TichHome

TichHome

TichHome is Raspberry Pi based home automation / smart home / smart house solution based on common and cheap components, works realiably and I think is very convenient to use.

It allows to control many things from any device in local network and also automate certain things (oh that hot coffe in the morning ready when you wake up).

My main goal for this home automation system was to create something that:

- would work inside of local network and be accessible from any device whether it is computer, phone or tablet for convenient use.

- not to be dependent in any way on anything outside of local network – there are no connections to cloud or some 3rd party services. It is all „inside" of local and independent from outside world. Internet is not needed here – just internal network will suffice.

- most important initial feature of this system was delayed auto disable. Devices that you configure into the system do not have to have that feature enabled but it definately turns out very practical in everyday usage – especially for lamp devices.

What it currently provides:

- web interface which allows to operate defined devices to anyone from your home network
- it allows control over radio operated devices/power switches
- it allows control over web operated devices/power switches
- provides way to schedule working time frames for devices by simply clicking in UI
- allows to define auto disable option with delay times after which devices would be turned off (energy saving for forgetting and salvation for lazy ones ;-))
- allows to define PIR sensors connected to Raspberry pi and basing on their signal can operate defined devices and/or trigger alarm/send alarm mails

Web interface looks like on picture below. It is tablet and phone friendly. Demo can also be viewed under this address: https://github.com/Sznapsollo/TichHome

Below few screens and brief description of what to expect. Detailed description of how to use TichHome will be provided in next chapters.

Radio Switch 🖮   2h 0m

Enable	Disable

Coffee machine 🖮   2h 0m

Enable	00:03:59 (19:00)

Humidfier Web Switch 🖮   30m

Enable	Disable

Living Room Lamp 🖮   30m

Enable	00:23:59 (19:20)

TV Lamp 🖮   2h 0m

Enable	Disable

Room Lamp 🖮   2h 0m

Enable	Disable

Computer TV 🖮 Check availability

Enable	Disable

TV stack 🖮

Enable	Disable

All lights

Enable	Disable

For devices that can be be auto disabled after delay there is default auto off time defined in such device definition but user can also manually change delay when tapping on device name and using the following slider:
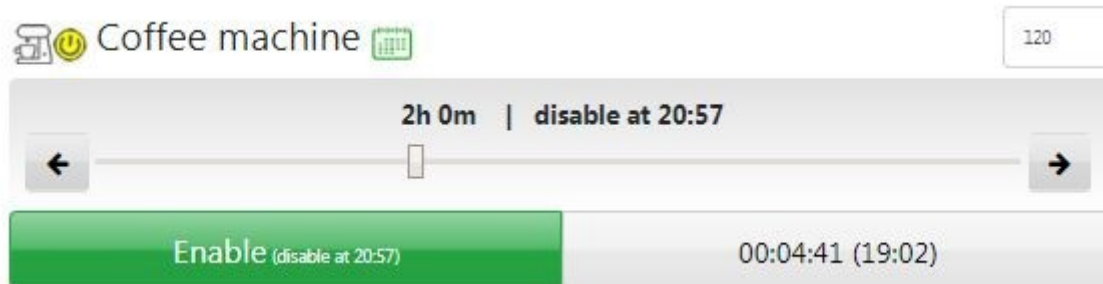
enabled device will then count down its life time on Off button

Schedulter timelines can be specified for the device by clicking on calendar icon and setting up its timelines:
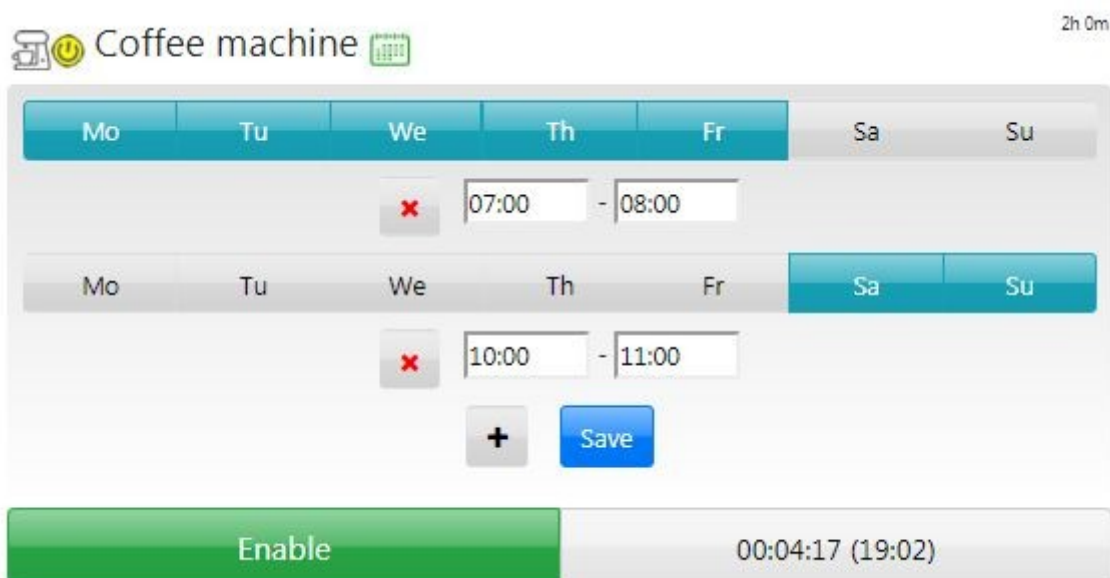
What Hardware is needed

1. Raspberry Pi with AC adapter/SD card/Wifi dongle. Because Raspberry Pi 3 has embedded wifi and is pretty fast I would recommend it but TichHome should work on any Rasberry Pi. I have run it on 1/2/3/4 and had no problems on any. However R1 is already a bit on the slow side.

2. If you are going to operate radio controlled power switches:

   1. You will need radio transmitter that can be connected to Raspberry PI. I use **XY-FST, XD-FST** radio transmitters.

   2. Radio signal receiver is optional and is needed if you dont know radio code that should be send in order to put it into the TichHome. In such case you read radio signal, that operates your radio controlled switch, with radio receiver connected to your Raspberry Pi and then put the code into the TichHome. I use **XY-MK-5V, XD-RF-5v** radio
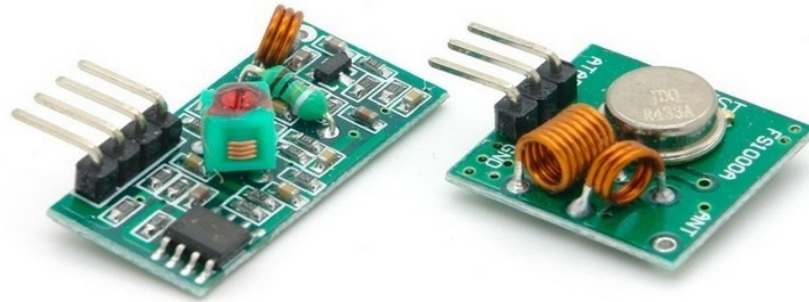
receivers.

