Liquidsoap. Systemd will also restart them when a crash happens - which is great for your radio's uptime. If you do so, the installer creates two systemd services with the same basename: for example, `radio_gel` (Showergel service associated to `radio`) and `radio_soap` (wrapper for the Liquidsoap script you provided for `radio`).

If you choose to not create a service, you will have to (re)start Showergel manually by calling `showergel radio.toml`.

Before exiting, the installer gives a recap of its actions.

Note: Please pay attention to the installer's recap: it lists how to run/stop Showergel and gives pointers to important files. Read it twice and copy this information safely.

Unless you configured another port, Showergel will be available at http://localhost:2345/. Showergel's and Liquidsoap's configurations are coupled. In perticular, they should define which port/URL should be used to contact each other: see *Configuring Showergel* and *Showergel integration in Liquidsoap scripts*.

If you installed Showergel as a system service, don't forget to restart showergel's service after editing its configuration file. If you installed your Liquidsoap script as a system service, restart this service after editing the script - and, of course, after running `liquidsoap --check my_script.liq` ! In both cases, log files are in the same folder as instance's other files.

2.2 Running multiple Liquidsoap scripts

An instance of the `showergel` program is the companion of *one* Liquidsoap stream. The two programs will communicate with each other to produce and follow this stream. An instance relies on a configuration file (`something.toml`) and a database (an SQLite file). It can be installed as a system service, named after information you provide at set-up time.

If you are running multiple Liquidsoap streams on the same machine, you'll have to set up one Showergel instance per stream. In that case,

- each instance should have a different name
- each instance should run on a different port number

We advise you to put all instance's files (configuration, DB, logs) in the same folder as the corresponding `.liq` file. When running multiple Liquidsoaps, create a folder per instance.

2.3 Install for back-end development

Depencencies, installation and packing is done by Poetry. Once Poetry is installed, create a Python3 environment, activate it, and run `poetry install` from a clone of Showergel's repository.

When developping, your Liquidsoap script and Showergel should be launched manually. Run `showergel_install --dev` to create an empty database (`showergel.db`) and a basic configuration file (`showergel.toml`) in the current folder. Read (and edit, maybe) `showergel.toml`, launch Liqudisoap, then run `showergel showergel.toml`. You'll likely want to enable the detailed log by setting `level=DEBUG` in the `logger_root` section of the toml file.

Test with `pytest`. See also *Packaging and publishing Showergel*.

2.4 Install for front-end development

The front-end is written in JavaScript packed with Yarn, with VueJS's single-file components. We use the Bulma CSS Framework.

To modify the front-end, you must beforehand install Yarn and Vue_CLI, then run `yarn install` from the repository root. Start the live-building server with `yarn serve`. If you don't have time to install the whole back-end, you can call the demo app by creating a `front/.env` file that contains:

```
VUE_APP_BACKEND_URL=https://showergel.fly.dev/
```

Similarly, a fully-working HTML/JS/CSS build is included in this repository, so one doesn't have to install `yarn` and Vue while working on the back-end. Those files are generated by `yarn build`.

Note: Please do **not** commit modifications in the `/showergel/www/` folder. In order to avoid complex and useless conflicts, commits concerning this folder should only happen on the `main` branch.

2.5 Deploy the demo to Fly.io

In demo mode, the application starts by putting fake data in the database. It's enabled by putting `demo = True` in the configuration file's `[listen]` section.

Source repository includes a configuration that can be pushed to Fly (thanks to `fly.toml`, `Procfile` and `procfile.toml`), so right after cloning you can `flyctl launch` (only the first time) then `fly deploy`.

We might need to update `requirements.txt` from time to time:

```
poetry export --with dev --without-hashes -f requirements.txt --output requirements.txt
```

`--with dev` is here because `requirements.txt` is also used by ReadTheDocs to compile the present documentation, which requires a Sphinx extension.

CHAPTER THREE

CONFIGURING SHOWERGEL

A configuration file is needed to start Showergel. It tells the program what to do, where data should be written, how to contact Liquidsoap, etc. Showergel's installer creates a minimal configuration, including comments if you're hurried to tweak it. This section will tell you all configuration details and their implications, if any.

3.1 What's a `.toml` configuration file

Showergel uses the TOML format. If you have never encountered it, let's start by describing *how* this configuration is written. It is a text file, encoded in UTF-8, that you can edit with the same program as your Liquidsoap script.

Configuration properties have a *key* (usually a word) and a *value* (a character string, a list, number, . . .). Properties are grouped by sections, like `[listen]` or `[db.sqlalchemy]`.

Do not mix properties between sections: the program will look for a property below one perticular section. *Comments* are notes that will be ignored by the program: they start with a `#`.

A TOML file looks like this:

```
[section]
property = "value"
somelist = ["foo", "bar", "baz"]
somenumber = 9876

[another_section]
# this line starts with a sharp : programs will ignore it
# you can use comments to note why a value is defined
# or to put aside old configuration values
debug = true  # comments can be at end of lines too
```

3.2 `[db.sqlalchemy]`

This section should at least include a path to the instance's database file:

```
url = "sqlite:////home/me/path/to/showergel.db"
```

Yes, that's four `/`. Use only 3 if you prefer a relative path (relative to Showergel's working directory), for example `sqlite:///showergel.db`.

