ForwardProxy manual version: V1.1.220421

ForwardProxy software version: V2.5.120421

“FowardProxy” (FWP) is an open source and free initiative to make OggFlac high fidelity radio stations available through Naim generation One streaming devices where this feature is not available, nor planned by Naim, at the moment of writing of this document.

I initially started this project as an answer to my own needs. Although after having programmed something that seemed “to hold the road”, I thought it might be useful so share my work.

Now, FWP has proven to be stable, scalable, resource friendly and portable. This was the trigger for me to let it go.

I hope that as a “user” you may enjoy the better sound quality on your system, versus the non-Flac streams. If you are a developer, I am looking forward to your enhancements; warm welcomed! ☺

I do not expect any kind of payment for this. However, if you are using it and you are happy with it, just drop me an email and tell me from where you are.

I initially set this up to listen to the HQ Naim (Jazz) streams. By applying some magic, I also got It working for some famous other streams. Do not kill or threaten me if it does not work for your favorite stream. Although, feel free to let me know, so I can have a look without guarantee for resolution.

For now, go on and enjoy exploring FWP as much as I did to program it. Or even better, enjoy the sound!

Kurt, Bornem, Belgium, 22 April 2021

kurt.lefevre@gmail.com

https://linkedin.com/in/lefevrekurt

https://github.com/kurt-lefevre/FWP

Ok, let us start… Well, the good thing is that it is less complicated than solving the Corona thing to get it running. On the other side, it is slightly more complex than yelling someone to get you a beer from the fridge ☺

Before diving into the architecture, there are some prerequisites in order to run FWP. If you cannot meet those, the story ends here. However, please let me know, so I can see if can address the issue.

Prerequisites and building blocks

A UNIX or Linux operating system. Although being developed on a Windows machine, **FWP only runs on a Linux or UNIX based operating system**. This has to do with the way the decoding process is launched as well as the "processkill" process, taking care of stale or stuck threads. Do not bother about this now; it will be explained later. The CPU type does not matter.

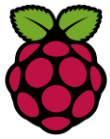
FWP is written in Java. Reason for choosing Java is the "write once, run anywhere" approach and its high performance. Indeed, FWP runs fine on your Linux laptop, Raspberry Pi, NAS, Cloud environment of choice, as long as you can run Java 11 (or above) as a JDK or JRE on it. **FWP has been targeted to Java 11** as this is the current Long Term Support (LTS) release. In addition, FWP also runs fine on the latest beta version of JDK 17 as well as on GraalVM.

If you are looking for a free Java download for your platform of choice, have a look at these sites:

- Azul: https://www.azul.com/downloads/zulu-community/?package=jdk
- Bellsoft: https://bell-sw.com/pages/downloads
- Oracle JDK: https://www.oracle.com/be/java/technologies/javase-downloads.html
- Amazon Corretto: https://aws.amazon.com/corretto/
- OpenJDK: https://jdk.java.net/

Note that a JRE is sufficient. There is no need for a full-blown JDK. As a JDK distribution also contains a JRE, a JDK is ok as well.

There is no preference for a specific JDK. I tested FWP on different JDK distributions and versions (11 and above) without any problems.





Next thing is the hardware and operating system

FWP requires a UNIX/Linux based operating system supporting (at least) Java 11. There are several options, not limited to the list below:

1) Run it on whatever machine running whatever UNIX/Linux distribution and Java 11+.
2) A Raspberry Pi running a Linux based OS. This is my personal favorite. I run FWP on a PI 4B with 4 GB RAM with the 64-bit Raspberry Pi OS (Debian). I also tested it with success on an older 32-bit Raspberry Pi B+.
3) NAS: All NAS appliances capable of running Java and providing the option to install additional (Java) applications.
4) The Cloud: Many options in this area. There is Amazon, Microsoft, Google, Oracle … I test and run FWP on a free Oracle Cloud instance running Ubuntu. The Cloud instance has 1 GB of RAM and 100 TB of disk space. As FWP requires few resources, this is a reliable, performant and free option. This is also my publically shared instance. More info about Oracle Free Tier: https://www.oracle.com/cloud/free/

As FWP is a lightweight application, older equipment with limited RAM or not a state of the art processor will do fine.

FFmpeg (https://www.ffmpeg.org) is popular open source, complete, cross-platform solution to record, convert and stream audio and video.

In the context of FWP, FFmpeg takes care of unwrapping the by the music service provided OggFlac envelope and converting the native Flac to a Wav format, being sent to the Naim. Later more on the process.

FWP comes with two decoding configurations, which are easy to configure, to extend, modify, optimize … according your needs or preferences. FWP does not impose using FFmpeg to transcode the OggFlac. Taking into consideration some conventions, you can use whatever transcoding engine with FWP.

Actually, FWP does not require FFmpeg. However, **if you would like to get started with the decode settings provided by the FWP package, you will need FFmpeg** as the decode schemes rely on FFmpeg. More details later.

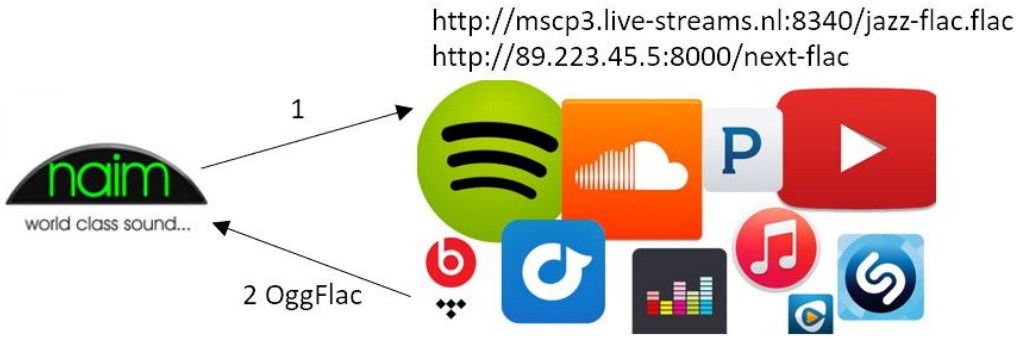
Architecture and design

If you are not interested in the internal working of FWP, you can skip this section and go on with the "Installation" chapter.

From a Naim point of view, FWP looks like any other custom radio station you configure in vTuner (https://naim.vtuner.com).