3. Note that above transmitters should suit frequency of devices that you want to handle. Above devices would handle 315Mhz, 330 Mhz and 433 Mhz frequencies. I have only worked with them for 433Mhz switches.

4. Below I puth url to youtube playlist that will contain any videos regarding TichHome that I create. It will include vids describing what these devices are, how to connect them and how to use them with some radio controlled power switches.

   1. https://www.youtube.com/watch?v=C19ARWDYR3c&list=PLjd2MVjW6mhFygrvXyVcdNoq6pHK8MdUW

3. TichHome also allows to define devices that will send web requests for ENABLE and DISABLE actions. These web requests operate as GET requests and can contain anything you define for ENABLE or DISABLE action. This actually presents many possibilities of use:

   1. For example you can operate **SONOFF** devices which are basically wifi operated powerswitches. SONOFF devices allow to upload flash them with your own firmware (lua scripts) which basically allows to to define how they are suppose to operate. I will create video describing SONOFF devices use, how to change their firmware and also provide script than can be used to connect SONOFF devices to TichHome. Below image of **basic model** but it would probably work on all models using ESP8266 chip. I have also used **TH10** and **TH16** models and they work great.
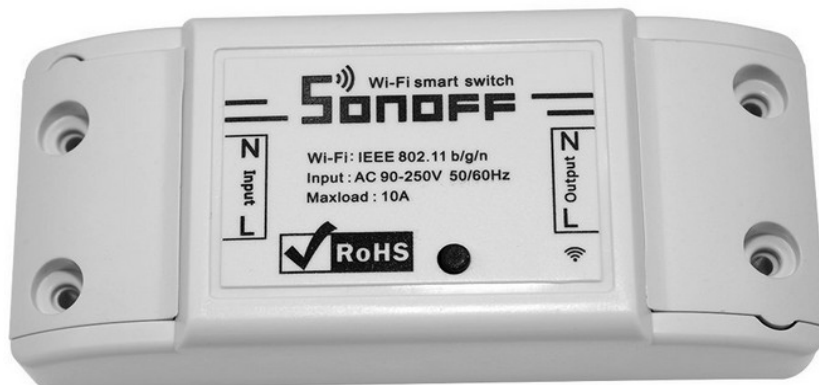
2. You can send requests to any device on your network. Requests can be different for ENABLE OR DISABLE action just the way you define them in TichHome configuration.

4. TichHome allows to connect and manage **HC SR501** PIR detector which can be configured to enable other devices in the system or to trigger alarms.

TichHome deployment in big short

I am not a doc guy so writing spesc and istructions is not easy for me. Believe me that I did put some effort into describing what should be done below and if this is still not clear please contact me at office@webproject.waw.pl . I will try to help if I can.

In big short please follow these steps – all of them i tried to describe in detail in the following chapters chapters to make it all work. The list seems long but once you do it it becomes super fast & easy any next time ;-) Enjoy :-)

// All installation steps super brief

- Install Raspbian

- setup hosts, hostname, enable vnc, ssh

- Initial run and setup wifi and static IP (from this point you are headless)

- sudo apt-get update

- sudo apt-get upgrade

- install ftp: sudo apt-get install proftpd

- install java: sudo apt install openjdk-8-jdk

- sudo update-alternatives --config java (choose java8)

- create tichhome folder

- copy webroot

- copy jar

- copy testfolders to main folder

- change settings file

- compile RadioDevices: sudo make, if it does not compile and RCSwitch.o changed its size from 12.6 to 12.1 copy RCSwitch.o again and retry compile

- test if scripts work: ./codesend 1070161, ./codesend 1070164

- test if python scripts work: python /home/pi/tichhome/processes/./run_radio_switch.py /home/pi/tichhome/RadioDevices/codesend 1070164

- test run: java -jar /home/pi/tichhome/tichhome-1.0-SNAPSHOT-fat.jar --conf /home/pi/tichhome/config/settings.json

- should be running http://192.168.0.44:8081/tichhome/#/

- create service: sudo cp /home/pi/tichhome/tichhome.service /etc/systemd/system/tichhome.service (add to solution)

- sudo systemctl enable tichhome.service

- sudo systemctl start tichhome.service

- [optional] sudo systemctl stop tichhome.service

- [optional] sudo systemctl status tichhome.service

- add items

- enjoy!

For more details see following chapters in this doc.

Rasberry Pi Raspbian installation and setup

This chapter is about setting up Rasberry Pi to be able to work with TichHome system. If you see point here that you already have installed just skip it and go to the next.

Raspbian System installation

In big short. Those who did this before even once will know whats what and will do this crazy fast. For those who did not – use below points as general hints. Use instructions and all helpul info from https://www.raspberrypi.org/learning/software-guide/ . This point can be skipped if you already have Raspberry Pi with system on it.

1. Download Noobs image from https://www.raspberrypi.org/downloads/noobs/
2. Format SD card according to instructions from raspberry pi site.
3. Copy extracted system files directly to SD card
4. Run Raspberry Pi and let it install system
5. after instllation run to have latest version of your system:
    1. sudo apt-get update
    2. sudo apt-get upgrade
    3. sudo apt-get dist-upgrade

Setup Machine name

1. `sudo nano /etc/hosts`

Leave all of the entries alone except for the very last entry labeled `127.0.1.1` with the hostname "`raspberrypi`". This is the only line you want to edit. Replace "raspberrypi" with whatever hostname you desire.

1. `sudo nano /etc/hostname`

2. sudo reboot

Setup password

Sudo raspi-config

```
1 Change User Password          Change password for the current u
2 Hostname                      Set the visible name for this Pi
```

Setup Wifi

sudo nano /etc/network/interfaces

```
  GNU nano 2.2.6          File: /etc/network/interfaces

# interfaces(5) file used by ifup(8) and ifdown(8)

# Please note that this file is written to be used with dhcpcd
# For static IP, consult /etc/dhcpcd.conf and 'man dhcpcd.conf'

# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

iface eth0 inet manual

allow-hotplug wlan0
iface wlan0 inet manual
    wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf

allow-hotplug wlan1
iface wlan1 inet manual
    wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
```

sudo nano /etc/wpa_supplicant/wpa_supplicant.conf

```
  GNU nano 2.7.4     File: /etc/wpa_supplicant/wpa_supplicant.conf     Modified

ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
ap_scan=1
update_config=1
network={
        ssid="MyNetworkName"
        psk="Supersecretpassword"
        key_mgmt=WPA-PSK
        scan_ssid=1
}
```

