Dr.Edgar Huckert V1.3 03-2020

MP3 player and foot switch for musicians

Contents

MP3 player and foot switch for musicians....1
Purpose and short description....1
GUI and application logic....2
A screen dump....3
Sample ini-file (Linux)....4
Sample make file (Linux)....4
Sample ini-file (Windows)....5
Sample make file (Windows)....5
The switch and the Arduino inside....6
The Arduino sketch....8
A screen dump produced with TeraTerm....9
Download link....10

Purpose and short description

I am a musician playing often solos (Jazz or classical music) with self produced MP3 accompaniments ("play along music") in the background. This is often my configuration:

- a mobile phone with Bluetooth
- a Bluetooth speaker (in my case JBL Charge 4)

This configuration however proved to be unreliable - not for technical reasons. The main problem was the high error rate when I had to manipulate the player on the phone: the chance to produce false actions is simply too high on a small device like a mobile phone. Even if used a laptopn instaed of a phone many manipulation errors occured (wrong mouse move, wrong mouse clicks etc.)

So I decided to follow a much simpler approach.:

- Use a foot switch to start/stop the MP3 player
- Use a small PC (laptop, notebook) instead of a mobile phone
- Use an extremely reduced set of GUI actions in order to reduce the number of errors

Using a foot switch is not new in music. Piano players use them always, guitar players use them for effects like chorus, distortion etc. I play wind instruments: my two hands must handle the instrument - no chance to do additional manipulations on a screen.

GUI and application logic

As always the GUI is the most complicated part. It could also be replaced by a simple CLI ("black window") application - but the GUI offers some additional features, among them the chance to start and stop the player via the buttons instead of the foot switch.

I have written the GUI in C++ as a *wxWidgets* application. *wxWidgets* is not a simple beast - so be warned. You have to choose the right wx*Widgets* version (here V3.1) and the right compiler version. I used g++ in version 4.9.2 under Linux and V 5.3.0 under Windows. Other version combinations between *wxWidgets* and g++ may not run.

The GUI and its **associated logic** is portable between Windows 32-Bit and Linux (Debian based). In the code you may find however a lot of passages with different code for Windows and Linux. *wxWidgets* is a pure GUI framework - it offers no features for the handling of the serial port.

What the GUI and its associated logic does:

- build the graphical interface
- read a configuration file (ini-file)
- open a serial line (under Windows: COM port)
- react on events coming from the buttons or a timer
- execute commands (system() call) that start an MP3 player or stop it

I have choosen a simple **timer solution** to evaluate the serial line and thus the Arduino in the foot switch. This is a polling solution. Alternative solutions may use a separate thread for this or an interrupt driven approach.

You find make files for Linux and Windows below and in the zip file offered for download. I never use IDEs (like Eclipse, Code Blocks etc.).

The GUI program also contains the **application logic** (serial port handling, starting / stopping the player). As much as possible I have choosen **standard commands** to control the MP3 player. My configuration uses the **mpg123 player** available under Linux and Windows - this player is however not pre-installed. Internally some Windows/Linux standard commands are used:

Windows: **tasklist** and **taskkill** (must be used with /F option!)

Linux: **ps** and **kill**

Under Linux extended rights are needed for the *kill* command. These rights can also be needed under Windows depending on the basic rights for the current user. The GUI program writes an auxiliary file - it cannot be started from a CDROM/DVD drive.

The GUI program reads essential parameters from an **ini-file** called *mp3PlayerW.ini* or *mp3PlayerU.ini.* Samples are given below. To a certain degree you can change here some

application parts - like the command for the MP3 player. Changes in the source code may then also be needed.

A note on the **USB/serial ports** under Windows an Linux. Under Windows this was COM27 on my machine (use a product like *TeraTerm* to test this). The COM number depends on the number of drivers loaded for the COM ports under Windows. Under Linux the port was */dev/ttyUSB0*. Use "ls -l /dev/tty*" to find it out and *minicom* to test it.

This GUI program can be enhanced: a *list widget* could be added to show the actual state of the player or to show error messages. But this is not really essential and does not contribute the the main design goal: **simplicity and robustness**.

A note on the **Start** action, that is activated by the corresponding button or by an "ON" command sent by the Arduino: I have implemented here a **toggle logic** for the foot switch, i.e. the first foot switch action start action starts the MP3 player, the second activation stops it.

