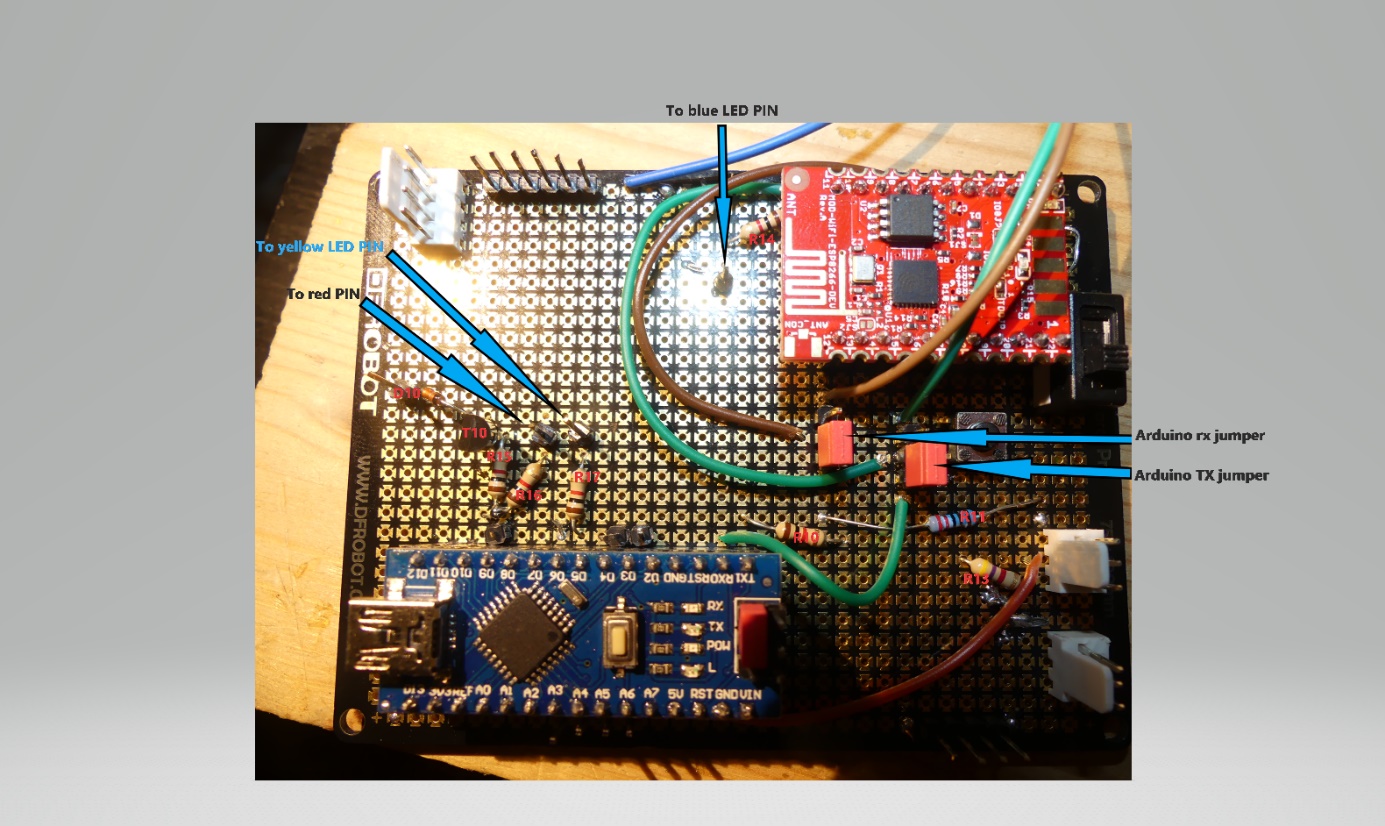
# 

Table des matières

[Thermostat documentation 2](#_Toc501267504)

[What is the purpose? 2](#_Toc501267505)

[Functions 2](#_Toc501267506)

[Basic services 2](#_Toc501267507)

[Extended services 2](#_Toc501267508)

[The thermostat can receive some external information to adjust temperature instruction: for instance, decrease temperature when you are out and have set your home is under alarm system. 2](#_Toc501267509)

[Architecture overview 3](#_Toc501267510)

[ 3](#_Toc501267511)

[Technical implementation 4](#_Toc501267512)

[Heating regulation 4](#_Toc501267513)

[Heating instruction remotely 4](#_Toc501267514)

[Hardware design 4](#_Toc501267515)

[Network connection 7](#_Toc501267516)

[Server components 9](#_Toc501267517)

[DIY instructions 11](#_Toc501267518)

[Build of material 11](#_Toc501267519)

[Build the power sources 12](#_Toc501267520)

[Build the electronics 12](#_Toc501267521)

[Let’s do with the gateway configuration 16](#_Toc501267522)

[Let’s do with the Arduino side 20](#_Toc501267523)

[Solder the components 25](#_Toc501267524)

[Solder the power supply 25](#_Toc501267525)

[Solder the microcontrollers PCB 26](#_Toc501267526)

[User guide 26](#_Toc501267527)

[Maintenance guide 27](#_Toc501267528)

[An open system 28](#_Toc501267529)

[Change than can be done easily 28](#_Toc501267530)

[Improvements that could be done 28](#_Toc501267531)

[Communication specifications 28](#_Toc501267532)

# Thermostat documentation

### What is the purpose?

* Increase your comfort by heating your house exactly as you want
* Make savings by heating your house only when you need
* On top of that be proud you did it by yourself

## Functions

### Basic services

Heating regulation algorithm adjusted to the environment provides a very stable temperature.

Heating instructions are defined on schedule based on weekly and daily calendars.

For instance: a weekly calendar to be used during the working periods and a daily calendar to be used when you spend holidays at home.

User expresses instruction as a couple (temperature,time) objective. For instance: instruction 20° at 8:30. The thermostat will start heating at the appropriate time depending on the inside and outside actual temperatures

User can at any time either with a remote controller

* switch between the calendars
* manually modify temperature instruction
* temporarily hold on the heating system

### Extended services

Even if the thermostat can work 100% autonomously its purpose is to be connected to provide additional services when connected to WIFI.

A Web application can be used for the user to monitor and control the heating. For instance, you will be able to increase the temperature remotely with your mobile on you way back from holiday.

The thermostat will provide a lot of information that could be used to better tune your installation and so increase your comfort and savings. Parameters can be modified remotely.

## The thermostat can receive some external information to adjust temperature instruction: for instance, decrease temperature when you are out and have set your home is under alarm system.

The clock of the thermostat is synchronized with the server and will automatically switch for the daylight saving time (summer time).

## Architecture overview



Internet WIFI

Router

Server

Tomcat Linux

WIFI Gateway

ESP8266

Micro Controller

Atmel ATmega

Boiler

## 

## Technical implementation

The core functions run on an Atmel ATmega micro-controller. It has been validated on the following Arduino platforms: Nano, Uno and Mega). Since code and parameters has been downloaded the Arduino and clock synchronized with the serer, it can run 100% autonomously. It communicates thru the serial link, eventually takes into account external information.