Setup Static IP

Sudo nano /etc/dhcpcd.conf

```
# Example static IP configuration:
interface wlan0
static ip_address=192.168.0.99/24
static routers=192.168.0.1
static domain_name_servers=192.168.0.1 8.8.8.8
```

Enable FTP

1. sudo raspi-config
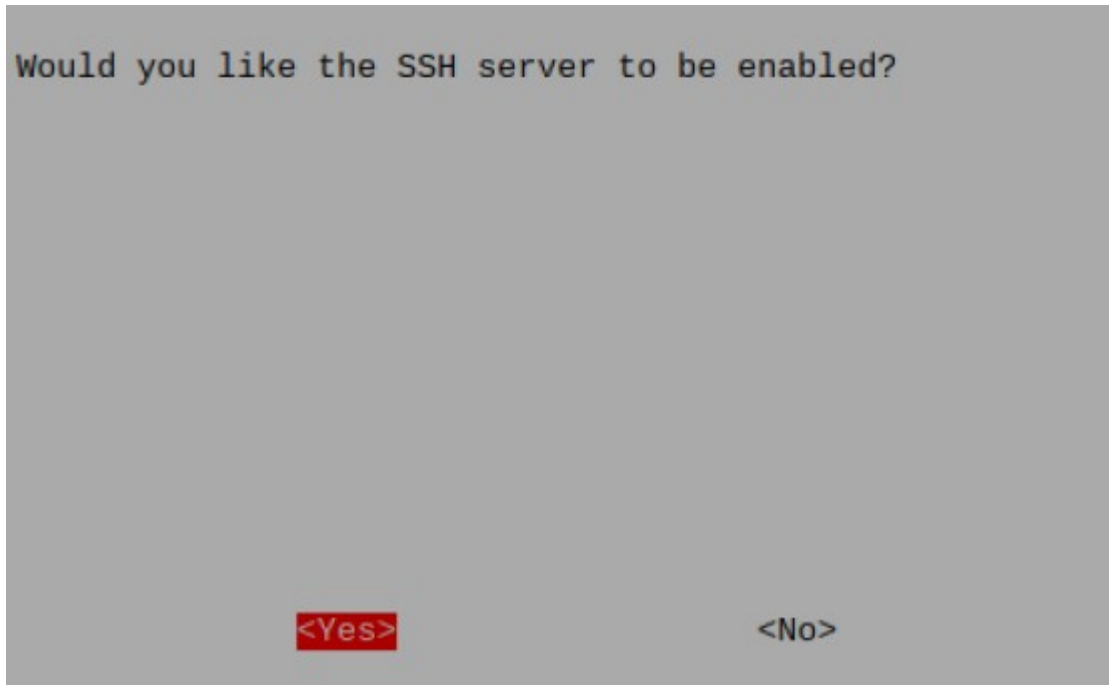
2. `sudo apt-get install proftpd`

   1. choose standalone
   2. [optional config changes] `sudo nano /etc/proftpd/proftpd.conf`
   3. `sudo service proftpd restart`

Enable VNC

3. sudo raspi-config
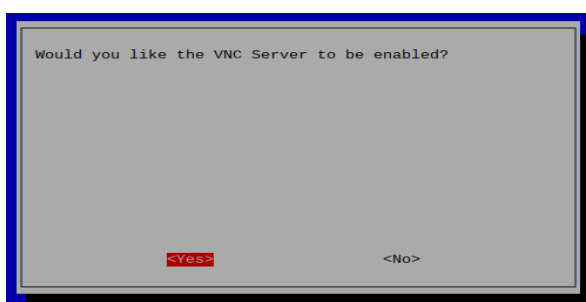
Now using VNC client you can connect to your Rasberry Pi using address yourstaticip:0, login: pi, password is you password that you set in „Setup password" chapter.

[Optional] Install wakeonlan

If you want to have option that would allow you to enable computers in your net via TichHome →
wakeonlan should be installed.

1. sudo apt-get install wakeonlan

Install and activate smbus and i2c

1. sudo apt-get install python-smbus
2. sudo raspi-config
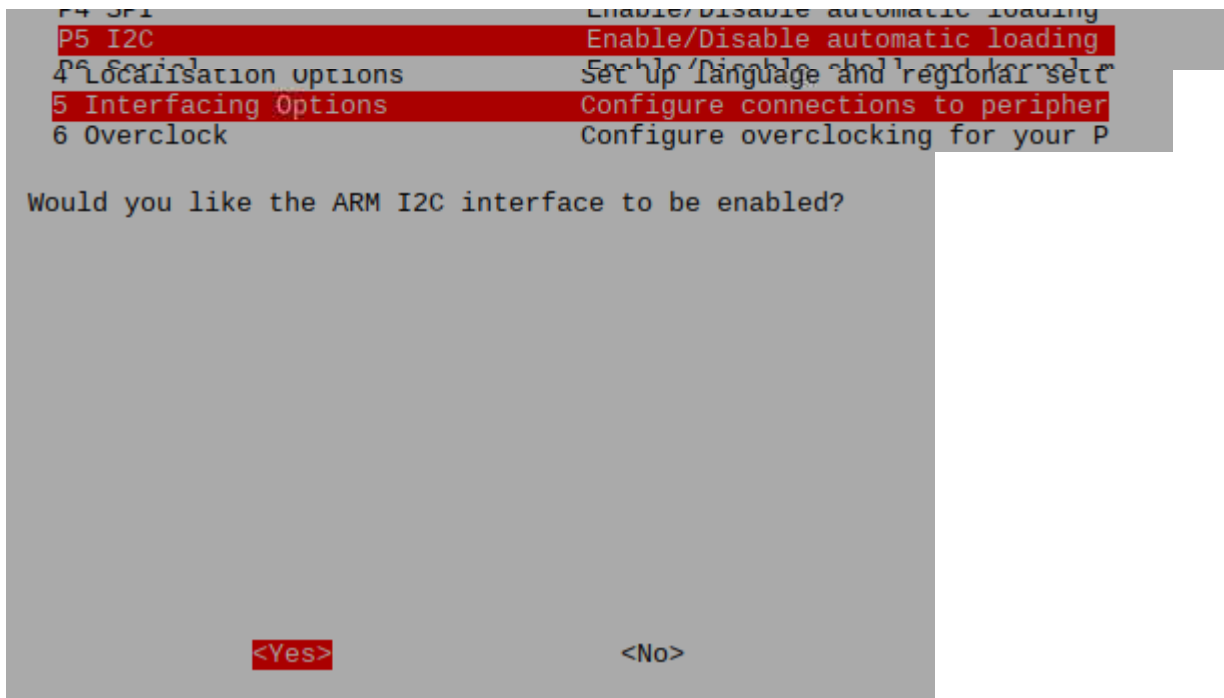
Install Java 8

1. sudo apt install openjdk-8-jdk

Now if on your Raspberry there are other versions of Java lets make Java 8 the default one.

