Dr. Edgar Huckert

11-2017 V1.2

Web: www.huckert.com

Controlling a Rasberry 3 based audio player via 433MHZ

Overview

I am an (advanced) amateur musician playing from time to time on a stage in a large auditorium on saxophone or clarinet. I use as accompagniment „play along" music that I prepare myself.

My problem is: I cannot control directly the **mixing console** (loudness, more basses, pan adjustment) as it stands **20 m away from me**. I tried therefor to control the mixing control (in fact: only my input to this console) by using a PC (here called **frontend** or **client**) standing near to me and an Raspberry 3 (here called ***backend*** or ***server***) connected to the mixing console by short audio cable. Between the PC and the Rasberry 3 commands are exchanged via a 433Mhz wireless connection.

I have written three solution versions for the given problem:

- a version based on WIFI/WLAN
- a version based on Bluetooth
- this version based on 433MHz communication

From these 3 versions I largely prefer the last one which uses two 433 MHz modules called "**HC-12**". These modules cost around 10 Euro per module. I have also tested other 433MHz modules: they are crap compared with the HC-12 modules. The advantages of the 433Mhz solution:

- very good **distance** coverage (I reached >100m).
- very **uncomplicated setup** (compared with Bluetooth)
- **no WLAN/WIFI** needed (not available in this auditorium)

I am able to control in real time the functions that seem important to me:

- start songs
- stop songs if necessary
- control the loudness of the „play along" music

I have written a very **simple graphical interface** (GUI) for a PC running under **Windows or Linux/GTK**. It was essential for me that this interface was **as simple as possible** as you have no time to issue complicated commands if you stand on a stage playing rather complicated music (in my case Jazz). The interface therefore only has some simple buttons for manipulating the remote player.

The frontend (PC side, TX)

The frontend is a PC based client. This is a graphical user interface (GUI) consisting mainly of large buttons enabling the start of songs (actually 9 song), the control of the loudness (4 buttons) and a STOP button to cancel the actual player process.

The frontend communicates with the backend by sending very simple ASCII based messages. You will see the frontend below under the chapter „The real scenario". When we mix software and hardware components the the frontend consists of:

- GUI software written in C++ with wxWidgetsI
- USB-serial driver for the Arduino
- Arduino 5V version (I used Arduino Nano)
- simple Arduino sketch
- HC-12 module 433MHZ (used as transmitter)

The **pure GUI component is portable** as it uses standard C++ with **wxWidgets** (Versions >= 3.0)**.** You may build and run the frontend software (GUI) under Windows or Linux/GTK. The functions for the serial communication over 433Mhz are however not so portable: I have put the respective versions between *#ifdef WIN32- #else* brackets. Keep in mind that the GUI client has also a very technical part: it must handle the HC-12 module acting as a transmitter.

The windows dimensions are calculated at start time – may be that this logic is a bit too simple. My goals was to get large buttons that can easily be manipulated on stage. A touch screen would even have been better – I don't own one.

You will find the GUI source code, a simple ini file and a **make file** in the ZIP file mentioned below. I used *Mingw g++* on Windows and GNU g++ on Linux as compiler and linker. You may have to **change some details in the make files**. Make files for projects including *wxWidgets* are not trivial: so don't expect to compile the GUI component via a simple CLI command.

Do not forget to enter the correct serial device names (Linux) or COM numbers in the ini-file *433MhzGUI.ini*. Under Linux this should be somethin like

„/dev/ttyUSB0" and under Windows just „3" oder „20". While testing I have seen a lot of variation for the COM ports: from COM3 to COM20.

A note for the Arduino to be used on the client side: I used an **Arduino Nano**. On one of my Windows 7 machines I had problems with the **driver** for the USB-serial port: It seems that the orginal Arduino Nano uses an FTDI chip whereas many clones use other chips like CH340 in my case (I even didn't know that my Arduino was a clone). I could solve the driver problem by installing the driver CH341SER on the Windows PC. On Linux/GTK the device /dev/ttyUSB0 mad no problems for the frontend.