Parameters are initially written in the Eeprom and can be modified remotely and saved in the Eeprom.

### Heating regulation

Temperature is driven by a PID regulator. The PID parameters (KpPID,KiPID,KdPID, thresholdPID, PIDCyleDelay) are initially defined when downloading the code and can be adjusted either or remotely. When the thermostat is set in PID tuning mode it continuously sends information to the server that are useful to adjust the PID parameters to your equipment.

A timer parameter (hysteresisDelay) is used to avoid to switch the boilier on/off to fast.

Internet provides actual external temperature at your location.

### Heating instruction remotely

Even if temperature can be set manually, the purpose of the thermostat is to run in automatic mode.

The system will choose between choose between 4 different temperature instructions (for instance morning, day, evening, night) according to your need. Your need has to be defined in 1 weekly and 2 daily agendas. For each half an hour you have to select one of the 4 temperature instructions. The schedule has to be downloaded with the code but can be modified remotely and saved in the eeprom.

A 5th temperature instruction is defined and used when you are out of home to keep your house out of freeze.

Temperature instructions are expressed as targets (be at a temperature at a time). To decide when to start heating the thermostat will take into account the difference between the actual internal temperature and the target and the actual outside temperature. This regulation can be tuned to your need with 2 parameters (reactivity, sizeAnticipation) that you can modify remotely and save in the eeprom. Each hour instruction used one byte: the first half byte is used to select the instruction for the first half hour and the second half byte is used to select the instruction for the second half hour. For instance, 0x12 means select temperature 1 between 0 to 29mn ad temperature 2 between 30 to 59mn.

2 parameters (maximumTemperature, minimumTemperature) define the maximum and minimum temperature than can be set by the user. It can be modified remotely and save in the eeprom.

### Hardware design

#### Boiling connection

Boiling is connected to a NO relay. The heating is set on by closing the relay and of by opening it.

#### Power

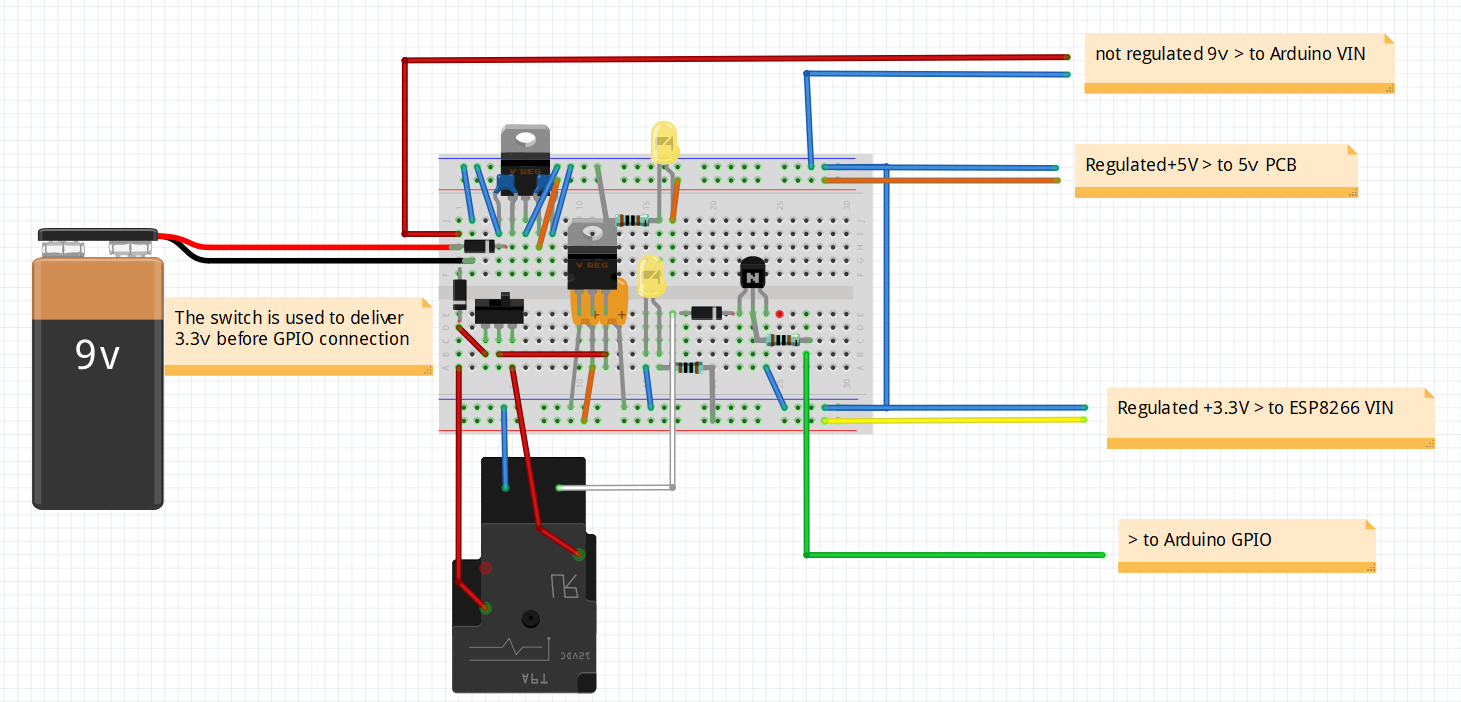
Power source can be delivered by any 9v DC 500mA not regulated. All components are soldered on a PCB. It delivers the original not regulated 9v power, a 5v and 3.3v regulated power. 3.3v is connected thru a NC relay to allow the Arduino to switch 3.3v On/Off. 2 diodes protect from wrong input connection.

9v powers the Arduino (VIN).

5v powers the 2 relays, the clock, the LCD, the LEDs and the infrared receiver.

3.3v power the ESP8266

The temperature sensor is powered by the Arduino 5v output.

[](https://github.com/cuillerj/Thermostat/blob/master/thermostatPower.fzz)

#### Electronics

Arduino is protected from relays by NPN transistors and diodes.

ESP8266 RX (3.3v) is protected from Arduino TX(5v) by a voltage divider.

The design allows to replace easily all the active components.

