HauntingHouse Documentation

Release 1

Michael Traut

Aug 26, 2016

CONTENTS

1 Intro **1**
1.1 Disclaimer . . . 1
1.2 Organization . . . 2
1.3 Acknowledgement . . . 2
1.4 License . . . 3

2 Part I **5**
2.1 The general idea . . . 5
2.1.1 Appartment server . . . 5
2.1.2 Basement server . . . 6
2.2 Basic tools . . . 6
2.2.1 Soldering iron . . . 6
2.2.2 Copper Wire . . . 6
2.2.3 Cable Connectors . . . 8
2.3 Appartment server . . . 10
2.3.1 Intro . . . 10
2.3.2 Hardware for the server . . . 10
2.3.3 Power supply . . . 11
Cabling . . . 12
2.3.4 Push Buttons / Digital Input . . . 14
Direct wiring . . . 14
Bus wiring . . . 15
Radio . . . 15
Conclusion . . . 16
2.3.5 Thermostat / Analog Input . . . 19
Requirement . . . 19
Direct wiring . . . 19
Radio . . . 19
Alternative . . . 19
Conclusion . . . 20
2.3.6 Relays / Digital output . . . 22
ULN 2803 APG . . . 23
Eltako Relay . . . 24

Heating Valve . . . 25
Driver module . . . 25
Connecting the Pi . . . 35
Distribution Box Patching . . . 37
2.3.7 Software stack . . . 42
Intro . . . 42
Raspbian . . . 43
Hmland . . . 44
Homegear . . . 44
Java . . . 45
openHAB . . . 45
2.3.8 Configuration details . . . 46
Raspbian . . . 46
Hmland . . . 47
Homegear . . . 47
openHAB . . . 50
2.3.9 openHAB Items . . . 53
The great plan . . . 54
Items . . . 54
GPIO . . . 55
Enocean . . . 56
Homematic . . . 57
2.3.10 openHAB Rules . . . 58
Overview . . . 58
System events . . . 59
Cron events . . . 59
Item events . . . 60
Lighting . . . 61
Heating . . . 61
2.3.11 openHAB Sites . . . 61
UI . . . 61
2.3.12 openHAB Persistence . . . 61
MapDB . . . 61
2.3.13 openHAB - more addons . . . 62
Persistence . . . 62
Pushing information . . . 63
Making sound . . . 63
Generic integration . . . 63
Receiving events . . . 64
External conditions . . . 64
Date & time based control . . . 64
Tracking . . . 65
2.3.14 openHAB - caveats . . . 66
Item initialization . . . 66
The item state . . . 66

The DateTime item . . . 67
The DateTime UI . . . 68
2.3.15 openHAB special use cases . . . 70
Intro . . . 70
Date based decisions . . . 70
2.4 Basement server . . . 73
2.4.1 Intro . . . 73
2.4.2 Hardware . . . 73
Server . . . 73
Switch . . . 73
Doorphone outdoor . . . 74
Doorphone indoor . . . 76
2.4.3 Software . . . 79
Raspbian . . . 79
Freeswitch . . . 79
Asterisk . . . 80
2.4.4 Configuration . . . 80
Asterisk . . . 80
GO/Box 100 . . . 84
Grandstream HT701 . . . 89

3 Part II **95**
3.1 Network setup . . . 95
3.1.1 Static IP . . . 95
3.1.2 Dynamic IP . . . 96
3.1.3 IP address setup . . . 96
3.1.4 Hostname and hosts . . . 96
3.1.5 Provide DHCP services . . . 97
3.2 Mass Storage . . . 98
3.2.1 USB Hub . . . 98
3.2.2 USB Disk . . . 99
3.2.3 Directory layout . . . 101
3.2.4 NFS . . . 101
3.2.5 Samba . . . 102
Misc . . . 102
Pi server . . . 102
Pi client . . . 103
Windows client . . . 104
More tweaks . . . 104
3.3 Backup . . . 105
3.3.1 So why? . . . 105
3.3.2 rsync . . . 105

4 Part III **109**
4.1 Access Point . . . 109

4.1.1 /etc/network/interfaces . . . 109
4.1.2 Disable power saving . . . 110
4.1.3 Switch hostapd . . . 111
4.1.4 Run hostapd as daemon . . . 111
4.1.5 Experience . . . 112
4.2 WAN access . . . 112
4.2.1 Surfstick . . . 113
4.2.2 SIM card . . . 115
4.2.3 Configuration . . . 115
4.2.4 Stick configuration . . . 116
4.2.5 wvdial . . . 117
4.2.6 SIM PIN . . . 121
Keep the PIN . . . 121
Unlock the SIM . . . 122
4.2.7 pppd . . . 122
4.2.8 Network config . . . 123
/etc/network/interfaces . . . 123
Manual shutdown and startup . . . 123
The routing table after successful connection . . . 123
Provide internet access for all other nodes . . . 124
4.2.9 Keep alive . . . 125
4.2.10 Auto disconnect . . . 126
4.2.11 Security . . . 126
4.2.12 Issues . . . 127
Modem not detected . . . 127
Network to fast . . . 127
4.2.13 More 128
4.3 Remote Control . . . 128
4.3.1 ssh . . . 128
Install ssh . . . 128
Basic usage . . . 129
Important options . . . 129
Public/Private key . . . 129
ssh daemon . . . 132
WinSCP . . . 132
ConnectBot . . . 135
fail2ban . . . 135
4.3.2 DynDNS . . . 135
4.3.3 Reverse tunneling . . . 136
Intro . . . 136
Weaved . . . 136
ssh Tunneling . . . 136
4.3.4 ssh Tunneling . . . 136
Straight tunnel . . . 137
Reverse ssh . . . 138

ssh daemon . . . 139
autossh . . . 140
Add to crontab . . . 140
Experience . . . 141
4.4 Thou shalt not write... . . . 141
4.4.1 Use high quality cards . . . 141
4.4.2 Use bigger card . . . 142
4.4.3 Do not write at all . . . 142
4.4.4 Use temporary file system . . . 142
4.4.5 File system flags . . . 142
4.4.6 Disable swap . . . 143
4.4.7 Reduce system logging . . . 143
4.4.8 Boot from another device . . . 143
4.5 Tweak installation . . . 143
4.5.1 Memory . . . 143
4.5.2 Watchdog . . . 144
4.6 What time is it? . . . 145
4.6.1 Overview . . . 145
4.6.2 Network Time . . . 145
4.6.3 Real time clock . . . 146
4.7 Improve logging . . . 147
4.7.1 Understand (r)syslog . . . 147
4.7.2 Log server . . . 147
4.7.3 Tweak logrotate . . . 147
4.7.4 Log to temporary file system . . . 148
4.7.5 Log to external disk . . . 149
4.7.6 Move logs . . . 149
rsyslog . . . 149
apt . . . 150
dpkg . . . 150
ppp . . . 151
samba . . . 151
msmtp . . . 151
asterisk . . . 152
hmland . . . 152
homegear . . . 153
openHAB . . . 154
4.7.7 Add logging to your python scripts . . . 156
4.8 Network monitoring . . . 158
4.8.1 Show statistics . . . 159
4.8.2 Send statistics . . . 159
4.9 Data logging . . . 159
4.9.1 Data sources . . . 159
Pi data . . . 159
openHAB data . . . 159