Most important parameter in the vTuner configuration of a custom radio station is the "Station URL". This is how the Naim knows where to connect to If you want to listen to a radio station. The URL has to be in the standard http or https format. The picture below gives you some examples.

The process, or flow, the Naim uses to connect to any music service is simple. It consists of a "handshake" process, represented by "1" in the picture and getting the continuous music stream after a **successful** handshake. This is show as "2 OggFlac" in the picture. Phase 2 is also the moment you hear the music playing through your speakers.



Although easier to understand than quantum mechanics for me, there are some catches in this story…

To develop and test FWP, I used three different stream players; my ND5 XS, a Mu-so Qb (both first generation) and VLC, an open source media player.

The handshake process is key to get music out of the speakers. The tricky thing about that step is that the handshake process is implemented in a different way on different devices. In addition to that, Naim, like any other proprietary solution, does not document or comment how a successful handshake should look like. As this becomes mainstream and common in modern audio equipment, I hope manufacturers change their mind one day and make this an open technology – just my 2c.

After spending days and nights on capturing and analyzing the bit stream, I gradually got a better view and more insight on this, although some things remain an assumption. The way it works is as follows:

1) The Naim connects to the custom radio station and announces itself as a streaming device. Consider this as a kind of a "Hi, I'm Naim!" message.
2) The radio service, or streaming service, responds to this by identifying itself by providing some information like station name, genre, **audio format**, bit rate, …
3) The Naim interprets this information and decides whether to continue the process of not.
4) In case the Naim considers the handshake as successful, the Naim sends another request to the streaming service like "Ok, send me the music!".

5) In response to this request, the streaming service responds to this by sending a continuous bit stream. This actually completes the handshake process. As from this point, the Naim gets the bit stream “forever” and starts playing the music.

Some important remarks about the handshake process that make things a little more complicated:

- Although mentioned in sequential steps, note that those steps take place in parallel. This means the Naim sends different request to the streaming service (more or less) at the same time. This is why FWP is fully multi-threaded.
- Sometimes, the Naim sends the same request twice. As Naim does not document the process, to me it remains a mystery why this happens. Fortunately, at the end, this has no impact on the handshake process.
- This is the way Naim handles the process. VLC for example approaches this in a slightly different way. Likewise, other streaming devices will do it also “their way”.
- As from the moment the bit stream starts playing, the Naim never sends a “stop” signal to the streaming services in case the unit is switched off, when you switch radio stations or when you switch to another input. Hence my quoted “forever” in step 5 of the handshake process. Instead, the Naim just drops the connection to the streaming service. Shortly after this, the streaming service will figure out it cannot send the bit stream to the Naim anymore, and will stop as well. Not the most polite way of closing connections, but it works and this is something to take care of in the design of FWP.

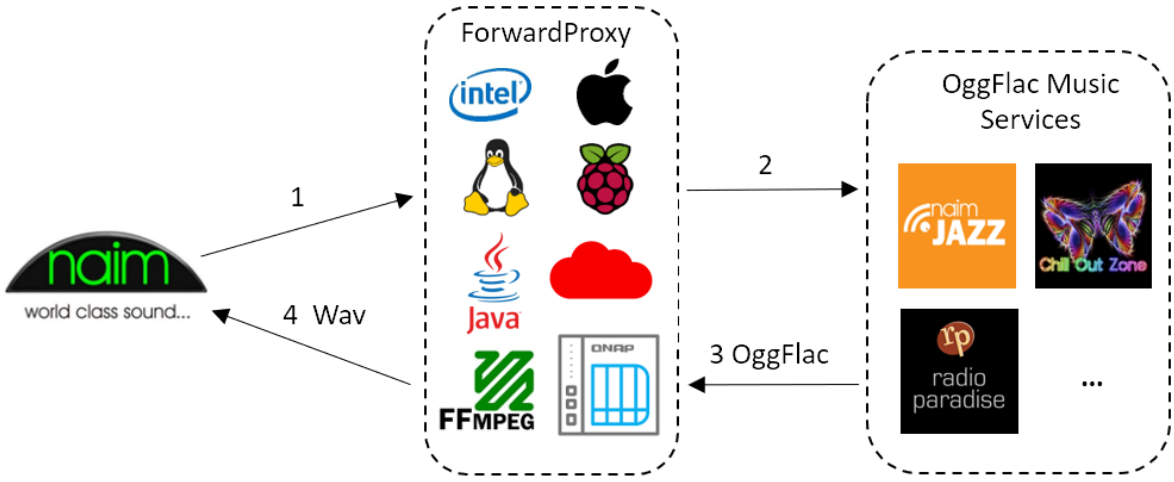
This process is quite similar for all audio formats. So, where does it go wrong when connecting to an OggFlac radio station with a first generation Naim device?

Short answer: “Step 3 in the handshake process”. In step 3 of the process, the Naim interprets the information (metadata) received from the streaming service in step 2. In case of an OggFlac stream, the Content-Type field, representing the audio format, is set to “application/ogg” or “audio/ogg” depending on the streaming service. Apparently, in case of a radio station, the Naim does not like this and drops the connection. This ends the handshake process at that stage and no music will play.

This conclusion was the starting point to think about a workaround. The idea that popped-up in my mind: *“If the Naim cannot deal with the OggFlac format, why not putting something in between the Naim and the streaming service that unwraps the OggFlac format in a native Flac or Wav, as the Naim can deal with this?”*. This idea resulted in the development of FWP.

Below you can find the schematic representation and flow how the Naim, FWP and Music Services interact with each other. I a nutshell this is how the flow works:

1) The Naim connects via a custom radio station to FWP. This has to be a HTTP URL pointing to the IP address and port of the system hosting FWP, followed by a user-configured path. More on this later. This is the above described, “Hi, I’m Naim!” request.
2) FWP maps the incoming request to the streaming service URL, connects to it and sends the unmodified request to it.
3) FWP receives the response, changes the above-mentioned “audio/ogg” Content-Type in the response to “audio/wav”, and then sends the response to the Naim.
4) The Naim receives the response, considers “Wav” as an acceptable format and continues the handshake process.



Those four steps are repeated once again for the “Ok, send me the music!” request from the Naim. When FWP detect this type of request, it will instruct FFmpeg to decode the OggFlac envelope, transform it on the fly to Wav, and return a Wav bit stream to the Naim. This is the moment the music starts playing through your speakers.