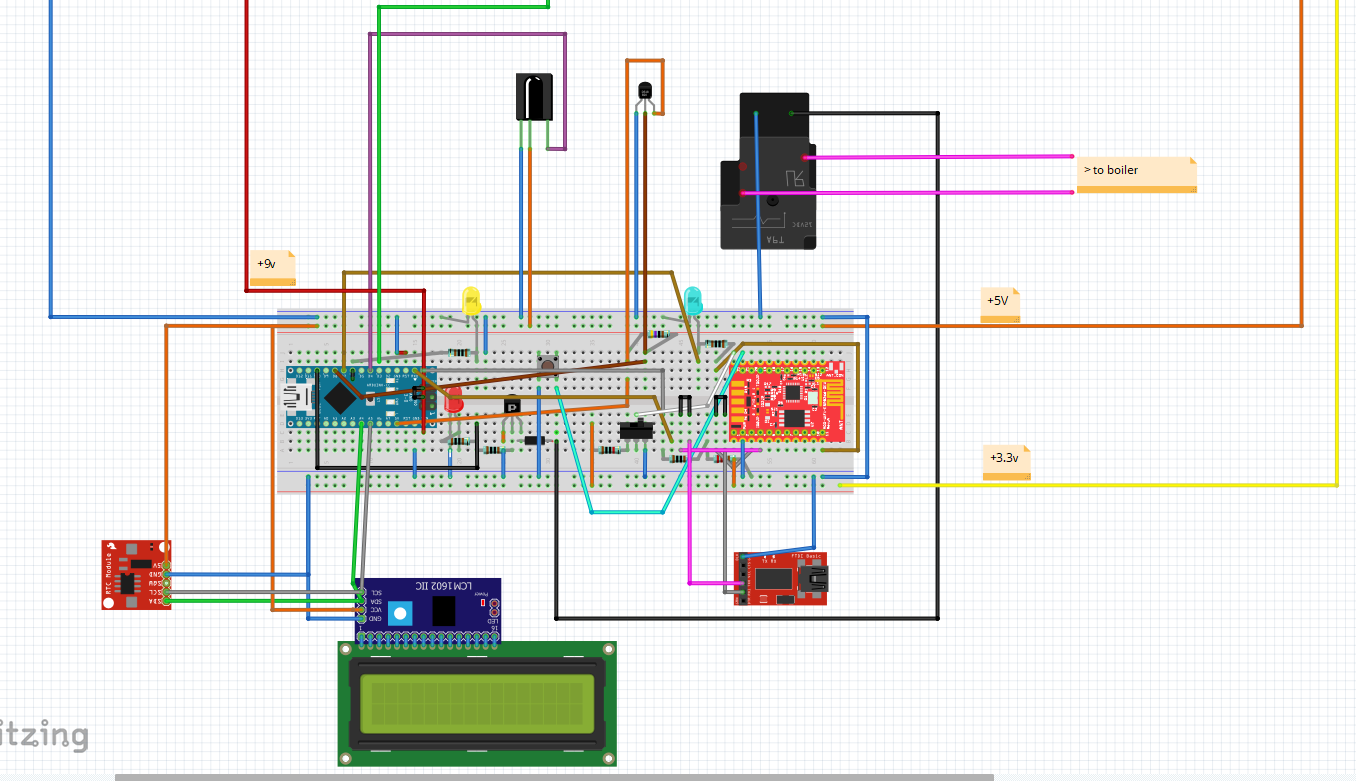
A push button allows to download ESP8266 program

2 connectors are used to switch the ESP8266 serial link from Arduino to the FTDI.

1 switch is used to set the ESP8266 in configuration mode

1 connector is used to set the Arduino in initialization mode

Arduino interacts with digital GPIO, serial link, One wire bus and I2C bus.

[](https://github.com/cuillerj/Thermostat/blob/master/thermostatElectronic.fzz)

### Network connection



The network connection is made with a ESP8266 WIFI microcontroller. It is based on the [gateway description ‘instructables”](https://www.instructables.com/id/How-to-Make-Your-Own-WIFI-Gateway-to-Connect-Your-/) . The following changes have been made from this description: some useless GPIOs for this project are not used and the Arduino and ESP8266 are soldered on the same PCB.

This module is connected from one side with the serial link from the other side to IP network with the Wifi. It acts as a black box. Data packets coming from the serial link are sent to an IP/Udp port and vis et versa. It can transfer either ASCII and binary data.

You just have to set your own configuration (IP, WIFI ...) once the first time you will power on the Gateway. This gateway configuration can be changed locally with USB TTY interface (see maintenance guide). Configuration is saved in eeprom.

A simple protocol is used between Server and Arduino as follow

* end to end 8bit CRC to check frame integrity
* send and receive frame numbers to check for missing frame
* frame sent can request or not request an acknowledgement from receiver

Both Server and Arduino initiate a dialog

* send information without being requested for
* send a request
* send a response to a request

[See communication specifications](#_Communication_specifications)

## Server components

Server runs Linux (developed with Ubuntu 14.04), mySQL server (developed with 5.5) and Tomcat7.

Batchs are in charge of communication with the Thermostat.

It reads “meteo” table to look for the actual outside temperature and prepare to send to the thermostat

It reads “ind\_desc” table to find parameter’s values that eventually need to be send to the thermostet

It writes in “ind\_value” table parameter’s values and mesurments sent by the thermostat

J2EE application is in charge of human interface. It reads and updates database. It also send command to the thermostat going thru batchs.

Thermostat

Read

Write

Request

Response/Information

Web application

J2EE

meteo

Ind\_value

Ind\_desc

Batchs

Java

2EE application are in charge of exchanges with the user. It reads and updates database. It also send command to the thermostat going thru batchs.

## DIY instructions

The first step is to gather all the parts you will need.

### Build of material

**You will need these main components**

*2 x micro-controllers*

* + 1 x Arduino - I chose a Nano 3.0 - you can find some at around 2.5$ (Aliexpress)
  + 1 x ESP8266 - I [chose -ESP8266-DEV Olimex - at 5.5€](https://www.olimex.com/Products/IoT/MOD-WIFI-ESP8266-DEV/open-source-hardware)

*1 x temperature sensor DS1820*

* I chose a waterproof one - you can get 5 for 9€ (Amazon)

*1 x double relay module (0 command)*

* I chose SONGLE SRD-05VDC - you can find some at 1.5€ (Amazon)

*1 x I2C LCD 2x16 characters*

I already had one - you can find some for less than 4$ (Aliexpress)

*1 x I2C*DS1307 *Real Time Module with CR2032 battery*

* I already had one - you can find some for less than 4$ (Aliexpress)

*you can find for a few euros*

*1 x Infrared receiver*

* I chose AX-1838HS you can find 5 for 4€

1 x FTDI

1 x IR remote controller (you can buy a dedicated on or use your TV one)