The **Stop action** evaluates the current **process list** to find the process ID of the player and stops it by using the given command in the ini-file.

The logic behind the buttons **Vol++** and **Vol--** is not yet implemented. I have problems under Windows to find a simple solution that can influence the sound volume. These buttons could also be controlled via one or two foot switches - probably not of the opener / closer type. Foot switches acting like a variable resistor are probably the simplest solution here. An Arduino Nano is able to accept and evalute analogue voltages (builtin ADC converter). Note that the signal cannot directly be influenced by this type of switch as the MP3 signal (in my scenario) is not accessible on a physical external line - it is sent over Bluetooth.

The C++ Code for the GUI and its associated logic is too large (ca. 2000 lines) to be included in this document. You will find it in the **download link** given at the end of this document.

A screen dump

Here is a screen dump for the Linux version (note the "&" at the end of the command):

The **Start** and **Stop** buttons can be replaced via external switches - the volume control buttons are not yet implemented.

Sample ini-file (Linux)

```
[general]
start=/usr/bin/mpg123 /home/huckert/Music/mp3/RitterFagV5.mp3 &
stop=/home/huckert/Music/mp3/RitterFagV5.mp3
serial=/dev/ttyUSB0
volincr=lauter
voldecr=leiser
```

Note: under Linux it is important to terminate the player start command with an „&“ (= run in the background) – else the application is blocked.

Sample make file (Linux)

Under Linux (Ubuntu LTS) I used:

- wxWidgets V3.1.0 (gtk2, unicode, static)
- g++ V 5.3.0 (minGW)

The utility *wx-config* is used here. It facilitates the need to specify exacts pathes. *wx-config* may not exist in all wxWidgets distributions:

```
OBJS=mp3Player.o hu_trace.o inifilewx.o

CFLAGS=-DLOGGG -DLINUX  `wx-config --cxxflags`

LFLAGS=`wx-config --libs`

mp3Player: $(OBJS)
     g++ -o mp3Player $(CFLAGS) $(OBJS) $(LFLAGS)

mp3Player.o: mp3Player.cpp
     g++ -c $(CFLAGS) mp3Player.cpp

hu_trace.o: hu_trace.cpp hu_trace.h
     g++ -c $(CFLAGS) hu_trace.cpp

inifilewx.o: inifilewx.cpp inifilewx.h
     g++ -c $(CFLAGS) inifilewx.cpp

clean:
     rm $(OBJS) mp3Player
```

Sample ini-file (Windows)

```
[general]
start=start c:\huckert\bin\mpg123 c:\huckert\mp3\RitterFagV5.mp3 /Q
stop=taskkill /F /pid
serial=27
volincr=lauter
voldecr=leiser
```

It is essential under Windows to use the *taskkill* command with the /F option!

Sample make file (Windows)

Under Windows I used:

- wxWidgets V3.1.0 (gtk2, unicode, static linking)
- g++ V 4.9.2

Note: the utility *wx-config* used in the Linux make file does not work as expected under Windows. This may be a problem of the make-utility. I redirected the output of *wx-config* in a file and pasted this output in the make file.

```
OBJS=mp3Player.o hu_trace.o
CFLAGS=-mthreads -DHAVE_W32API_H -D__WXMSW__ -D__WXDEBUG__ -D_UNICODE
-Ic:\huckert\wxWidgets-3.1.0\lib\gcc_lib\mswud -Ic:\huckert\wxWidgets-
3.1.0\include -Wno-ctor-dtor-privacy -pipe -fmessage-length=0
LFLAGS=-mthreads -Lc:\huckert\wxWidgets-3.1.0\lib\gcc_lib -lwxmsw31ud
-lwxtiffd -lwxjpegd -lwxpngd -lwxzlibd -lwxregexud -lwxexpatd -
lkernel32 -luser32 -lgdi32 -lcomdlg32 -lwxregexud -lwinspool -lwinmm -
lshell32 -lcomctl32 -lole32 -loleaut32 -luuid -lrpcrt4 -ladvapi32 -
lwsock32  -lversion -lshlwapi


mp3Player.exe: $(OBJS) mp3Player.res
     g++ -o mp3Player.exe $(CFLAGS) $(OBJS) $(LFLAGS)
mp3Player.o: mp3Player.cpp
     g++ -c $(CFLAGS) mp3Player.cpp
hu_trace.o: hu_trace.cpp hu_trace.h
     g++ -c $(CFLAGS) hu_trace.cpp
clean:
```

```
del $(OBJS) mp3Player.exe
```

The switch and the Arduino inside

The **foot switch** is a **heavy duty type** model "LT3" (costs around 20 Euro). For me as a musician it is important to use a **heavy switch** type as this model cannot easily be moved accidentally.

