Streamer **5** Software Development

[The fundament - MVC 2](#_Toc210211533)

[Model 2](#_Toc210211534)

[View 2](#_Toc210211535)

[Control 2](#_Toc210211536)

[Issues to be addressed 3](#_Toc210211537)

[View - Commands and Information 4](#_Toc210211538)

[Start page 4](#_Toc210211539)

[Start Page template 6](#_Toc210211540)

[Open start page call behaviour 6](#_Toc210211541)

[Bluetooth Page 7](#_Toc210211542)

[Bluetooth page template 8](#_Toc210211543)

[Network Page 9](#_Toc210211544)

[Network Page template 9](#_Toc210211545)

[About page 10](#_Toc210211546)

[Notes on HTML 11](#_Toc210211547)

[Model - states implementation 12](#_Toc210211548)

[States 12](#_Toc210211549)

[Detect Spotify and Airplay 13](#_Toc210211550)

[Detect Bluetooth audio streaming 14](#_Toc210211551)

[Detect wireless and wired and Internet access. 14](#_Toc210211552)

[Detect Bluetooth devices connects and disconnects 14](#_Toc210211553)

[Timers, write and read intervals for Bluetooth detection 15](#_Toc210211554)

[When is Bluetooth streaming to the Bluetooth speaker? 15](#_Toc210211555)

[Control - implementation 16](#_Toc210211556)

[Boot phase 16](#_Toc210211557)

[Streaming services control 16](#_Toc210211558)

[Bluetooth audio streaming 17](#_Toc210211559)

[More on detection of Bluetooth speaker 17](#_Toc210211560)

[Best practise setup for frontend 19](#_Toc210211561)

[Appendix: Essential Linux CLI 20](#_Toc210211562)

[Amplifier 20](#_Toc210211563)

[Bluetooth connections 20](#_Toc210211564)

[Bluetooth sink or source 21](#_Toc210211565)

[Bluetooth scan for sinks 21](#_Toc210211566)

[Wireless connections 23](#_Toc210211567)

[Access Point – hotspot 24](#_Toc210211568)

[alsa and bluealsa -analogue amplifier 25](#_Toc210211569)

[Internet 25](#_Toc210211570)

[Wired connection 26](#_Toc210211571)

[Important internal data 26](#_Toc210211572)

[Cleansing 26](#_Toc210211573)

[Appendix: Legacy Player 4.016 27](#_Toc210211574)

[Developer's Notice to Reader 27](#_Toc210211575)

[Player services control was loops 29](#_Toc210211576)

[Some Player pages 30](#_Toc210211577)

# The fundament - MVC

## Model

### Connections

* Wired, Ethernet LAN cable.
* Wi-Fi, connected to Wi-Fi network.
* Bluetooth, point-to-point connections.
* AP, Streamer’s own Wi-Fi network (hotspot).
* Internet access, indirectly via wired or wireless LANs.

### Services

* Airplay, version 1 and 2.
* Spotify.
* Bluetooth audio stream.

### Output

* Amplifier, four binding posts with analogue sound.
* Volume of amplifier.
* One Bluetooth speaker and audio stream.

### Input

* Apps as sources.
* Web browser as user interface.

## View

Web pages that will:

* Manage and show connections.
* Show service status.
* Manage and show outputs.
* Manage and show Streamer system.

## Control

Start Streamer.

Execute commands from web pages.

Deliver connection states.

Deliver output states.

Detect state changes.

Reset Streamer.

Shutdown Streamer.

### Main control actions

Watch - Detect - Identify Change - Act on Change - [Render].

Page event - keep track of connected pages.

User event - command calls.

Start and stop of Streamer.

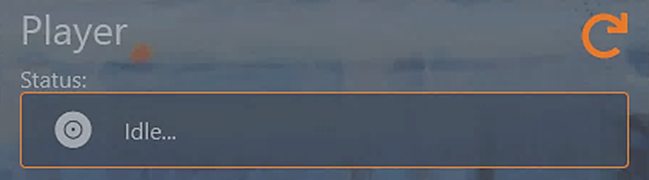
# Issues to be addressed

1. In Player software there are many events, loops watching for events and tokens floating around. They have to be reduced.
2. All different states are stored in Player. Keeping track and keep it consistent is tedious.
3. A simpler and better-defined protocol between front and backend.
4. The service bluetoothctl is a catastrophe. It has to be managed in another way. There are so many writings of information. Maybe 3 - 5 per minute. Refactor this!
5. There are no hooks provided for bluetooth events. What is the best way to figure out the status of bluetooth streaming?
6. Horrible dependencies with mpd and upnpa – will be removed.  
   mpd and upnpa is not provided anymore

# View - Commands and Information

## Start page

#### Streaming information



**Command** – Render, Spotify stop, Airplay stop, Bluetooth stop

**Streaming status**: Idle… | Spotify | Airplay | Streaming Bluetooth Audio

**Dynamic frame**: external events change the content. Persistent.

#### Volume information



**Command** - Volume

**Volume**: integer 0 – 100

**Static frame**: only the user can use this frame. Persistent.

#### Main commands



**Commands** - Bluetooth, Wireless, [vacant – since USB play is deprecated], Settings

Static frame: persistent

#### Bluetooth information

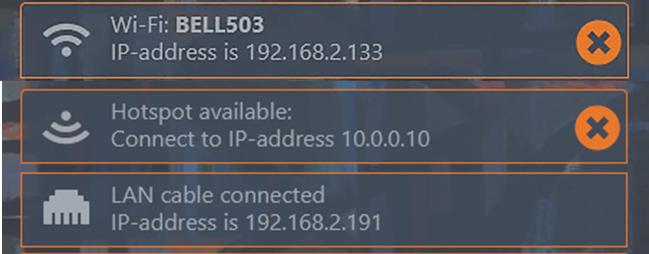


**Commands** - Bluetooth off, Disconnect speaker

**Bluetooth status**: yes | no, **Bluetooth Speaker**: name. The amplifier status is an appendix to the connected speaker frame as a reminder and it is rendered at the same time, but in its own frame.

**Dynamic frames**: events on the Bluetooth Page change these frames.

#### Network connection information



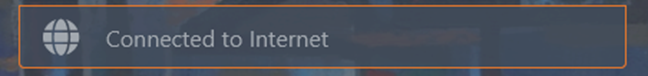
**Commands** - Wireless disconnect, AP down

**Wireless status**: [ssid , ipaddress] | []

**AP status**: up | down, **Wired status**: ip address | [].

**Dynamic frames**: external events and events on the Wi-Fi page change these frames. Lan event is external and the frame is persistent.