*2 x power regulators (3.3v & 5v)*

* I chose I x LM1086 3.3v & 1 x L7850CV 5v

**And some few stuff**

5 x LED

9 x 1K resistors

1 x 2.2K resistor

1 x 4.7K resistor

1 x 100 microF ceramic capacitor

1 x 330 microF ceramic capacitor

2 x 1 microF tentalum capacitor

2 x NPN transistors

4 x Diodes

2 PCB breadboard

2 x 3 pins switches

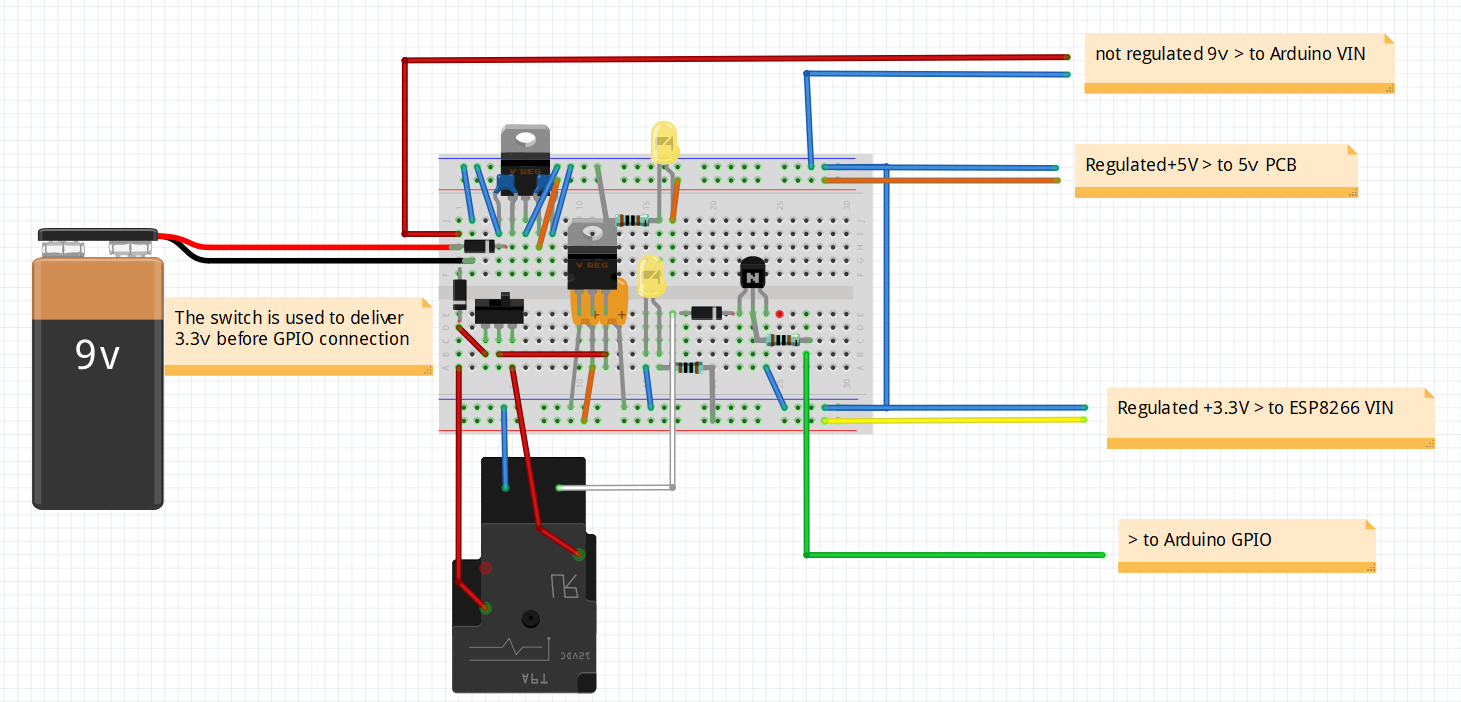
Some connectors and wires

Of course, soldering iron and tin to do it.

### Build the power sources

[This fritzing file describes what to do.](https://github.com/cuillerj/Thermostat/blob/master/thermostatPower.fzz)

It is better to start to build the power sources with a breadboard even if there are no difficulties.

[](https://github.com/cuillerj/Thermostat/blob/master/thermostatPower.fzz)

Regulators can easily be replaced by other ones: just modify connections and capacitors according to your regulators characteristics.

Check it delivers a constant 5v and 3.3v even with a load (100 ohms resistors for instance).

You can now solder all the components on a breadboard PCB as below.

### Build the electronics

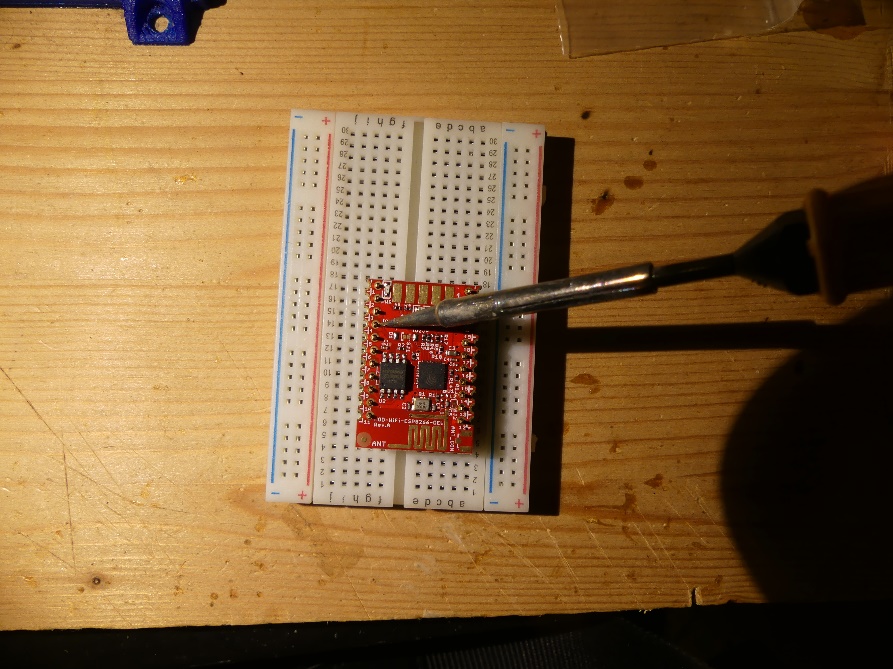
[This fritzing file describes what to do.](https://github.com/cuillerj/Thermostat/blob/master/thermostatElectronic.fzz)

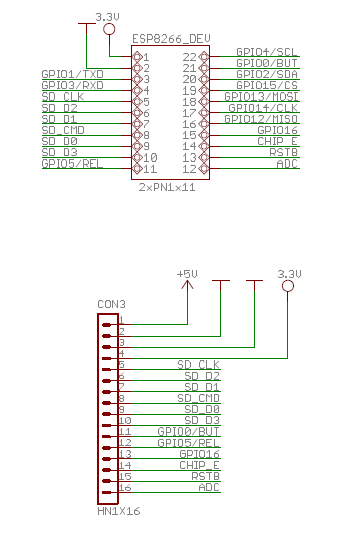
I strongly suggest starting to build the electronics with a breadboard.