2. sudo update-alternatives --config java
3. Choose Java 8

Java 8 will be necessary to build TichHome solution.

TichHome files deployment

Compile TichHome

Open TichHome solution in your IDE (for example Visual Studio Code) and compile it

1. ./gradlew shadowJar

Succesful compilation will result with creation of tichhome-1.0-SNAPSHOT-fat.jar file in your build/libs folder

Deploy files to Raspberry pi

In your home directory in Raspbian create tichhome directory (or any other). Remember that the name you choose will have to be later used in settings files to properly configure TichHome. For the sake of this tutorial i created in home folder „tichhome" directory.
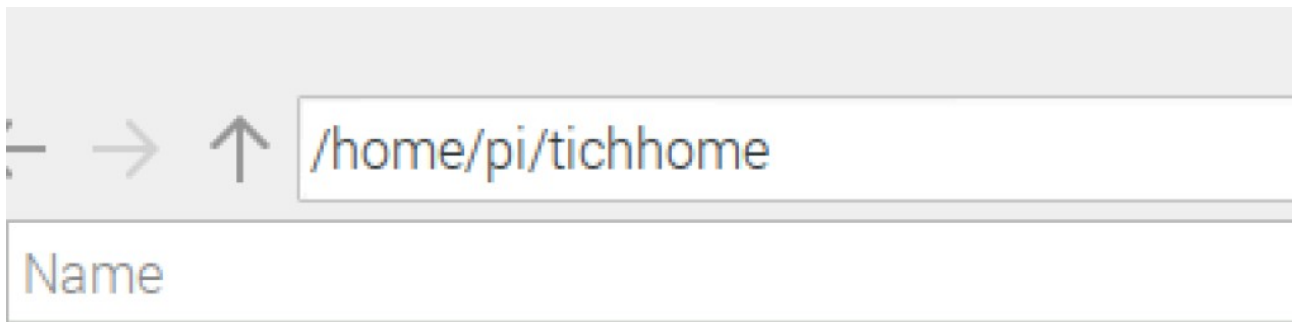
1. Copy [your machine tichhome git folder] build/libs/ tichhome-1.0-SNAPSHOT-fat.jar to [Raspberrypi] home⁄pi/tichhome

2. Copy [your machine tichhome git folder] testFolders content to [Raspberrypi] home⁄pi/tichhome

3. Copy [your machine tichhome git folder] tichhome.service to [Raspberrypi] home⁄pi/tichhome

The endresult should look somewhat like that:
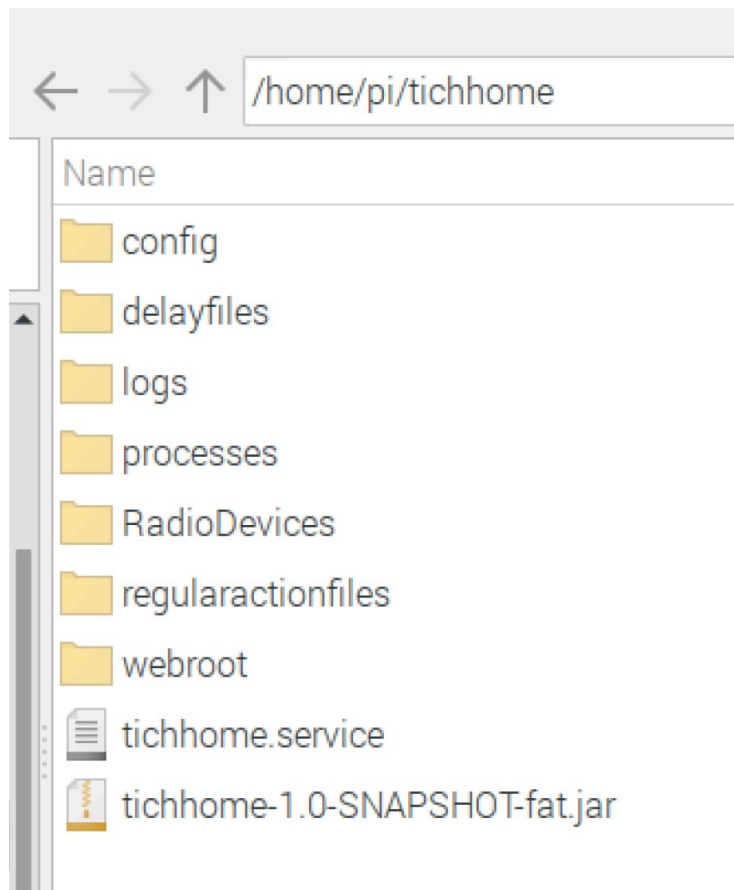
Modify settings file

Open /home/pi/config/settings.json

Change nodes visible below to match your server address and your paths. Below i put screen of how this should look in case of tichhome folder on your Raspberry and considering that your raspberry operates on 192.168.0.44 ip address in your local network. For now you can ignore all settings that are below the ones from screen (do not modify them).

```
settings.json   ✕
1  ⊟ {
2         "datagram.address" : "0.0.0.0",
3         "datagram.port" : 8082,
4         "eventbus.messages.delay": 4000,
5         "eventbus.subscription.idle.lifespan": 300000,
6         "http.server.address": "0.0.0.0",
7         "http.server.port": 8081,
8         "http.server.eventbus.path": "eventbus",
9         "http.server.root.path": "tichhome",
10        "www.path": "/home/pi/tichhome/webroot",
11        "backups.path": "backups",
12        "nodes.file.path": "/home/pi/tichhome/config/nodes.json",
13        "sensors.file.path": "/home/pi/tichhome/config/sensors.json",
14        "web.server.address":"http://192.168.0.44:8081/tichhome/",
15        "satellite.server.addresses":["http://192.168.0.44:8081/tichhome/"],
16        "processes.files.path": "/home/pi/tichhome/processes/",
17        "logs.folder.path": "/home/pi/tichhome/logs/",
18        "delay.files.path":"/home/pi/tichhome/delayfiles/",
19        "regularaction.files.path":"/home/pi/tichhome/regularactionfiles/",
20        "sensor.settings.files.path":"/home/pi/tichhome/sensorsettingsfiles/",
21        "code.send.path":"/home/pi/tichhome/RadioDevices/codesend",
22        "conrad.code.send.path":"/home/pi/tichhome/RadioDevices/codesendConrad",
```

Compile signal sending files

Go to directory
  – cd /home/pi/tichhome/RadioDevices

and perform „sudo make" operation to recompile script files that will be used for transmitting signals. The results should look somehow like this. You will need to have wiringPi installed on your Raspberry pi for this compilation to succeed. If you do not have wiringPi you should install it by following these steps: http://wiringpi.com/download-and-install/


```
pi@raspi4:~/tichhome/RadioDevices $ sudo make
g++    RCSwitch.o send.o -o send -lwiringPi
g++    RCSwitch.o codesend.o -o codesend -lwiringPi
g++    RCSwitch.o paringConrad.o -o paringConrad -lwiringPi
g++    RCSwitch.o receive.o -o receive -lwiringPi
g++    RCSwitch.o RFSniffer.o -o RFSniffer -lwiringPi
```

I did receive some feedback that for some people there are problems with this compilation after which compilation returns error and RCSwitch.o file changes its size from 12.6Kb to 12.1Kb

RCSwitch.o   12.6 KiB

If that happens.