#### Internet access

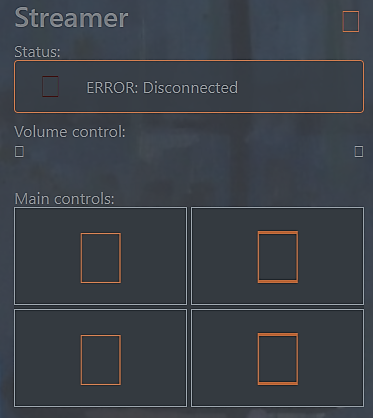


**Commands** – Check Internet

**Internet access**: connected | disconnected

**Dynamic frame**: external events change this frame. Persistent.

## Start Page template



streamer.html

## Open start page call behaviour

### IP address used - or URL used

DOM ready generates the first rendering.

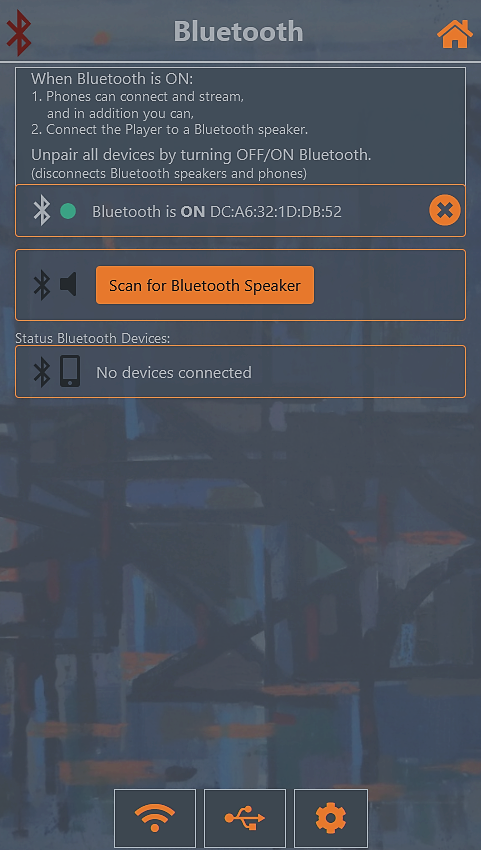
**Note**: just entering “1” in the browser address box and the auto fill-in of “191.168.0.131” shows up, that causes an immediate first render. A second rendering comes after CR. Typing streamer.local is the same.

### After reconnect

Render happens when the last socket emit after disconnect was discovered by the page eventually reaches the restarted/reconnected Streamer backend.

## Bluetooth Page

#### Bluetooth information

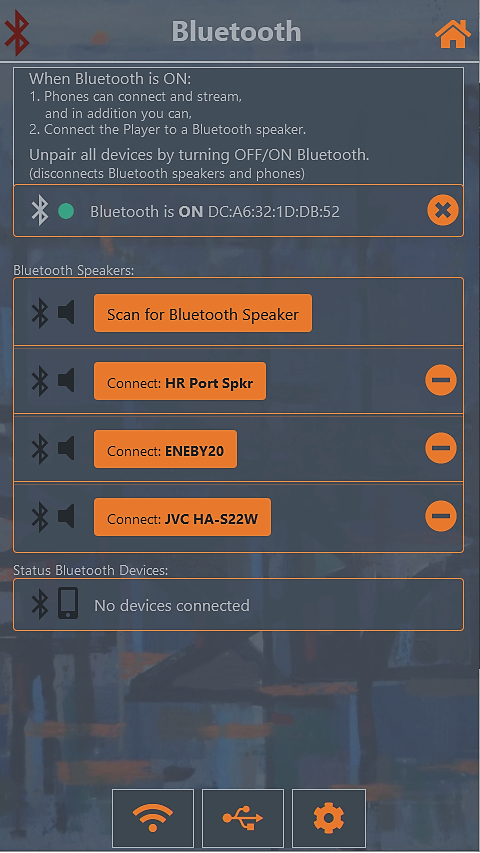


**Commands** - Bluetooth on or off

**Bluetooth status**: yes | no Note: BD-address will not be displayed.

This frame is persistent.

#### Bluetooth Speakers information



**Commands** - Scan for speaker, disconnect speaker, unpair speakers

**Connected Bluetooth speaker**: name | []

**Bluetooth speaker list**: [name,…] | []

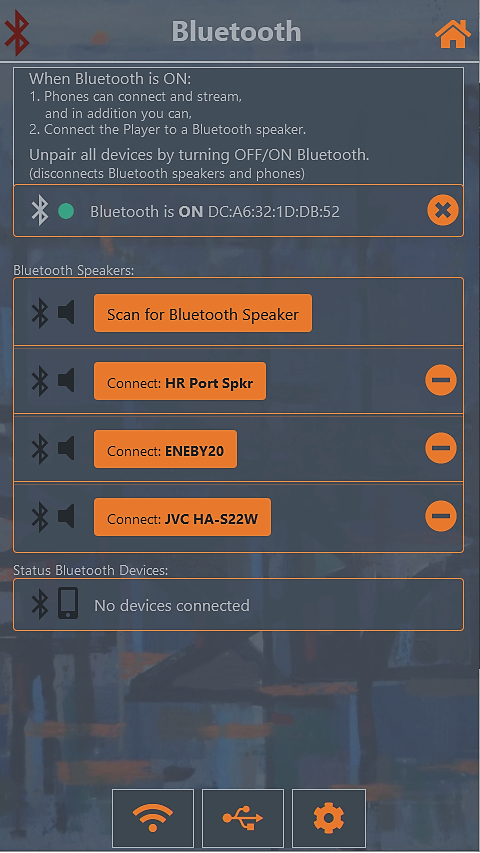
Frame for scanning for speakers is persistent.

**Dynamic frames**: Connect/disconnect, remove speakers and events (external and internal).

**List of speakers**: names | [], **Chosen speaker:** name | []

**Modal input frame**: for choosing a speaker.

#### Bluetooth devices information



**Command** - disconnect devices, unpair devices

**Connected Bluetooth devices**: [name, … | []

**Paired only Bluetooth devices**: [name, … | []

**Dynamic frames**: Connect/disconnect, pair/unpair devices and events (external and internal).



**Commands** - open network page /open start page /open setting.

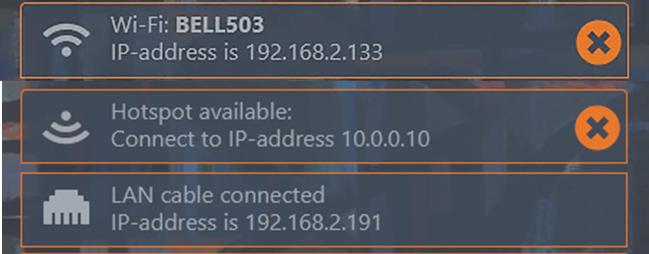
## Bluetooth page template



streamerbluetooth.html

## Network Page

### Network information



**Commands** - Wireless connect/disconnect, AP up/down

**Wireless status**: [ssid , ipaddress] | []

**AP status**: up | down, **Wired status**: ip address | [].

**Dynamic frames**: Wired events (external) and events for connect/disconnect t Wi-Fi and turn AP up or down. All frames are persistent.