#### But even with the breadboard you will have to do some soldering as a first step.

Actually, the ESP8266-Dev from Olimex is coming as below

|  |  |
| --- | --- |
|  |  |

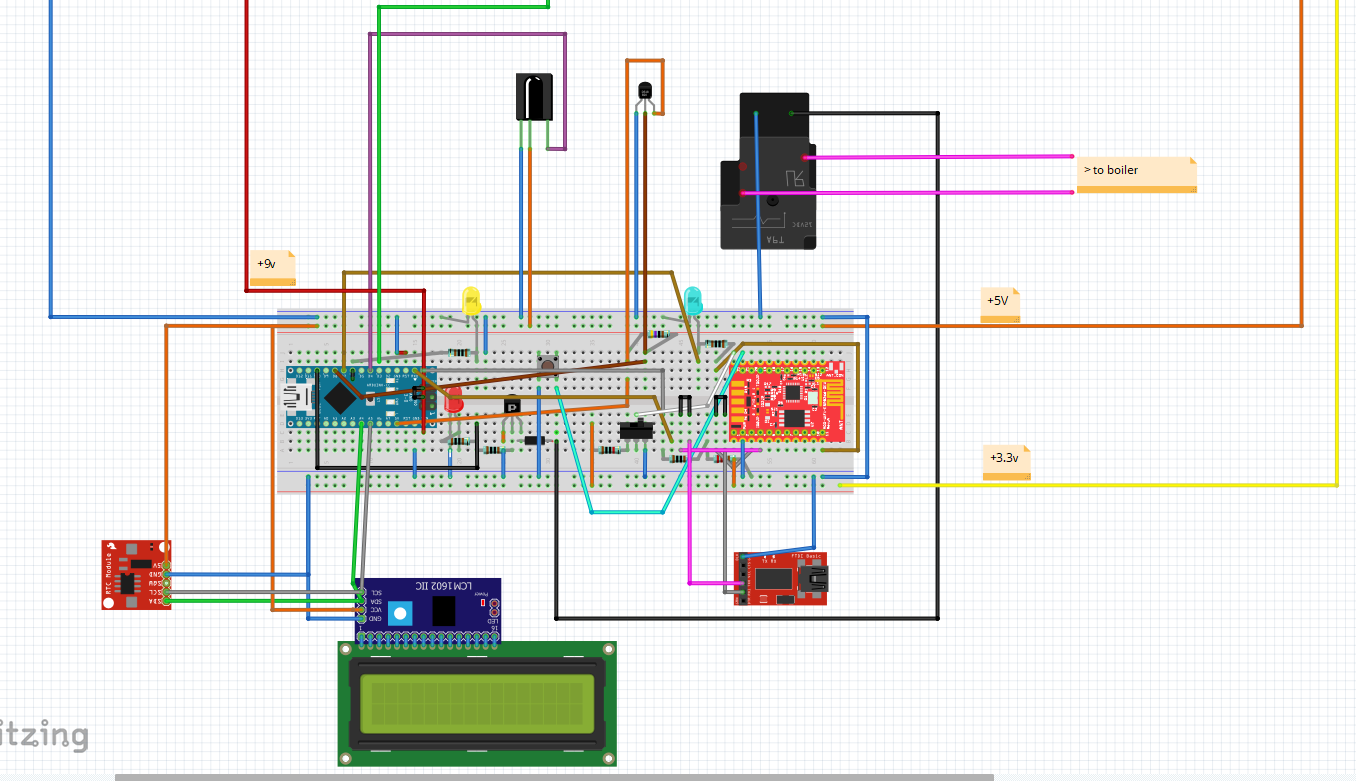
Plug your ESP8266 in a breadboard for an easiest soldering a below

The ESP8266 Olimex-Dev schema

#### Now you are ready to put it all together on the breadboard

[Reproduce the Fritzing reference.](https://github.com/cuillerj/Thermostat/blob/master/thermostatElectronic.fzz)

There is no specific difficulty. Just take your time and check twice!

[](https://github.com/cuillerj/Thermostat/blob/master/thermostatElectronic.fzz)

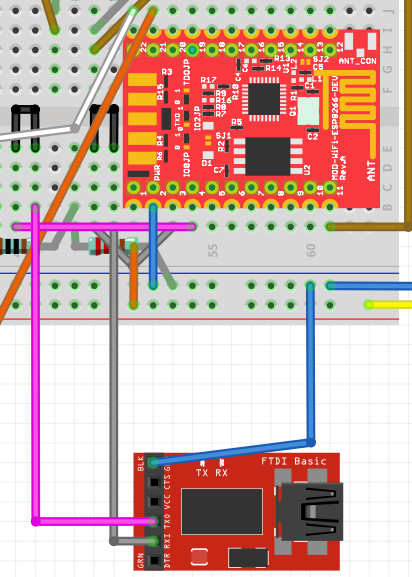
#### Connect carefully power sources

Check power LEDs on the Arduino and ESP8266.

The LCD must light on.

### Let’s do with the gateway configuration

#### Connect the FTDI USB to your development station.

 Set the serial link switch in order to connect ESP8266 to the FTDI as this

#### Prepare to download the gateway code

Start Arduino on your workstation.

You need ESP8266 to be known as board by the IDE.

Select the USB port and the appropriate board with Tools / boards menu.

If you do not see any ESP266 in the list that means you may have to install ESP8266 Arduino Addon ([you can find here the procedure](https://github.com/esp8266/Arduino#installing-with-boards-manager)).

All the code you need is available on GitHub. It is time to download it !

The main code of the Gateway is there: [https://github.com/cuillerj/Esp8266IPSerialGateway...](https://github.com/cuillerj/Esp8266IPSerialGateway/blob/master/Esp8266IPSerialGateway.ino)

On top of standard Arduino and ESP8266 includes the main code need these 2 includes:  
LookFoString that is used to manipulate strings and is there: <https://github.com/cuillerj/LookForString>

ManageParamEeprom that is used to read and store parameters in Eeprom ans is there: <https://github.com/cuillerj/ManageParamEeprom>

Once you get all the code it's time to upload it into the ESP8266.  
First connect the FTDI to an USB port of your computer.

I suggest you check the connection before trying to upload.

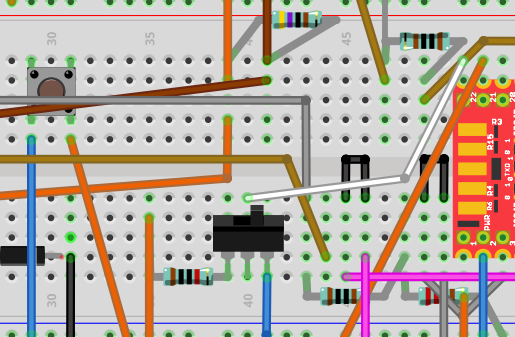
* Set the Arduino serial monitor to the new USB port.
* Set the speed to 115200 both cr nl (defaut speed for Olimex)
* Power on the breadboard (ESP8266 comes with software that deals with AT commands)
* Send "AT" with the serial tool.
* You must get "OK" in return.

If not check your connection and look at your ESP8266 specifications.

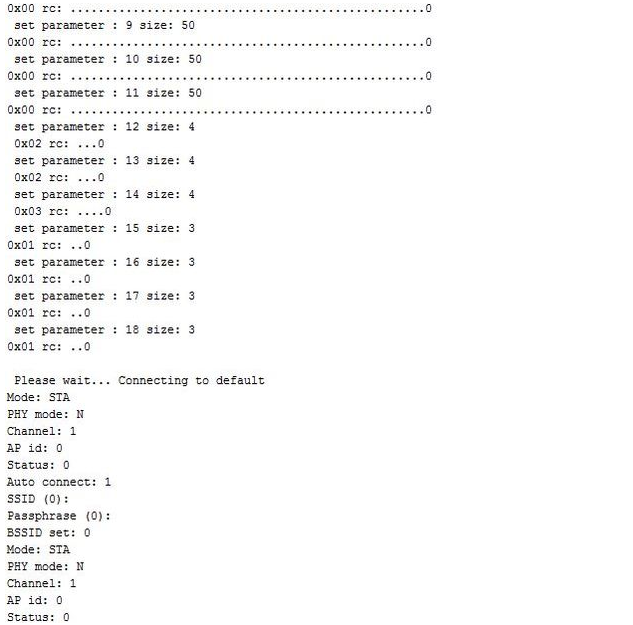
**If you got "OK" your are ready to upload the code**.