1. Delete RCSwitch.o file.

2. Copy [your machine tichhome git folder] testFolders/RadioDevices/RCSwitch.o content to [Raspberrypi] home/pi/tichhome/RadioDevices

3. on raspberry go to home/pi/tichhome/RadioDevices folder

4. sudo make

5. It should compile fine now

RadioDevices is based on RCSwitch arduino library: https://github.com/sui77/rc-switch/

Radio Controlled Switches setup

At this point I assume that you deployed TichHome files onto Raspberry Pi and also recompiled RFSniffer libraries as described in „Compile signal sending files" chapter

How to set up radio controlled power switches I describe in vids that are on playlist below:

- https://www.youtube.com/watch?v=C19ARWDYR3c&list=PLjd2MVjW6mhFygrvXyVcdNoq6pHK8MdUW

To proceed with reading your powerswitches signals and testing them using RFSniffer in chapters below you should have your radio transmitter (example: XY-FST) and radio receiver (example: XY-MK-5V) conneted to Raspberry Pi in a way shown in vids above.

In case of TichHome system transmitter data pin should be connected to pin **11** (**GPIO17**) and receiver data pin should be connected to pin **13** (**GPIO27**).

[Optional] Test running of sending signal scripts (RCSwitch)

If you have RF transmitter connected to Raspberry pi and have compiled signal sending files you can perform simple tests to check if they work. To make this test you should also have some RF switch and know the codes to this switch ON/OFF. How to obtain these codes is described later in chapter describing usage of RFSniffer tool.

For the sake of this test klets say that we have RF switch device which is turned on by signal 263505 and turned off by signal 265506

go to home/pi/tichhome/RadioDevices folder

./codesend 263505

The output should be as following and device should turn on.

./codesend 263506

The output should be as following adn device should turn off.

```
pi@raspi4:~/tichhome/RadioDevices $ ./codesend  263506
sending code[263506]
```

If these tests produce similar output then this test is considered as a success. Scripts work we can check next things.

If it did not that is not the end of the world since the pulse signal that your device use may be different than codesend file is actually using.

For example TichHome system uses different file for radio controlled powerswitches manufactured by Conrad company. You can check out contents of this file in /home/pi/tichhome/RadioDevices/codesendConrad.cpp. See that different pulse is set there which is suitable for Conrad devices (701). It might be that your devices need also different pulse. To find out simplies would be to google it and with luck find proper setting for RFSniffer. Then overwrite codesend.cpp (previously backing it up) or creating new file, then compile it using make and try again. Good luck!

[Optional] Test running of python sending signal scripts

TichHome uses some python scripts to operate on RCSwitch and provide some additional functionality. At this point assuming that, like mentioned in previous chapter, you already have RCSwitch scripts compiled and have connected transmitter and have some radio switches ready – you can perform some simple tests to see if all works ok.

On raspberry run the following line

python /home/pi/tichhome/processes/./run_radio_switch.py /home/pi/tichhome/RadioDevices/codesend 263505

The output should be as following and device should turn on.


```
pi@raspi4:~/tichhome/RadioDevices $ python /home/pi/tichhome/processes/./run_rad
io_switch.py /home/pi/tichhome/RadioDevices/codesend 263505
[2021-10-06 20:17:59]: [run_radio_switch] execute radio switch...

sending code[263505]
```

On raspberry run the following line

python /home/pi/tichhome/processes/./run_radio_switch.py /home/pi/tichhome/RadioDevices/codesend 263506

The output should be as following and device should turn off.


```
pi@raspi4:~/tichhome/RadioDevices $ python /home/pi/tichhome/processes/./run_rad
io_switch.py /home/pi/tichhome/RadioDevices/codesend 263506
[2021-10-06 20:19:20]: [run_radio_switch] execute radio switch...

sending code[263506]
```

Test running TichHome solution

If you set up your files as mentioned in previous chapter run the following line on your raspberry.

java -jar /home/pi/tichhome/tichhome-1.0-SNAPSHOT-fat.jar --conf /home/pi/tichhome/config/settings.json

This should run TichHome solution. If it runs without problems the console output should look like

this:

And assuming that you did setup as mentioned in previous chapters the TichHome website will be accessible under URL http://192.168.0.44:8081/tichhome/ If it is so please stop TichHome by ending running of the jar (ctrl+c in console) and get to next chapter.

Setup TichHome as service

The goal is to setup TichHome as autostarting service. There is ready service file provided here and it is easily done by performing the following steps. (I assume that you did copy tichhome.service file to your Raspberry)

On Raspberry open console and run the following command

sudo cp /home/pi/tichhome/tichhome.service /etc/systemd/system/tichhome.service

Then enable service

sudo systemctl enable tichhome.service

and start service

- sudo systemctl start tichhome.service

then you can check status of the service. If all went ok it should look like this:

Reading Radio controlled power switch signal using RFSniffer

So you have connected transmitter and receiver to your Raspberry. You have compiled RCSwitch scripts and have some Raio switches ready to be used in your TichHome setup. What you need now is to read ON/OFF signal for those switches so you can later on use them to configure your TichHome homepage devices.

Prepare your radio controlled power switch and remote that controls it.

Go to directory
  – cd /home/pi/tichhome/RadioDevices
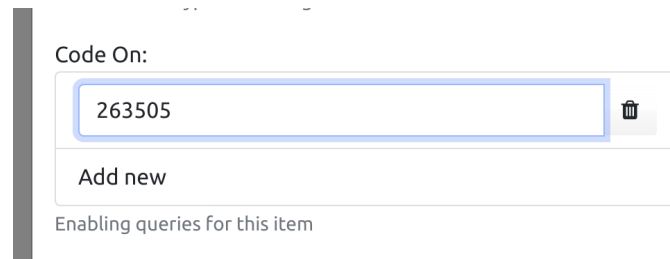
Type:
  – sudo ./RFSniffer

This will initiate RFSniffer which will constantly read radio signals. Place your remote radio transmitter near radio receiver connected to Raspberry Pi (example: XY-MK-5V). Slowly and with some time gaps push buttons responsible for actions on your radio controlled power switch and observer what RFSniffer outputs. Push each button a couple of times to ensure that RFSniffer returns repeatadly same values for same button to eliminate any signal distortions. Write down codes returned by RFSniffer

The output should look somehow like this (in given example one button was pressed repeatedly. Since value 263505 repeats we will write it down as value that represents signal generated by this button):

In this case ON code is 263505 and i will put it in TichHome when adding Item as CodeOn

Code On:

263505 🗑

Add new

Enabling queries for this item

Web Controller Switches Setup

TichHome can also be configured with web controller switches. That means that unlike Radio Controlled switches described above, system will send web requests on disable and enable actions and web device receiving these requests will act accordingly. Many web devices or even services can be used and paired with TichHome because as you will see below in node usage description this system is actually quite elastic regarding what you use.