In summary, from a Naim device point of view, FWP looks like any other custom radio station.

Behind the scenes, FWP uses a configurable bit stream buffering mechanism and sends the music bit stream in chunks of the same size to the Naim. As the Naim only sees FWP as its iRadio endpoint or radio station URL, having FWP running at home, or in a Cloud in your country, could be a solution for latency issues (drops) due to the long distance and network hops between the location where the Naim is and the location running FWP.

Although, also some catches here:

- Native Flac is not supported via iRadio (?). Although the native Flac format is supported via the UPnP input, it looks this is not the case for the iRadio input. My initial idea was to unwrap the OggFlac envelope and send the native Flac contents to the Naim. I have many CDs ripped to Flac and play them from a NAS with MinimServer. As this works pretty well, I supposed this approach should do it also for the iRadio input. It doesn’t; the Naim drops the connection. Initially I thought it had to do with the used FFmpeg decoder. To exclude FFmpeg I wrote my own OggFlac decoder based on the official documentation and specifications (https://xiph.org/vorbis/doc/framing.html, https://xiph.org/flac/ogg_mapping.html and https://xiph.org/flac/format.html), but also this did not fly. As a last attempt and to avoid any coding bugs from my side, I took a perfect playing Flac file from my NAS and sent it to the Naim. Exactly the same behavior. I raised the question to Naim about the iRadio input and Flac support, but even after 1.5 months and some reminders, I still don’t have an answer ☹. Due to the lack of feedback from Naim, I just can guess iRadio does not support native Flac as an input format on first generation devices. This is why FFmpeg transcodes to Wav instead of Flac. Both Flac and Wav formats are lossless, so there is no loss of sound quality.
- Running FWP in the same network domain as the Naim, makes the Naim app thinking it is a UPnP device. Indeed, if you add FWP as a custom radio station and have the FWP IP address in the same network domain as your Naim, when you select that radio station, the app will

start playing the radio station and will bring you to the UPnP music play page. I created a second network domain in my house to run FWP, and this works as expected. It looks like the Naim app considers local iRadio stations as UPnP sources…

FWP is fully multi-threaded and supports multi devices. This means that multiple (Naim) devices can connect simultaneously to FWP to the same or different radio stations without interference.

One of the advantages of the OggFlac format is the fact it is optimized for decoding speed (penalty is paid during encoding). This results in fast on the fly decoding combined with low memory and CPU consumption. The latter on its turn results in lower energy consumption. Transcoding the OggFlac to Wav offloads the transcoding process from the Naim, so the Naim runs more "at ease".

Concerning security, as usual a general security infrastructure blocking any malicious requests as soon as possible is the recommend approach. However, this may be overkill for "just" a proxy solution like FWP. With this requirement in mind, FWP is designed to block any not compliant requests immediately the moment they reach FWP. As I run FWP on a non-firewalled public Cloud instance, FWP is attacked on a regular basis. This is something you can figure in the log files. However, no performance penalty. Evil requests get kicked out the moment they try to enter FWP!

Installation

The installation may require some basic Linux skills. All screenshots and Linux commands used in FWP are Raspberry OS (aka Debian) based.

Upfront hints:

- **On a Linux Operating System, all commands and parameters are case sensitive**. So unlike Windows, on a Linux or UNIX based Operating System, "Java" is not the same as "java".
- Be **careful with copy/paste operations from within a PDF file**, especially when line breaks are involved.

Terminal emulation

The installation requires command line access to your Linux/UNIX environment. There are numerous terminal emulation programs available. Popular free Windows applications in this domain are MobaXterm (https://mobaxterm.mobatek.net) and PuTTY (https://www.putty.org). From within an existing Linux environment, a remote terminal connection will do this without any additional software.

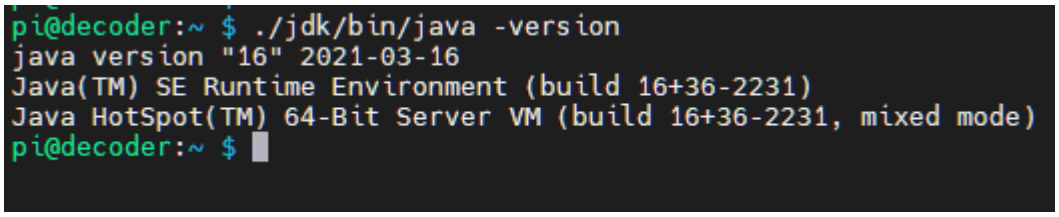
Java

"No Java, no play". FWP requires Java 11 or higher. Figure out if you have Java on your system and its version just type "Java -version" on the command line. Do not copy or type the quotes. I just

added them for readability purposed. If you get something like this, it means that 1) you do not have Java installed yet or 2) Java is not in your path.

If you do not have Java, you can download it via one of the links mentioned above. I prefer downloading a tar.gz distribution over an installer, so things are not mixed.

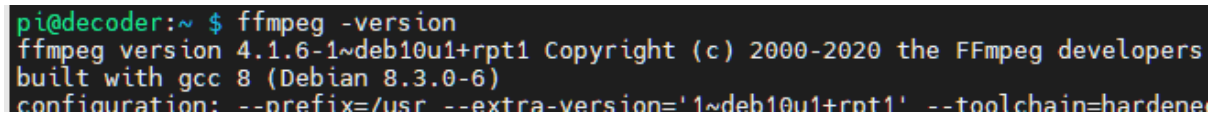
After a successful installation, and depending if Java is your path or not, it should give you something like shown below. As you can see, I run it on Java 16, but if you install 11, it will show 11 of course.



FFmpeg

FFmpeg (https://www.ffmpeg.org) is used to transcode the OggFlac envelope coming from the music service to a Wav format. If you want to use the decode settings provided by the FWP package, you will need FFmpeg. However, it is also possible to use another transcoder with FWP. This will be explained later.

In order to use FFmpeg with the FWP provided package, FFmpeg has to be in the system path. This means that typing “ffmpeg -version” should give you something like this:



If you do not want FFmpeg in the system path, you can also use the absolute path to FFmpeg. This will also be explained later. If FFmpeg is missing, on a Debian based Linux distribution, “sudo apt install ffmpeg” should do the job.

Having Java 11+ and FFmpeg installed is a prerequisite as a starting point to get FWP working.