There are two type of foot switches on the market:

- simple opener/closer types
- volume pedal types: these are normally variable resistors

I have choose a simple but rugged closer switch type.

This massive switch has inside enough **space to host an Arduino Nano**- nearly every other type should also work if it is small enough to fit into the switch.

The **Arduino Nano** has the following advantages:

- It can be **powered** via USB (no additional power supply needed)
- It has an **USB component** that can interact with a PC by simulating a **serial** line

The Arduino decodes the state of a line connecting the 3.3 V output of the Arduino and the D5 pin configured as input. This pin D5 is also connected via a resistor to GND thus providing a stable basic state (state=0). The switch is a closer type, i.e. it is normally open - no voltage is applied to pin D5. When activated it closes the 3.3V line thus setting the D5 input to state=1. This state change is detected in the Arduino sketch: it sends then an "ON\r\n" string to the PC.

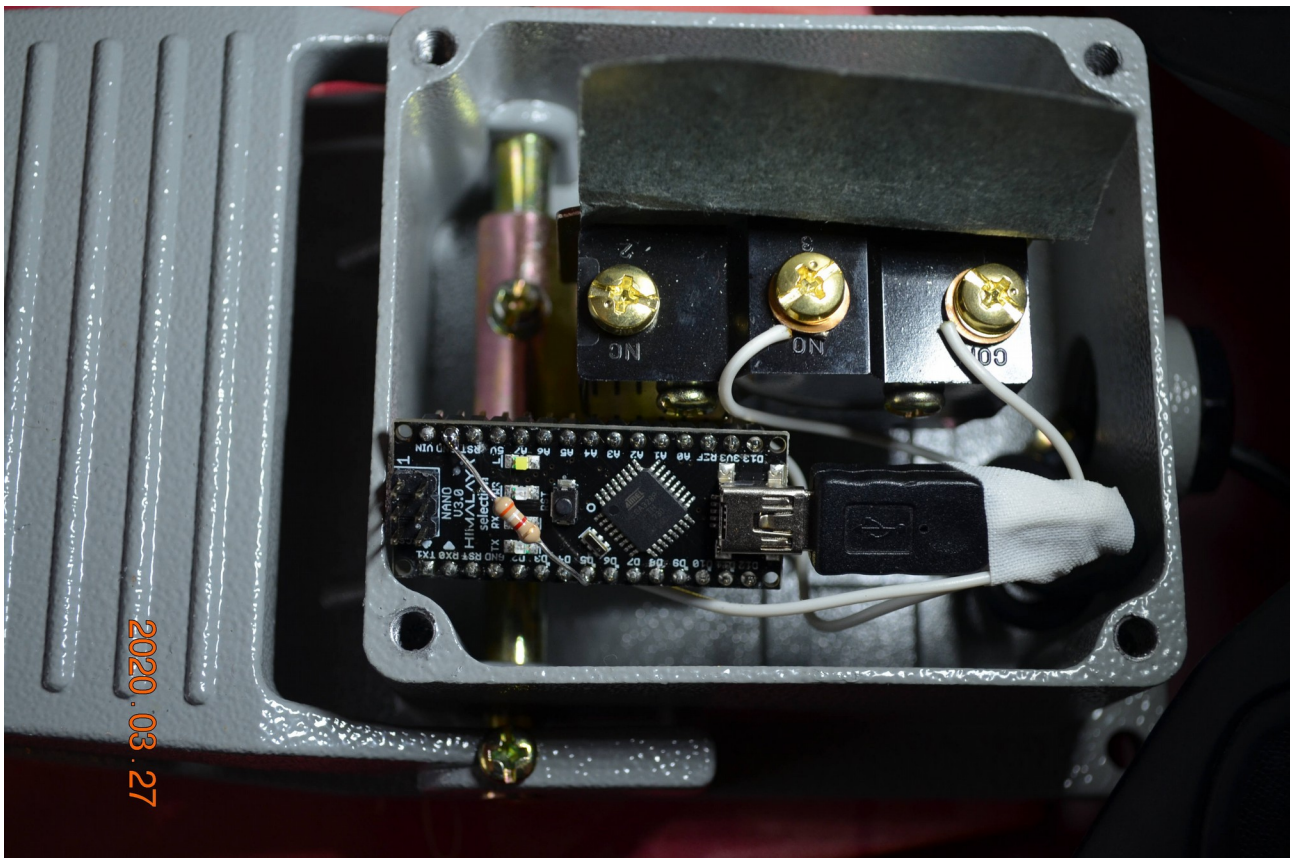
The Arduino must be a 5V type as we use the 5V voltage provided by the USB interface to power the Arduino. If a 3.3V type is used then this voltag probably must be lowered by a cheap voltage regulator.