Heating data (Oekofen pellematic) . . . 162
Internet data . . . 165
4.9.2 Data sinks . . . 165
Plain RRD . . . 165
Outside temperature . . . 166
Heating . . . 166
Room data . . . 167
rrdtool graphing . . . 167
openHAB persistence . . . 167
Hosted cloud services . . . 167
Own services . . . 168
4.9.3 Data visualization . . . 168
4.10 Event notifications . . . 168
4.10.1 Sending eMail . . . 168
msmtp . . . 170

5 Part IV **171**
5.1 More event notifications... . . . 171
5.1.1 Pushover . . . 171
5.1.2 Sending SMS . . . 172
5.1.3 Twitter . . . 172
5.1.4 Other push message platforms . . . 172
5.2 Adding a UI device . . . 172
5.3 Alternate actor technology . . . 172
5.4 Access control . . . 172
5.5 Irrigation control . . . 172
5.6 RTC . . . 172

6 Part V **173**
6.1 Radicale . . . 173
6.1.1 Overview . . . 173
6.1.2 Installation . . . 173
6.1.3 Test . . . 174
6.1.4 Update . . . 174
6.1.5 Authentication . . . 174
6.1.6 More security . . . 175
6.1.7 Clients . . . 175
6.2 minicom . . . 176
6.3 screen . . . 176

7 Part VI **177**
7.1 Troubleshooting . . . 177
7.2 Oekofen Pellematic Datapoints . . . 177
7.3 Abbreviations . . . 180
7.4 *nix 101 . . . 180

7.4.1 Package management . . . 180
7.4.2 mount . . . 181
7.4.3 shebang . . . 181
7.4.4 python . . . 181
7.4.5 Users & Passwords . . . 181
Change password . . . 181
Remove password . . . 182
Create a user . . . 182
Remove a user . . . 182
7.4.6 Service Management . . . 182
7.4.7 Send command to background . . . 182
7.4.8 Concatenating output . . . 183
7.4.9 whoami . . . 183
7.4.10 groups . . . 183
7.4.11 rm . . . 183
7.4.12 File system magic . . . 183
List of open files . . . 183
Show users of file . . . 183
Show file system of a file . . . 183
7.4.13 Networking . . . 183
netstat . . . 184
7.4.14 crontab . . . 184
7.5 Tools . . . 185
7.5.1 Connecting . . . 185
7.5.2 Some network tools . . . 185
wget . . . 185
NMAP . . . 185
netcat . . . 185
7.6 File listings . . . 185
7.6.1 /etc/fstab . . . 185

8 Indices and tables **187**

CHAPTER

ONE

INTRO

This is the story of a home automation project. I hope it may serve as help for anyone to come who wants to make the same mistakes again...

After having setup some "simple" Homematic scenarios before, I wanted to try the "real thing". Setup all hardware and software components by myself, in an environment that is not limited by existing solutions. I played around with a Raspberry, made some demo installs, made some LEDs switch on and off, clicked some relays...

That's when I got the chance to make it all come true: We started to renovate a building completely, including new electricity. The challenge: The solution should not be (much) more expensive than a classical one and the tenants to come should be able to live with it, both from a usability and reliability point of view.

As far as existing "home stories" on home automation, scattered everywhere on the internet, are concerned, this is an important difference (in my humble opinion): There's no way for your wife to shout in despair "Darling, hurry up, reboot the house, the oven is offline and the Smiths are coming...".

1.1 Disclaimer

All content has been compiled with care and to the best of my knowledge. Everything you find here is working fine with me.

Much of the content is a derived work of tips and tutorials on the internet. I think what makes this book unique is the "putting the pieces together".

However, I cannot assume liability for the up-to-dateness, completeness or accuracy of the content. Any damage or loss is up to you. Please, double check what you are doing.

I'm in no way "expert" in the domains to be explored, neither in unix nor in electrics nor electronics. So, naturally you will probably not find the best, the cheapest or nicest solution. It's a collection of working technology...

If you belong to the folks who claim "when there's still a button, it's not home automation" - then simply look at it as the basic infrastructure for all the glorious things to come.

If you find any information that's wrong or have ideas for improvement, feel free to send them to wachenburgweg@gmail.com.

And, finally: Stay away from high voltages. Anything that has to do with the plain old powerline is out of scope here.

1.2 Organization

You may not believe it (yet) - but this is a quite complex project. I tried my best to organize the book in way to break down the tasks and issues in small manageable parts.

This way you can decide, what's your level of sophistication and automation you want to achieve.

The first part will give you the general idea of what is expected. Then we will immediately select the tools to do so and bring up a "minimal" working system. Until now we will have designed an "island" system that can control a single apartment.

The second part will combine the islands and you will see: the whole is more than the sum of the details.

The third part will add concrete bells and whistles. This is the part that will never end, there's always one more step to go.

Part four is about the whistles to add sooner or later and some alternative approaches.

Part five is a giant appendix.

1.3 Acknowledgement

There's quite few in this book invented here. I hope it's the collection of topics, the production setting or the careful composition of know how that makes it worth reading.

Nevertheless - this book is written "on the shoulders of giants". Contributions that made this book possible are countless.

- Countless bloggers that build up my "Linux Know How"
- Another countless bloggers that built my "electronics expertise"
- And then the ones that made me an "Raspberry expert"
- Countless programmers that provided an incredible software stack for **free**

If you can improve this book by corrections or ideas, feel free to contact me.

And then, a special thanks to my loving and beloved wife: If she would have suspected what I am about to do and where this will end, this project would have never seen the light of day

1.4 License

This book is covered by a Creative Commons License, the "Attribution - Non Commercial" one (CC BY-NC 3.0). In non-legalese this means you can do whatever you want with the content as long as you mention where it is from and do not take commercial advantage from it.

See http://creativecommons.org/licenses/by-nc/3.0/legalcode.

CHAPTER

TWO

PART I

Here you will learn what this project is all about and what are the initial steps to set up the automation scenario.

2.1 The general idea

The idea of the architecture is simple:

We have 3 apartments, each getting its own HA central. Another HA central, with different duties, is in the basement. All systems are interconnected but the components are shielded from unauthorized access. For remote access one fine day I want to setup a cable modem or surfstick based connection.

Any part of the system should be in some sense "open" - i.e. **not** a proprietary technology that cannot be replaced at its "API level" by another component more or less instantly. For the purpose of this project it does not have to be "free", publicly documented or self-made.