#### Download the gateway code

* Power off the breadboard, wait a few seconds,
* press on the push button of the breadboard and power on.



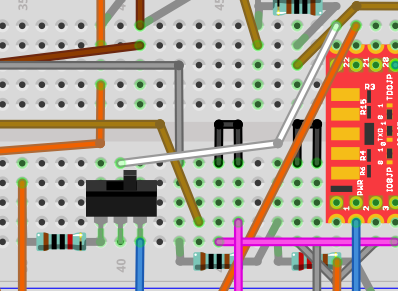
* release the push button It is normal to get some garbage on the serial monitor.
* Press on the upload IDE as for an Arduino.
* After the upload completed set serial speed to 38400.

You will see something as in the picture. 

Congratulation you successfully uploaded the code !

#### Set your own gateway parameters

* Keep opened the Serial Monitor (speed 38400) of the IDE
* Power off the breadboard, wait a few seconds
* Use the switch to set the configGPIO to 1 (3.3v)



Scan the WIFI by entering the command: ScanWifi. You will see a list of the detected network.

* Then set your SSID by entering "SSID1=yournetwork"
* Then set your password by enterind "PSW1=yourpassword"
* Then enter "SSID=1" to define the current network
* Enter "Restart" to connect the Gateway to your WIFI.
* You can verify you got an IP by entering "ShowWifi".
* The blue LED will be on and the red LED blinking.

It's time to define your IP server address by entering the 4 subaddresses (server that will run the Java test code). For instance for IP=192.168.1.10 enter :

* "IP1=192"
* "IP2=168"
* "IP3=1"
* "IP4=10"

Define IP ports as:

* routePort=1840 (or else according to your application configuration see “Server installation guide”)

Enter "ShowEeprom" to check what you just stored in Eeprom

Now set the GPIO2 to ground to leave the configuration mode (use the switch to do so)

Your Gateway is ready to work !

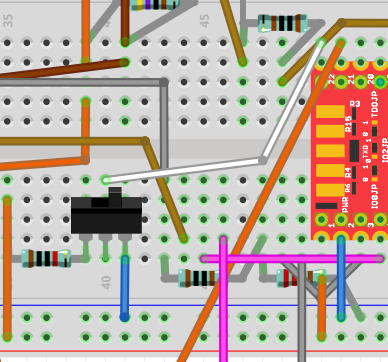
The blue LED must go on as soon as the gateway is connected to your WIFI.

*There are some others commands you could find in the gateway documentation.*

## Let’s do with the Arduino side

Before working with the Thermostat code let’s do some tests with the IDE example sources

Firstly, unplug the serial link connectors to avoid USB conflict.



Connect the Arduino USB to your workstation.

Chose Serial Port, set speed to 9600 and set card type to Nano.

#### Check the temperature sensor

Open Files/ examples /Max31850Onewire / DS18x20\_Temperature and modify OneWire ds(8); (8 instead of 10).

Upload and check it works. In case not check your DS1820 connections.

#### Check the clock

Open Files/ examples / DS1307RTC / setTime program

Upload the code and check you get the right time.

#### Check the LCD

Open Files/ examples / liquid cristal / HelloWorld program

Upload the code and check you get the message.

#### Check the remote control

Open Files/ examples / ArduinoIRremotemaster / IRrecvDemo program

Modify the PIN to 4 – upload the code

Use your remote controller and check you get the IRs code on the monitor.

It is time to choose the remote control 8 different keys you want to use as below:

* increase temperature instruction
* decrease temperature instruction
* switch off the thermostat
* select the week agenda mode
* select the first day agenda mode
* select the second day agenda mode
* select the not freezing mode
* power on/off the WIFI gateway

Since you made your choice use the key, copy and save in a text document the received codes. You will need this information later.

#### Check the network connection

To check your work the best is to use the Arduino and Java examples.

##### Arduino

You can download it there: <https://github.com/cuillerj/TestSerialLinkLibrary>

It includes SerialNetwork library that is here: <https://github.com/cuillerj/SerialNetwork>

Just upload the code inside your Arduino.

##### Server

The server example is a Java program that you can download here: [https://github.com/cuillerj/ESP8266SerialUdpGatewa...](https://github.com/cuillerj/ESP8266SerialUdpGatewayExample/blob/master/src/ESP8266SerialUdpGatewayExample.java)

Just run it

Look at the Java console.

Look at the Arduino monitor.

Arduino send 2 different packets.

* The first one contains the digital pins 2 to 6 status.
* The second one contains 2 random values, the voltage level of A0 in mV and incremental count.

The Java program

* print the received data in hexadecimal format
* reply to the first kind of data with a random on/off value to set on/off the Arduino LED
* reply to the second kind of data with the received count and a random value.

You must see something like this.

|  |  |
| --- | --- |
| Arduino console | Java console |
|  |  |

**You are now ready to work on Thermostat code!**

Thermostat sources are available on GitHub

First [download this library](https://github.com/cuillerj/thermostatParameters) and copy files in your usual library.

[Then download these sources](https://github.com/cuillerj/Thermostat/blob/master/thermostatElectronic.fzz) and copy files in your usual Arduino sources folder.

Open Thermosat.ico and compile and check you do not get errors

Connect the Arduino USB to your workstation.

Set speed to 38400.

We need to set the Arduino in configuration mode

Plug a connector on the ICSP so that GPIO 11 is set to 1 (5.5v)

|  |  |
| --- | --- |
|  |  |

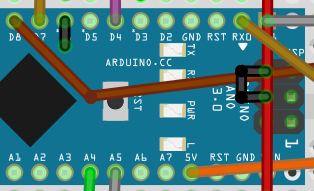
Upload the Arduino code.

The Arduino will start automatically.

Wait for the message “end init eeprom".

The default parameter’s values are now written in the eeprom.

Plug the connector on the ICSP so that GPIO 11 is set to 0 (ground) to set the Arduino in running mode.



Reset the Arduino.

You must see time on the LCD and the yellow LED must be on. (You will see 0:0 if the clock has not be synchronized or time lost (powered of and no battery)).

After a few seconds you will see alternatively this kind of screen on the LCD.

|  |  |
| --- | --- |
| Screen 1 | Screen 2 |

**Look at the LCD.**

You will see alternatively 3 different screens. I will describe the 2 above. The 3rd one is described in the maintenance guide.