FWP

I recommend starting with the standard FWP package. The installation package is a compressed file in the format “forwardproxyDDMMYY.tar.gz” and contains all required assets to start with. Once comfortable, you can start playing with different decode settings or decoders, configuration parameters etc.

FWP requires minimum these files:

- “**ForwardProxy.jar**”: This is the FWP application.

- A **configuration file**: This plain text file contains the configuration settings and station definition of FWP. The by the package provided configuration file is called “config.txt”, but you can give it whatever name.
- **Decode script**(s): The script(s) contain(s) the decode configuration to convert the FlacOgg to Wav. Based on the type of bit stream it may require different decode settings. The for a radio station to be used decode setting has to be specified in the configuration file. By convention, a decode script should have following format: “decodeX.sh”, where X represents a number corresponding the number specified in the configuration file for that radio station. The script requires execution rights and has to be in the same directory as FWP. The package comes with two (example) scripts. You should modify these scripts, or create additional scripts, if you want to use another decoder, modify the decode settings etc. If you start modifying the scripts, note that they run each time you start a radio station. So, keep the script logic as simple as possible. Hence the reason why the transcode settings in the package are just one line.
- **Kill script**: As the Naim often closes connections in an abrupt way (read above), it may leave the FWP decoding thread in an unstable state. To avoid resource leaks, unused processes, memory leaks, needles CPU consumption, FWP has a StaleTheadMonitor that monitors inactive decoders and shut them down nicely when not terminated properly. The kill script is part of this process. By convention, this script has to be called “killprocs.sh”, requires execution rights, and has to be in the same directory as FWP. Although no need to alter this, you can modify it at your convenience.

All of this has to land somewhere in a folder. To keep things easy I created a folder called “forwardproxy” in the home directory of my Raspberry Pi. All supplementary scripts rely on this location. **Keep in mind to adapt the scrips when choosing another installation folder.**

Best to extract the FWP package on the Linux system itself and not via 7-ZIP, WinZip or something similar on a Windows machine. Using a Samba share to the Raspberry Pi may change file access rights in this case.

```
pi@decoder:~/forwardproxy $ pwd
/home/pi/forwardproxy
pi@decoder:~/forwardproxy $
```

A successful extraction should look like this. Make sure the ones listed above are present and have the appropriate execute rights. This is the “rwx” at the beginning of the line. Hit “ls -l” in the installation directory to find out.

```
-rwxr--r-- 1 pi pi  1207 Apr 20 10:28 config.txt
-rwxr--r-- 1 pi pi    49 Apr 20 10:26 decode1.sh
-rwxr--r-- 1 pi pi    65 Apr 20 10:26 decode2.sh
-rwxr--r-- 1 pi pi   120 Apr 20 10:26 forwardproxy-bg.sh
-rwxr--r-- 1 pi pi 25022 Apr 20 10:26 ForwardProxy.jar
-rwxr--r-- 1 pi pi    43 Apr 20 10:26 forwardproxy-restart.sh
-rwxr--r-- 1 pi pi    75 Apr 20 10:26 forwardproxy.sh
-rwxr--r-- 1 pi pi    64 Apr 20 10:26 forwardproxy-stop.sh
-rwxr--r-- 1 pi pi    21 Apr 20 10:26 killprocs.sh
```

Do not bother about all individual files. This will be explained later. If you have all of this, let us do three simple tests. First, we shall test if you succeed in starting FWP. Just go into the folder you extracted the package and run “java -jar ./ForwardProxy.jar”. If Java is not in your system path, you will need to use the absolute path as I did. If everything works fine, FWP should display the message below (version may be different). This proves you can start FWP, but that FWP is still missing the configuration file, which is ok.

Make sure you get this working properly before moving on. After successful completion of this step, it means Java and FWP are setup correctly.

```
pi@decoder:~/forwardproxy $ /home/pi/jdk/bin/java -jar ./ForwardProxy.jar
ForwardProxy V2.5.120421
========================
USAGE:
  ForwardProxy {configuration file}
pi@decoder:~/forwardproxy $
```

As a second test, let us see if FWP can read the by the package provided default configuration file.

From within the installation folder, type “java -jar ./ForwardProxy.jar ./config.txt”. This is similar to the first test, but now, we pass an argument, being the configuration file.

This will launch FWP. As FWP is passed an argument, it will open that file and parse it. The in the FWP packages configuration file obviously works and FWP should startup without failure and warning. This should give an output as listed below, listing the configured stations and mentioning it is listening for requests on port 9000 and that is has started the monitor on port 9001.

However, this still does not guarantee proper function of FWP. At this stage, FWP proves it can read the configuration file. It is as from the moment you connect to it via the Naim, it will also trigger the decoding process. Let us test this in a later phase.

```
pi@decoder:~/forwardproxy $ /home/pi/jdk/bin/java -jar ./ForwardProxy.jar ./config.txt
14/04 13:02:23 ForwardProxy V2.5.120421
14/04 13:02:23 ========================
14/04 13:02:23
ForwardProxy V2.5.120421
========================

Device type      naim
Logging          ON
Debug mode       OFF
Logfile size     1024 kB
I/O buffer size  8 kB
Content type     audio/wav

Stations
--------
    440Hz-Radio (440hz_radio)
    Chill Out Zone Plus (chill_out_zone)
    Intense Radio (intense_radio)
    Naim Classical (naim_classical)
    Naim Jazz (naim_jazz)
    Naim Radio (naim_radio)
    Radio Paradise Main Mix (radio_paradise_main)
    Radio Paradise Mellow Mix (radio_paradise_mellow)
    Radio Paradise World Mix (radio_paradise_world)
    Sector 80s (sector80s)
    Sector 90s (sector90s)
    The Cheese (the_cheese)
```

FWP requires two TCP ports, or sockets. One of them listens to the iRadio requests coming from the Naim; the other one is a monitoring port, providing you a basic webpage to check the status of FWP. The in the package provided configuration file binds the iRadio socket to port 9000 and the monitor port to 9001.

If one, or both, of these ports are already in use, you will get the message below. This may also be an indication an instance of FWP is already running (in the background).

```
14/04 18:36:11 Failed to bind ForwardProxy to port 9000: Address already in use
14/04 18:36:11 Failed to bind Monitor to port 9001: Address already in use
pi@decoder:~/forwardproxy $
```

If you are in this situation, press Ctrl-C to stop FWP. Next, edit the config.txt file and modify the "proxy_port" and "monitor_port" values. Try again until you get both ports not already in use.

