

# ESP32-radio V2.0

This document describes the realization of an Internet radio based on an ESP32 WiFi module. For the ESP32 an Internet radio was build in 1917 using a VS1053 mp3 decoder. Now the software is updated to support I2S output as well. A 1.8 TFT color display to give some information about the radio station that's playing. Other types of display are also supported, including NEXTION.

## Features:

- Can connect to thousands of Internet radio stations that broadcast MP3 or AAC audio streams.
- Output to VS1053, I2S<sup>1</sup>).
- MP3 playlists supported.
- Uses a minimal number of components.
- Has a preset list of a maximum of 100 favorite radio stations as preferences in flash.
- Can be controlled by a tablet or other device through the built-in webserver.
- Can be controlled by MQTT commands. Status info is published by using MQTT.
- Can be controlled by commands on the serial input.
- Can be controlled by IR.
- Can be controlled by rotary switch encoder.
- Up to 14 input ports can be configured as control buttons like volume, skip to the next/previous/first/favorite preset station.
- I/O pins for TFT, VS1053, I2S, IR and rotary encoder can be configured in preferences.
- The strongest available WiFi network is automatically selected. Passwords are kept as preferences in flash. Heavily commented source code, easy to add extra functionality.
- Debug information through serial output.
- 2<sup>nd</sup> processor is used for smooth playback.
- Update of software over WiFi (OTA) through PlatformIO IDE.
- Parameters like WiFi information and presets can be edited in the web interface.
- Preset, volume, bass and treble settings are saved in preferences.
- Displays time of day on TFT.
- Optional displays remaining battery charge.
- Bit-rates up to 320 kbps.
- Control of TFT back-light.
- PCB available.
- 3D printable box available.
- Support for Lilygo TTGO TM Music Albums with PCM5102A converter and 320x240 display.

## Restrictions for I2S and internal DAC output.

Restrictions for I2S output.

There is no tone control for I2S output. You have to use analog controls on the external amplifier.

Restrictions for DAC output.

There is no tone control for internal DAC . You have to use analog controls on the external amplifier. The internal DAC uses only 8 bits for the digital to analog conversion. This affects the sound quality.

## Bluetooth output.

If I2S output is enabled and you are using a PCM5102 converter, you can connect a “Bluetooth 5.0 4.2 Receiver Transmitter”. See [here](#). Due to technical limits, it is not possible to use the internal ESP bluetooth.

---

<sup>1</sup>See restrictions for I2S output  
ESP32 Radio V2 -- page 1



## Software:

The software for the radio is supplied as a PlatformIO project using the Arduino platform.

The following libraries are used:

- Wire
- SD(esp32) @^1.0.5
- knolleary/PubSubClient@^2.8
- adafruit/Adafruit ST7735 and ST7789 Library@^1.6.0
- adafruit/Adafruit BusIO@^1.9.3
- adafruit/Adafruit GFX Library@^1.107.4
- adafruit/Adafruit ILI9341 @ ^1.5.9
- djuseeq/Ch376msc @ ^1.4.4
- ESP Async WebServer@^1.2.3
- AsyncTCP@^1.1.1WiFi – for establishing the communication with WiFi, part of ESP32 core library

The data directory of the project contains the files for the web interface: index.html, favicon.ico, style.css, search.html, config.html, defaultprefs.txt, and about.html.

The directory structure of the project is developed for PlatformIO. If you want to build the project using the Arduino IDE, you should follow the instructions in chapter 17 of this document.

## Configuration in config.h.

In order to enable or disable various features the config.h file can be edited before compiling.

You can set a fixed WiFi network to simplify the initial setup, define the MP3 decoder, the display and the type of the rotary encoder.

Note that, if the Lilygo TTGO TM Music Albums module is used, both DEC\_HELIX must be defined as the MP3/AAC decoder and TTGOTM defined as the display.

## Preferences

Preferences for ESP32-Radio are kept in Flash memory (NVS).

You may change the contents of NVS with the “Config”-button in the web interface. An example of the contents is:

```
bat0    = 2178    # ADC for 0% battery capacity left
bat100  = 3256    # ACD for 100% battery capacity
#
clk_dst = 1
clk_offset = -1
clk_server = pool.ntp.org
#
gpio_00 = uppreset = 1
gpio_12 = upvolume = 2
gpio_13 = downvolume = 2
#
ir_40BF = upvolume = 2
ir_C03F = downvolume = 2
#
mqttbroker = none
mqttpasswd = none
mqttport = 1883
mqttuser = carol
#
pin_enc_clk = 25    # GPIO rotary encoder CLK
pin_enc_dt = 26     # GPIO rotary encoder DT
pin_enc_sw = 27     # GPIO rotary encoder SW
pin_ir = 35         # GPIO IR receiver VS1838B
pin_tft_cs = 15     # GPIO TFT CS
pin_tft_dc = 2      # GPIO TFT DC
pin_vs_cs = 5       # GPIO VS1053 CS
pin_vs_dcs = 16     # GPIO VS1053 DCS
pin_vs_dreq = 4     # GPIO VS1053 DREQ
#
preset = 5
preset_000 = 109.206.96.34:8100                # 0 - NAXI LOVE RADIO, Belgrade, Serbia
preset_001 = airspectrum.cdnstream1.com:8114/1648_128 # 1 - Easy Hits Florida
preset_002 = streamer.radio.co/sa7bf61271/listen # 2 - JB's Rock N Roll
preset_003 = airspectrum.cdnstream1.com:8000/1261_192 # 3 - Magic Oldies Florida
preset_004 = stream.laut.fm/kinderlieder-123        # 4 - Kinderlieder
preset_005 = icecast.omroep.nl:80/radio1-bb-mp3     # 5 - Radio 1, NL
preset_006 = icecast.omroep.nl:80/radio5-bb-mp3     # 6 - Radio 5, NL
preset_007 = icecast.omroep.nl:80/radio2-bb-mp3     # 7 - Radio 2, NL#
#
toneha = 0
tonehf = 0
tonela = 0
tonelf = 0
#
volume = 72
#
wifi_00 = ADSL-11_plus/xxxxxx
wifi_01 = NETGEAR-11/xxxxxx
wifi_02 = ADSL-11/xxxxxx
```

Note that the size of the NVS is limited to 20 kB.

Lines starting with “#” are comment lines and are ignored. Comments per line (after “#”) will also be ignored, except for the “preset\_” lines. The comments on the “preset\_” lines are used to identify the presets in the web interface.

The maximum line length is 150 characters.

Note that the preset numbers ranges from 000 to 199. The range can be extended by the definition of MAXPRESETS. Note that there is also a limit on the NVS space. If the highest numbered station is reached, the next station will be 00 again. URLs with mp3 files or mp3 playlists (.m3u) are allowed.

Preferences in the form "pin\_XXXX" specify pin numbers for the various interfaces. For a FEATERBOARD you have to change some settings, especially for the display. The pins for SPI default to SCK=18, MISO=19 and MOSI=23, but they may be configured by "pin\_spi\_XXX" commands.

The complete list of configurable interface pins is:

Configuration name	Function
pin_ir	Pin for IR sensor ( <small>VS1838B</small> )
pin_enc_clk	Rotary encoder CLK pin
pin_enc_dt	Rotary encoder DT pin
pin_enc_up	ZIPPY B5 side switch up
pin_enc_dwn	ZIPPY B5 side switch down
pin_enc_sw	Rotary encoder switch down
pin_tft_cs	Display SPI version CS pin
pin_tft_dc	Display SPI version DC pin
pin_tft_scl	Display I2C version SCL. For OLED, LCD1602, LCD2004
pin_tft_sda	Display I2C version SDA For OLED, LCD1602, LCD2004
pin_tft_bl	Display back-light. The BL is disabled after some time if there is no activity. This helps to reduce power when running on batteries.
pin_tft_blx	Display back-light (inversed logic)
pin_nxt_rx	NEXTION (input from NEXTION)
pin_nxt_tx	NEXTION (output to NEXTION)
pin_sd_cs	SD card select
pin_vs_cs	VS1053 CS pin
pin_vs_dcs	VS1053 DCS pin
pin_vs_dreq	VS1053 DREQ pin
pin_shutdown	Amplifier shutdown pin. Pin will be set if the radio is not playing or the volume is set to zero. This output-pin can be used to shut down the amplifier.
pin_shutdownx	Amplifier shutdown pin (inversed logic)
pin_i2s_bck	I2S BCK signal for I2S devices
pin_i2s_lck	I2S L(R)CK signal for I2S devices
pin_i2s_din	I2S DIN signal for I2S devices
pin_spi_sck	SPI bus SCK pin, default pin 18
pin_spi_miso	SPI bus MISO pin, default pin 19
pin_spi_mosi	SPI bus MOSI pin, default pin 16
pin_eth_mdc	For boards with Ethernet, MDC signal, default pin 23
pin_eth_mdio	For boards with Ethernet, MDIO signal, default pin 18
pin_eth_power	For boards with Ethernet, POWER signal, default pin 16

The "gpio\_" lines specify input pins and the command that is executed if the input pin goes from HIGH to LOW.

The "touch\_" lines specify the GPIO input pins for this signal and the command that is executed if the input pin is activated.