A note for the Arduino under Windows: depending on the Arduino type and on the Arduino supplier an additional USB/COM port driver must be installed. Some Arduinos (with non FTDI chips) are not accepted by standard Windows.

Here is a photo of the switch. The black cable seen on the right side is a standard USB cable connecting the *Arduino Nano* in the switch (not visible) and the laptop:

Do not forget to install a **cord grip** in the hole for the USB cable.

Here is a photo showing the Arduino **inside the foot switch** (plastic isolation removed):



The Arduino sketch

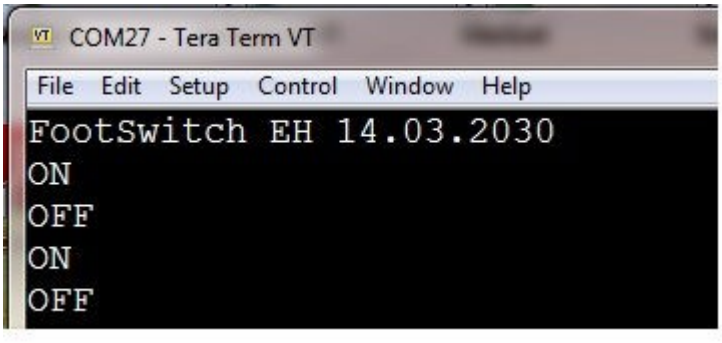
This is the very simple source code for the Arduino Nano sketch in the switch. The standard serial line (object Serial) is initialized for 9600 Baud. The switch interrupts digital input line D5 on the Arduino. As said above it sends "ON" if the switch is pressed down. The string "FootSwitch..." is sent once at the beginning:

```
// module FootSwitch.ino
// Transmits oder serial line the state changes of a foot switch
// Serial params: 9600,8,N
// Written by Dr.E.Huckert 03-2020

const int ledPin     = 13;      // the number of the LED pin
const int switchPin  = 5;       // D5 connected to the switch
int       lastState  = 0;
int ledState         = LOW;

// ---------------------------------------------
void setup()
{
  // set the digital pin as output:
  pinMode(ledPin, OUTPUT);
```

```
  pinMode(switchPin, INPUT);
  Serial.begin(9600);
  delay(100);
  Serial.println("FootSwitch EH 18.03.2020");
}

// ------------------------------------------------
void loop()
{
  int state;
  //
  if (ledState == LOW) ledState = HIGH;
  else                 ledState = LOW;
  digitalWrite(ledPin, ledState);
  //
  state = digitalRead(switchPin);
  // Serial.print("state="); Serial.println(state);
  if (lastState == 0)
  {
    if (state == 1)
      Serial.print("ON\r\n");
  }
  else
  {
    // lastState is here 1
    //if (state == 0)
    //  Serial.print("OFF\r\n");
  }
  lastState = state;
  //
  delay(100);
}   // end loop
```

A screen dump produced with TeraTerm

When you connect the switch (with the Arduino inside) via USB to a **serial monitor** (like *TeraTerm* under Windows or *minicom* under Linux) then the output on the serial monitor should look as follows (the switch has been pressed twice). The serial monitor replaces here the GUI and is just used as a test bed:



Download link

You may download the complete software from this link:

www.huckert.com/ehuckert/programs/Mp3PlayerFoot.zip

This zip file contains:

- GUI C++ code (3 *.cpp* files and 2 *.h* files)
- make files for Windows and Linux
- Arduino sketch (extension *.ino*)
- sample ini files (must be accomodated)
- Windows executable (32 Bit, statically linked)