The reason for this is simple: many components are too complex to be manufactured completely yourself in a timely fashion, in production quality or within a reasonable budget (and some of them all of this...). But you can tell the components that will lock you in a certain architecture (I only say : cloud, cloud, cloud) or provider and leave them alone.

2.1.1 Appartment server

A "simple" home automation device with initially the following tasks

- Control lighting via push buttons
- Control shutters via push buttons
- Control heating via thermostat and some "simple" rules

Clearly, this has the potential to grow....

2.1.2 Basement server

This one will eventually provide the connectivity to the whole wide world, allowing for remote operation of the complete system.

Second, I want to provide door and house communication via this device.

Last not least one can setup more central services for all participants and control for HVACR components.

2.2 Basic tools

2.2.1 Soldering iron

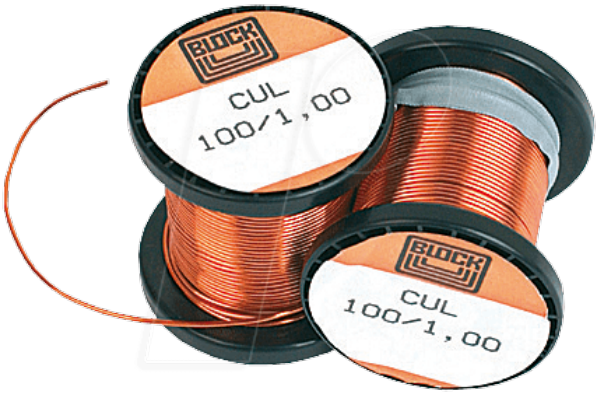
If you too want to build the driver component yourself, you will need an electronic soldering iron. This has not to be something special, we will not solder SMDs or expensive chips.

Having an iron at hand will be reasonable anyway if you plan to do tinkering with electric components yourself.

2.2.2 Copper Wire

For my project I needed two types of copper wire.

Jumper Wire (or hook-up wire?), 0,8 mm diameter with plastic insulation. This is used for wiring the low voltage components within the electric distribution.



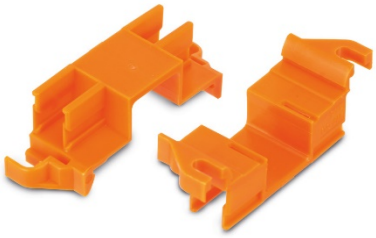
Jumper wire with coating. 0,28 mm diameter. This is used for wiring the electronic components on the proto board.

In this project we will deal with 24V and 5V components and currencies up to 500mA (that will sum up, for sure).

2.2.3 Cable Connectors

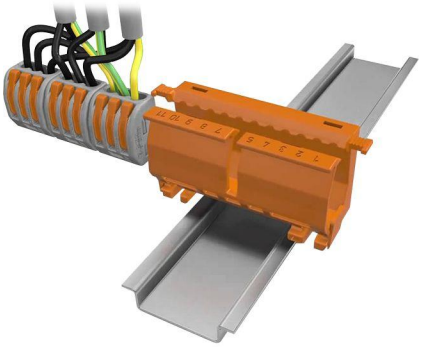
For patching the communication and other low voltage cables there is a simple connector system from "Wago", series 243.

The Wago 243 connector. You can get this in different colors and sizes.



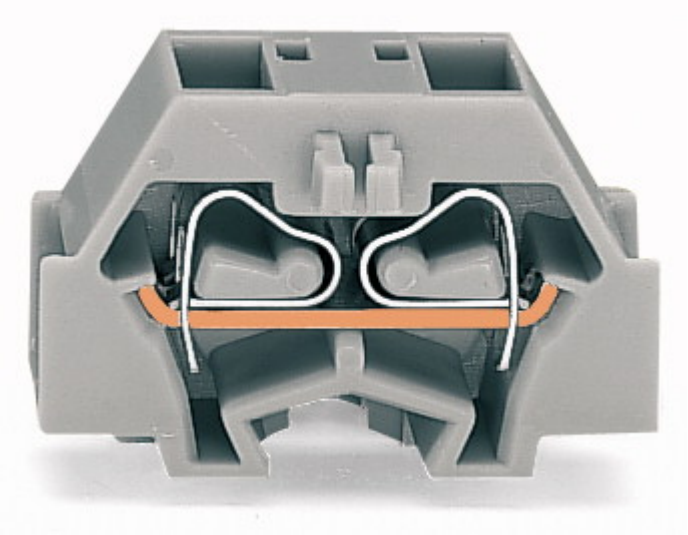
The Wago 243 holder is a great tool to mount the connectors on a DIN rail.

For patching the 240V lines I used the Wago 222 series, the 3 and 5 slot ones.



Again, the Wago 222 has a holder to mount - wherever you want.

Another very practical item is the WAGO "miniature" terminal block, e.g. 260-331.



There are a couple of other parts in this series that make life easier, just look through the catalog. I've used these items successfully for wiring the heating valves. As DIN rail mounts are available, using them in the cabinet for patching is another "hot spot" for the parts.

2.3 Appartment server

2.3.1 Intro

The apartment server will control the internals of a single unit. Its complexity is "modest". We will only provide basic control, no sophisticated automation so far. It's the modern, scalable adaption of a classical apartment installation.

First, we have a look at the hardware components, then we select and install a software stack. Finally we make some tweaks that I consider "basic requirements" for a production system.

2.3.2 Hardware for the server

For the central platform of this project we have the following constraints

- price tag
- power consumption
- availability of software
- availability of hardware
- availability of advice
- interoperability features

- stability/reliability

It's not very hard to come across the Raspberry Pi seeing this list. I must admit, I am not a microcontroller or single board computer nerd. The selection process was finally driven by the question "availability" of all kind. Other SBCs may be better suited (and may be are interchangeable in the design to come). But, whatever software and hardware component you may want to use, if you don't find a tutorial for the Pi, you will find none.

The greatest disadvantages (let's name it challenges) I found so far for this platform:

- **stability/reliability** Based on the SD card as filesystem, one can expect failures on a regular base
- **USB** The USB connectivity for the Pi is known to be a weak spot. One may expect outages if network communication and other USB controlled communication will exceed a certain threshold.

2.3.3 Power supply

A major concern of this project is stability and as such everybody really recommends a good power supply. Using the Pi in a central installation gives you the added benefit of using a DIN rail component like the Meanwell DR-60-05. This should be enough to serve for all 5v components planned so far in the "rack".

Here you can see a "preview" for the new electric distribution with the power supply. The power supply on the left, a 24V will drive the relays.