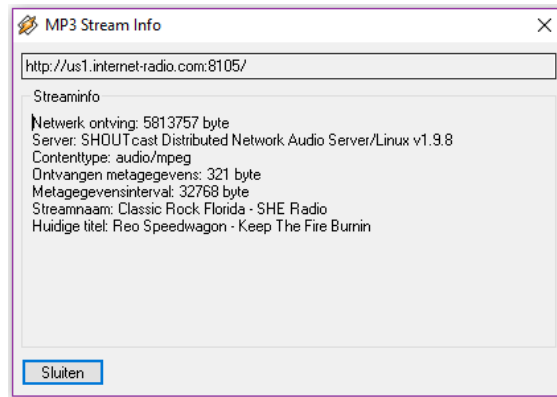
The "ir\_XXXX" lines specify IR-codes and the command that is executed if the IR-code is received. The code part is a hexadecimal number in upper-case characters.

The "clk\_" lines are used for the display of the current time on the TFT. Please change the values for your timezone. The values in the example are valid for the Netherlands.

The “bat0” and “bat100” lines are for displaying the remaining battery capacity on the display. See the details at the paragraph with optional features.

The preferences can be edited in the web interface. Changes will in some cases be effective after restart of the Esp-radio. If the list of preferences is empty (first start), you may use the default button. The list will show some default values which can be edited.

Using Winamp to find out the correct preset line for a station.



Press Alt-3 in the main window (left picture). You will see info for the playing station (right picture). The top line (with “http”) will contain the information for the preset, in this example: “us1.internet-radio.com:8105”. The complete line for this station in the preferences would be:

preset_05 = us1.internet-radio.com:8105	# 5 - Classic Rock Florida - SHE Radio
---	--

## Optional features:

### Digital control through input pins:

Normally the radio is controlled by the web interface. However, free digital inputs (GPIO) may be connected to buttons to control the radio. Their function can be programmed using the web interface.

You can assign commands to the digital inputs by adding lines in the configuration (Webinterface, "Config" page). Examples:

```
gpio_00 = uppreset = 1
gpio_12 = upvolume = 2
gpio_13 = downvolume = 2
gpio_21 = station = icecast.omroep.nl:80/radio1-bb-mp3
```

In this example the ESP32-Radio will execute the command "uppreset=1" if GPIO00 will go from HIGH to LOW. The commands are equal to the commands that are handled by the serial input or by the MQTT interface.

The same format can be used for the touch-inputs:

```
touch_04 = uppreset = 1
touch_13 = upvolume = 2
```

In this example the ESP32-Radio will execute the command "uppreset=1" if TOUCH0 (GPIO04) will be activated. The commands are equal to the commands that are handled by the serial input or by the MQTT interface.

### IR Interface.

The radio can be controlled by an IR remote control like this:



To use this interface, the "out" pin of a VS1838B receiver must be connected to a GPIO pin of the ESP32:



Add the assigned GPIO pin to the preferences through the config page in the web interface. Example:

```
pin_ir = 35 # GPIO Pin number for IR receiver VS1838B
```

VCC is connected to 3.3 Volt. A 220  $\mu$ F capacitor should be connected between VCC and GND.

The software will read the raw code of the IR transmitter, making it possible to use virtually any remote control to be used. I tested it with the 21 button remote as well as with an LG TV remote.

To assign functions to the buttons, watch the debug log output while pressing a button. For example, press the +volume button. You will see something like:

```
D: IR code 807F received, but not found in preferences!
```

Now add the command:

```
ir_807F = upvolume = 2
```

to the preferences in the config page of the web interface. Likewise you can assign functions to all buttons, for example:

```
ir_8A31 = uppreset = 1
ir_719A = station = us1.internet-radio.com:8105
ir_1F6B = mute
```

### Display remaining battery capacity:

The remaining battery capacity is computed by measuring the battery voltage on pin GPIO36 (ADC0).

The battery voltage must be supplied to ADC0 through a resistor voltage divider. The maximum



voltage fort his input is 1.0 volt. I used 100 k $\Omega$  and 22 k $\Omega$  fort his purpose. Furthermore you need to specify ADC value for both the full and the empty voltages. These values must be in the preferences like:

bat0	=	2178	#	ADC for 0% battery capacity left
bat100	=	2690	#	ACD for 100% battery capacity

To calibrate the settings you may use the “TEST” command in the serial monitor. This will display the current reading of the ADC0 pin. Do this for a fully charge battery and for an (almost) empty one. Note that it will take about 6 seconds before the reading is stable. This is a result of a filter in the measurement of the ADC.

The remaining capacity is displayed as a green/red bar near the top of the display.

## Hardware:

The radio is built with the following widely available hardware:

- An ESP-32 module. This is basically an ESP32 on a small print. I used a DOIT ESP32 Development Board. See figure 1 below. The ESP32 is running on 160 MHz. On Aliexpress: [this](#).
- An optional VS1053 module. See figure 2.
- An optional PCM5102, 1334A or similar I2S to analog module. See figure 14.
- A 1.8 inch color TFT display 160x128. Optional, see figure 3 and 4. On Aliexpress: [this](#) or [this](#).
- A 1.44 inch color TFT display 128x128. Optional, see figure 9. On Aliexpress: [this](#).
- An OLED 128x64 display. Optional, see figure 7. On Aliexpress: [this](#).
- An 1602 LCD display with I2C backpack. Optional, see figure 8.
- An 2004 LCD display with I2C backpack. Optional, see figure 13.
- A NEXTION display. Optional, see figure 10. Tested with a NX3224T024\_011.
- Two small speakers.
- A Class D stereo amplifier to drive the speakers. Best quality if powered by a separate power source.
- A rotary encoder switch. Optional, see figure 5.
- A ZIPPY B5 side switch. Optional. This switch may replace the rotary decoder. Define “ZIPPYB5” in config.h and define “pin\_enc\_dwn” and “pin\_enc\_sw” in the config page of the webinterface. See figure 15.
- A IR receiver. Optional, see figure 6.
- Two audio transformers. Optional, see figure 11.
- Optional CH376 drive for USB stick, see figure 12.
- Lilygo TTGO TM Music Albums ESP-32 module with integrated PCM5102A I2S converter and 320x240 display, see figure 16.



Fig. 1

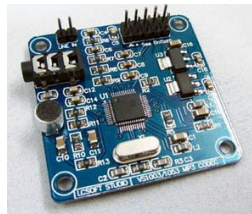


Fig. 2

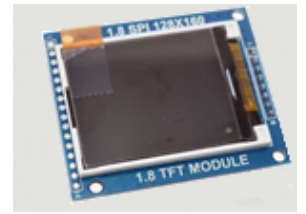


Fig. 3



Fig. 4



Fig. 5



Fig. 6



Fig. 7



8

Ffig.