In demo mode, you might want to set `sqlite:///:memory:` to avoid writing a file.

If you have a DB server at hand, you can use it for Showergel too. You will have to install the DBAPI package youself, and refer to SQLAlchemy's documentation for the URL format - see pages about PostgreSQL or MySQL.

When debugging, you may display DB requests in Showergel's log by setting the property `echo = true`.

3.3 [interface]

You may change the name displayed in the interface's left bar:

```
name = "Showergel Radio 98.7 FM"
```

3.4 [listen]

This defines Showergel's address for your browser and liquidsoap. For example:

```
address = "localhost"
port = 2345
```

Using these (default) values, Showergel's interface will be available at the URL http://localhost:2345/. If you are running multiple instances of Showergel on the same machine, be careful to set a different `port` for each one.

For some features (user authentication, metadata logging), Showergel's URL must be set in your Liquidsoap script too. See *Showergel integration in Liquidsoap scripts*.

Warning: The `address` property may be an IP address. If you change the address, ensure it is only accessible on a private network.

To have a more detailed server log you can add `debug = true`.

3.5 [liquidsoap]

This section defines how Showgel can contact Liquidsoap:

```
method = "telnet"
host = "localhost"
port = 1234
```

This should match Liquidsoap's telnet parameters - see *Showergel integration in Liquidsoap scripts*.

Other values can be set as `method`:

- `none` if you don't want to enable Showergel's "current track" display.
- `demo` will simulate a Liquidsoap connection. In that case `host` and `port` are ignored. This is used by Showergel's online demo.
- anything else, or if the parameter is missing, will simulate a Liquidsoap connection by generating different data each time it's called. This should only be used for Showergel's unit tests.

You can also add a line stating `ouput = "identifier"` to force Showergel to get its "Now playing" information from the output having `id="identifier"` in your Liquidsoap script (see *Display/skip current track*).

3.6 [metadata_log]

This section configures how Showergel stores tracks' metadata. It may contain `extra_fields`: a list of metadata fields that should be stored, when available.

```
[metadata_log]
extra_fields = [
    "genre",
    "language",
    "year",
    "track*",
]
```

A `*` in the field name represents any characters or nothing. In the example above, `track*` will match `track`, but also `track_number` or `tracktotal`. Field names are stored as they are, so Showergel will store `track_number` or `tracktotal`.

3.7 Logging configuration

This follows Python's configuration dictionary schema for logging.

CHAPTER

FOUR

SHOWERGEL INTEGRATION IN LIQUIDSOAP SCRIPTS

Most Showergel features require your Liquidsoap script to implement a link with Showergel. This page presents these links, as seen from Liquidsoap. **All examples below assume your Liquidsoap script starts with a variable defining Showergel's URL, as:**

```
SHOWERGEL = "http://localhost:2345"
```

4.1 Showergel's quickstart.liq

We showcase the complete Showergel integration in a "quick-start script" you can download `here`. It is tested against Liquidsoap 2.1.x. This `.liq` file defines typical radio sources and is heavily commented: you can use it to start your first stream, or pick portions that would improve your existing script (it also contain a few Liquidsoap tricks!).

Warning: Showergel and Liquidsoap are **not** secured against malicious access. In the worst case, this could result in innapropriate control of your radio's program. Please isolate the machine running Showergel on both physical and network levels.

Sections below discuss implementation details on integrating each Showergel feature.

4.2 Display/skip current track

You need to enable Liquidsoap's telnet server. For example:

```
settings.server.telnet.set(true)
settings.server.telnet.bind_addr.set("127.0.0.1")
settings.server.telnet.port.set(1234)
```

`127.0.0.1` is the IP address of `localhost`. The `port` value should match the one in Showergel's configuration's *[liquidsoap]* section. If you are running multiple instances of Liquidsoap on the same machine, be careful to set a different `port` for each one.

Warning: Do not use a public IP address as `bind_addr`. This would open your Liquidsoap instance to the Internet, and someone might connect and mess up your programs.

If your script has multiple outputs, ensure the main one has an identifier by setting its `id="identifier"` parameter. This identifier should be copied as `output` in the *[liquidsoap]* section.