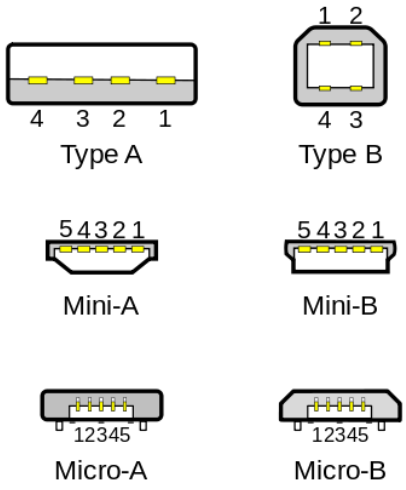
In addition, the internet advice is to always connect the Pi via the USB plug, this ensures leveraging the fuse and some magic on the Pi's board take care of good "vibrations".

Cabling

The PI needs power via USB cable (you may opt to power via the GPIO Pins, but this is not easier in this case and seems to bring some issues if you do so). So, simple solution is to buy some preconfigured USB, cut the host adapter and patch the power line to the 5V supply, using the Wago 243 "distribution" (you can see on the left of the supply).

You will most probably find that the cables are not suited to be patched directly - this is the first time you will have to heat up the soldering iron and add some copper wire at the end.

For your reference I enclose the connector pinout here.



USB 1 standard pinout

Pin	Name	Color	Description
1	V bus	Red	+5V
2	D-	White	Data-
3	D+	Green	Data+
4	GND	Black	Ground

USB micro pinout

Pin	Name	Color	Description
1	V bus	Red	+5V
2	D-	White	Data-
3	D+	Green	Data+
4	ID		USB OTG ID
4	GND	Black	Ground

2.3.4 Push Buttons / Digital Input

Even in the age of smartphones you will not omit plain buttons for lights and shutters (at least if you want to avoid trouble with your sweetheart).

I looked around a lot for this one and it is not easy to decide:

Direct wiring

One can setup a wire based scenario, using "cheap" buttons that finally switch some GPIO directly, preferably using 24V technique. Every button is connected to a GPIO line directly. In this small use case with around 10 switches planned upfront it is a viable solution. Performed correctly one fine day you may switch the 1:1 cabling to a bus system, so there's an upgrade possibility.

The greatest advantage of this solution (at least for my wife) was: if anything goes wrong, one can still switch to a more conservative solution later on.

The drawbacks:

- you still pay for the cabling, which is not so cheap if you don't do it yourself.
- you loose a great deal of flexibility - you have to decide upfront how many buttons you need and where they will go finally
- it's not scalable (you need a single cable for each button planned)
- it's still homegrown (not the buttons and infrastructure, but the central)

Bus wiring

Next one is a bus system. Using KNX is an option here, but not if you want to install a "cheap" solution (there may be other opinions on this..).

There are also a couple of interesting "proprietary" solutions (e.g. Eltako and Homematic wired) and if it wasn't for the fun of the project, I would have chosen it. There's lot of information around, of the shelf products are available.

More interesting is the possibility of using an "open" bus system like Modbus, RS485, CAN or OneWire. The greatest problem here in my opinion is the availability of the button hardware itself. I didn't find any off the shelf product. One can (must) design his button and the bus endpoint logic himself. While there's lot of information available this will result in lot of work for creating you proprietary button yourself, besides the fact that this button is **really** not cheap finally.

Don't forget that you have to design the central gateway of the bus to the server also.

The advantage of bus wiring for me is the reliability while we still have a scalable solution (if you call it scalable that you **can** add another component, but you still have to tear apart your wall...).

This would have been the solution if I had unlimited time, I think. But I don't have...

Radio

Again there is a very wide choice of solutions.

Homematic, FS20, Zigbee, ZWave, Enocean,... Countless providers for radio solutions and every day more...

To be honest, I didn't test and setup a prototype for all of them, but I think important criteria is

- **Power consumption** You won't get happy if you have to switch batteries 4 times a year...
- **Reliability** Pressing a button should switch the light. Every time.
- **Range** And don't forget to differentiate between the nominal and the real one. Take into account the specific anatomy of your building. Maybe you should consider making real measurements - it may seem that the devices are up and running. But in reality the connection quality may be so bad that numerous retransmissions are necessary, rapidly destroying your overall bandwidth.
- **Scalability** The different technologies differ in the way they use the radio band und the legal limitations to do so. In a small apartment this may be neglected, but mid sized or great installations might have to take action to circumvent the problem.

Some technology comes along with built-in repeater features, some technology does not provide such things at all.

- **Component availability** Best is: all components, sensors and actors, are available from your preferred technology. But, as you will see, thanks to gateways and freely available software, this is not a real problem..
- **Gateway availability** Going for a SBC and open source controlled system, this is a real showstopper. Be sure there is a component you can plug into your SBC along with the appropriate drivers for your platform.
- **Number or size of providers** Personally I would not mind going for a new gadget that may get lost in 2 years. But a commercial grade project for three apartments has a lifecycle of at least 10 to 20 years (I hope...).
- **Price** Well...
- **Design** As you like...

Conclusion

For this project I choose Enocean based buttons. The reasoning:

- **Power consumption** Well, it's "self contained". Power is harvested via a piezo element upon button press.
- **Reliability** It's an old and widespread technology, so I hoped that in this small scale project I will not meet any issues (which is not really true so far).

Time Travel

As of today, Enocean is not really without problems. One of the original requirements "push a button triggers an action - every time" is not really true. Sometimes it is necessary to repeat a button press to get it recognized. That's a bit annoying but not really a showstopper.

More of a problem was the unexplicable degradation of a single button/receiver combination. Only after changing the receiver stick, things got better.

- **Range** Range as specified and tested is enough in my max. 10m, single floor scenario.
- **Scalability** There exist really large installation, so I do not mind this one.
- **Component availability** There's a broad range of components from different suppliers.
- **Gateway availability** There are even different hardware gateways for the Pi.
- **Number or size of providers** See above.

- **Price** This are not the cheapest ones, especially in comparison with some other 433 MHZ technology, but from time to time you can hit a special offer.
- **Design** It's OK - and you can integrate to other standard (55mm) providers.

So here it is:

A plain old button - besides the interesting fact that it is radio based.

The back is even more interesting (and yes, I know, one should not show the serial). Here you find all the information to be applied when we integrate the button in the software.

This is the official Enocean USB adapter.

Time Travel

Maybe today i would evaluate implementing radio based buttons using the ESP8266 - an inexpensive microcontroller with on board WIFI. This would fit perfectly in a HA design and the component may be used for other tasks, too.

It has all kind of IOs on board, compare it with a very cheap Arduino with built in WIFI

2.3.5 Thermostat / Analog Input

Requirement

In Germany one is legally forced (!) to provide a thermostat unit for each (!) room - even if we have a floor heating as in our project. Anyway, my unit must have a temperature sensor, a display and some means to enter the target temperature.

Direct wiring

Again, in theory you have the option of wiring a such a device directly. In practice I do not know of any unit that fulfills the requirements and has a "public" protocol that I could use with the Pi.