Fig. 9



Fig. 10

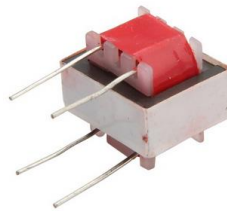


Fig. 11



Fig. 12



Fig. 13



Fig. 14

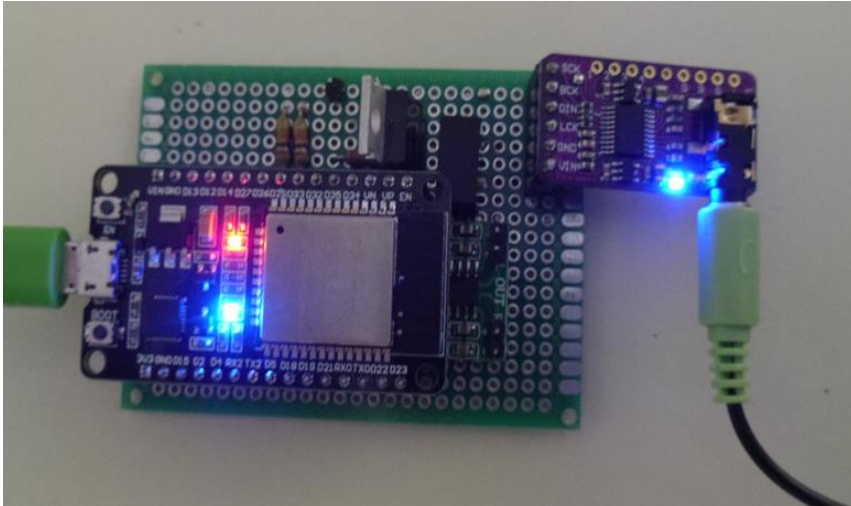


Fig. 15



Fig. 16

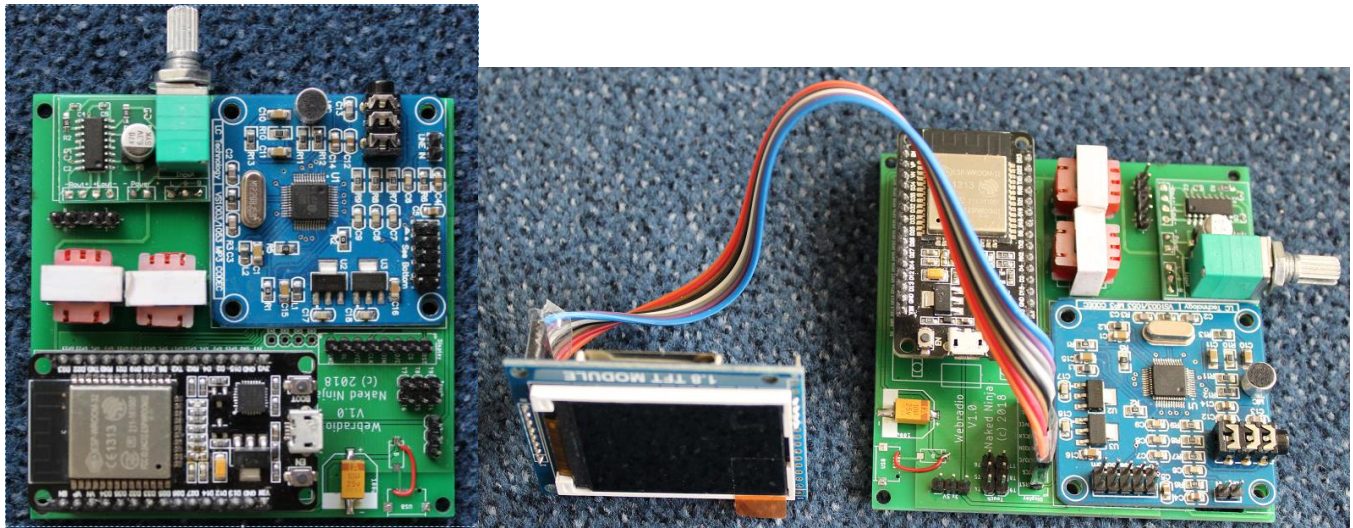
Here is a picture of the radio in a test configuration with software MP3 decoding. Only the PCM5102 is connected. There is also a PAM8303 mounted under the ESP32-board to test internal DAC output.





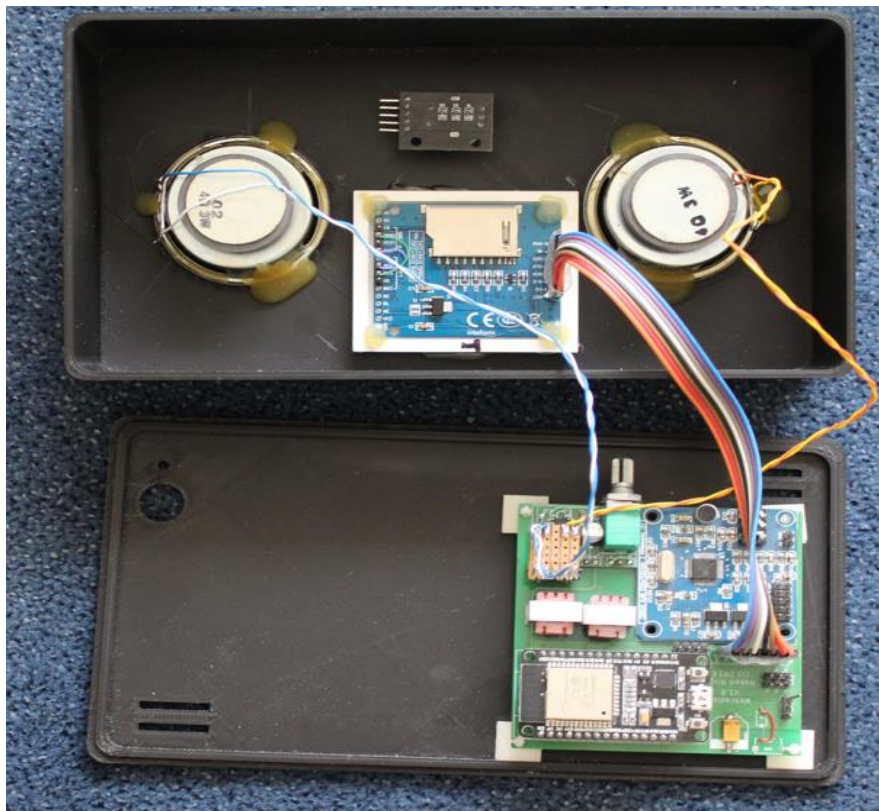
## PCB.

Harm Verbeek, see <https://github.com/Edzelf/ESP32-Radio/issues/198>, designed a PCB for this radio. I have built the radio on this PCB:



There is another PCB version available, see <https://github.com/Edzelf/ESP32-Radio/issues/165>.

Hans Vink designed a #D printable case for this radio. Mounted in tis case is looks like this:



See <https://www.thingiverse.com/thing:3346460>

Note that power supply and rotary switched are not connected here.

And playing:



## Wiring:

The logic wiring in the table below. The analog amplifier and the speakers are not included. Note that the GPIO pins are just an example. The assignment of GPIO pins is defined in the preferences that can be edited in the "config" page of the web interface.

ESP32dev	Signal	Wired to LCD	Wired to VS1053	Wired to the rest
-----	-----	-----	-----	-----
GPIO16		-	pin 1 XDCS	-
GPIO5		-	pin 2 XCS	-
GPIO4		-	pin 4 DREQ	-
GPIO2		pin 3 D/C or A0	-	-
GPIO22		-	-	-
GPIO21	RXD2	-	-	TX of NEXTION (if in use)
GPIO17	TXD2	-	-	RX of NEXTION (if in use)
GPIO18	SCK	pin 5 CLK or SCK	pin 5 SCK	-
GPIO19	MISO	-	pin 7 MISO	-
GPIO23	MOSI	pin 4 DIN or SDA	pin 6 MOSI	-
GPIO15		pin 2 CS	-	-
GPIO3	RXD0	-	-	Reserved serial input
GPIO1	TXD0	-	-	Reserved serial output
GPIO34	-	-	-	Optional pull-up resistor
GPIO35	-	-	-	Infrared receiver VS1838B
GPIO25	-	-	-	Rotary encoder CLK
GPIO26	-	-	-	Rotary encoder DT
GPIO27	-	-	-	Rotary encoder SW
-----	-----	-----	-----	-----
GND	-	pin 8 GND	pin 8 GND	Power supply GND
VCC 5 V	-	pin 7 BL	-	Power supply
VCC 5 V	-	pin 6 VCC	pin 9 5V	Power supply
EN	-	pin 1 RST	pin 3 XRST	-

The wiring of the 1.8 inch TFT module is on the right side of the board.

## Uploading firmware and data.

Find out what COM-poort is used by the USB to Serial convertor on the ESP32 board. That should be visible in the Windows device manager. In PlatformIO, edit the COM port settings. Example for COM9:

```
monitor_port = COM9  
upload_port = COM9
```

Then open a terminal openen in PlatformIO with this button:



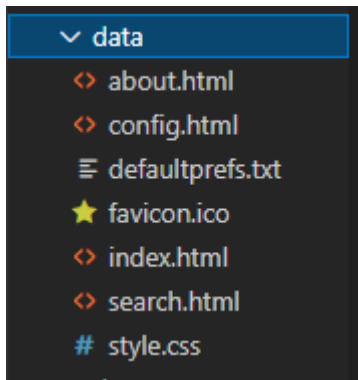
Flash the firmware with PlatformIO. Use this command in the terminal:

```
pio run -t upload
```

Now upload the filesystem for the web interface from the data directory with the command:

```
pio run -t uploadfs
```

Your data directory should look like this:



Start in PlatformIO the serial monitor with the button:



Press reset on the ESP32.

In the serial output you see lines like this:

D: Connect to WiFi

D: WiFi Failed! Trying to setup AP with name ESP32Radio and password ESP32Radio.

D: IP = 192.168.4.1

Connect to the access point “ESP32Radio” on your Windows machine. Password is “ESP32Radio”.

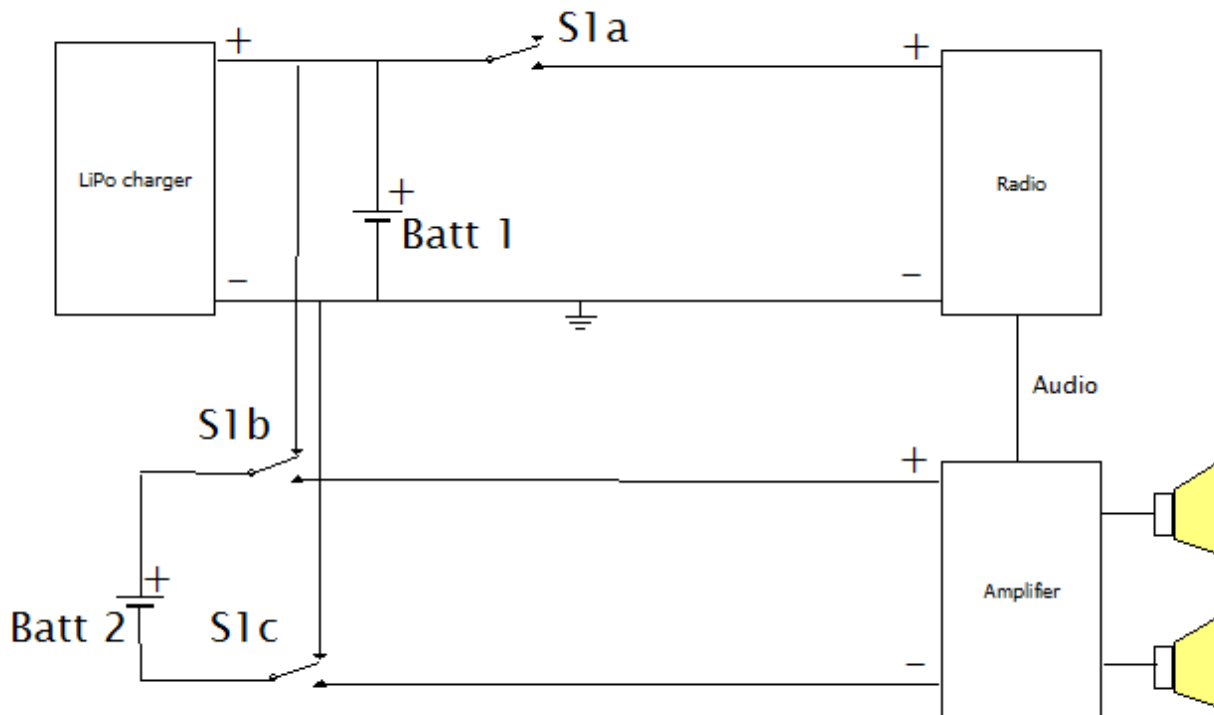
Open in your web browser “http://192.168.4.1”. You should see the home page of the radio.