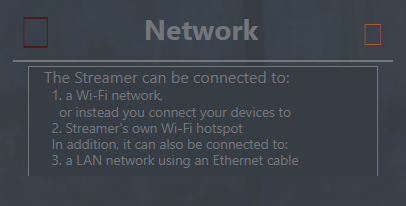
**List of Wi-Fis**: ssids | [] ssid |[], **Chosen Wi-Fi:** ssid | [], **Connect to the Wi-Fi**: ssid, password | [] []

**Modal input frames**: for choosing a Wi-Fi and then enter password.

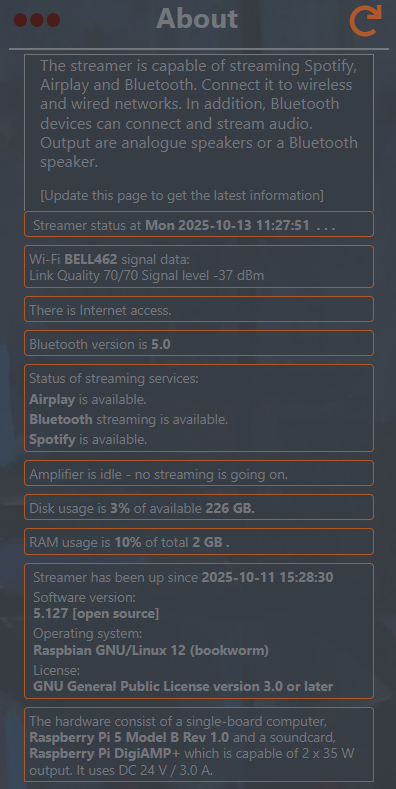


**Commands** - open bluetooth page / open start page / open setting.

## Network Page template



## About page



## Notes on HTML

### Notyf

Notyf out of the box doesn’t work. There is a conflict between ES-module and the Common.js syntax. More detailed, the line on top Object.defineProperty(exports, '\_\_esModule', { value: true }); is a common pattern seen in JavaScript files, particularly those compiled from TypeScript or transpiled by tools like Babel. Its purpose is to mark the current module as an ES module (ECMAScript module). That did not work.

The insight is that there are no exports needed since in Streamer Notyf only lives on the web pages (unlike Player).

### Fixes

Remove ‘use strict’

Remove Object.defineProperty(exports, '\_\_esModule', { value: true });

Remove all references to export, there are five of them at the end (lines 447 - 455)

### Add declarations

var NotyfArrayEvent = [];

var NotyfEvent = [];

# Model - states implementation

## States

Defined states and their defined values:

1. Streaming services: Idle, Spotify, Bluetooth, Airplay
2. Amplifier volume: 0-100%
3. Bluetooth service: yes or no
4. Connected Bluetooth speaker: name or ""
5. Wi-FI: [ssid, ip-addr] or []
6. AP: up or down
7. LAN connection: IP address or ""
8. Internet: access or ""
9. Trusted Bluetooth sinks: [name,..] or []
10. Connected Bluetooth sources: [name,..] or []
11. Paired Bluetooth sources: [name,..] or []

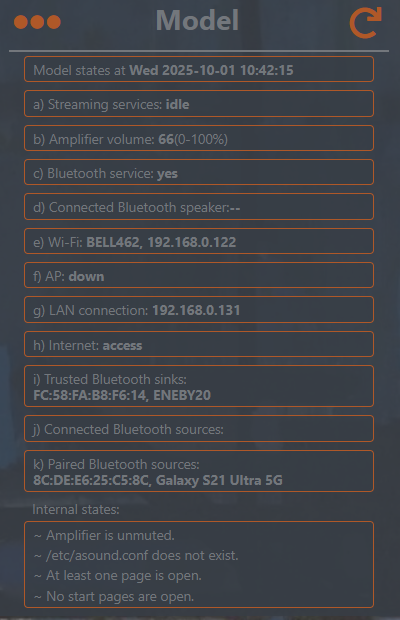
Internal states not shown on the web pages:

Key internal states that have impact of Control:

* Are there any web pages socket.io connected (open).
* Is the amplifier muted.
* Is the /etc/asound.conf file in place or not

Other internal states which are needed:

* Accessible Wi-Fi ssids.
* Is there a connected Bluetooth devices.
* Bluetooth devices by name and its BD-address (connected, trusted or paired).



When there is any dynamic behaviour on a web page it is necessary to know if there are any web pages connected. In order to keep track of open pages they are marked up in the socket.data object. The object stores arbitrary data and extra attributes, called for example ‘startpage’. If the page is the start page it is set to ‘true’, otherwise ‘undefined’. This attribute is checked and used by mod.isOpenStartPage(). References: <https://socket.io/docs/v4/server-instance/#utility-methods>

## Detect Spotify and Airplay

The librespot and shairport-sync streaming services write on file /var/log/streamsensor.log when it starts and when it stops. Follows the format used in Player (see [appendix](#_Player_services_control)). The format is basically: 'idle:stop', '<service>:start', '<service>:stop'

## Detect Bluetooth audio streaming

There are no hooks that can generate start and stop events for bluetooth streaming. Streaming from a source (phone) to Streamer (sink) or streaming from Streamer (source) to a bluetooth speaker (sink). Or both combined. These are the key parts involved:

1. bluetooth bluetooth services managed by bluetoothctl,  
   used for connections, turned on/off with rfkill block bluetooth.
2. bluealsa bluetooth audio streaming to alsa with different codecs,  
   always on. Check bluetooth uses the alsa or if it is idle.
3. bluealsa-aplay bluetooth streaming using alsa or a bluetooth speaker,  
   the actual bluetooth streaming audio service, on or off, but needs restart when the speaker connects or disconnects. In addition, the asound.conf file is needed for the speaker

## Detect wireless and wired and Internet access.

If there is at least one page open and the streamer is connected to a Wi-Fi - it needs to be monitored.