4.3 Logging metadata

You need to define a function that will post metadata to Showergel:

```
def post_to_showergel(md)
    response = http.post("#{SHOWERGEL}/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", "Error while posting metadata to Showergel: #{response} #
↪{response.status_code} #{response.status_message}")
    end
end

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

We advise to plug the function with source.on_metadata, but source.on_track may work too.

Warning: Many Liquidsoap operators repeat previous track's metadata when switching from a source to a another. This concerns operators whose `replay_metadata` parameter defaults to `true`. This often yields duplicate entries in the log, although Showergel does its best to ignore duplicates.

In other words, if you get duplicates in the metadata log, you might avoid them by adding `replay_metadata=false` to your `fallback/random/rotate/switch` operators. Especially if they're track-insensitive.

4.4 Authenticating users on harbor

Liquidsoap's input.harbor can require authentication by giving `user` and `password` parameters. But this implies

- storing the clear password in your `.liq` script
- sharing those credentials
- restarting the Liquidsoap stream when you want to update those credentials

This is unconvenient and not enough secured.

Instead, you can rely on Showergel to hold the list of users and their passwords - encrypted. Then you will be able to add/edit crendentials from Showergel's web interface. This method requires creating an authentication function (in your `.liq`) passed to `intput.harbor`'s `auth` parameter (instead of `user` and `password`).

This function can be written as:

```
def auth_function(login) =
    response = http.post("#{SHOWERGEL}/login",
        headers=[("Content-Type", "application/json")],
        data=json.stringify(login)
    )
    if response.status_code == 200 then
        log("Access granted to #{login.user}")
```

(continues on next page)

(continued from previous page)

```
        true
    else
        log("Access denied to #{login.user}")
        false
    end
end

harbor = input.harbor(auth=auth_function, ...
```

CHAPTER

FIVE

RESTFUL INTERFACE

`GET /metadata_log`

Without parameters, `GET /metadata_log` returns the 10 most recent metadata items played. The following query parameters can be provided

Query Parameters

- **`start`** (*`string`*) – (ISO 8601 format) define an inclusive query interval
- **`end`** (*`string`*) – define an inclusive query interval
- **`limit`** (*`int`*) – restricts the number of results (and defaults to 10). It is ignored if `start` and end are provided.
- **`chronological`** (*`bool`*) – may be set to anything non-empty (use `1` or `true`), otherwise results are sorted recent first. Doesn't affect the interpretation of `start` and `end`.

Response JSON Object

- **`metadata_log`** – logged metadata matching the query parameters.

`POST /metadata_log`

Should be called by Liquidsoap to save tracks' metadata. See *Logging metadata*.

`POST /login`

Should be called by Liquidsoap to authenticate harbor users. It returns the matched user information as a JSONobject, or a 404 error. See *Authenticating users on harbor*.

`GET /users`

Response JSON Array of Objects

- **`username`** –
- **`created_at`** –
- **`modified_at`** –

`PUT /users`

User registration

Request JSON Object

- **`username`** –
- **`password`** –

Response JSON Object

- **`username`** –

- `created_at` –
- `modified_at` –

`POST /users/`(*username*)

Update user attributes

Request JSON Object

- `password` – optional

Response JSON Object

- `username` –
- `created_at` –
- `modified_at` –

`DELETE /users/`(*username*)

Delete someone's user account

Response JSON Object

- `deleted` – username

`GET /live`

The returned JSON object might contain many more fields, depending on what's in the current track's metadata. You can reasonably expect `title` and `artist`.

Response JSON Object

- `source` – name of the currently playing source
- `on_air` – current track start time
- `status` – status of the current source ("playing" or "connected to …")
- `server_time` – server's datetime
- `remaining` – *maybe* remaining duration of current source, in seconds

`GET /parameters`

This returns values from the `[interface]` section of the configuration file.

Response JSON Object

- `name` – instance name (appears as interface's title)
- `version` – showergel's version
- `commands` – list of available Liquidsoap commands

`DELETE /live`

Skips current track: this sends a skip command to the first Liquidsoap output.

`PUT /schedule`

Schedule event creation. Note that at most one event can be registered at a given date.

Request JSON Object

- `command` – Liquidsoap command
- `when` – Event time (ISO 8601 with time zone info)

Response JSON Object

- `event_id` – created event's ID

`GET /schedule`

List upcoming events. For each event, expect:

- `event_id`
- `when` (ISO 8601)
- `command`

Response JSON Object

- **`schedule`** – list of scheduled events

`OPTIONS /(:re:.*)`

this only ensures we support `OPTIONS` requests (instead of returning 405). CORS headers will be returned iff `send_cors` is enabled.

CHAPTER

SIX

PACKAGING AND PUBLISHING SHOWERGEL

This is the maintainer's check-list.

6.1 Pre-publication check-list

- Run `poetry update`
- Run `pytest` - all tests should pass
- From the `front` folder, run `yarn upgrade`.

If this is done right after a feature freeze, the upcoming version should have a pre-release marker: `-alpha.0`. Run this pre-release for a few days in a realistic setup.

Build and check the documentation: `cd docs; make html`.

6.2 Update version markers

- `version` in `pyproject.toml`
- `version` in `front/package.json`
- `release` in `docs/conf.py`

6.3 Package

- From the `front` folder, run `yarn build`
- `poetry export --with dev --without-hashes -f requirements.txt --output requirements.txt`
- `git commit,push,tag`
- `poetry build`
- `poetry publish`

Showergel is released under the GPL3 license.

CHAPTER SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

HTTP ROUTING TABLE