Go to the config tab and do some configuration.



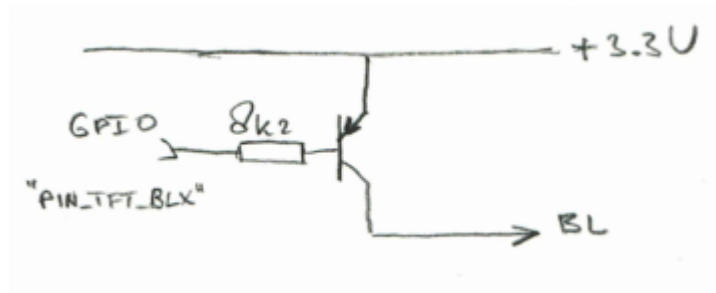
## Amplifier and power circuit (VS1053 version).

The amplifier is a class D stereo amplifier. If the power is shared with the power supply of the radio, you will hear much noise. So I used a separate LiPo battery (Batt 2) for the amplifier. During operation only Batt 1 will be charged. If the radio is switched off, both batteries will be recharged by the LiPo charger. S1a, S1b and S1c is a triple On-On switch. Note that there may be high currents in the “off”-position if one of the batteries fully discharged. Use protected batteries only!



A better approach is to use 2 small audio transformers to isolate the output of the VS1053 from the power circuit. This is used on the PCB made by Harm Verbeek.

## Backlight control circuit.



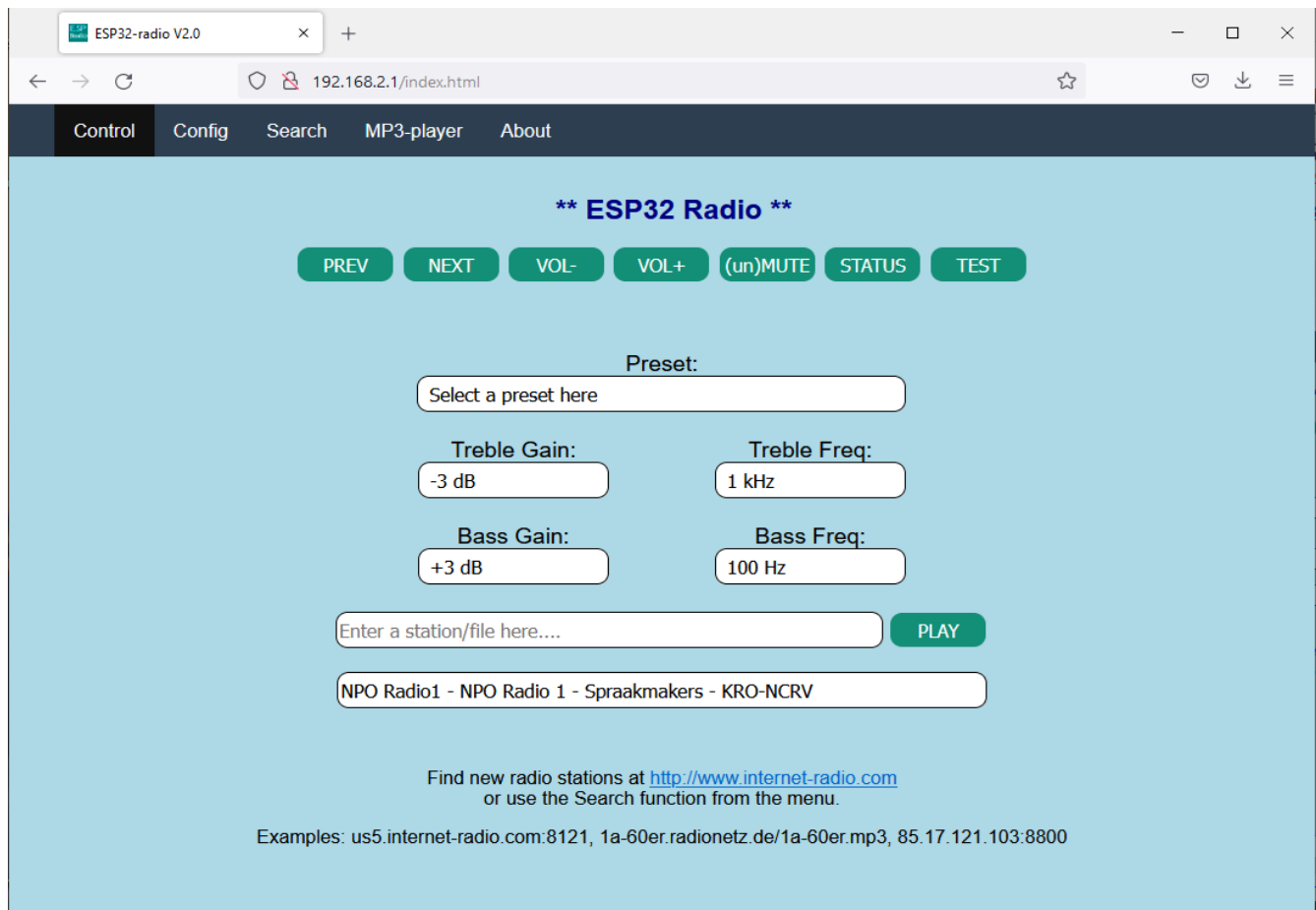
## Web interface.

The web interface is simple and can be adapted to your needs. The basic idea is to have a html page with embedded javascript that displays an interface to the radio. Command to the radio can be sent to the http server on the ESP32. The IP address of the webserver will be displayed on the TFT during startup. The webpages are defined in the data directory.

## Capabilities of the webserver:

Let's assume that the IP of the ESP32-radio is 192.168.2.12. From your browser you can show a simple root page by entering the following URL: `http://192.168.2.12`. This will display the `index.html` page from the ESP32 as well as `favicon.ico`.

If your computer is configured for mDNS, you can also use `http://ESP32Radio.local` in your browser. The following simple web interface will be displayed (VS1053 version):



Clicking on one of the available buttons will control the ESP32-radio. The reply of the webserver will be visible in the status box below the buttons. A click will be translated into a command to the ESP32-radio in the form:

`http://192.168.2.12/?<parameter>=<value>`

For example: `http://192.168.2.12/?upvolume=2`

The "STATUS" and "TEST" buttons give some info about the playing stream/file. The "Search" menu option can be used to discover stations. The list of stations is maintained by Community Radio Browser. See <https://www.radio-browser.info>.

Not all functions are available as buttons in the web interface shown above. Commands may also come from MQTT or serial input. Not all commands are meaningful on MQTT or serial input.

Working commands are:

preset	= 12	Select start preset to connect to
preset_00	= <mp3 stream>	Specify station for a preset 00-max *)
volume	= 95	Percentage between 0 and 100
upvolume	= 2	Add percentage to current volume
downvolume	= 2	Subtract percentage from current volume
toneha	= <0..15>	Setting treble gain
tonehf	= <0..15>	Setting treble frequency
tonela	= <0..15>	Setting bass gain
tonelf	= <0..15>	Setting treble frequency
station	= <mp3 stream>	Select new station (will not be saved)
station	= <URL>.mp3	Play standalone .mp3 file (not saved)
station	= <URL>.m3u	Select playlist (will not be saved)
mute		Mute/unmute the music (toggle)
wifi_00	= mySSID/mypassword	Set WiFi SSID and password *)
mqttbroker	= mybroker.com	Set MQTT broker to use *)
mqtttprefix	= XP93g	Set MQTT broker to use
mqtttport	= 1883	Set MQTT port to use, default 1883 *)
mqtttuser	= myuser	Set MQTT user for authentication *)
mqtttpasswd	= mypassword	Set MQTT password for authentication *)
clk_server	= pool.ntp.org	Time server to be used *)
clk_offset	= <-11..+14>	Offset with respect to UTC in hours *)
clk_dst	= <1..2>	Offset during dst in hours *)
settings		Returns setting like presets and tone
status		Show current URL to play
test		For test purposes
debug	= 0 or 1	Switch debugging on or off
reset		Restart the ESP32
bat0	= 2318	ADC value for an empty battery
bat100	= 2916	ADC value for a fully charged battery

Commands marked with "\*" are sensible during power-up only.