Common to screen 1 & 2:

* at the left of the top: the actual time
* at the left of the bottom: the actual temperature instruction
* at the middle of the bottom the: actual inside temperature (DS1820)

Screen 1:

* at the middle of the top: actual running mode

Screen 2:

* at the middle of the top: actual day of the week
* at the right of the top: day & month numbers

#### Test the relays

##### Test the Gateway relay

At this stage you must be WIFI connected and the blue LED must light on.

Press the remote controller key you selected to power on/off the WIFI gateway. The relay must switch off the ESP8266 and the blue LED.

Wait a few seconds and press again the remote controller key. The WIFI gateway must be powered on.

Within a minute le the gateway must be connected, and the blue LED must light on.

##### Test the boiler relay

First look at the red LED. If temperature instruction is much higher than the inside temperature the LED must light on. It takes a few minutes after the start for the Arduino to get enough data to decide weather or not to heat.

If the red LED is on, decrease the temperature instruction to set it low below the inside temperature. Within a few seconds the relay must switch off and the red LED light off.

If the red LED is off, increase the temperature instruction to set it low below the inside temperature. Within a few seconds the relay must switch on and the red LED light on.

If you do it more time, bear in mind that the system will not react immediately to avoid too fast switch the boiler.

That’s the end of the breadboard work.

## Solder the components

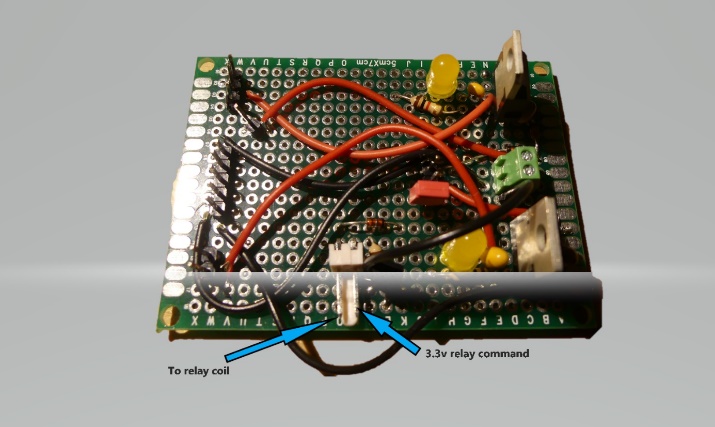
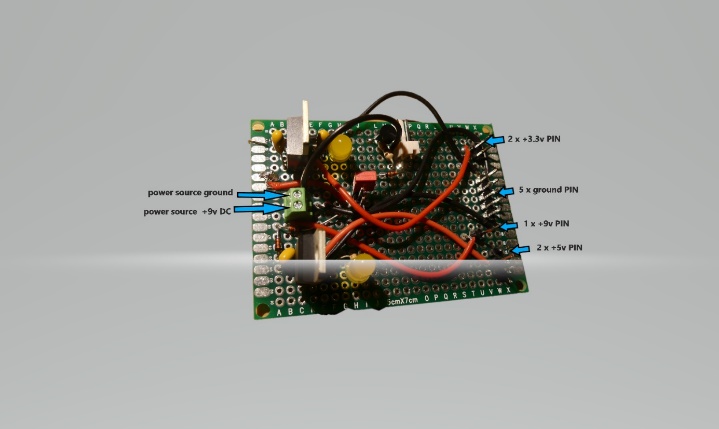
I suggest using 2 different PCB: one for the power supply and one for the micro-controllers.

### Solder the power supply

[Here is the Frizting to follow !](https://github.com/cuillerj/Thermostat/blob/master/thermostatPower.fzz)

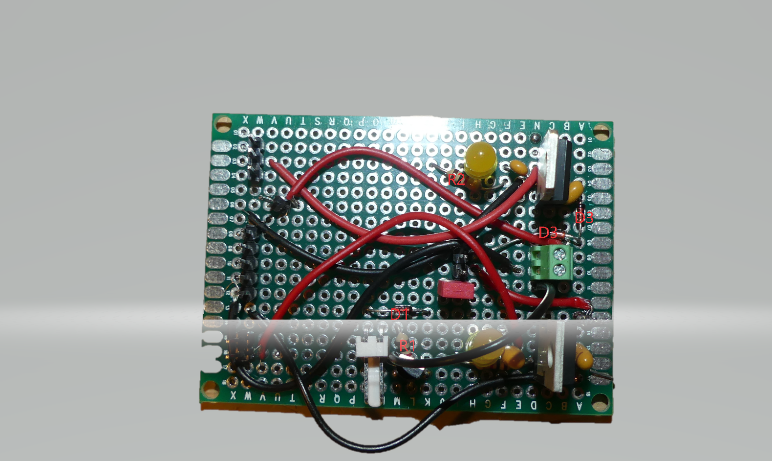
You will need connectors for;

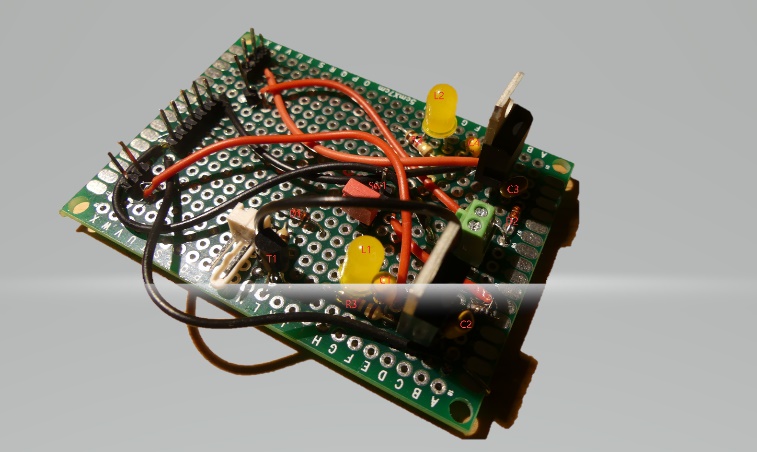
* 2 for 9v input power supply
* 1 for +9v output
* 1 for +3.3v output (I did 2)
* 2 for +5v output (I did 3)
* 2 for relay command
* 2 for relay power



|  |  |
| --- | --- |
|  |  |

You can see below the parts numbers according to the Fritzing model.