If you decide to map something that is present on your network into TichHome system you just need to define this „something" in TichHome system by introducing its node in nodes.js file. Then provide its „address" property which will define what the address is. And then provide „On" and „Off" properties which define what arguments will be present in „on" and „off" requests going from TichHome system into this particular device.

For example if you configure device with data address: 192.1.1.1, codeOn: [„pin=1"], codeOff: „pin=0" then on „Enable" action TichHome will send the following request „http://192.1.1.1/?pin=1" and on „Disable" action TichHome will send the following request „http://192.1.1.1/?pin=0". So regarding web devices that you can pair with TichHome it can be anything .. device, service.. you name it. Below how this would look like in TichHome when you create/configure such item.

Item

Send Option:

Web Item

Defines what type of sending is used for this item

Address:

http://192.1.1.1

Address for web type device. Contains mail url base while 'code on' and 'code odd' would contain url arguments

Code On:

pin=1 🗑

Add new

Enabling queries for this item

Code Off:

pin=0

Disabling query for this item

Delay(s):

10800

Defines time in seconds after which this item will auto turn off

Check availability(ip):

Ip by wchich 'Check availability' option will be displayed allowing to check if ip is reachable

Web Controlled Electric switches

If you are thinking about web controlled electric switches there is also very nice solution. On the market you can get relatively cheap and very realiable **Sonoff TH01** and **TH10**, **TH16** web controlled electric switches. These switches are based on **ESP8266** wifi module. By default these switches come with Sonoff software which operates in the cloud and allows you to toggle these devices from your home or perhaps pair it with other systems.

However because it uses ESP8266 component which is well known among automation enthusiasts these devices can be easily flashed which allows you to get rid of original software and install your own .lua scripts which do exactly what you want them to do.

On my GitHub repository **https://github.com/Sznapsollo/AK_Sonoff_nodeMCU_LUA_automation** I uploaded .lua scripts that I use to pair Sonoff devices with TichHome system. These scripts were defined in such way which allows TichHome system to also know what is the current state of each device which is an advantage over radio controlled devices where you cannot do this.

AK_Sonoff_nodeMCU_LUA_automation repository also contains description of how to flash Sonoff devices, where to get necessary tools from and also link to Youtube video showcasing the whole process.

For more information how to define web controlled device in TichHome System please also check content in next chapter.

How to set up Sonoff web controlled power switches I describe in vids that are on playlist below:

- https://youtu.be/AlX1ZiVodwY

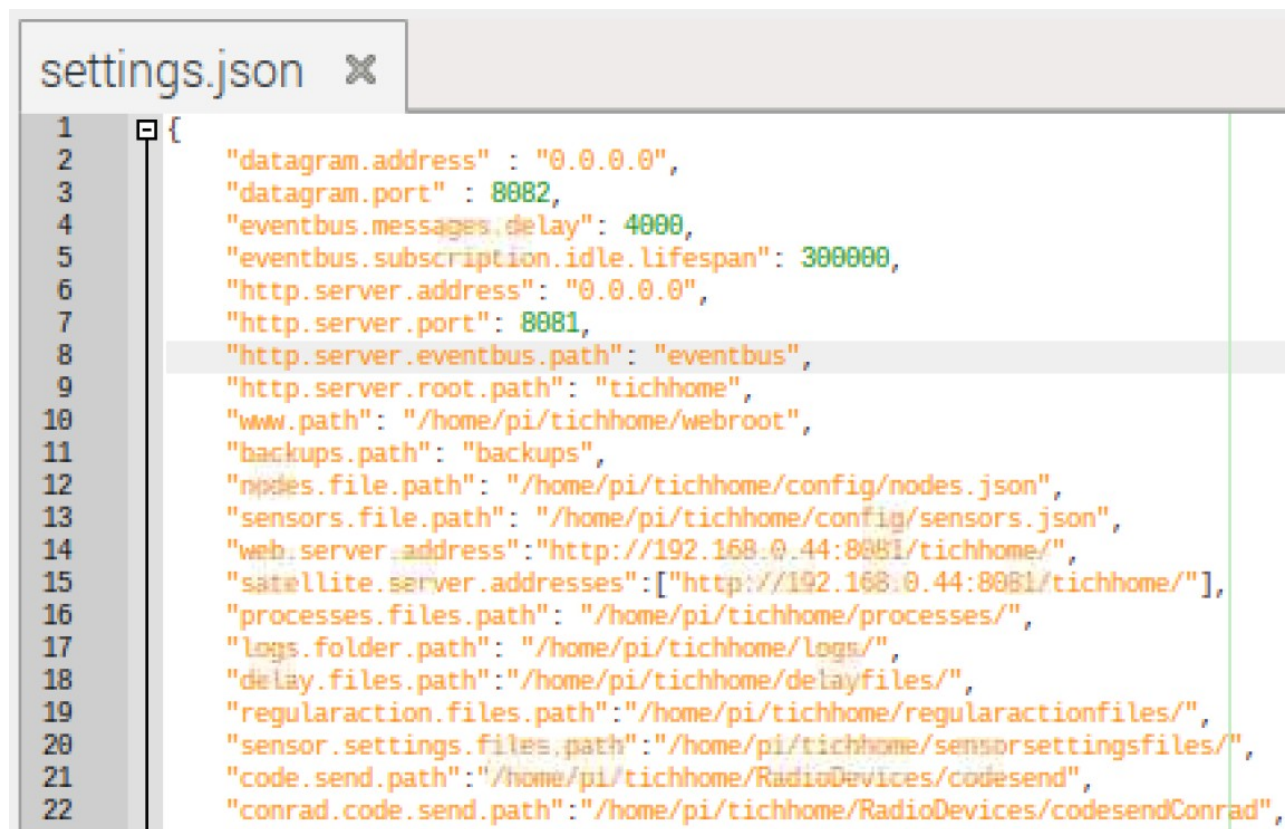- https://www.youtube.com/watch?v=C19ARWDYR3c&list=PLjd2MVjW6mhFygrvXyVcdNoq6pHK8MdUW

TichHome configuration

Settings.json file setup

Settings.json file is placed in the following location:
- **cd /home/pi/tichhome/config/settings.json**

It contains basic settings of TichHome.