If there is at least one page open the LAN cable has to be detected as attached or not attached.

Internet access is checked by “pinging” the following web pages in shuffled[[1]](#footnote-1) order:

"https://www.google.com", "https://www.youtube.com", "https://www.facebook.com", "https://www.weather.com","https://www.bing.com", "https://www.instagram.com", "https://www.wikipedia.org/", “<https://www.reddit.com>”

… as soon a web site responds the check is true.

## Detect Bluetooth devices connects and disconnects

### Bluetooth source devices

Check with the CLI bluetoothctl.

### Bluetooth speakers

Key criteria for managing bluetooth speakers connects and disconnects:

1. muted amplifier - when the bluetooth speaker is connected
2. the connected speaker itself
3. at least one trusted speaker
4. the /etc/asound.conf file - when the bluetooth speaker is connected

If there is at least one page open:

Check if the amplifier is muted.

Monitor the connected bluetooth speaker - check disconnect.

Monitor the connected source device - check disconnected

## Timers, write and read intervals for event detection

The defining timer 0 is the node-watch, watcherInterval, read interval every 2,001[[2]](#footnote-2) seconds.

'idle:stop', '<service>:start', '<service>:stop' file: /var/log/streamsensor.log

Bluetooth timer 1: intervals between writes of the names of file descriptors[[3]](#footnote-3) for open files of bluealsa-aplay: btDetectInterval = watcherInterval \* 1,1

…writes to /var/log/bluetoothdetect.log using the script fdinfobluesalsa.sh provided with the bluealsa-aplay process id as a parameter.

Bluetooth timer 2: read interval of the file written to by timer 1 and analyses the patterns.

btLoopInterval = btDetectInterval \* 1,27

…always reads the number of names written to /var/log/bluetoothdetect.log

…writes at events 'bt:start', 'bt:stop' to file: /var/log/streamsensor.log

## When is Bluetooth streaming to the Bluetooth speaker?

Streaming is going on if the number of file descriptor names are 11.

When there is no streaming and no bluetooth sources connected there are seven file descriptor names and when there is no bluetooth devices connected at all there are only five,

Example: using CLI: readlink /proc/*<pid>/*fd/\*

|  |  |
| --- | --- |
| /dev/null | /dev/null |
| socket:[662584] | socket:[662584] |
| pipe:[1374960] | socket:[662584] |
| socket:[1374962] | anon\_inode:[eventfd] |
| socket:[662584] | socket:[660127] |
| anon\_inode:[eventfd] | pipe:[814868] |
| socket:[660127] | socket:[814870] |
| socket:[1377312] |
| 7anon\_inode:[eventfd] |
| pipe:[814868] |
| socket:[814870] |

[count: **11**] [count: **7**]

## Loop conditional statements

Loops are used to monitor the model of the Streamer and detect changes. If there are pages open and Bluetooth is on there are 115 linux calls per minute. If no pages and no Bluetooth there are 72 calls per minute for loop conditional statements.

Get the calls:

journalctl | grep COMMAND

Check if the amplifier is unmuted:

time /usr/bin/amixer -c 0 get Digital -- 0m0.003s

Check if there is Bluetooth service running

time /usr/sbin/rfkill list -- 0m0.002s

Check Bluetooth devices:

time /usr/bin/bluetoothctl devices Connected --0m0.014s

time /usr/bin/bluetoothctl devices Trusted -- 0m0.013s

Check LAN cable:

time /usr/sbin/ip -f inet -o addr show eth0 -- 0m0.002

Check Bluetooth streaming to Streamer and alsa:

time pmap -q -A ffff1000,ffff1100 20573 -- 0m0.007s

time cat /proc/asound/card0/pcm0p/sub0/status -- 0m0.001s

Check Bluetooth streaming to Bluetooth speaker:

time systemctl status bluealsa-aplay -- 0m0.018s

time /streamer/fdinfobluesalsa.sh 20573 -- 0m0.002s

time echo "idle" > /var/log/bluetoothdetect.log -- 0m0.000s

time find /etc/asound.conf -- 0m0.001s

## Loop structure

pollStreamingStatus -- changes of file /var/log/streamsensor.log

[watcher()]

bluetoothLoop -- start bluetooth detect or not

[bluetoothStatus() == "yes"]

then

detectAlsaBluetoothStreaming -- detect bluetooth streaming to alsa

[areConnectedBluetoothSources() === true]

[alsa.isAmpMuted() === false] + GV whatStreams

or then…

detectBluetoohSpeakerStreaming -- detect bluetooth streaming to speaker

[alsa.isAmpMuted() === true]

bluetoothDetect --is bluetooth streaming detected in file system

[alsa.isAmpMuted() === true] + GV whatStreams

# Control - implementation

## Boot phase

Streamer is now booting: . . . . . . . . . . . . . . . . . . . . .

Version: 5.121 [open source]

Copyright 2025 by Retro Audiophile Designs, license GPL 3.0 or later

10:43:05:555 ====================================================

10:43:05:567 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

10:43:05:567 Boot Phase: time critical------------[1] started

10:43:05:605 aux: partition /dev/sda2 --> read and write mode

10:43:06:139 detect ctl: log files reset

10:43:06:156 alsa: Read file, start up value returned is 65

find: ‘/etc/asound.conf’: No such file or directory

10:43:06:243 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

10:43:06:243 Boot Phase: resets-------------------[2] started

find: ‘/etc/asound.conf’: No such file or directory

Error: 'Hotspot-1' is not an active connection.

Error: no active connection provided.

10:43:06:434 Wireless is UP... AP goes down.

10:43:06:435 view: Webserver and io.socket up...

10:43:06:667 Bluetooth started...

10:43:06:686 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

10:43:06:687 Boot Phase: controls-----------------[3] started

10:43:06:704 ctl: there is no speaker connected...

10:43:06:708 bt loops: timer [1] step 1 & step 2 starts

10:43:06:737 detect: trusted speaker monitor STARTS

10:43:07:003 detect: bt speaker connection monitor is READY [ <| ]

10:43:07:004 detect: LAN cable status at boot = 192.168.0.131

10:43:07:004 detect: Internet access at boot = access

10:43:07:203 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

10:43:07:204 Boot Phase: last part----------------[4] started

10:43:07:392 air: system service started for shairport-sync ----- # 9384

10:43:07:392 spot: system service started for librespot ----- # 9395

10:43:07:392 ctl: system service started for bluealsa ----- # 9411

10:43:07:392 Boot Phase: NO alsa.startupSound() [!]