A really short **Arduino sketch** named *PlayerFront.ino* is in the ZIP file. This sketch just accepts bytes from the USB-serial port and forwards them to the HC-12 module that is wired with the Arduino. The Arduino program is more or less a **simple echo program**: it just reads bytes from the serial line and forwards them to the HC-12 transmitter. The Arduino is needed as you cannot normally manipulate directly pins on a PC!

It is important the the frontend HC-12 has only the TX-pin connected to the RX pin of the Arduino. The RX pin must not be connected - it is used by the USB/serial driver internally.

The backend (Raspberry 3 side, RX)

The server must be an interface for the receiver module HC-12. I have connected this module directly to the pins of my Raspberry 3 so that **no USB driver is needed,** just the standard serial device */dev/ttyS0*. The receiver side is thus simpler that the transmitter side.

The server software consists mainly of a server program receiving messages from the client. I have written a program called *433Mhz_server* in standard C. After having received a message it starts a WAVE player process (called *playw*), stops a player process or controls the loudness of the player using the ALSA standard command *amixer*. Unfortunately this command is badly documented. On my Linux machines each *amixer* version shows different behaviour. Be prepared to modify the *amixer* calls in the code if necessary. If the *vol-* or *vol+* buttons don't work as expected make you own experiences with the *amixer* ALSA program. We need in the backend just ALSA – no *Pulseaudio* nor *Jack* needed!

If we mix again software and hardware components then the backend consist of:

- a Raspberry V3
- serial driver (normally */dev/ttyS0*) - no USB needed
- server program *433Mhz_server*
- player program *playw*
- HC-12 module 433MHZ (used as receiver)
- standard ALSA program *amixer*

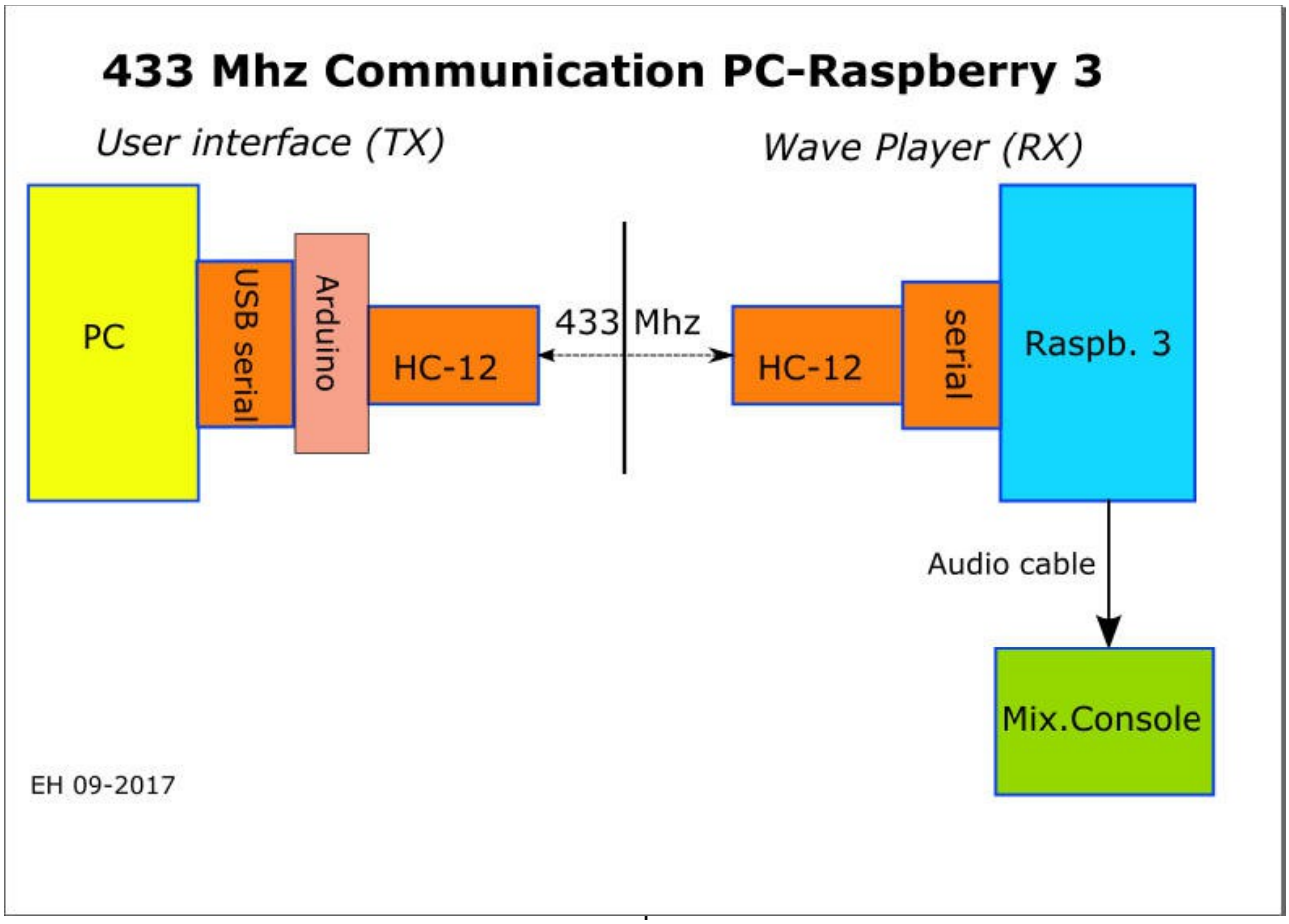
Note that I don't recommand a Raspberry 2: the audio module is worse that on the Raspberrry 3.