Settings.json nodes

- **web.server.address** – this node describes web path to TichHome. It should be ip of your raspberry and proper port (http.server.port) and folder name

- **satellite.server.addresses** – TichHome allows to pass signals to also other machines on your network, which for example could be situated in different areas of your flat to increase signal range. This node would contain list of web paths to TichHome on other machines. Please be advised that these other instances should not have satelliteServerAddresses containing initially calling machine as that would lead to some loops ;-). Initially this node is empty. If it contained some values it would look somehow like this:

  - "**satellite.server.addresses**":[
      "http://x.x.x.1:8081/tichhome/", "http://x.x.x.2/:8081tichhome/",
    "http://x.x.x.3:8081/tichhome/"
      ],

- **www.path** – this node defines where TichHome frontend is located on local machine

- **delay.files.path** – TichHome allows for delayed devices disabling. The delay is actually managed by files which are created in folder defined in this node.

- **regularaction.files.path** – TichHome allows for setting up scheduler actions per each of your devices defining during which days and time they should work. User sets this up from webpage by clicking on calendar icon next to each item (if item has regular actions feature enabled from configuration). Files that contain configuration set up by user will be stored and managed in folder defined by regularaction.files.path setting.

- **sensor.settings.files.path** – if there are any sensors connected to TichHome system, their initial setup is defined in nodes.js configuration file. However use can also change this setup for each sensor from UI. If such customization is performed then config files containing new settings will be generated and contained within directory defined in this setting.

- **code.send.path** – this node defines where RCSwitch libraries, including codesend file responsible for transmitting radio devices, is placed. This file is used for radio devices that will have sending option defined as "sendOption": 0

- **conrad.code.Send.path** - his node defines where RCSwitch libraries codesendConrad file responsible for transmitting radio devices is placed. This file is used for radio devices that will have sending option defined as "sendOption": 1. Please have alook at vids mentioned in Error: Reference source not found for more ino on this

- **sensorAlarmMail** – this option will be used if you have some sensors defined in nodes.json and will want to use alarm option and sending mails on alarm.

  - **enableMailingOnAlarm** – if enabled will attempt to send mail on alarm event

  - **fromaddr** – source mail account address

  - **toaddr** – where the alarm mail should be send

  - **password** – password to source mail account

  - **smtpServerAddress** – smtp server of source mail account

  - **smtpServerPort** – port of smtp mail account

- **save.daily.logs.to.file** – if set to true it will create daily log file of all enable actions. This file will be places in executables/logs/actions folder and will be titled actions_YYYYMMDD.json (where YYYYMMDD will correspond to day date)

- **save.daily.sensor.logs.to.file** – if set to true it will create daily log file of all sensor enable

actions. This file will be places in executables/logs/sensors folder and will be titled sensors_YYYYMMDD.json (where YYYYMMDD will correspond to day date)

- **canChangeAlarmSettings** – if set to true, it will enable alarm changing icons on „Sensors" tab per every available sensor. These icons allow user to change sensor's timeline and operated devices data

- **logsDropdownFilter** – json object containing dropdown values for log popup (which can be open for any day that contains logs from „Action Logs" and „Sensor Logs" tabs)

- **translations** - this node contains strings that will be presented in webpage. They can be changed/translated here

It also contains list of sensors that could be connected to TichHome that would work if check_sensors.py is running. (see „Setup TichHome as service" how to enable it)

Nodes.json file setup

nodes.json file is placed in the following location:
- cd /home/pi/tichhome/config/nodes.json

It contains list of devices that will be used in TichHome.

There can be few device types defined depending on device sendOption properties as described in a moment. There can also be device defined which will enable/disable whole groups of devices.

Communication witch device is determined by **sendOption** property which can have the following values:

- 0 - this is radio controlled device. Signal will be send using file defined in settings.json → codeSendPath setting

- 1 - this is radio controlled device but controlled by different signal pulse length than 0. Signal will be send using file defined in settings.json → conradCodeSendPath setting

- 2 – this is web operated device. Such device should have „address" property defined which defines address of device. CodeOn and codeOff for such defvice should contain GET request parameters that will be included in request url

- 3 – this is wakeonlan device type that will allow to send wakeonlan signals to specified device by this node.

- 4 – this is shell type that will allow to run shell commandson server. Can be used for variety of purposes to control the server or run any scripts on the server..

Now Lets see some examples of different device types and below I will exmplain all properties that we can assign to devices and what they do.

Example radio controlled device

```json
{
    "name": "kitchenlights",
    "image": "kitchenlights.jpg",
    "category": "general",
    "header": "Kitchen lights",
    "regularActions" : true,
    "codeOn": [
        44335
    ],
    "codeOff": 44365,
    "delay": 7200,
    "sendOption": 0,
    "enableOn": true,
    "enableOff": true
},
```

Example radio controlled device (alternative sendOption)

```json
{
    "name": "coffee",
    "image": "coffee.jpg",
    "category": "general",
    "header": "Coffee machine",
    "regularActions" : true,
    "codeOn": [
        966491648
    ],
    "codeOff": 899382784,
    "delay": 7200,
    "sendOption": 1,
    "enableOn": true,
    "enableOff": true,
    "notifySatellites": false
},
```

Example Web controlled device

```
{
    "name": "humidfier",
    "category": "general",
    "header": "Humidifier",
    "regularActions" : true,
    "address": "http://192.168.20.121",
    "codeOn": [
        "pin=ON1"
    ],
    "codeOff": "pin=OFF1",
    "delay": 1800,
    "sendOption": 2,
    "enableOn": true,
    "enableOff": true,
    "codeDev": "---------"
},
```

Example group device

```
{
    "name": "alllights",
    "image": "alllights.jpg",
    "category": "general",
    "header": "All lights",
    "questionOn": "Really? All lights?",
    "itemIDs": [
        "tvlamp",
        "roomlamp",
        "roomlampsecond",
        "kitchenlights",
        "lavalamp"
    ],
    "enableOn": true,
    "enableOff": true
},
```

Example confirm device

```
{
    "name": "tvstack",
    "category": "general",
    "header": "TV stack",
    "regularActions" : true,
    "questionOff": "Are you sure you want to turn it all off?",
    "codeOn": [
        12312
    ],
    "codeOff": 2612312313508,
    "delay": 0,
    "sendOption": 0,
    "enableOn": true,
    "enableOff": true,
    "codeDev": "-----"
}
```

Example wakeonlan device

Devices properties explained

Ok lets go one by one property

- **name** – unique name for each device

- **image** – optional property. It holds image name that would be displayed next to device in UI. These images are taken from home\graphics\icons\ folder. There are some predefined icons there. If this property is not provided – no image would be displayed.

- **icon** – optional property. Used to fill „class" for <i class="..."></i> and if filled displayed next to given item. For example out of the box it can be provided with any icon coming from font-awesome-4.3.0 package which is currently used by TichHome. Example „fa fa-whatsapp fa-2x" (also showcased in demo) will display whatsapp icon.