For sure you can fall back to the countless OneWire components with regard to sensors. In this case you have to find an alternative solution for entering the temperature (see below).

Radio

There's a bunch of (cheap) radio thermostats - but no public documentation for the protocol. So, if you find some, get it..

There are some enocean based units - but they are really expensive. As I need 12 of them, I skipped this option.

Finally I found a good offer for the Homematic thermostat. A cute unit with all requirements and even humidity sensor. The drawback is, that I have to add another gateway to my Pi - but that's fine anyway. It's not too expensive and now we have access to the two most used radio technologies in Germany (today).

Alternative

Instead of distributing a complete sensor/actor component in every room you may look at the option of including a central control panel. E.g. there are countless cheap options for a TFT touch screen in any size.

Then you can fall back to simple thermostat sensors in every room, while the temperature selection is available only on the control panel.

Such a panel can range from a simple numeric display with some buttons over a computer connected touch display (as a "console") to a remote browser client, again with different hardware options.

While I plan to have a look at such an option later on, in the first phase of the project I will stick with a console-less approach.

Time Travel

Displays and tablets get cheaper and cheaper. Maybe today I'd evaluate a complete tablet based frontend and simple sensors in each room. While this will involve more software design on the other hand you get a frontend for all other items in your installation, too.

Conclusion

The conservative approach: we take a pack of these

Homematic Thermostat (HM-TC-IT-WM-W-EU).

Homematic USB adapter

2.3.6 Relays / Digital output

Next task is the interaction with the output side. This is a relatively small installation, so we do not need a lot of outputs (you will see, the 16 GPIOS used later will be enough for me). If this is not the case, a multiplexing solution is more appropriate.

Here in the first step we choose a straight forward solution:

- Use 16 GPIO pins for output
- Run every output through an amplifying stage - here a darlington arry
- Connect the darlington output to either the terminal component (if output constraints are satisfied) or drive a relay.

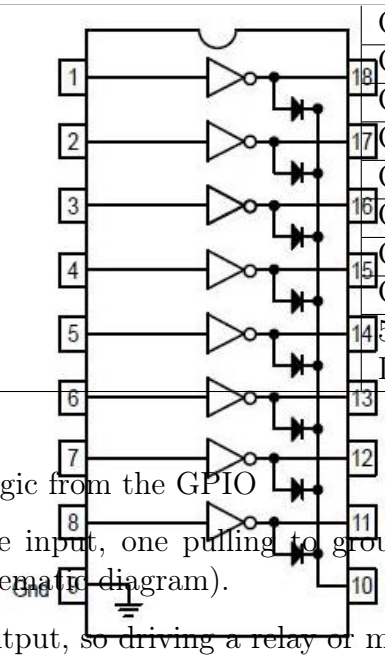
ULN 2803 APG

Controlling relays or other components means nearly always driving currencies that are too high for a GPIO pin. There's lots of discussion in the forums on which solution will fry the pi and which will work. I think the best way to go is: make it "right" and make it flexible.

Still you will find several solution for this problem, one of them is using a darlington transistor or in this case an array of it.

The ULN 2803 takes care you don't fry your Pi. It allows for up to 50V, 500mA per Output and as such, with a price tag of ~ 70 Euro Cent, it is suitable as a joker for nearly all IO. Btw. there are some similar chips to this one, even cheaper, with comparable capabilities. Just look around for finding good tutorials.

IN 1	OUT 1
IN 2	OUT 2
IN 3	OUT 3
IN 4	OUT 4
IN 5	OUT 5
IN 6	OUT 6
IN 7	OUT 7
IN 8	OUT 8
GND You need to connect Pi and secondary power GND	5V to 50V DC



Some more advantages:

- The chip allows to work with "active high" logic from the GPIO
- You have built in protective resistors on the input, one pulling to ground and one constraining input current (see datasheet schematic diagram).
- You have built in protective diodes on the output, so driving a relay or motor is fine

And some advice:

Keep in mind, that while you may use the 5V from Pi's power, consuming too much current will make the Pi instable or completely dysfunctional. If you need the chip to decouple and protect the Pi, chances are good that you should use a separate power supply anyway.

Remember to always provide common ground to Pi, ULN2803 and secondary power.

The concrete use of this chip in my project is multipurpose. On one hand this should drive high voltage relays (see beyond), on the other hand I want to drive heating valves directly. Some outputs will remain unused, but the multipurpose chip will surely fit in all output scenarios to come.

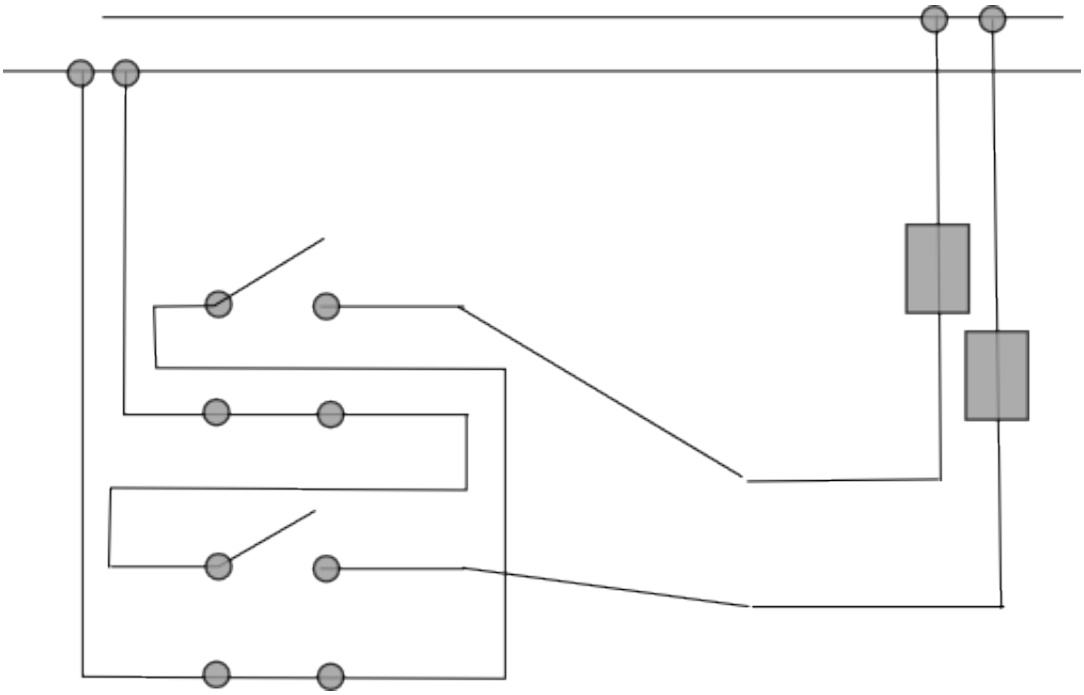
Eltako Relay

The power lines are switched using plain relays for mounting in the service cabinet.

here the R12-100 for switching light and comparable und the R12-110 for switching the shutters. I selected the 24VDC type, needing < 100mA.