Station may also be of the form "skonto.ls.lv:8002/mp3". The default port is 80.

Station may also point to an mp3 playlist. Example: "www.rockantenne.de/webradio/rockantenne.m3u".

Station may be an .mp3-file on a remote server. Example: "www.stephaniequinn.com/Music/Rondeau.mp3".

It is allowed to have multiple (max 100) "preset\_" lines. The number after the "\_" will be used as the preset number. The comment part (after the "#") will be shown in the web interface.

It is also allowed to have multiple "wifi\_" lines. The strongest Wifi access point will be used.

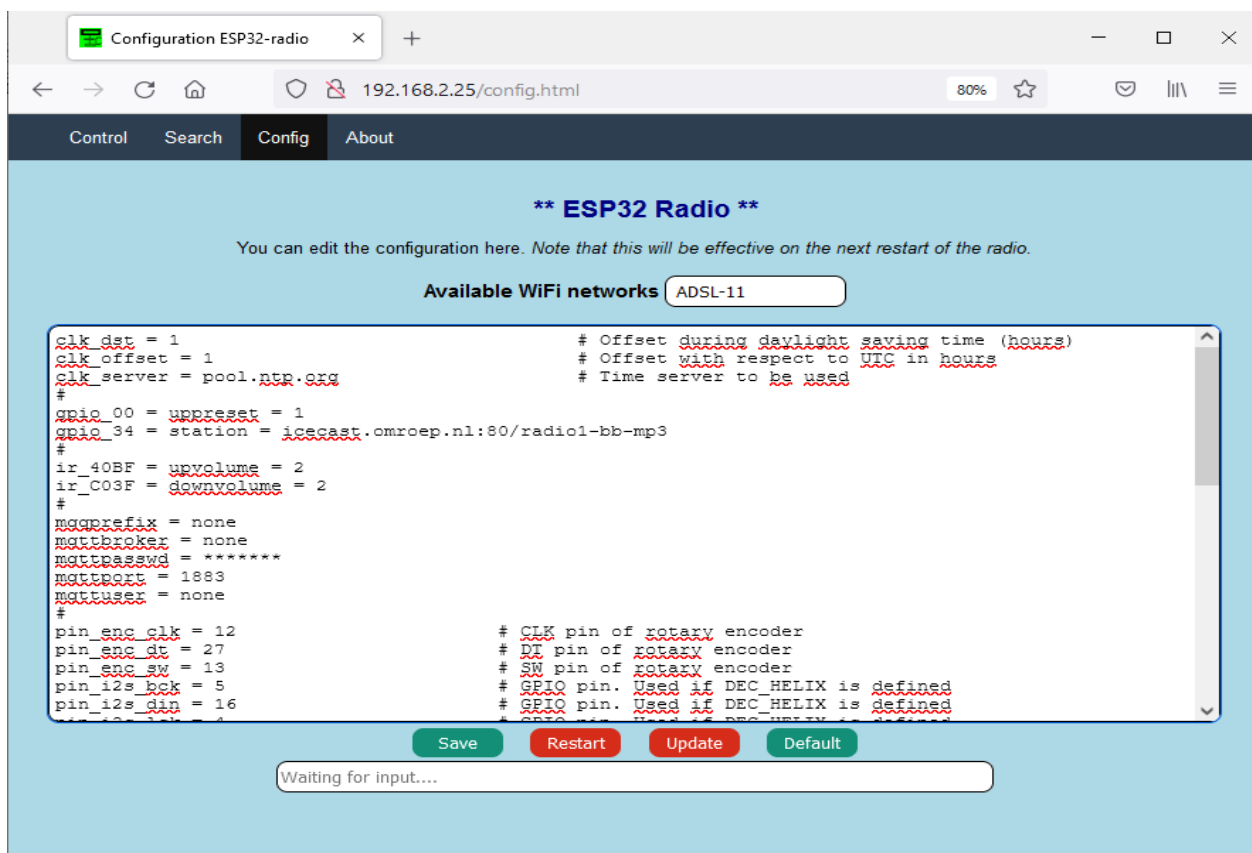
## Configuration

The "Config" button will bring up a second screen. Here you can edit the preferences. The available Wifi networks are listed as well. The config screen will be shown automatically if the ESP32 cannot connect to one of the WiFi stations specified in the preferences. In that case the ESP32 will act as an accesspoint with the name "ESP32Radio". You have to connect to this AP with password "ESP32Radio". Then the ESP32-radio can be reached at [http:// 192.168.4.1](http://192.168.4.1).

The configuration screen will be displayed and you have the opportunity to edit the preferences. A "default" button is available to fill the editbox with some example data. For a quick start, just fill in you WiFi network name and password and click "Save". Restart the radio afterwards.

Note that passwords are hidden in this screen to give some protection against abuses. You may however change them here. If unchanged, the original values are preserved.

After changing the contents of this page, it must be saved to the preferences by clicking the "Save" button. Changes will have effect on the next restart of the ESP-radio, so click the "Restart" button.



It may take some time to fill this screen.

## **Rotary encoder interface.**

The rotary encoder switch can control some essential functions of the ESP32-radio. The "GND"- and "+"-pins must be connected to ground and 3.3 Volt pins of the ESP32 DEV module. Note that some encoders do not have a "+" connection.

### **Rotary encoder volume control.**

The default function is volume control. Turning the knob will result in lower or higher volume. Pressing the knob will mute/unmute the signal. The text "Mute" or "Unmute" will be shown on the TFT during 4 seconds.

### **Rotary encoder preset mode.**

A double click selects the preset-mode. Rotation of the switch will select one of the preset stations. The preset will be shown on the TFT. Once a preset is selected, you can activate this preset by a single click.

Without rotation, the next preset is selected.

After an inactivity of 4 seconds the rotary encoder will return to its default function (VOLUME).

### **Rotary encoder SD track select.**

A triple click will select SD mode. Rotate to scroll through the list of tracks and press the knob to play the track.

### **Rotary encoder stop player.**

A long click will stop the radio/mp3player.

## Supported displays.

The preferred display is a 1.8 inch TFT with 160x128 pixels. Other displays may also be used:

- No display. Edit the sketch (around line 123), so that "DUMMYTFT" is defined and all other display types are commented out.
- The preferred display 160x128 pixels. Define "BLUETFT" and comment out the other types.
- An ILI9341 display, default 320x240 pixels. Define "ILI9341" and comment out the other types.
- A 128x128 1.44 inch display. Edit bluetft.h (around line 58), use "INISTR\_144GREENTAB" and set dsp\_getwidth() to 128.
- A ST7789 240 x 240 display. Define "ST7789" and comment out the other types.
- A 128x64 OLED. Define ""OLED" and comment out the other types.
- An LCD 1602 display with I2C backpack. Define "LCD1602I2C" and comment out the other types. Note that this display is limited as it has only 32 characters.
- An LCD 2004 display with I2C backpack. Define "LCD2004I2C" and comment out the other types. Note that this display is limited as it has only 80 characters.
- A NEXTON model NX#@ @\$T024\_11 display connected to GPIO 16 and 17.
- A 320x240 2.4 inch TFT on the Lilygo TTGO TM Music Albums module. Note that this must be defined as the TTGOTM display in config.h. Sample pin definitions are provided in file TTGO TM example config.txt

The OLED, 2004 and 1602 displays use the serial I2C bus. There must be definitions for this in the preferences, for example:

<pre>pin_tft_scl = 13 pin_tft_sda = 14</pre>	<pre># GPIO Pin number for SCL # GPIO Pin number for SDA</pre>
--	--

The I2C address of the 2004 and 1602 displays is defined in the corresponding module.

## Wired (ethernet) version.

A wired version of the radio is also possible. For testing I used a WT32-ETH01 module that has a built-in LAN8720 ethernet controller. The number of free pins is therefore limited. With the VS1053 and the BLUETFT display I used the following pin setting in the configuration page of the web interface:

```
pin_vs_cs = 2      # GPIO VS1053 CS
pin_vs_dcs = 12    # GPIO VS1053 DCS
pin_vs_dreq = 4    # GPIO VS1053 DREQ
#
pin_spi_sck = 5    # GPIO SPI SCK
pin_spi_miso = 33  # GPIO SPI MISO
pin_spi_mosi = 17  # GPIO SPI MOSI
#
pin_tft_cs = 15    # GPIO BLUETFT CS
pin_tft_dc = 14    # GPIO BLUETFT A0
```

This module has also been tested with I2S output. The pins used above for the VS1053 were used for the 3 I2S signals.

## AI Thinker Audio kit V2.1.

This board has an I2S DAC on-board and there are speaker outputs as well as an output for headphones.

The number of free GPIOs is limited. The internal DAC is a AC101. Note that later versions of this board may have a different DAC. The software works for the AC101 only!

GPIOs that are used internally are 2, 4, 13, 14, 15, 25, 26, 27, 32, 33.

GPIOs that are free and available at the 12 pin header are 21, 22, 19, 23, 18, 5.

If you want to use the KEYs, only 21 and 22 are free.