10:43:07:392 ============================================|

10:43:07:393 Streamer Boot Completed in:

Startup finished in 3.734s (kernel) + 8.773s (userspace) = 12.508s

multi-user.target reached after 8.761s in userspace.

Boot residue . . . . . . . . . . . . . . . . . . . . . . . |

## Streamer process

sudo ps -aux | grep /bin/nodejs | head -n1

. . . /bin/nodejs /streamer/streamer-control.js

## Streaming services control

### When a streaming service[[4]](#footnote-4) is actually streaming audio

* No streaming services actively streaming audio.  
  Let all streaming services be on.
* A streaming service starts streaming audio.  
  Stop all other streaming services.
* The streaming service that has the alsa stops streaming audio.  
  All other streaming services starts again

## Bluetooth audio streaming

Bluetooth streaming may have two parts. One, streaming audio from a phone to the Streamer’s amplifier. Two, the Streamer is streaming audio to the connected bluetooth speaker and the amplifier is not used (muted). They can be combined, even if that make less sense.

Unfortunately, there are no in-built events (hooks) to catch for bluetooth start/stop streaming audio as for example Spotify. There is therefore important to understand when bluetooth streaming might occur:

1. detect functions for analogue speakers must run when the amplifier is unmuted.
2. detect functions for bluetooth speaker must run when the amplifier is muted.

Bluetooth streaming audio detection is not needed when

1. [analogue speakers] amplifier is muted or   
    another streaming service is running or   
    when no Bluetooth device is connected.
2. [bluetooth speaker] when the amplifier is unmuted,  
    but other streaming services might use the bluetooth speaker and  
    there might be no Bluetooth devices connected either.
3. [booth] when bluetooth is off.

## More on detection of Bluetooth speaker

The interactive bluetoothctl cannot really be used. It is way too verbose. Instead, other key criteria must be combined in the following way:

Can a Bluetooth speaker be connected? Then the following criteria must be in place:

1. No speaker is connected.
2. The amplifier is unmuted.
3. The file etc/asound.conf is not in place.

An external connect occurs when a trusted speaker reconnects itself, then the following criteria is valid:

1. The amplifier is unmuted.
2. The file etc/asound.conf is not in place.
3. There is a trusted speaker connected.

An external disconnect might occur when the speaker is turned off, then the following criteria is valid:

1. The amplifier is still muted.
2. The file etc/asound.conf exists.
3. There is no speaker connected.

### Bluetooth speaker configuration

In order to direct the audio to the speaker an asound.conf file has to be present. It is a bluealsa-plugin. The BlueALSA ALSA PCM plugin communicates with the bluealsad systemd service, called bluealsa. Remember: no file - no sound.

This is the content:

pcm.btspeaker {

type plug

slave.pcm {

type bluealsa

device '"${bdaddr}"'

profile 'a2dp'

# delay 20000 -- delay, period\_time, period are defined here, only if needed!

}

hint {

show on

description 'Bluetooth speaker, ${bdaddr}'

}

}

ctl.btspeaker {

type bluealsa

}

pcm.!default {

type plug

slave.pcm "btspeaker"

hint {

show on

description 'btspeaker'

}

}

ctl.!default {

type bluealsa

}

The upper part is an ALSA configuration node defining a PCM with type bluealsa. Do not confuse the PCM type bluealsa with the PCM named ‘bluealsa’.

The type itself does not perform any audio conversions, therefor the mid part is needed to define PCMs with type plug to achieve the conversion. There the Bluetooth Device address of the connected speaker has to be defined, ‘bdaddr’, every time a Bluetooth speaker is connected. To make the BlueALSA PCM visible via that API it is necessary to add a "hint" section to the ALSA configuration as well.

The last part, the pcm.!default makes the Bluetooth speaker as the default sound output device, so adding that section enables streaming services using alsa to give the sound output to that bluetooth device as well - note: that is why the analogue speakers (the alsa) is muted. ctl. is the mixer.

Use the default values of delay, period\_time and period. This is a mess.

## Best practise setup for frontend

Chris Ferdinandi (go-make-things)

* A static site generator (SSG) that serves mostly HTML, enhanced with Web Components.
* Vanilla CSS customized with variables/custom properties (--color-text).
* Let the SSG load modular CSS and JS as needed (maybe everything on every page, maybe only certain things on certain pages). [Hugo](https://click.convertkit-mail2.com/n4u8egw34wsvhxnkem9h6h6pq0kggalhoz2nw/3ohphkh3nlm862ir/aHR0cHM6Ly9nb2h1Z28uaW8=), [11ty](https://click.convertkit-mail2.com/n4u8egw34wsvhxnkem9h6h6pq0kggalhoz2nw/n2hohvhvkpq4wgf6/aHR0cHM6Ly93d3cuMTF0eS5kZXY=), [Astro](https://click.convertkit-mail2.com/n4u8egw34wsvhxnkem9h6h6pq0kggalhoz2nw/wnh2hghq3p7d93f7/aHR0cHM6Ly9hc3Ryby5idWlsZA==), and more all have mechanisms for doing this easily.
* [Playwright](https://click.convertkit-mail2.com/n4u8egw34wsvhxnkem9h6h6pq0kggalhoz2nw/reh8hohmerdp95f2/aHR0cHM6Ly9wbGF5d3JpZ2h0LmRldg==) for testing, using [a TDD approach](https://click.convertkit-mail2.com/n4u8egw34wsvhxnkem9h6h6pq0kggalhoz2nw/08hwh9h2k5e40mcl/aHR0cHM6Ly9nb21ha2V0aGluZ3MuY29tL3RkZC1mb3ItamF2YXNjcmlwdC13aXRoLWJ1aWxkbGVzcy1tb2NoYS1hbmQtY2hhaS8=). Only test the user/developer-facing surface areas, not implementation details.
* [Biome](https://click.convertkit-mail2.com/n4u8egw34wsvhxnkem9h6h6pq0kggalhoz2nw/vqh3hrhod7062ofg/aHR0cHM6Ly9iaW9tZWpzLmRldg==) for linting, and *maybe* formatting, mostly with the default settings. Enable [an IDE extension](https://click.convertkit-mail2.com/n4u8egw34wsvhxnkem9h6h6pq0kggalhoz2nw/m2h7h5h37p5xkeim/aHR0cHM6Ly9iaW9tZWpzLmRldi9ndWlkZXMvZWRpdG9ycy9maXJzdC1wYXJ0eS1leHRlbnNpb25zLw==) for live-linting as you type.
* [JSDoc](https://click.convertkit-mail2.com/n4u8egw34wsvhxnkem9h6h6pq0kggalhoz2nw/dpheh0hemg6xzrbm/aHR0cHM6Ly9nb21ha2V0aGluZ3MuY29tL2hvdy10by1kb2N1bWVudC1vYmplY3QtcHJvcGVydGllcy13aXRoLWpzZG9jLw==) for in-code documentation.
* *No* typescript files, but [use it with JSDoc to typecheck](https://click.convertkit-mail2.com/n4u8egw34wsvhxnkem9h6h6pq0kggalhoz2nw/e0hph7h72gvz9ma8/aHR0cHM6Ly9mZXR0YmxvZy5ldS90eXBlc2NyaXB0LWpzZG9jLXN1cGVycG93ZXJzLw==) plain old JS files. You can have it live typecheck in your IDE, which is really helpful!
* An LSP extension for [Sublime](https://click.convertkit-mail2.com/n4u8egw34wsvhxnkem9h6h6pq0kggalhoz2nw/7qh7h8h9m7levqiz/aHR0cHM6Ly9sc3Auc3VibGltZXRleHQuaW8vbGFuZ3VhZ2Vfc2VydmVycy8=), [VSCode](https://click.convertkit-mail2.com/n4u8egw34wsvhxnkem9h6h6pq0kggalhoz2nw/owhkhqhw7v9e39fv/aHR0cHM6Ly9jb2RlLnZpc3VhbHN0dWRpby5jb20vYXBpL2xhbmd1YWdlLWV4dGVuc2lvbnMvbGFuZ3VhZ2Utc2VydmVyLWV4dGVuc2lvbi1ndWlkZQ==" \t "_blank" \o "https://click.convertkit-mail2.com/n4u8egw34wsvhxnkem9h6h6pq0kggalhoz2nw/owhkhqhw7v9e39fv/aHR0cHM6Ly9jb2RlLnZpc3VhbHN0dWRpby5jb20vYXBpL2xhbmd1YWdlLWV4dGVuc2lvbnMvbGFuZ3VhZ2Utc2VydmVyLWV4dGVuc2lvbi1ndWlkZQ==), or your text editor of choice.