Using the R12-100 type is straightforward. The control pins are connected to a GPIO and 24V.

The R12-110 controls the shutter motor, you need one for up and one for down. The R12-110 has an "opener" and a "closer". This feature is used to forestall switching both circuits on. Here's the wiring scheme



These relays are under heavy load, especially when reversing up and down in short period. You really should add mechanisms in the control software to delay for 200-400 ms between changing direction.

Heating Valve

The heating valves are 24VDC, the power consumption should be <100mA (Danfoss 088H3141)

The valves are "normally open", meaning that a correctly installed heating system should never need any power. In case of failure the heating is on.

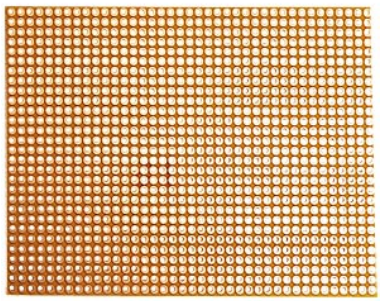
Driver module

The "driver module" is a simple connector for the Pi to the outside world, based on the ULN2803. I was eager to learn some details, so I decided to start this component from scratch.

All in all I can not recommend following this strategy: It's a lot of work if you don't have a PCB, it's error prone and it's not really cheap. In some later chapters I will talk about some alternatives.

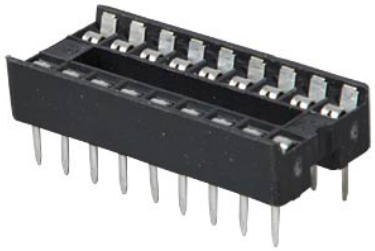
That said, let's get together the components we need (besides the basic stuff mentioned in the beginning).

We have two ULN2803 for a total of 16 "amplified" IO ports. That's enough for me for the



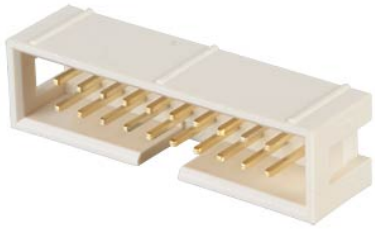
moment, controlling 4 heat valves, some lights and some shutters.

We have the prototyping board,



two IC sockets for the ULN2803,

some screw terminals. These are used for external power supply and all outputs.

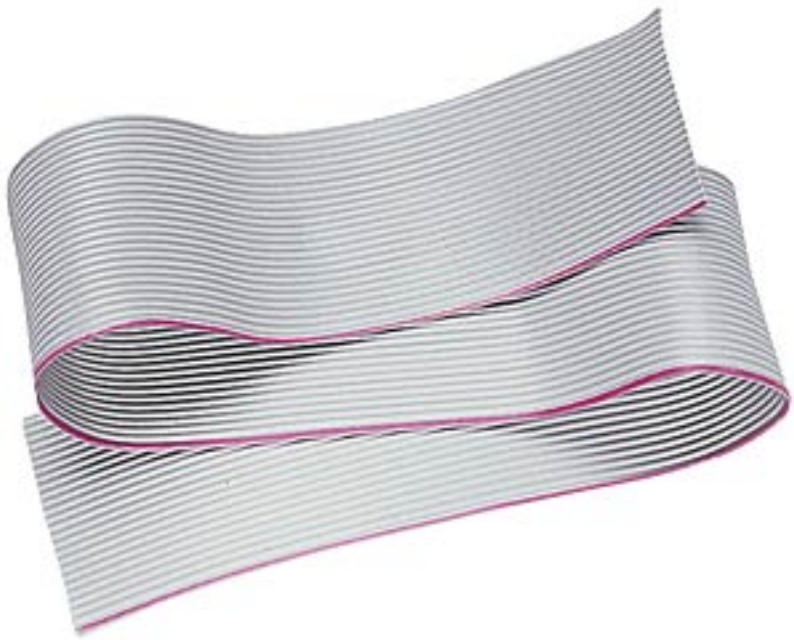


A 20 pin male connector. This is used for connecting the components inputs.