If everything went ok, then let us do the third test and check the web page of the monitor interface.

In order to get there, you will need to know the IP address of the device running FWP. Typically, as you already connected to it, this should be the connection address specified in your terminal configuration. Although, this has not be like that. On a Raspberry Pi, you can have for example a wireless and a wired network interface active at the same time.

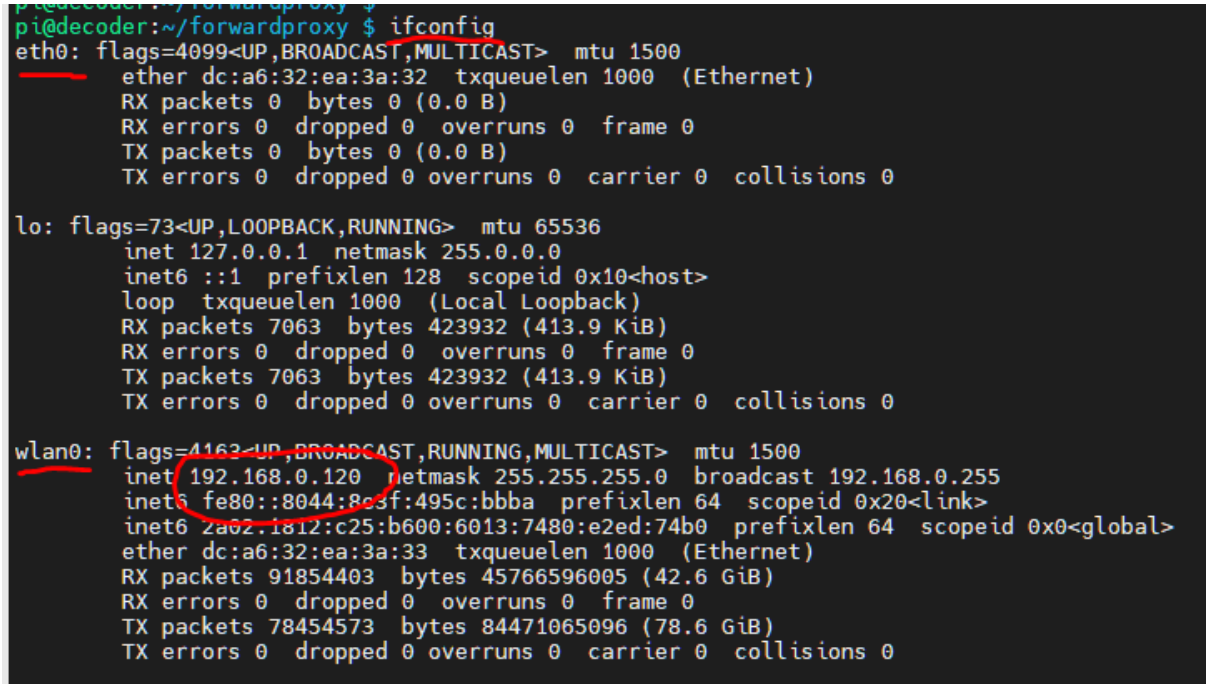
The address you need is the IP address that will be used to connect to the FWP application from your Naim device. You can figure this out by typing "ifconfig" on the command line. This should you give something similar as what is shown below. In my case, the Raspberry Pi is connected wireless (wlan0) and has the IP address 192.168.0.120. For sure, in your environment this will be different. If your device is connected via an Ethernet cable, than look at the eth0 interface. If you run it in the cloud, than look at your public IP address.

In case of a Cloud installation, make sure the configured FWP ports are open. This is NOT a requirement if you run it in your own local network!

It is recommend configuring this address as a "static" IP address, to avoid this address changes after a reboot of the device. Typically, this is no issue for Cloud based installation, but it is an attention point for, for example, Raspberry Pi installations, who are typically DHCP based.

Note following information down as you will need it afterwards:

- IP address of the device running FWP (**fwp-ip-address**): ……..
- FWP iRadio listening port (**fwp-radio-port**): …… (default: 9000)
- FWP monitor port (**fwp-monitor-port**): …… (default: 9001)



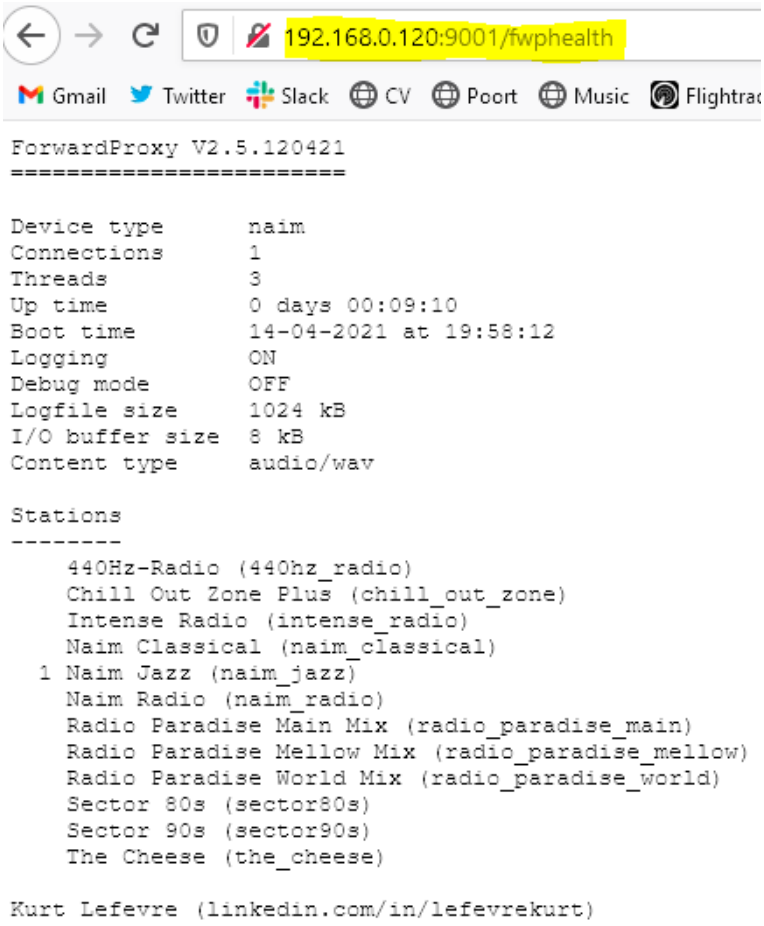
Ok. Now you should have FWP running and know the IP address of the device running FWP. Now, try to start the FWP monitoring page on http://fwp-ip-address:fwp-monitor-port/fwphealth . You have to replace “fwp-ip-address” with the IP address of the device running FWP; same for the port. See the example below.