# Appendix: Essential Linux CLI

## Amplifier

#### Introducing the -M option

Use the mapped volume for evaluating the percentage representation like alsamixer, to be more natural for human ear.

amixer -Mc0 set Digital 40% --the option -M uses mapped volume for %

Simple mixer control 'Digital',0

Capabilities: pvolume pswitch

Playback channels: Front Left - Front Right

Limits: Playback 0 - 207

Mono:

Front Left: Playback 159 [40%] [-24.00dB] [off]

Front Right: Playback 159 [40%] [-24.00dB] [off]

Gets the right volume %

amixer -Mc 0 sget Digital | grep 'Right:' | awk -F'[][%]' '{ print $2 }'

33

amixer -Mc 0 sget Digital | grep 'Right:' | awk -F'[][%]' '{ print $2 }'

amixer -c 0 set Digital toggle -- mute and unmute

amixer -Mc 0 set Digital 25% -- -c is card, set volume

## Bluetooth connections

bluetoothctl devices Connected or Paired or Trusted

Device FC:58:FA:1B:06:66 JVC HA-S22W

Device 8C:DE:E6:25:C5:8C Galaxy S21 Ultra 5G

bluetoothctl devices Connected | awk '{print substr($0, index($0,$2))}'

FC:58:FA:1B:06:66 JVC HA-S22W

8C:DE:E6:25:C5:8C Galaxy S21 Ultra 5G

Regarding awk; uses substr to extract a substring of the line starting from the beginning of the second field ($2). index($0,$2) finds the starting position of the second field within the entire line ($0).

##### The bluetootctl device string

Device = 7 chars (including tailing space.

8C:DE:E6:25:C5:8C = start 8 position, 17 chars.

8C:DE:E6:25:C5:8C = end 28 position.

Name =start 30 position to end

bluetoothctl devices

Device FC:58:FA:1B:06:66 JVC HA-S22W

Device 10:CA:BF:01:80:5F S7c0023c8ffc8df99C

Device 8C:DE:E6:25:C5:8C Galaxy S21 Ultra 5G

bluetoothctl devices | cut -d' ' -f2

10:CA:BF:01:80:5F

FC:58:FA:1B:06:66

8C:DE:E6:25:C5:8C

bluetoothctl devices Connected | cut -d' ' -f2

FC:58:FA:1B:06:66

8C:DE:E6:25:C5:8C

## Bluetooth sink or source

bluetoothctl info FC:58:FA:1B:06:66 | grep "UUID: Audio Sink " | cut -d' ' -f3

Sink

bluetoothctl info FC:58:FA:1B:06:66 | grep "Name: " | cut -d' ' -f2-

JVC HA-S22W

## Bluetooth scan for sinks

bluetoothctl scan on

Discovery started

[CHG] Controller 2C:CF:67:75:F6:A1 Discovering: yes

[NEW] Device FC:58:FA:1B:06:66 JVC HA-S22W

[CHG] Device FC:58:FA:1B:06:66 RSSI: -53

[NEW] Device 8C:DE:E6:25:C5:8C Galaxy S21 Ultra 5G

[CHG] Device 8C:DE:E6:25:C5:8C RSSI: -61

: : : -- it goes on and on… has to be stopped!

Note: it has to scan for at least 15 secs, the longer the better…

Find the pid for bluetoothctl scan:

pgrep -a bluetoothctl | grep "bluetoothctl scan on" |cut -d' ' -f1

641

Note: option '-a' yields full command line so it is possible to check the status of scan on the returned:  
641 bluetoothctl scan on -- this is obviously the scan process

ps aux | grep bluetoothctl

root 641 0.0 0.1 8148 3744 pts/0 S+ 14:13 0:00 bluetoothctl scan on

root 679 0.0 0.0 7520 1868 pts/1 S+ 14:21 0:00 grep bluetoothctl

Stop the bluetoothctl scan with:

kill 641 -- killed with a smooth clean-up (use this).

kill -9 641 -- brutally killed with no clean up

What did bluetoothctl find: -- only valid for max 3 minutes after scan stopped

bluetoothctl devices

Device FC:58:FA:1B:06:66 JVC HA-S22W

Device 8C:DE:E6:25:C5:8C Galaxy S21 Ultra 5G

bluetoothctl devices| awk '{print substr($0, index($0,$2))}'

FC:58:FA:1B:06:66 JVC HA-S22W

8C:DE:E6:25:C5:8C Galaxy S21 Ultra 5G

bt-device -l

Added devices:

JVC HA-S22W (FC:58:FA:1B:06:66)

S7c0023c8ffc8df99C (10:CA:BF:01:80:5F)

Galaxy S21 Ultra 5G (8C:DE:E6:25:C5:8C)

bt-device -l | awk 'NR > 1 { print }'

JVC HA-S22W (FC:58:FA:1B:06:66)

Galaxy S21 Ultra 5G (8C:DE:E6:25:C5:8C)

Someone was able to poll for a connection by looking for: (event4 does not exists…)

ls /dev/input/event4

### Bluetooth speaker

aplay -D bluealsa:DEV=FC:58:FA:ED:57:60 /streamer/audio/startup.wav &

Test the Eneby 20 speaker.