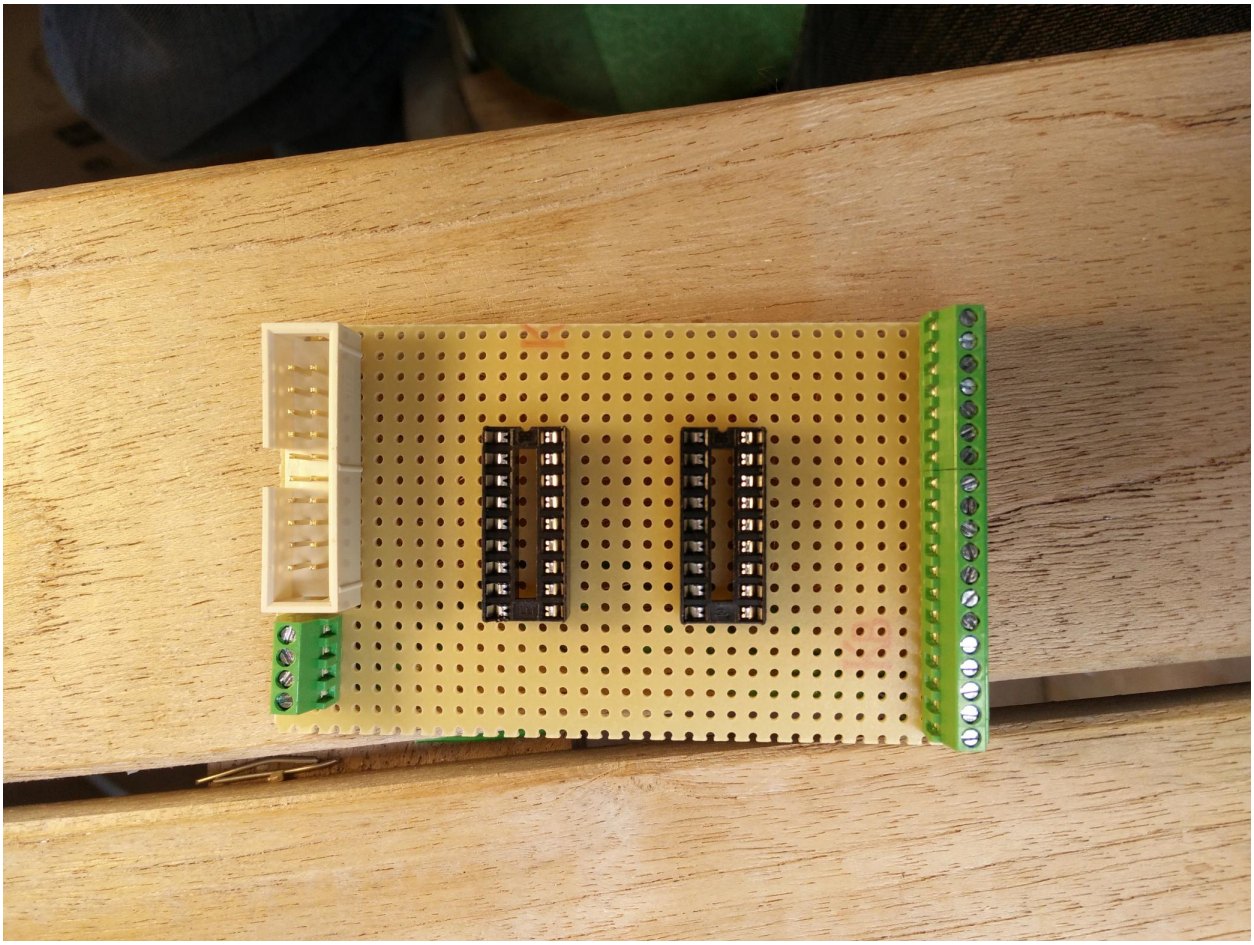
A DIN rail case, 3 units

A 20 pin female connector for the component input.

A 40 pin female connector for the Pi GPIO.



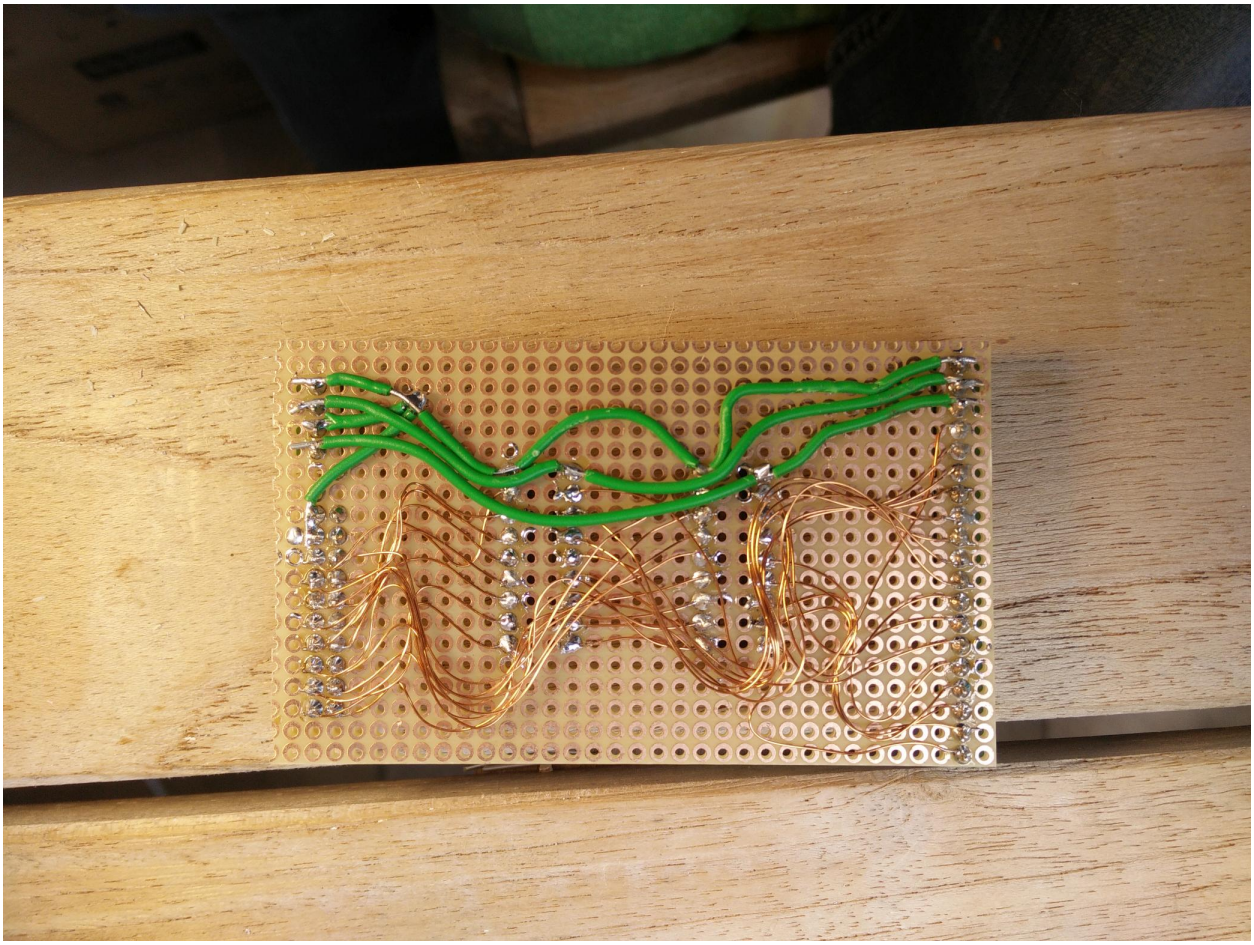
A ribbon cable, 20 cores



The module is set up on a generic prototyping board and deployed in a din rail box. The input side is a 20 pin board connector for the 16 GPIOS and GND and 4 small clamps for the secondary power. Output power ULN2803 can be supplied separately this way.