Note **the protocol for the monitor is http and not https**. No need to worry about security. This only shows some information and no confidential information. In addition, http is a more lean-and-mean protocol, so less resource consuming than https.

This will display the FWP monitoring page. The interface is simply, yet useful and self-explanatory. The shown information is partially pulled from the configuration file, combined with the actual number of connected devices and consulted radio stations at that moment.

If some of the parameters look strange, do not panic, this will be explained in the “Configuration” section.

Time information is displayed in a 24h format as HH:MM:SS. Dates are listed in a DD-MM-YYYY format.



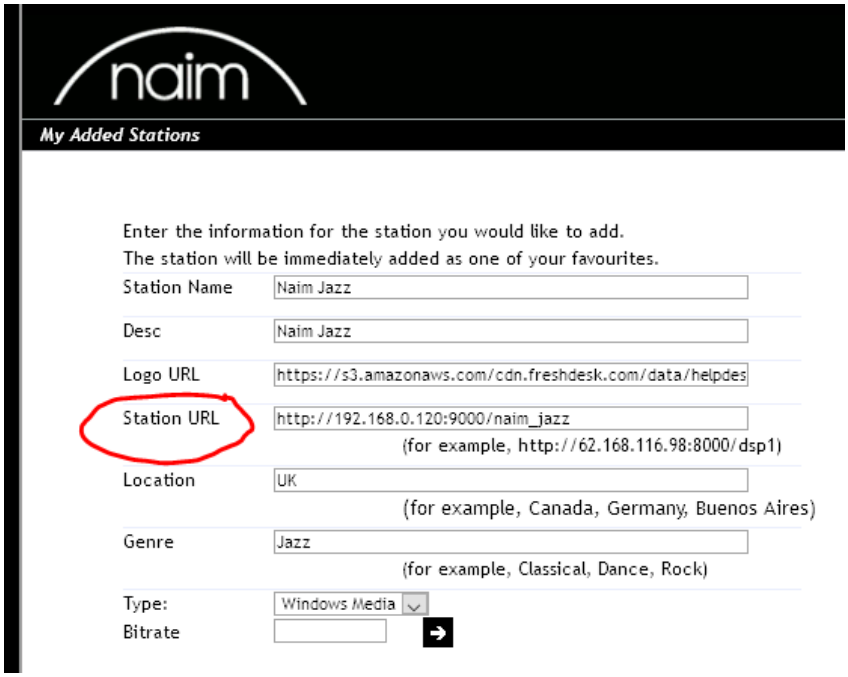
Ok, if you are still on track, let us try to connect to it from the Naim device… If you manage to get here, than you are ready to go (yourself). ☺

As we are still on a test, do not press Ctrl-C or Ctrl-D as this will kill your current session and the FWP application. Later on, we shall see how we can run away and leave FWP running. So, keep the terminal window open for now!

When using the by the FWP package provided configuration file, go to vTuner (https://naim.vtuner.com) and add a custom radio station. The most important parameter is the "Station URL". Other parameters are optional, some are mandatory, but are not important to FWP.

Remember, the Naim should NOT directly connect to the music service itself, but to FWP instead. Hence, the **FWP station URL has the following syntax: http://fwp-ip-address:fwp-radio-port/station-path** . Keep this in mind for later when the configuration file is covered in detail.

For now, do not bother about the "station-path" thing. As a test, add the following in vTuner: "http://fwp-ip-address:fwp-radio-port/naim_jazz", without the quotes. Note the underscore. You have to replace "fwp-ip-address" with the IP address of your device running FWP; same for the "fwp-radio-port".



Once this is done, go to the Naim app; find your just added station under “Added Stations” and select it; **it should start playing**. On your terminal screen, you should see the connection requests coming in.

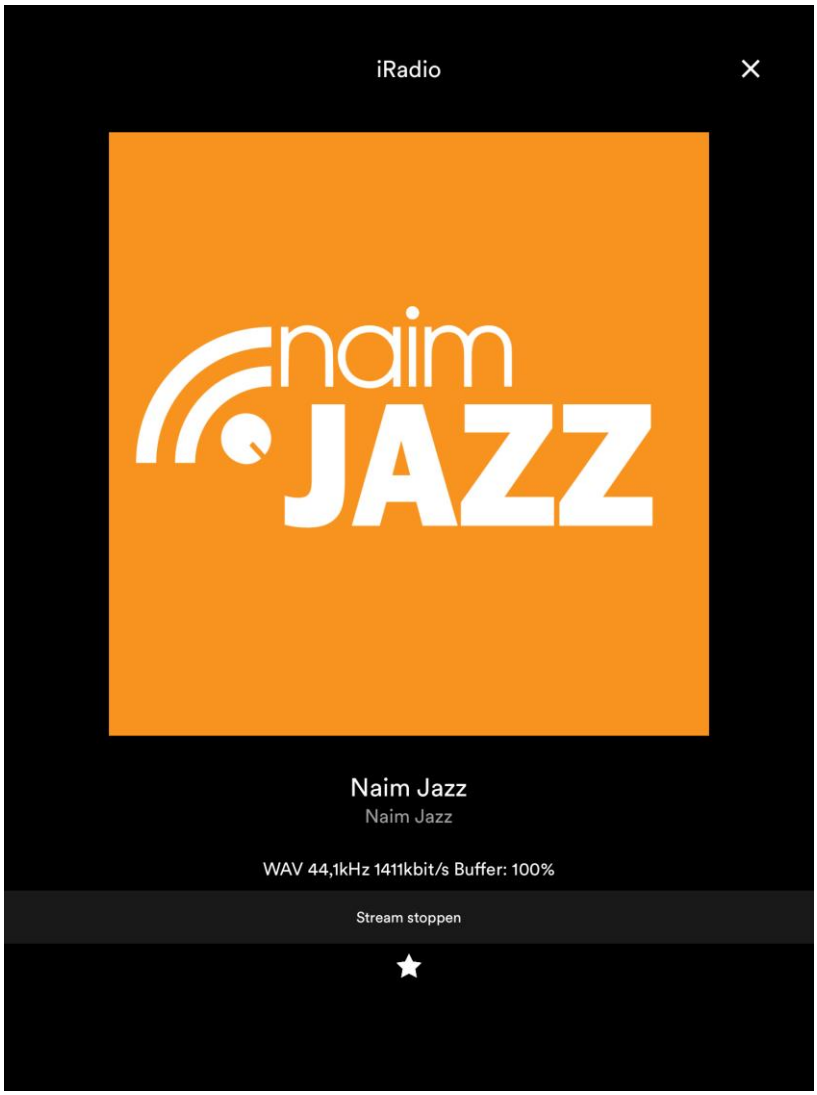
This is actually the part that cannot be tested without connecting a (Naim) device to FWP. In this phase, Java, ForwardProxy and FFmpeg are supposed to be installed and working correctly.