While audio is playing verify the PCM\* settings with:

sudo cat /proc/asound/card\*/pcm\*p/sub\*/hw\_params

amixer scontrols -D bluealsa

Acquires name of your mixer device. Example output:

Simple mixer control 'Speaker - A2DP',0

bluealsa-aplay -l -- devices

\*\*\*\* List of PLAYBACK Bluetooth Devices \*\*\*\*

hci0: FC:58:FA:ED:57:60 [ENEBY30], trusted audio-headphones

A2DP (SBC): S16\_LE 2 channels 44100 Hz

\*\*\*\* List of CAPTURE Bluetooth Devices \*\*\*\*

hci0: 8C:DE:E6:25:C5:8C [Galaxy S21 Ultra 5G], phone

A2DP (aptX): S16\_LE 2 channels 44100 Hz

bluealsa-aplay -L -- pcm

bluealsa:DEV=FC:58:FA:ED:57:60,PROFILE=a2dp,SRV=org.bluealsa

ENEBY30, trusted audio-headphones, playback

A2DP (SBC): S16\_LE 2 channels 44100 Hz

bluealsa:DEV=8C:DE:E6:25:C5:8C,PROFILE=a2dp,SRV=org.bluealsa

Galaxy S21 Ultra 5G, phone, capture

A2DP (aptX): S16\_LE 2 channels 44100 Hz

### Optional bluetooth CLI

bluetoothctl devices

Device FC:58:FA:1B:06:66 JVC HA-S22W

Device 8C:DE:E6:25:C5:8C Galaxy S21 Ultra 5G

All information about devices in Bluetoothctl can be found in the file:

/var/lib/bluetooth/DC:A6:32:00:32:B2/34:14:5F:48:32:F8/info

DC:A6:32:00:32:B2 = Agent BD and 34:14:5F:48:32:F8 = Device BD

bt-device -l | grep BD | cut -d " " -f1-2

bluetoothctl devices | grep BD | cut -d " " -f3-

bluetoothctl info BD | grep '^\s\*Connected:' | awk '{print $NF}'

bt-device -i BD | awk '/Connected/ {print $2}'

bt-device -i BD | grep Connected

bluetoothctl > /dev/null

hcitool scan --flush

rfkill list | grep -A2 hci0

rfkill block bluetooth or rfkill unblock bluetooth

## Wireless connections

Unlike Bluetooth, Wireless is never turned off. The streamer can be connected to Wi-Fi or be disconnected.

nmcli device status | grep wlan1 | awk '{ print $3}'

connected

But it is the SSID that is interesting:

nmcli device status | grep wlan1 | awk '{print $4}'

BELL462

… and the ip address:

ip -f inet -o addr show wlan1 | awk '{ print $4 }' | cut -d'/' -f1

192.168.0.122

Connect to Wi-Fi:

sudo nmcli dev wifi connect "BELL462" password "4DFD4EFC155C" ifname wlan1

Device 'wlan0' successfully activated with 'd8848859-d784-4885-96c7-2efc599d1115'.

The ifname option defines which wlan.

Get the UUID in order to disconnect the right connection

nmcli -t -f general device show wlan1 | grep GENERAL.CON-UUID: |cut -d':' -f2

18ce02f6-ca72-46a9-8f3b-674ff9cdbfac -- add | tr -d '\n' to get a clean string

Scan for Wi-Fi;

nmcli dev wifi list -- below we got two scans, first is wlan0 and the other is wlan1

IN-USE BSSID SSID MODE CHAN RATE SIGNAL BARS SECURITY

D8:44:89:5B:0B:6E BELL462 Infra 6 130 Mbit/s 100 ▂▄▆█ WPA2

DE:44:89:5B:0B:6E -- Infra 6 130 Mbit/s 100 ▂▄▆█ WPA2

D8:44:89:5B:0B:6D BELL462 Infra 48 270 Mbit/s 100 ▂▄▆█ WPA2

0C:AC:8A:8E:48:90 -- Infra 157 540 Mbit/s 100 ▂▄▆█ WPA2

F0:A7:31:81:21:EB BELL462 Infra 6 130 Mbit/s 85 ▂▄▆█ WPA2

12:A7:31:81:21:EB -- Infra 6 130 Mbit/s 85 ▂▄▆█ WPA2

F0:A7:31:81:21:EA BELL462 Infra 48 270 Mbit/s 64 ▂▄▆\_ WPA2

12:A7:31:81:21:EA -- Infra 48 270 Mbit/s 62 ▂▄▆\_ WPA2

IN-USE BSSID SSID MODE CHAN RATE SIGNAL BARS SECURITY

D8:44:89:5B:0B:6E BELL462 Infra 6 130 Mbit/s 100 ▂▄▆█ WPA2

DE:44:89:5B:0B:6E -- Infra 6 130 Mbit/s 100 ▂▄▆█ WPA2

F0:A7:31:81:21:EB BELL462 Infra 6 130 Mbit/s 72 ▂▄▆\_ WPA2

12:A7:31:81:21:EB -- Infra 6 130 Mbit/s 72 ▂▄▆\_ WPA2

Scan, and then show the result for wlan1:

nmcli dev wifi list --rescan yes | awk -v RS= 'NR==2'| awk '!/--/ && !/BSSID/ {print $2,$4}'

BELL462 6

BELL462 6

BELL462 6

BELL462 6

… and then disconnect

nmcli connection delete 18ce02f6-ca72-46a9-8f3b-674ff9cdbfac

Connection 'BELL462' (18ce02f6-ca72-46a9-8f3b-674ff9cdbfac) successfully deleted.