Since there are only a few GPIO pins are available, the best option for the human interface is probably a NEXTION display. For testing I used “pin\_nxt\_rx = 18” and “pin\_nxt\_tx = 5” in de config page of the web interface.

The GPIO configuration in the web interface could look like this:

pin_i2s_bck = 27	# AI GPIO Pin for I2S "BCK"
pin_i2s_din = 25	# AI GPIO Pin for I2S "DIN"
pin_i2s_lck = 26	# AI GPIO Pin for I2S "L(R)CK"
pin_nxt_rx = 18	# NEXTION serial input
pin_nxt_tx = 5	# NEXTION serial output
pin_sd_cs = 13	# AI GPIO Pin for Chip select SD card
pin_spi_miso = 2	# AI GPIO Pin for SPI MISO
pin_spi_mosi = 15	# AI GPIO Pin for SPI MOSI
pin_spi_sck = 14	# AI GPIO Pin for SPI SCK

Before compilation, change the config.h file like this:

Section “..MP3/AAC decoder”:

```
#define DEC_HELIX_AI // Software decoder for AI Audio kip (AC101)
```

Section “..display..”:

```
#define NEXTION // Nextion display
```

## Lilygo TTGO TM Music Albums

This board has a I2S DAC and a 320x240 TFT with an ST7789 controller on board  
The number of free GPIOs is limited. The internal DAC is a PCM5102A

GPIOs that are used internally are 2, 5, 13, 14, 15, 16, 17, 18, 19, 23, 25, 26, 37, 38, 39.

GPIOs that are free and available at the 24 pin header are 12, 27, 32, 33, 34, 35.

The GPIO configuration in the web interface could look like this:

pin_i2s_bck = 26	# TTGO GPIO Pin for I2S "BCK"
pin_i2s_din = 19	# TTGO GPIO Pin for I2S "DIN"
pin_i2s_lck = 25	# TTGO GPIO Pin for I2S "L(R)CK"
pin_sd_cs = 21	# TTGO GPIO Pin for Chip select SD card
pin_tft_cs = 5	# TTGO GPIO Pin for TFT CS
pin_tft_dc = 16	# TTGO GPIO Pin for TFT DC

Before compilation, change the config.h file like this:

Section “..MP3/AAC decoder”:

```
#define DEC_HELIX // Software decoder for MP3, AAC. I2S output
```

Section “..display..”:

```
#define TTGOTM // TTGO TM Music Albums
```



## MQTT interface.

The MQTT interface can handle the same commands as the web interface.

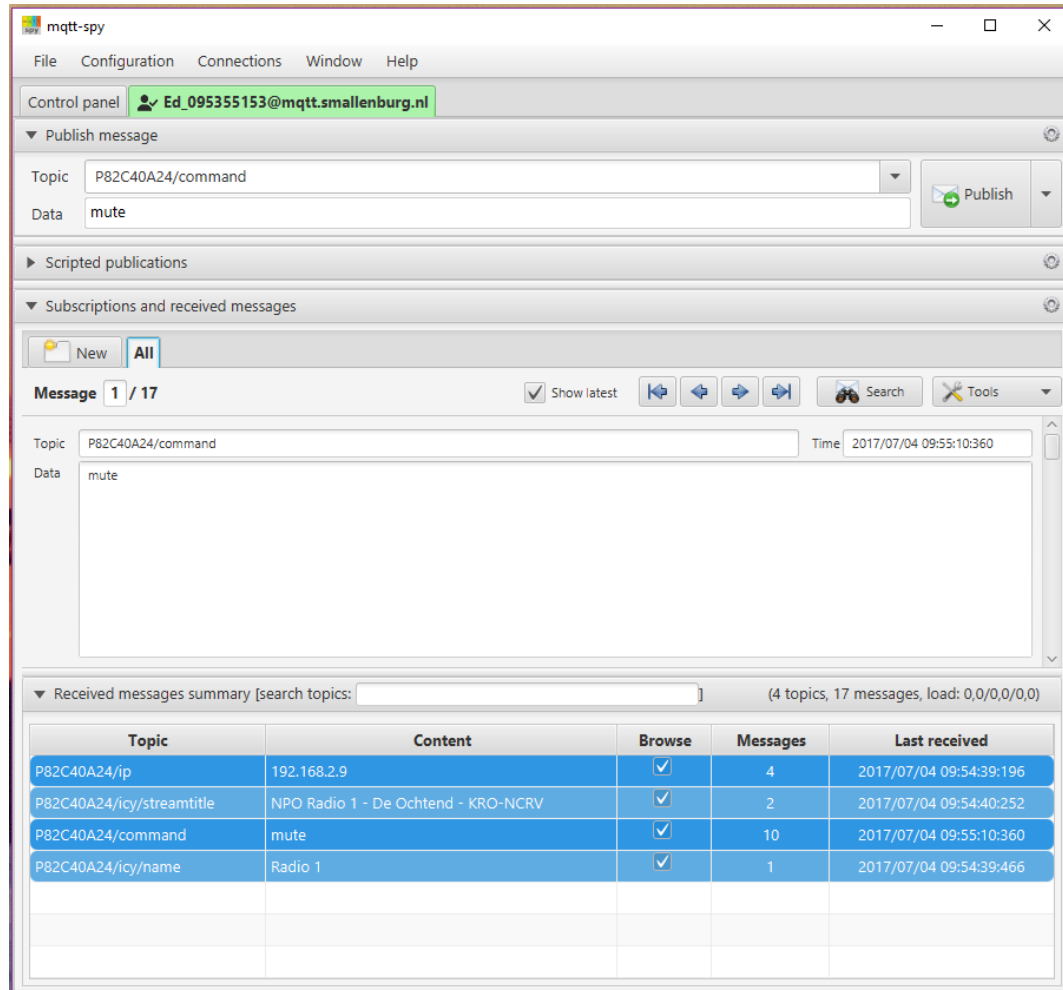
As publish command on a Linux system may look like:

```
$ mosquitto_pub -h broker.hivemq.com -t espradio -m volume=80
```

Note that `broker.hivemq.com` is heavily used, this may cause some delay. If you use your own broker the reaction on commands will be much better.

Remove the lines starting with “mqtt” if no MQTT is required or set `mqttbroker` to “none”.

You can use an MQTT client like <https://kamilfb.github.io/mqtt-spy/> to view the MQTT interface.



Subscribe to “P82C40A24/ip” and you will see the IP-address of your radio. You will see the IP address of your radio or the IP address of a different user. So be sure to use unique names for your topics. This can be accomplished by specifying a unique prefix in the preferences, see below. The parameters in preferences for this example are:

```
# MQTT broker, credentials and topic to subscribe
mqttbroker = broker.hivemq.com      # Broker to connect with
mqttprefix = P82C40A24              # Prefix for pub/sub. Default is part of MAC-address.
mqttport = 1883                     # Portnumber (1883 is default)
mqttuser = none                     # (No) username for broker
mqttpasswd = none                   # (No) password for broker
#
```

In this example I published the command “mute” to the radio. The radio published the IP-address 192.168.2.8 to the broker (once every 10 minutes).

MQTT PUB topics.

In this version 7 topics will be published to MQTT and can be subscribed to by an MQTT client:

prefix/ip	The IP-address of the ESP32-Radio
prefix/icy/name	The name of the station
prefix/icy/streamtitle	The name of the stream
prefix/nowplaying	Track information
prefix/preset	Preset currently plaining
prefix/volume	Current volume setting
prefix/playing	1 if playing, 0 if stopped

Command to control the ESP32-Radio can be published to "prefix/command".

Alternative way of configuration.

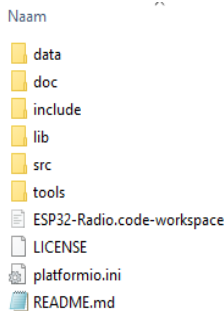
An extra sketch "Esp32\_radio\_init.ino" is supplied as an alternative to initialize the preferences (in Non-Volatile Storage of the ESP32). Just change lines 39 and 40 (the specs for WiFi networks) to match your network(s).

Upload and run the sketch once and then load the ESP32-Radio.

## Compiling with Arduino IDE.

The preferred way to compile this project is to using PlatformIO. For the ArduinoIDE, some extra steps must be taken:

- Download the project from github (<https://github.com/Edzelf/ESP32Radio-V2/archive/refs/tags/main.zip>). Store the zip file in any directory.
- Go to this directory and unzip the zip file. This will produce a directory with the name "ESP32Radio-V2-main".
- Rename "ESP32Radio-V2-main" to "ESP32Radio". That will be your work directory. It will



have this structure:

- Goto this directory and remove "doc", "tools", "ESP32-Radio.code-workspace", "LICENCE", "platformio.ini" and "README.md"
- Move the 3 files in "src" to your work directory. And remove the (now empty) "src" directory.
- Move all files from "include" to your work directory and remove the "include" directory.
- Edit the file config.h to meet your configuration (display, codec, ..).
- Move the files in "lib/codecs/src" to your work directory. If you use VS1053, you may skip the aac\* and mp3\* files.
- Depending on your type of display, move the files in "lib/ILI9341/src" to your work directory, or move the files in "lib/LCD1602/src" to your work directory, or move the files in "lib/LCD2004/src" to your work directory, or move the files in "lib/NEXTION/src" to your work directory, or move the files in "lib/bluetft/src" to your work directory, or move the files in "lib/dummytft/src" to your work directory, or move the files in "lib/OLED/src" to your work directory, or move the files in "lib/ST7789/src" to your work directory, or move the files in "lib/TTGOTM/src" to your work directory.
- Remove the "lib" directory if you want to save some disk space.
- Rename the file "main.cpp" to "ESP32Radio.ino". The Arduino IDE should start.
- Your work directory will now contain "data", "doc", "nextion", "tools", the source files and maybe "lib".
- Start the Arduino IDE by double clicking "ESP32Radio.ino".
- Select the right board (ESP32 Dev Module) in Tools and set the right Port.
- Edit the "config.h" file to your needs.
- You can now verify the sketch. You probably have to install some libraries.