As the provided configuration file has no hard dependencies, this should work independent from the installation type.

In the Naim app, you should see something like the picture below. If you are missing the logo, do not panic. This is because you did not provide a picture URL in the “Logo URL” field. As this is only testing, nothing to worry about. There is plenty of time to fix this afterwards in vTuner.

When you refresh the FWP monitoring page, a “1” should appear next to the “Naim Jazz” radio station. The number on the left side of the station name in the “Stations” list represents the number of connected devices to the particular radio station.

Pressing Ctrl-D or Ctrl-C will stop FWP. We shall see later on how to keep it running.



However, if this fails…

- Note my remarks about the side effects of running it in a local network.
- As logging is active in the by the package provided configuration file, check for failures on screen, or in the created log file ”ForwardProxy_0.log” in the installation folder.
- Check whether a connection request came in. Firewall rules may block incoming connections. Sole option is to open the port(s) in the firewall configuration.
- Check whether there is no error dealing with launching the decoder.
- Check whether the streaming service URL is active (ping, wget, curl, browser, …)

If you cannot get it working via the by the package provided configuration file, drop me an email.

Configuration

The configuration file is crucial to make FWP work.

The configuration file is a free format plain vanilla text file. Therefore, this means you can create and edit it with your editor of choice and you structure it the way you like. Although, some rules have to be respected:

- All entries follow the “key = value” format.
- Blanc lines are allowed, but are not interpreted.
- Lines starting with a **#** character are not interpreted and are considered as comment.
- Sequence/order of keys does not matter.
- Some keys are mandatory, some keys are optional. Will be explained later.
- Optional keys will use default values when not defined in the configuration file.
- Optional known keys with unknown values will assume default values.
- Unknown keys are ignored.
- Mandatory known keys with unknown values terminate FWP.
- Missing mandatory keys (for whatever reason) will terminate FWP.
- The “STATION” key uses a comma as field delimiter. Names should not contain a comma.
- Key names are case insensitive. For example: station = Station = STATION.
- Values for MONITOR_PORT and PROXY_PORT should be different.

For performance reasons, the configuration file is parsed once at startup and kept in memory during execution of FWP.

Parse anomalies are reported during the startup process and FWP will (try to) continue. Exceptions and serious errors will stop FWP at startup.

Modifications to the configuration file of a running FWP instance require a restart of FWP.

Following keys are valid in the configuration file:

- ***PROXY_PORT***: Mandatory. FWP uses this port to listen to iRadio requests from the Naim and to return the Wav stream to it. This should be an (arbitrary chosen) numeric value. Values as from 5000 would do, unless the port is already in use by another application or by your operating system. If this is the case, an “already in use message” will be reported and FWP will stop. See above. The PROXY_PORT value has to be different from the MONITOR_PORT value.
- ***MONITOR_PORT***: Mandatory. FWP uses this TCP port for the monitoring interface. This should be an (arbitrary chosen) numeric value. Values as from 5000 would do, unless the port is already in use by another application or by your operating system. If this is the case, an “already in use message” will be reported and FWP will stop. See above. The MONITOR_PORT value has to be different from the PROXY_PORT value.
- ***CONTENT_TYPE***: Mandatory. This has to be set to “**audio/wav**”. The initial idea was to provide the possibility to send a Wav or a native Flac bit stream to the Naim. FWP is designed with this in mind. As I concluded the first generation Naim devices just do not support Flac as an iRadio format (see above), Wav is the sole possible option for now.
- ***STATION***: Mandatory. At least one station has to be defined. FWP will stop if it cannot find a station definition. Stations may be defined in random order, but will be sorted based on the friendly name. The STATION definition is a particular case and will be explained more in detail below.
- ***LOG***: Optional, default = false. By default, logging is turned off. The LOG key provides you useful information about incoming requests, lifetime of the requests, ip address of the incoming connection. The logged information is written to a file called

"ForwardProxy_0.log". When the log file size becomes more than 1024 kB, or a different value when specified via the LOGFILE_SIZE_KB key, a file called "ForwardProxy_1.log" will be created, containing the contents of "ForwardProxy_0.log" and the latter one will be truncated. In other words, "ForwardProxy_0.log" is always the most recent log file. There is a maximum of two log files. The log file content, e.g. *"**19/04 09:39:59 [531:1:1]** OggDecoder: Start for Radio Paradise Mellow Mix"*, always starts with the in bold printed information: date in day/month format, followed by the time in 24h format. The values between square brackets represent the thread ID, thread count and number of decoders running. The thread ID allows correlating messages, as FWP is fully multi-threaded. The remaining part of the logged content is free format.