### Optional network CLIs:

Sometimes CLIs seem to use multiple spaces or inconsistent delimiters for status, that is why tr (translate) has to be used. sed can also normalise the delimiters.

nmcli device status | grep wlan1 | grep " connected "

wlan1 wifi connected BELL462

wpa\_cli status | grep "ip\_address=" |cut -d'=' -f2 -- DEPRECATED

192.168.0.122

This gives a lot of information even when it is limited to general:

nmcli -f general device show wlan1

GENERAL.DEVICE: wlan1

GENERAL.TYPE: wifi

GENERAL.NM-TYPE: NMDeviceWifi

nmcli device status | grep wlan1 | tr -s ' ' |cut -d' ' -f3

connected -- added tr -s ‘ ‘ removes extra spaces

ip -f inet -o addr show wlan1 | awk -F'/' '{print $1}' | tr -s ' ' | cut -d' ' -f4

192.168.0.122

ifconfig wlan1 | grep "inet " | tr -s ' ' | cut -d' ' -f3

192.168.0.122 -- have to add tr -s ‘ ‘ as well… Deprecated

ip -br a |grep eth0 | grep -Po ' \K[\d.]+' -- eth0 can be replaced with wlan1

192.168.0.131

## Access Point – hotspot

The AP is off or on.

nmcli device status | grep Hotspot-1

wlan0 wifi connected Hotspot-1 - - connected!

- - - disconnected!

nmcli connection up Hotspot-1 - up, Hotspot-1 is the connection profile

nmcli connection down Hotspot-1 - down

### Optional

nmcli device status | grep "wlan0 " | awk '{ print $3}'

connected

disconnected -- wlan0 cannot catch connect because of p2p-dev-wlan0

nmcli device status | grep Hotspot-1 | awk '{ print $3}'

connected

## alsa and bluealsa -analogue amplifier

To set bluealsa as default sound output device on Linux, one typically create an etc/asoundrc file to define a new ALSA "default" PCM that points to the bluealsa device, like this: pcm.!default { type plug slave.pcm "btreceiver" }, after defining a pcm.btreceiver that links to the specific Bluetooth address and profile.

Which is the process id for bluealsa-aplay:

systemctl status bluealsa-aplay | grep "Main PID: " | cut -d' ' -f6 | tr -d '\n'

538

amixer -Mqc 0 set Digital 100% -- set full volume, used at boot

- no message since -q option is set

amixer -c 0 set Digital 201 -- using 0-207 scale instead of %

Simple mixer control 'Digital',0

Capabilities: pvolume pswitch

Playback channels: Front Left - Front Right

Limits: Playback 0 - 207

Mono:

Front Left: Playback 201 [97%] [-3.00dB] [on]

Front Right: Playback 201 [97%] [-3.00dB] [on]

rm -f /etc/asound.conf -- remove asound.conf for bluetooth speaker

/proc/asound/card0/pcm0p/sub0/status -- f alsa is running the pid is here

pmap -q -A ffff1000,ffff1100 14295 -- 14293 is the pid in this case

14293: /bin/librespot -n Streamer -b 320 --initial-volume 35 --volume-ctrl cubic --disable-audio-cache --enable-volume-normalisation

Processes using alsa:

fuser /dev/snd/\*

/dev/snd/pcmC0D0p: 521 -- not always the main pid

bluealsa-aplay -v

bluealsa-aplay: I: Selected configuration:

BlueALSA service: org.bluealsa

ALSA PCM device: default

ALSA PCM buffer time: 200000 us

ALSA PCM period time: 50000 us

ALSA mixer device: default

ALSA mixer element: 'Master',0

Volume control type: auto

Bluetooth device(s): ANY

Profile: A2DP

systemctl edit bluealsa-aplay

Will open the system service file in use.

amixer -D bluealsa scontrols

Simple mixer control 'Galaxy S21 Ultra 5G A2DP',0

## Internet

These web pages are tried in the shuffled order: "https://www.google.com", "https://www.youtube.com", "https://www.facebook.com", "https://www.weather.com",

"https://www.bing.com", "https://www.instagram.com", "https://www.wikipedia.org/", "https://www.reddit.com"

curl -Is https://www.google.com | grep "HTTP/2 200"

HTTP/2 200 -- maybe add | tr -d '\n' at the end

### Optional

wget --spider https://www.google.com

Spider mode enabled. Check if remote file exists.

--2025-08-18 11:09:20-- <https://www.google.com/>

: : :

## Wired connection

ip -f inet -o addr show eth0 | awk '{ print $4 }' | cut -d'/' -f1

192.168.0.131

hostname -I|cut -d' ' -f1 -- LAN IP

curl -s4 ifconfig.co --. Public IP

### Optional

Check the file /sys/class/net/eth0/carrier for state of the LAN port. If "1" the cable is physically connected, if "0" no cable in LAN port.

## Important internal data

### Process id

systemctl status <service> | grep "Main PID:" | cut -d' ' -f6

ps aux | grep <process name> -- too verbose

All about SSD devices:

udevadm info --query=all --name=/dev/sdb -- detailed and verbose

blkid -o export -- nice output

### Disk

df -h / | awk '/\// {print "Disk Size="$2", Used="$3", Available="$4", "$5" of disk used"}'

du -sh /\* 2>/dev/null

Top level view of space of all directories.

du -sh /streamer/\* | sort -h

Next level, this is /streamer directory.

du -h | sort

User root directory

du -h /streamer | sort -rh | head -n 10

10 largest files in /streamer.

### Temperature & voltage

#### dmesg messages

dmesg -H | grep -i temperature

dmesg -H | grep -i voltage

dmesg -H | grep -i throttle

This is the dmesg to watch:

dmesg -HTwx | grep -i voltage --interactive mode

vcgencmd measure\_volts

volt=0.7500V

vcgencmd pmic\_read\_adc

Generates a long list of voltages.

### Throttled?

vcgencmd get\_throttled

throttled=0x50000 -- anything other than 0x0 then it is, or has been, throttled.

0x50000 means throttled has occurred since the last reboot. 0x5000 indicates that the system has detected conditions requiring a reduction in CPU frequency to prevent damage or maintain stability. This usually points to either thermal issues (overheating) or power supply issues (undervoltage). One can also use watch vcgencmd get\_throttled to continuously monitor the status.

Reset

vcgencmd get\_throttled 0x05

throttled=0x0

cat /sys/class/thermal/thermal\_zone0/temp | awk '{printf "%.2f", $1/1000}'

1. Fisher-Yates (Not anymore just plain random.) [↑](#footnote-ref-1)
2. Needs to be a prime integer. [↑](#footnote-ref-2)
3. File Descriptor is a process identifier for a file or other input/output resource, such as a pipe or network socket. Files are always there but the number of pipes and sockets varies. [↑](#footnote-ref-3)
4. Streaming services are Spotify (librespot, Airplay (shairport-sync),… not bluealsa-aplay. [↑](#footnote-ref-4)