### Some of the libraries used (depends on configuration):

WiFi at version 2.0.0  
PubSubClient at version 2.8  
ESPmDNS at version 2.0.0  
SPI at version 2.0.0  
ArduinoOTA at version 2.0.0  
Update at version 2.0.0  
ESPAsyncWebServer-master at version 1.2.0  
FS at version 2.0.0  
AsyncTCP-master at version 1.0.3  
SPIFFS at version 2.0.0  
Adafruit\_ST7735\_and\_ST7789\_Library at version 1.7.4  
Adafruit\_GFX\_Library at version 1.10.12  
Adafruit\_BusIO at version 1.9.3  
Wire at version 2.0.0

# Compiler/upload output.

## Software upload.

Processing esp32 (platform: espressif32; board: esp32doit-devkit-v1; framework: arduino)

```
-----
-Verbose mode can be enabled via `-v, --verbose` option
CONFIGURATION: https://docs.platformio.org/page/boards/espressif32/esp32doit-devkit-v1.html
PLATFORM: Espressif 32 (3.5.0) > DOIT ESP32 DEVKIT V1
HARDWARE: ESP32 240MHz, 320KB RAM, 4MB Flash
DEBUG: Current (esp-prog) External (esp-prog, iot-bus-jtag, jlink, minimodule, olimex-arm-usb-ocd,
olimex-arm-usb-ocd-h, olimex-arm-usb-tiny-h, olimex-jtag-tiny, tumpa)
PACKAGES:
- framework-arduinoespressif32 3.10006.210326 (1.0.6)
- tool-esptoolpy 1.30100.210531 (3.1.0)
- tool-mkspiffs 2.230.0 (2.30)
- toolchain-xtensa32 2.50200.97 (5.2.0)
LDF: Library Dependency Finder -> https://bit.ly/configure-pio-ldf
LDF Modes: Finder ~ chain, Compatibility ~ soft
Found 46 compatible libraries
Scanning dependencies...
Dependency Graph
|-- <Wire> 1.0.1
|-- <SD(esp32)> 1.0.5
|   |-- <FS> 1.0
|   |-- <SPI> 1.0
|-- <PubSubClient> 2.8.0
|-- <Adafruit BusIO> 1.11.3
|   |-- <Wire> 1.0.1
|   |-- <SPI> 1.0
|-- <Adafruit GFX Library> 1.10.14
|   |-- <SPI> 1.0
|   |-- <Adafruit BusIO> 1.11.3
|       |-- <Wire> 1.0.1
|       |-- <SPI> 1.0
|       |-- <Wire> 1.0.1
|-- <Adafruit ILI9341> 1.5.10
|   |-- <Adafruit GFX Library> 1.10.14
|       |-- <SPI> 1.0
|       |-- <Adafruit BusIO> 1.11.3
|           |-- <Wire> 1.0.1
|           |-- <SPI> 1.0
|           |-- <Wire> 1.0.1
|           |-- <SPI> 1.0
|-- <Adafruit ST7735 and ST7789 Library> 1.9.3
|   |-- <Adafruit GFX Library> 1.10.14
|       |-- <SPI> 1.0
|       |-- <Adafruit BusIO> 1.11.3
|           |-- <Wire> 1.0.1
|           |-- <SPI> 1.0
|           |-- <Wire> 1.0.1
|           |-- <SPI> 1.0
|-- <ESP Async WebServer> 1.2.3
|   |-- <AsyncTCP> 1.1.1
|   |-- <FS> 1.0
|   |-- <WiFi> 1.0
|-- <AsyncTCP> 1.1.1
|-- <codecs>
|   |-- <SPI> 1.0
|-- <bluetft>
|   |-- <Adafruit ST7735 and ST7789 Library> 1.9.3
|       |-- <Adafruit GFX Library> 1.10.14
|           |-- <SPI> 1.0
|           |-- <Adafruit BusIO> 1.11.3
|               |-- <Wire> 1.0.1
|               |-- <SPI> 1.0
|               |-- <Wire> 1.0.1
|               |-- <SPI> 1.0
|-- <dummytft>
|-- <ILI9341>
|   |-- <Adafruit ILI9341> 1.5.10
|       |-- <Adafruit GFX Library> 1.10.14
|           |-- <SPI> 1.0
|           |-- <Adafruit BusIO> 1.11.3
|               |-- <Wire> 1.0.1
```

```

| | | | | |-- <SPI> 1.0
| | | | | |-- <Wire> 1.0.1
| | | | | |-- <SPI> 1.0
|-- <LCD1602>
|-- <LCD2004>
|-- <NEXTION>
|-- <OLED>
|-- <ST7789>
| | |-- <Adafruit ST7735 and ST7789 Library> 1.9.3
| | | |-- <Adafruit GFX Library> 1.10.14
| | | | |-- <SPI> 1.0
| | | | |-- <Adafruit BusIO> 1.11.3
| | | | | |-- <Wire> 1.0.1
| | | | | |-- <SPI> 1.0
| | | | | |-- <Wire> 1.0.1
| | | | | |-- <SPI> 1.0
|-- <ArduinoOTA> 1.0
| | |-- <Update> 1.0
| | |-- <WiFi> 1.0
| | |-- <ESPmDNS> 1.0
| | | |-- <WiFi> 1.0
|-- <ESPmDNS> 1.0
| | |-- <WiFi> 1.0
|-- <SPI> 1.0
|-- <SPIFFS> 1.0
| | |-- <FS> 1.0
|-- <WiFi> 1.0
|-- <FS> 1.0
Building in release mode
Compiling .pio\build\esp32\src\main.cpp.o
Linking .pio\build\esp32\firmware.elf
Retrieving maximum program size .pio\build\esp32\firmware.elf
Checking size .pio\build\esp32\firmware.elf
Advanced Memory Usage is available via "PlatformIO Home > Project Inspect"
RAM: [== ] 18.0% (used 58932 bytes from 327680 bytes)
Flash: [===== ] 85.5% (used 1120346 bytes from 1310720 bytes)
Building .pio\build\esp32\firmware.bin
esptool.py v3.1
Merged 1 ELF section
Configuring upload protocol...
AVAILABLE: esp-prog, espota, esptool, iot-bus-jtag, jlink, minimodule, olimex-arm-usb-ocd, olimex-arm-usb-ocd-h, olimex-arm-usb-tiny-h, olimex-jtag-tiny, tumpa
CURRENT: upload_protocol = esptool
Looking for upload port...
Use manually specified: COM3
Uploading .pio\build\esp32\firmware.bin
esptool.py v3.1
Serial port COM3
Connecting.....
Chip is ESP32-D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: cc:50:e3:95:88:90
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Flash will be erased from 0x00001000 to 0x00005fff...
Flash will be erased from 0x00008000 to 0x00008fff...
Flash will be erased from 0x0000e000 to 0x0000ffff...
Flash will be erased from 0x00010000 to 0x00121fff...
Compressed 17104 bytes to 11191...
Writing at 0x00001000... (100 %)
Wrote 17104 bytes (11191 compressed) at 0x00001000 in 0.6 seconds (effective 234.7 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 128...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (128 compressed) at 0x00008000 in 0.1 seconds (effective 394.0 kbit/s)...
Hash of data verified.
Compressed 8192 bytes to 47...
Writing at 0x0000e000... (100 %)
Wrote 8192 bytes (47 compressed) at 0x0000e000 in 0.1 seconds (effective 478.3 kbit/s)...
Hash of data verified.