One of the bigger problems were the *STOP* commands: I had to manipulate the process list in order to find the actual player processes that were to be killed.

The player used in this backend is my own program *playw* (the source is included). If you don't like this program you can also use standard ALSA commands like *aplay*. The player is alway started as a separate background process – it is not tightly coupled with the backend program *433Mhz_server*.

You will find the server source code and a make-file in the ZIP file mentioned below. I used *Mingw g++* as compiler and linker.

I include a diagram describing the structure of frontend and backend:



Note that the frontend and backend sides are **asymmetric**:The **frontend** uses an Arduino (in my case an Arduino Nano) to control the HC-12 module (the transmitter). A USB/serial driver as used in normal Arduino IDE installations is used as link between the PC and the Arduino. This driver controls a COM port: in my case "COM3:" - but may be called differently depending on the Arduino versions and the Windows versions.

The **backend** needs no Arduino: the Raspberry 3 can control the HC-12 module (the receiver) directly - there is even no USB needed as the serial /dev/ttyS0 driver can control the HC-12 module directly.

Testing the communication

Before using the real frontend - the GUI application - and the real backend - the audio player - we can test the basic communication by using:

- a terminal emulation like ***TeraTerm*** under Windows(used as sender) on the PC side or ***Putty*** under Windows or Linux
- a terminal emulation like ***minicom*** (used as receiver) on the Raspberry 3 side

Both sides should be configured with *9600/n/1*. This can be done in the terminal emulation program or with this command:

```
sudo stty -F /dev/ttyS0 9600 raw
```

On the Rasberry 3 side you should test if a *getty* process (a login process) is running on */dev/ttyS0*. This *getty* process can seriously disturb the reception of commands on the Raspberry side. Proceed as follows under Raspian V8.0:

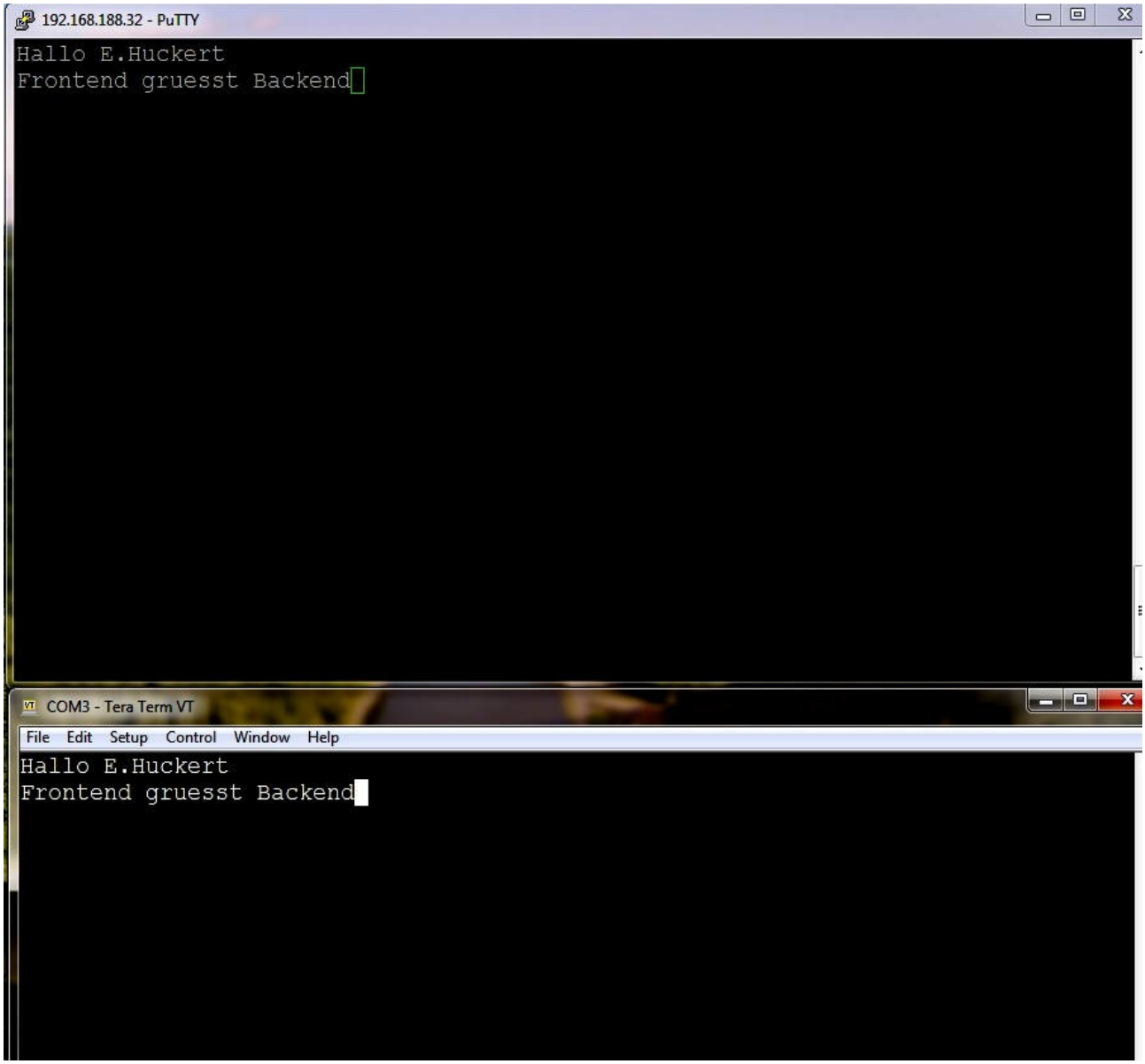
```
ps ax | grep getty
```

If *getty* or *agetty* is shown in the result then:

```
sudo systemctl stop serial-getty@ttyS0.service
sudo systemctl disable serial-getty@ttyS0.service
```

These commands stop and disable the *getty* or *agetty* processe only for the current session. The processes will reappear on the next boot. I have placed these commands in a short procedure called *stopGetty*.