Solder the microcontrollers on PCB

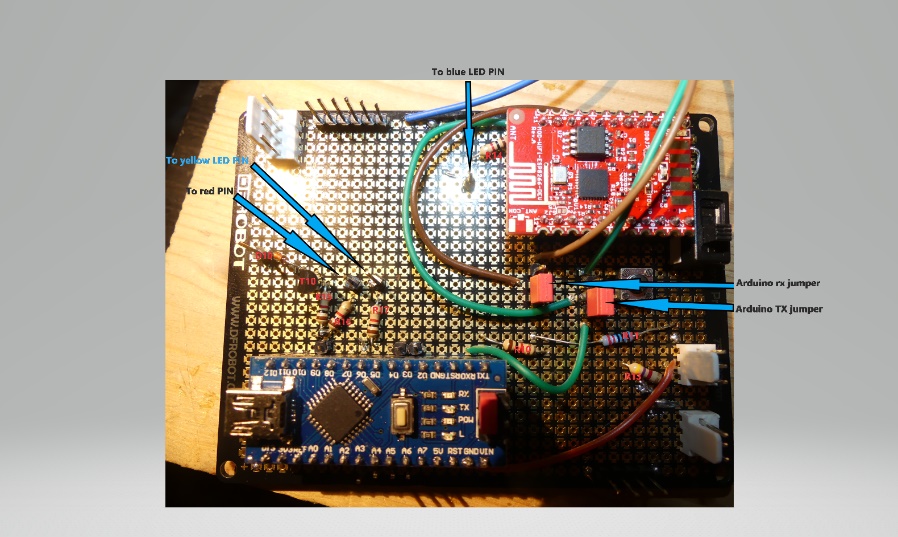
[Here is the Frizting to follow !](https://github.com/cuillerj/Thermostat/blob/master/thermostatElectronicDesign.fzz)

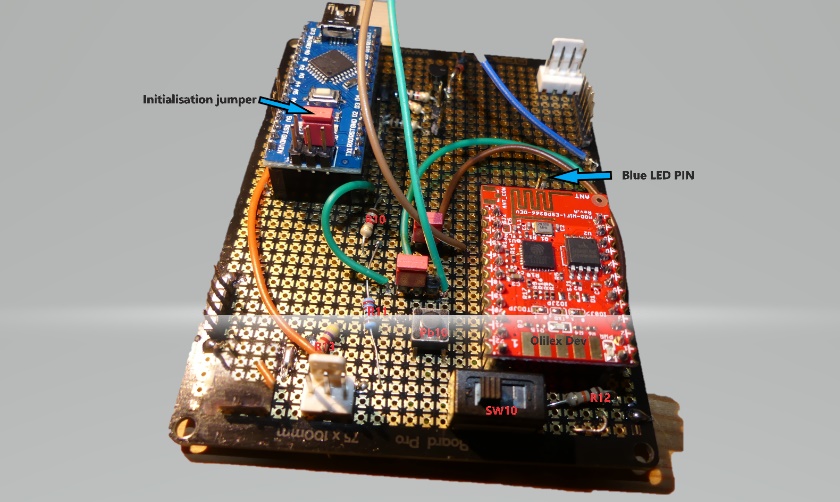
You will need connectors for:

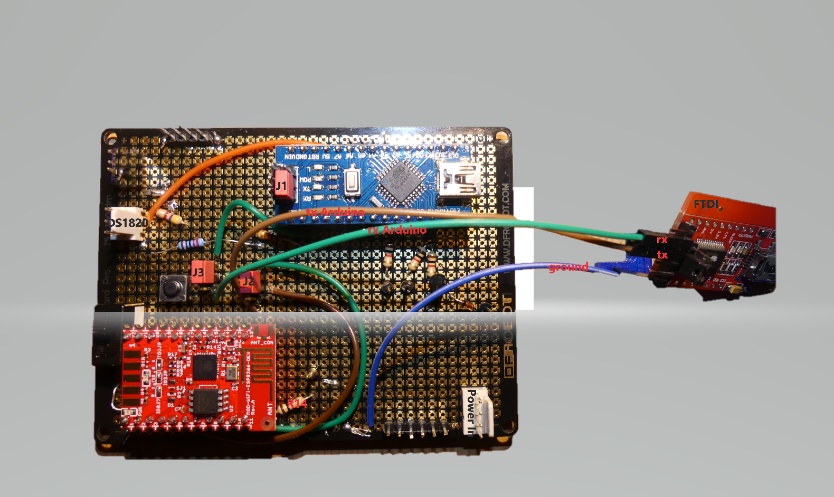
* 3 x +5v (I did one spare)
* 6 x ground
* 3 x for DS1820
* 3 x for LED
* 1 x IR receiver
* 2 x for relay command
* 4 x for I2C bus

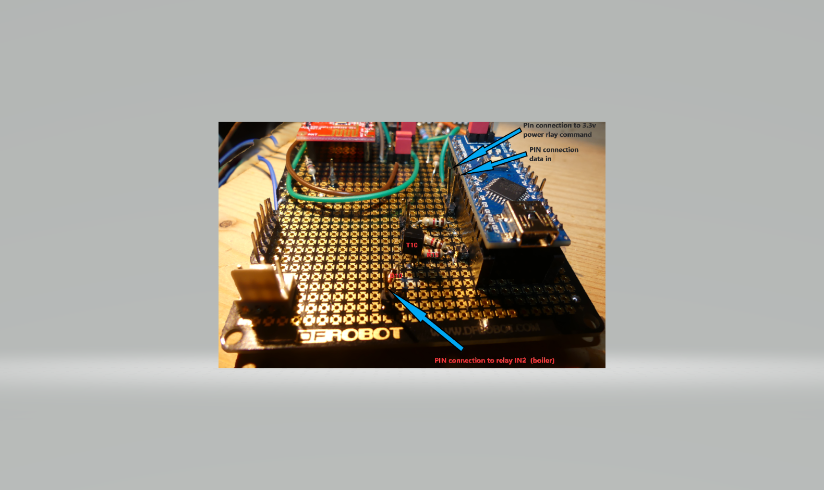


You can see below the parts numbers according to the Fritzing model.









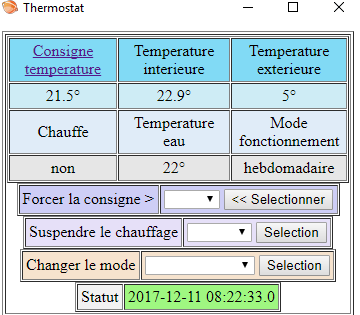
## 

The I2C connection

Connect and check altogether before putting in the box

## 

## User guide



## Maintenance guide

## An open system

### Change than can be done easily

Replace internet weather forecast by a DIY external temperature sensor.

Replace Linux server with a Windows one.

### Improvements that could be done

Replace the breadboard PCB with a printed PCB.

### Communication specifications

The ESp8266 Serial/UDP gateway allows to route up to 30 bytes. No other constraints for ASCII data. Regarding binary data it can send any 2 consecutive bytes. Those 3 consecutive bytes "0x7f7f7f & 0x7f7e7f" that are forbidden.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| serial link level | | | UDP Level | | | | | |
| 0x7f | 0x7e | 0x7f | FNumber | 0x00 | Req | 0x00 | FLen | DataBytes |

FNumber: frame number modulo 256

Req format:

* the bit b1000 0000 determines request or response frame
* the bit b0100 0000 determines if the frame has to be acknowledged or not
* the bit b0000 0001 reserved

FLen: number if bytes at UDP level

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DataBytes | | | | |
| unitG | unitID | Cmd | Data bytes | CRC |

unitG: 1 byte unit group unitID: 1 byte unit number 256\*unitG + unitID = the thermostat identifier

Cmd: 1 byte command identifier (commands list are defined in ThermostatDefines.h)

CRC: 1 byte CRC