- **category** – web UI contains main page and also „Advanced" tab. This property when set to „general" will display device on main page and if set as „advanced" will display it in advanced tab.

- **enabled** - optional property. When set to „false", it will disable this node eliminating it from being visible in UI and also from any processing. If this property is not present, item is considered as enabled: true.

- **header** – text that will be displayed for this device in web UI

- **questionOn** – optional property. Confirmation question that will be displayed in case of hitting enable button. If this property has value, the enable button will have ble color.

- **questionOff** – optional property. Confirmation question that will be displayed in case of hitting enable button. If this property has value, the disable button will have ble color..

- **address** – web address which is used for sendOption: 3 devices

- **codeOn** – collection of enabling codes for given device.

- **codeOff** – disabling code for given device

- **delay** – optional property. Default delay for given device after which delay device would be disabled (unless user sets different delay from UI). Value in seconds. Value -1 means there will be no delay applied. If this property is not set, the delay slider options that shop up when clicking on device header will no longer be active.

- **regularActions** - optional property. When set to „true", it will display calendar icon which is used for setting regular actions for particular device.

- **sendOption** – communication with device. Possible values 0/1/2/3 – described at the beginning ofthis section.

- **enableOn** – if true then Enable button for this device will be displayed in web UI

- **enableOff** – if true then Disable button for this device will be displayed in web UI

- **codeDev** – purely optional - just some comments for internal use of configurator

- **itemIDs** – list of device names that will be used for group device. Devices by these names should exist/be configured in the system.

Sensors.json file setup

sensors.json file is placed in the following location:
- cd /home/pi/tichhome/config/sensors.json

It contains list of sensors that will be used in TichHome.

If you have connected HC SR501 PIR detector to your Raspberry PI and enabled check_sensros.py script to be run on system start (see „Setup TichHome as service" how to enable it) then you can define sensors in node.js defining what devices should they trigger, what alarma at what timeframes etc.

sensros node is a list of sensor json objects which will be described below. We will use the following example to describe all properties.

```
"sensors": [{
    "id": "pir_ben_dover",
    "header": "Pir Label",
    "pin": 32,
    "rebound": 20,
    "timeUnits":[
        {"timeStart":"00:00","timeEnd":"08:00","daysOfWeek":",0,1,2,3,4"},
        {"timeStart":"15:30","timeEnd":"23:59","daysOfWeek":",0,1,2,3,4"},
        {"timeStart":"00:00","timeEnd":"23:59","daysOfWeek":",5,6"}
    ],
    "alarmTimeUnits":[
        {"timeStart":"09:00","timeEnd":"15:00","daysOfWeek":",1,2,4"}
    ],
    "on": [
    {
        "id": "roomlamp",
        "delay": 600,
        "rebound": 0,
        "dependencyMethod": "checkLuminosity",
        "dependencyOperation": "lwr",
        "dependencyValue": 25
    }],
    "onAlarm": [
    {
        "id": "testswitch1",
        "delay": 60
    }]
}],
```

- **id** – unique identifier for each sensor provided in this listopad

- **header** – User friendly label for given sensor which will be used to present sensor data from UI.
- **pin** – pin to which sensor data output has been connected to Raspberry Pi
- **rebound** – in seconds value that determines when next signal will be analyzed after last valid signal
- **timeUnits** – list of settings specifying when sensor should be active to operate devices
    - **timeStart** – time the sensors starts being active
    - **timeEnd** – time sensor stops being active
    - **daysOfWeek** – days during sensor should be active and consider timeStart and timeEnd defined in same object.
- **alarmTimeUnits** – list of settings specifying when sensor should be active to operate trigger alarm
    - **time** settings are done same as in case of timeUnits
    - if sensor will trigger during timeFrame defined by this setting it can send mail if it has been enabled ins ettings.json
- **on** – list of devices that should be triggered when sensor detects something withing active „timeUnits"
    - **id** – device identifier that corresponds to existing entry in nodes.json->nodes collection
    - **delay** – delay after which device should be disabled. Can be negative in which case no delay would be applied. In case of 0 default delay for this device would be applied (from mdevice configuration definition)
    - **dependencyMethod** – this is optional and can be removed. It supports checkLuminosity methow which would read light levels if light sensor would be connected to Raspberry Pi pin #39
    - **dependencyOperation** – this is optional and can be removed. Corresponds to dependencyMethod. Can have values „lwr", „grtr" which define logical condition between dependencyMethod and dependencyValue.
    - **dependencyValue** – this is optional and can be removed. Contains value which is compared to dependencyMethod.

Troubleshooting and exceptions

Sometimes there are problems. Here are some hints how to narrow down the reason for them.

In case there are some exceptions thrown during operation of python scripts log folder will be created containg exception files grouped by days (for one day there will be one exception file containing all excepons from that day).

Log folder will be situated in:

- **/home/pi/tichhome/logs**

In TichHome you can check exceptions by going to Exceptions Logs tab and if there are some daily files you can check their content.

Exception Logs

Items per page: 48 ∨   **Page:** 1 ∨

---

exceptions_20210929.log   ✕

Change Sort | Refresh | [Szukaj] | 1

2021-09-29 19:53:59:groovy.lang.MissingPropertyException: No such property:
regularActionRandomStart for class:
io.vertx.core.json.JsonObjectgroovy.lang.MissingPropertyException: No such property:
regularActionRandomStart for class: io.vertx.core.json.JsonObject --
groovy.lang.MissingPropertyException: No such property: regularActionRandomStart for class:
io.vertx.core.json.JsonObject at
org.codehaus.groovy.runtime.ScriptBytecodeAdapter.unwrap(ScriptBytecodeAdapter.java:53) at
org.codehaus.groovy.runtime.ScriptBytecodeAdapter.setProperty(ScriptBytecodeAdapter.java:486) at
com.ak.Server.checkRegularActionData(Server.groovy:581) at

If system does not seem to work but there are no exceptions in log file we can also try to debug
server request response from within the javascript. Go to chrome inspector and try to analyze
network requests and responses.

Using TichHome

You can check online demo here (just bear in mind that it is accessigble by all online demo so not
working on any actual devices and has not working all elements) – it is just to show how TichHome
frontend looks like. You can find it here: http://cultrides.com/test/Github/TichHome/

I did some video tutorials of how to use this system. It might be easier to check how it works by
watching them: https://www.youtube.com/watch?
v=C19ARWDYR3c&list=PLjd2MVjW6mhFygrvXyVcdNoq6pHK8MdUW

Devices List

Tbd

Delay Settings

Tbd

Regular action settings

Tbd

Create/Edit switch item

Tbd

Create/Edit sensor item

Tbd

View Action Logs

Tbd

View Sensor logs

Tbd

View Exception logs

Tbd

View Sessions

Tbd