The following screen shot shows my communication test scenario: the upper „black" window shows a *Putty* screen running *minicom* on the Raspberry 3, the lower black window shows TeraTerm running on a Windows PC:



Not much surprise here: the input on the lower window is echoed on the upper window.

Testing the backend

We can now test the backend audio server. This is program ***433Mhz_server.c*** running on the Raspberry 3. If we are sure that no getty process is running on our serial line and after configuring this line for *9600/n/1* we start the backend process on the command line if by typing

```
sudo ./433Mhz_server -v
```

In a real context we would rather start this audio server in the background (appending a „&" to the command) or start it at boot time via *systemd* or via *crontab.*The „-v" parameter ensures that you see something on the Raspberry side as soon as a command arrives.

On the frontend side (PC) *TeraTerm* (Windows) *or Putty* (Windows or Linux) *are* good to send commands. Please remember that our backend expects commands terminated by LF ('\n'). If you want to send the name of a WAVE file to be played then the commands are identical to the names of the WAVE files (here W1..W9). Be sure that you have configured you terminal emulation accordingly or enter explict linefeeds after each command string (CTL-J will normally do the job).

After having **placed some WAVE files in the Wave directory** you can enter commands in the terminal emulation of the frontend side. The commands accepted can be something like the following:

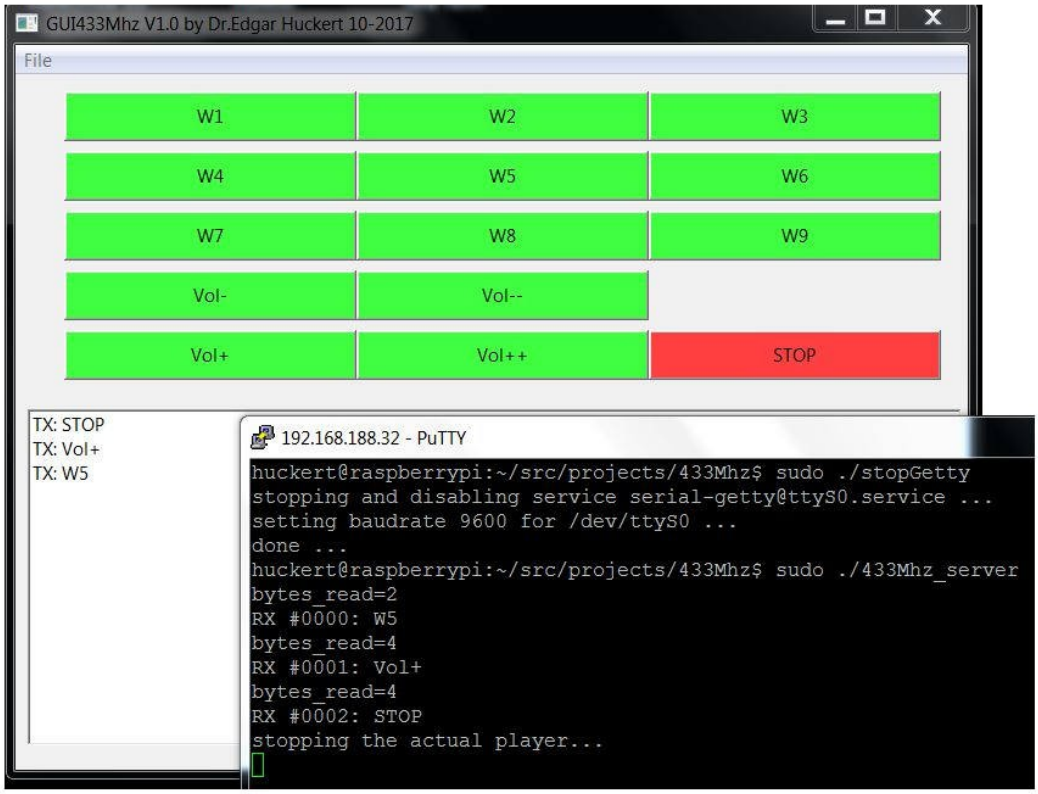
W1	[means: play file W1.wav in the Wave directory]
STOP	[means: stop the current Wave file]
+	[means: play it louder]
-	[means: lower the volume]

The real frontend (the GUI explained below) will do nothing else than issuing these commands. It is easy to expand the command set on the server side. But do not forget that the client side (GUI) must also be expanded – which is not so easy.