- ***DEBUG***: Optional, default = false. By default, debugging is turned off. The DEBUG key provides you very detailed information about incoming and outgoing requests, start and stop of the decoding engine etc. Debug information is not written to the log file and is only display via the standard output; typical the console screen. As the DEBUG key generates a lot of information, debugging should only be turned on for troubleshooting or debugging purposes as it may slow down processing.
- ***LOGFILE_SIZE_KB***: Optional, default = 1024 kB. The size of the log file is by default limited at 1024 kB. The LOGFILE_SIZE_KB allows you to change that value. This only applies if logging is turned on.
- ***COMPATIBILITY_MODE***: Optional, default = false. FWP is written to support first generation Naim streaming devices. As above described, the handshake process amongst the streaming device and the music service is somewhat particular and differs amongst streaming devices. By setting COMPATIBILITY_MODE to false, FWP will be optimized for Naim devices. This setting is listed as "Naim" as "Device type" on the FWP monitoring page. In this configuration, the decoder is only started when the Naim sends the request to receive the bit stream. If you want to test with other streaming devices or applications, like VLC, you may need to set COMPATIBILITY_MODE to true to make it work. This setting is listed as "generic" as "Device type" on the monitoring page. In this configuration, the decoder is started for every request received from the streaming device. Setting COMPATIBILITY_MODE to true also works with Naim devices, but is slightly less performant.
- ***IO_BUFFER_SIZE***: Optional, default = 8 kB. FWP sends the bit stream in fixed sized chunks or buckets to the Naim. Using a buffer approach reduces possible interruptions. In case you suffer with hick-ups, increasing this value may help. Note that using a (too) large value will result in a longer wait time before the streaming device receives the bit steam, which could lead to a timeout. It actually has to do with network latency. There is no ideal value. Experimenting will make you know what is best for you, or just leave the default setting.

The ***STATION*** key requires some more explanation. The definition of the STATION entry in the configuration file is as follows:

STATION = station-path, friendly-name, music-service-URL, decode-script-number

Note: station-path, friendly-name and music-service-URL values should not contain a comma. The configuration file parser uses the comma as a value separator.

- ***station-path*****:** Remember from above, the Naim does NOT directly connect to the music service itself, but to FWP instead. **The station-path is a unique user-defined string used in the station configuration in vTuner**. The FWP station URL has the following syntax:

http://fwp-ip-address:fwp-radio-port/station-path. The station-path values are listed between brackets on the monitoring page after the station names. See example of the FWP monitoring page above.

- ***friendly-name:*** This a user-friendly definition of the radio station. The friendly-name value is used in log files and on the FWP monitoring page when displaying the stations. Stations are sorted based on this value. The comma is not allowed here, other values, including spaces are.
- ***music-service-URL:*** This represents the actual music service address. Both http and https protocols are supported. As http is a more lean-and-mean protocol, so less resource consuming than https, I recommend using http.
- ***decode-script-number:*** Different stations may require different decode settings due to, for example, different bit rates. **The decode-script-number, X, refers to the decodeX.sh script that will be launched**. In the example below, decode**1**.sh will be launched for the "Naim Jazz" and some other stations requiring the same decode settings (16 bit). Intense Radio provides 24-bit audio. For this stream, FWP will launch decode**2**.sh, containing specific 24-bit decode settings. This mechanism allows you to easily reuse decode settings, or define specific decode settings for specific streams. See below.

Below, you can find an example of a snapshot of the configuration file provided in the standard FWP package:

```
pi@decoder:~/forwardproxy $ cat config.txt
proxy_port = 9000
monitor_port = 9001
content_type = audio/wav
log = true
#logfile_size_kb = 512
#compatibility_mode = true
#debug = true
#io_buffer_size_kb = 1024

station = naim_jazz, Naim Jazz, http://mscp3.live-streams.nl:8340/jazz-flac.flac, 1
station = naim_radio, Naim Radio, http://mscp3.live-streams.nl:8360/flac.flac, 1
station = naim_classical, Naim Classical, http://mscp3.live-streams.nl:8250/class-flac.flac, 1
station = radio_paradise_main, Radio Paradise Main Mix, http://stream.radioparadise.com/flac, 1
station = radio_paradise_mellow, Radio Paradise Mellow Mix, http://stream.radioparadise.com/mellow-flac, 1
station = radio_paradise_world, Radio Paradise World Mix, http://stream.radioparadise.com/world-etc-flac, 1
station = intense_radio, Intense Radio, http://secure.live-streams.nl/flac.flac, 2
station = the_cheese, The Cheese, http://thecheese.ddns.net:8004/stream, 1
station = sector80s, Sector 80s, http://89.223.45.5:8000/geny-flac, 1
station = sector90s, Sector 90s, http://89.223.45.5:8000/next-flac, 1
station = chill_out_zone, Chill Out Zone Plus, http://chillout.zone/chillout_plus, 1
station = 440hz_radio, 440Hz-Radio, http://stream.440hz-radio.de:8080/440hz.flac.ogg, 1
pi@decoder:~/forwardproxy $
```

Starting FWP at boot time

It is convenient to start FWP automatically the moment the system starts up. Linux provides different ways of achieving this. To keep it simple, I put my start commands in the **/etc/rc.local** file. This approach works on Raspberry Pi, but may also work on other Linux distributions.

This may require sudo rights to edit. Just before the last line (exit 0), you have to add this in italics printed line: *su -c '/home/pi/forwardproxy/forwardproxy.sh /home/pi/forwardproxy/config.txt' - pi &*

This command line will execute in the background, as user “pi”, the forwardproxy.sh script using config.txt as parameter. To make things easy, I used absolute paths. In order to make this work, you have to adapt the paths to correspond to your settings. To find out if this works, you have to reboot the system.

Helper scripts

The FWP package contains, next to the required decode and kill scripts some useful helper scripts I wrote during the development of this application. Those scripts are not mandatory to run FWP and you can modify, enhance, enrich … at your convenience.

Note that all scripts ending on .sh require execute rights. This is the “rwx” you see when you list the folder contents (ls -l). If this is not the case, executing chmod 744 my-file, will fix that.

If you want to make use of these helper scripts, have a look at the path settings in the script, as I mostly used absolute paths.

- ***forwardproxy.sh:*** This script starts FWP on the command line, or on the foreground, and expects the configuration file as parameter. It abstracts the Java command line to start FWP. Pressing Ctrl-C or Ctrl-D will terminate FWP. This is a useful script to start FWP in debug mode.
- ***forwardproxy-bg.sh:*** This script starts FWP on the background, supposing config.txt is the name of configuration file. Pressing Ctrl-C or Ctrl-D will NOT terminate FWP. This is useful when you want to start FWP on the command line and terminate the terminal session afterwards. FWP works in normal circumstances this way.
- ***forwardproxy-stop.sh:*** This script stops the FWP running instance.
- ***forwardproxy-restart.sh:*** This script will restart FWP in the background, supposing config.txt is the name of configuration file. This is a composite script that first stops FWP by running forwardproxy-stop.sh, followed by starting FWP by running forwardproxy-bg.sh.

- THE END -