```

```
Compressed 1120448 bytes to 644211...
Writing at 0x00010000... (2 %)
Writing at 0x0001a3c4... (5 %)
Writing at 0x00027b54... (7 %)
Writing at 0x00032214... (10 %)
..
..
Writing at 0x00113ca2... (95 %)
Writing at 0x00119941... (97 %)
Writing at 0x0011f4c0... (100 %)
Wrote 1120448 bytes (644211 compressed) at 0x00010000 in 15.5 seconds (effective 576.9 kbit/s)...
Hash of data verified.
```

```
Leaving...
Hard resetting via RTS pin...
```

## Filesystem upload.

```
PS D:\docs\PlatformIO\Projects\ESP32-Radio> pio run -t uploadfs
Processing esp32doit-devkit-v1 (platform: espressif32; board: esp32doit-devkit-v1; framework: arduino)
-----
-----Verbose mode can be enabled via `-v, --verbose` option
CONFIGURATION: https://docs.platformio.org/page/boards/espressif32/esp32doit-devkit-v1.html
PLATFORM: Espressif 32 (3.3.2) > DOIT ESP32 DEVKIT V1
HARDWARE: ESP32 240MHz, 320KB RAM, 4MB Flash
DEBUG: Current (esp-prog) External (esp-prog, iot-bus-jtag, jlink, minimodule, olimex-arm-usb-ocd, olimex-arm-usb-ocd-h, olimex-arm-usb-tiny-h, olimex-jtag-tiny, tumpa)
PACKAGES:
- framework-arduinoespressif32 3.10006.210326 (1.0.6)
- tool-esptoolpy 1.30100.210531 (3.1.0)
- tool-mkspiffs 2.230.0 (2.30)
- toolchain-xtensa32 2.50200.97 (5.2.0)
LDF: Library Dependency Finder -> http://bit.ly/configure-pio-ldf
LDF Modes: Finder ~ chain, Compatibility ~ soft
Found 47 compatible libraries
Scanning dependencies...
Dependency Graph
|-- <Wire> 1.0.1
|-- <SD(esp32)> 1.0.5
|   |-- <FS> 1.0
|   |   |-- <SPI> 1.0
|-- <PubSubClient> 2.8.0
|-- <Adafruit ST7735 and ST7789 Library> 1.7.4
|   |-- <Adafruit GFX Library> 1.10.12
|   |   |-- <SPI> 1.0
|   |   |   |-- <Adafruit BusIO> 1.9.3
|   |   |   |   |-- <Wire> 1.0.1
|   |   |   |   |-- <SPI> 1.0
|   |   |   |   |-- <Wire> 1.0.1
|   |   |   |   |-- <SPI> 1.0
|-- <Adafruit BusIO> 1.9.3
|   |-- <Wire> 1.0.1
|   |-- <SPI> 1.0
|-- <Adafruit GFX Library> 1.10.12
|   |-- <SPI> 1.0
|   |   |-- <Adafruit BusIO> 1.9.3
|   |   |   |-- <Wire> 1.0.1
|   |   |   |-- <SPI> 1.0
|   |   |   |-- <Wire> 1.0.1
|-- <Adafruit ILI9341> 1.5.9
|   |-- <Adafruit GFX Library> 1.10.12
|   |   |-- <SPI> 1.0
|   |   |   |-- <Adafruit BusIO> 1.9.3
|   |   |   |   |-- <Wire> 1.0.1
|   |   |   |   |-- <SPI> 1.0
|   |   |   |   |-- <Wire> 1.0.1
|   |   |   |   |-- <SPI> 1.0
|-- <Ch376msc> 1.4.5
|   |-- <SPI> 1.0
|-- <ESP Async WebServer> 1.2.3
|   |-- <AsyncTCP> 1.1.1
|   |-- <FS> 1.0
|   |-- <WiFi> 1.0
|-- <AsyncTCP> 1.1.1
|-- <codecs>
|   |-- <SPI> 1.0
|-- <bluetooth>
|   |-- <Adafruit ST7735 and ST7789 Library> 1.7.4
|   |   |-- <Adafruit GFX Library> 1.10.12
|   |   |   |-- <SPI> 1.0
|   |   |   |   |-- <Adafruit BusIO> 1.9.3
|   |   |   |   |   |-- <Wire> 1.0.1
|   |   |   |   |   |-- <SPI> 1.0
```

```

| | | |-- <Wire> 1.0.1
| | |-- <SPI> 1.0
|-- <dummytft>
|-- <ILI9341>
| |-- <Adafruit ILI9341> 1.5.9
| | |-- <Adafruit GFX Library> 1.10.12
| | | |-- <SPI> 1.0
| | | |-- <Adafruit BusIO> 1.9.3
| | | | |-- <Wire> 1.0.1
| | | | |-- <SPI> 1.0
| | | |-- <Wire> 1.0.1
| | |-- <SPI> 1.0
|-- <LCD1602>
|-- <LCD2004>
|-- <NEXTION>
|-- <ArduinoOTA> 1.0
| |-- <Update> 1.0
| |-- <WiFi> 1.0
| |-- <ESPmDNS> 1.0
| | |-- <WiFi> 1.0
|-- <ESPmDNS> 1.0
| |-- <WiFi> 1.0
|-- <SPI> 1.0
|-- <SPIFFS> 1.0
| |-- <FS> 1.0
|-- <WiFi> 1.0
Building in release mode
Building SPIFFS image from 'data' directory to .pio\build\esp32doit-devkit-v1\spiffs.bin
/about.html
/config.html
/defaultprefs.txt
/favicon.ico
/index.html
/search.html
/style.css
Looking for upload port...
Use manually specified: COM4
Uploading .pio\build\esp32doit-devkit-v1\spiffs.bin
esptool.py v3.1
Serial port COM4
Connecting....
Chip is ESP32-D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: 24:6f:28:b0:7b:e0
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Flash will be erased from 0x00290000 to 0x003ffffff...
Compressed 1507328 bytes to 12150...
Writing at 0x00290000... (100 %)
Wrote 1507328 bytes (12150 compressed) at 0x00290000 in 8.0 seconds (effective 1505.9 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...

```



# Debug (serial 115200 Baud) output.

This is an example of the debug output:

```
D: Starting ESP32-radio running on CPU 1 at 240 MHz.  Version Thu, 14 Apr 2022 06:50:00 GMT.  Free memory 113792
D: Display type is BLUETFT
D: SPIFFS is okay, space 1378241, used 33634
D: Found partition 'nvs'      ' at offset 0x009000 with size      20480
D: Found partition 'otadata'  ' at offset 0x00E000 with size      8192
D: Found partition 'spiffs'   ' at offset 0x290000 with size    1507328
D: Read 38 keys from NVS
D: pin_ir set to 35
D: pin_enc_clk set to 12
D: pin_enc_dt set to 27
D: pin_enc_sw set to 13
D: pin_tft_cs set to 15
D: pin_tft_dc set to 2
D: pin_sd_cs set to 14
D: pin_vs_cs set to 5
D: pin_vs_dcs set to 16
D: pin_vs_dreg set to 4
D: pin_spi_sck set to 18
D: pin_spi_miso set to 19
D: pin_spi_mosi set to 23
D: GPIO0 is HIGH
D: GPIO2 is LOW, probably no PULL-UP
D: GPIO4 is HIGH
D: GPIO5 is HIGH
D: GPIO12 is HIGH
D: GPIO13 is HIGH
D: GPIO14 is HIGH
D: GPIO15 is HIGH
D: GPIO16 is HIGH
D: GPIO17 is HIGH
D: GPIO18 is HIGH
D: GPIO19 is HIGH
D: GPIO21 is HIGH
D: GPIO22 is HIGH
D: GPIO23 is HIGH
D: GPIO25 is HIGH
D: GPIO26 is HIGH
D: GPIO27 is HIGH
D: GPIO32 is HIGH
D: GPIO33 is HIGH
D: GPIO34 is LOW, probably no PULL-UP
D: GPIO35 is LOW, probably no PULL-UP
D: GPIO39 is LOW, probably no PULL-UP
D: gpio_00 will execute uppreset = 1
D: Enable pin 35 for IR
D: Start display
D: Create list with acceptable WiFi networks
D: Added MYWIFI to list of networks
D: Added ADSL-l1 to list of networks
D: Scan Networks
D: Scan completed
D: Number of available networks: 5
D: 1 - instain          Signal: -46 dBm, Encryption WPA2_PSK,
D: 2 - ADSL-l1          Signal: -82 dBm, Encryption WPA2_PSK, Acceptable
D: 3 - Ziggo8571367     Signal: -86 dBm, Encryption WPA2_PSK,
D: 4 - Ziggo            Signal: -86 dBm, Encryption ????,
D: 5 - VGV75190073AE    Signal: -93 dBm, Encryption WPA_WPA2_PSK,
D: End of list
D: Slow SPI, Testing VS1053 read/write registers...
D: Fast SPI, Testing VS1053 read/write registers again...
D: Connect to WiFi
D: SSID = ADSL-l1
D: IP = 192.168.2.1
D: Start server for commands
D: Network found. Starting mp3 client, mqtt and OTA
D: MDNS responder started
D: Rotary encoder is enabled
D: Sync TOD
D: Sync TOD, new value is 10:56:07
D: Radiofuncs cmd is 1
D: Host selected is 0:109.206.96.34:8100 # 0 - NAXI LOVE RADIO, Belgrade, Serbia
D: Connect to host 109.206.96.34:8100
D: Connect to 109.206.96.34 on port 8100, extension /
D: Song stopped correctly after 0 msec
D: Connected to MP3 host at 109.206.96.34 on port 8100
D: send GET command
D: Switch to HEADER
D: Headerline: icy-name:NAXI LOVE RADIO (NAXI,Belgrade,Serbia, NAXI,Beograd,Srbija) - 128k
D: Headerline: icy-genre:Love and Romance
D: Headerline: icy-url:http://www.naxi.rs
D: Headerline: content-type:audio/mpeg
D: audio/mpeg seen.
D: Headerline: icy-pub:0
D: Headerline: icy-metaint:8192
D: Headerline: icy-br:128
D: Switch to DATA, bitrate is 128, metaint is 8192
```