You should start tests on the backend only if the **ALSA architecture** including the development package on the Raspberry 3 is installed and working. Commands like *aplay* provide the base for the most basic tests. If *aplay* produces no sound then the more complicated tests will fail also.

The real scenario

The following screen dump shows the GUI client (program *433MhzGUI*) running



on a Windows Laptop in the upper window. The output from the server program *433Mhz_server* on my Raspberry 3 is shown in the lower black window. This output is produced by *Putty* connecting via WLAN the PC and the Raspberry 3. Putty is used just to demonstrate the arrival of messages in the backend. **You don't need a WLAN connection** nor *Putty* – the server can run without this.

Be sure that to start it on the Raspberry 3 with **root rights** or change permanently the access rights for the serial line (in my case */dev/ttyS0*).

The procedure *stopGetty* that can be seen in the first line of the Putty windows disables an eventual *agetty* process running on this serial line. It also configures the serial line for 9600 Baud, no parity, 1 stop bit. *stopGetty* contains the commands mentioned under „Testing the communication".

Preparing the Raspberry

On the Raspberry 3 side I don't use an ini-file. Some configuration must therefore be done in the source code *433Mhz_server.c*:

- change the device name (standard is */dev/ttyS0*)

- change the directory for the executable (see *exe_dir*)
- change the directory for the WAVE file (see wave_dir)
- change the name of the player command (see *play_cmd*)

Don't forget to compile the program *433Mhz_server.c* after having changed the parameters.

It is very important for this simple solution to **place all WAVE files** (or MP3 if you use a different player) **in the same directory**.

Note that the names of the WAVE files **correspond to the button names** in the GUI. If you want to use more elaborate WAVE file names then you have to introduce an indirection level on the server side (a mapping table) or on the GUI side.

If you change the name of the **player command** then the *STOP* logic must also be changed: the backend program inspects the process table in order to find the player process by name.

If you have tested the 433Mhz_server manually and if everything works fine then you can add enrties to file */etc/crontab* in order to **start the backend at boot time.** In my case I added two lines lines at the end of /etc/crontab:

```
@reboot huckert amixer  -c 0 sset PCM 92%
@reboot root /home/huckert/bin/433Mhz_server -v >/tmp/433Mhz.txt &
```

The first line presets the volume of Alsa device PCM on channel 0 to 92% (this may change depending on the Linux and Alsa variants) and the second line starts a background process for the backend *433Mhz_server*.

Wiring

Connections **Raspberry 3** GPIO to **HC-12** (Backend):

```
 2 - 1 (5V)
 6 - 2 (GND)
 8 - 3 (RX)
10 - 4 (TX)
```

Connections **Arduino Nano** to **HC-12** (Frontend):

```
 27 - 1 (5V)
 29 - 2 (GND)
  1 - 3 (RX)
```

Download links for Windows and Linux/GTK:

You will find the source code for frontend and backend, the Arduino sketch, a make file for the GUI and even a Win32 executable under this link **for Windows**: www.huckert.com/ehuckert/programs/433MhzEH.zip

The **version for Linux/GTK** under www.huckert.com/ehuckert/programs/433MhzEHUnx.zip is more or less the same, except for the executable: as Linux runs on many very different CPUs a pre-built executable would not be helpful. The ZIP file offers a make file for Linux/GTK and a C++ source that is slightly adapted for the GUI. The server part is identical to the Windows ZIP above.