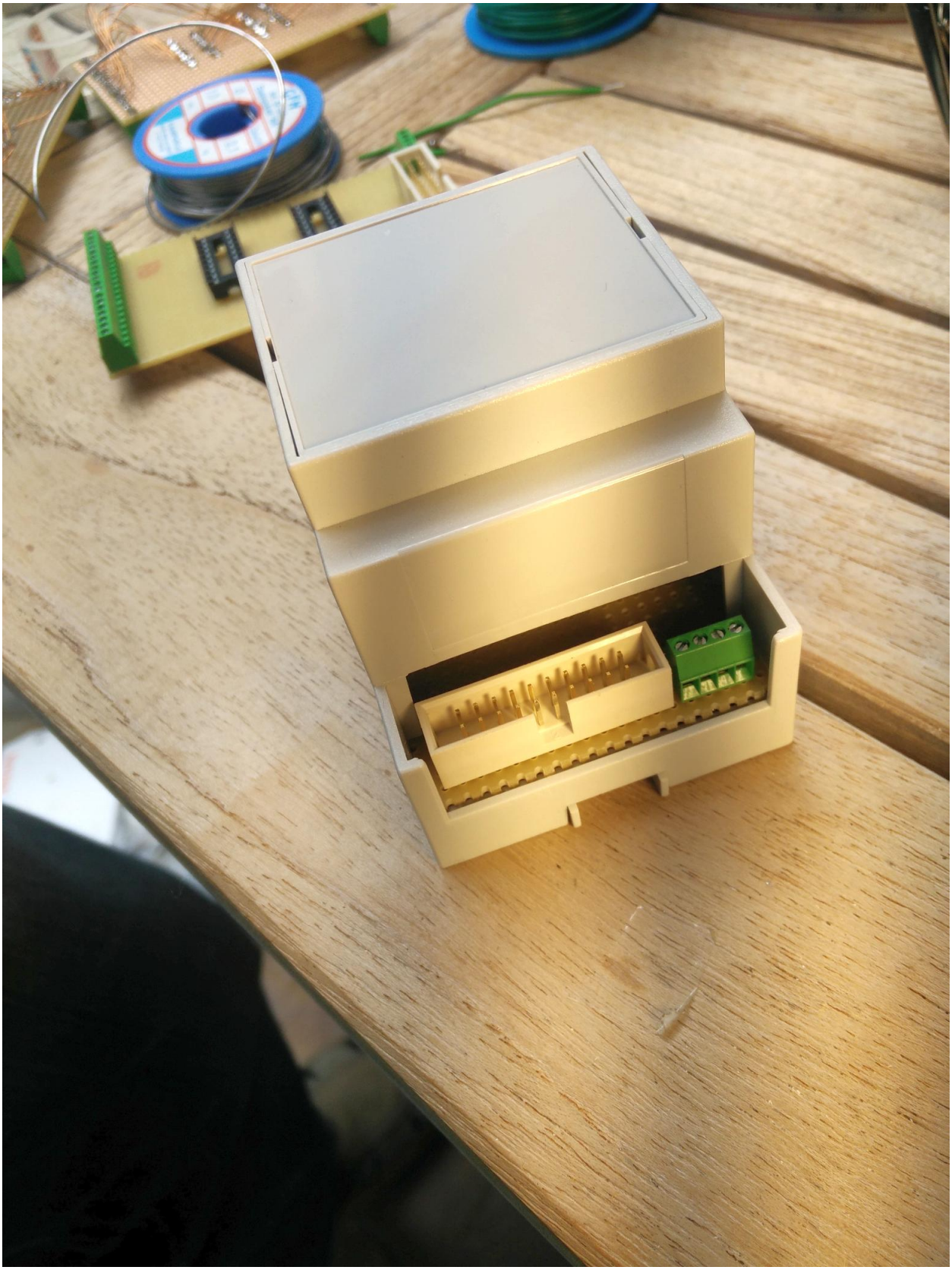
The output is 19 small clamps (well - they just fitted in a 3 unit din rail..) for 16 power outputs, common ground and the two secondary power inputs.

In the middle we have the two ULN2803 (on a socket, accounting for both my poor soldering abilities and the possibility that I may fry one of them later on). The "top" one, near to the input lets call "A", the other "B".



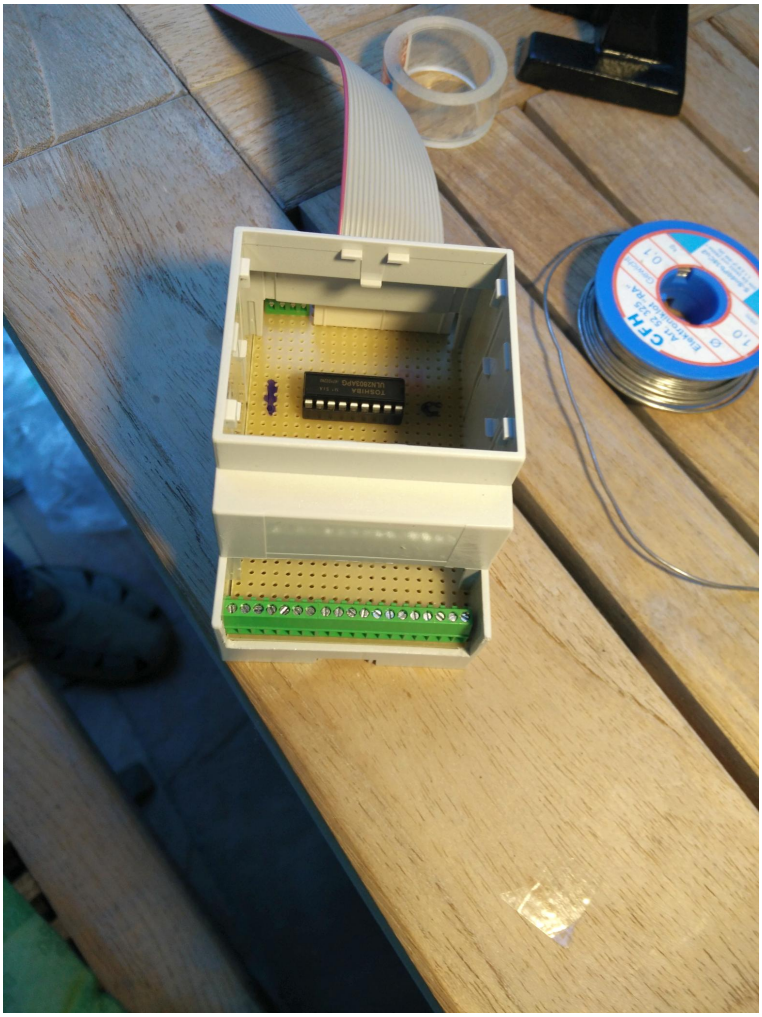
For your amusement I show the soldering side here. It's ok for me, but I can testify that these are the most expensive modules you can have. For me it was at least 2 hours per unit, 5 units in total.

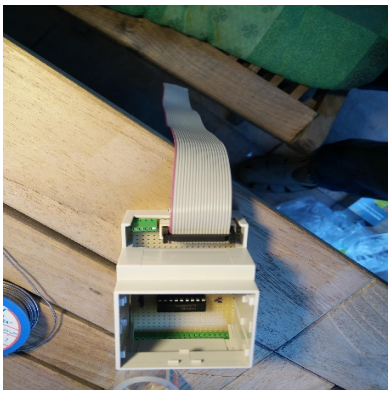
Keep in mind that the overall current **may** be quite high and use appropriate cables for GND and plus side.



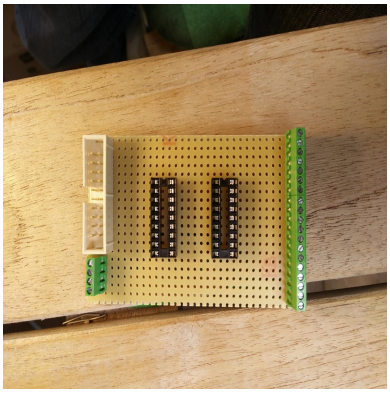
In the din rail housing, where you can't see the nasty details it already looks a little profes-

sional.





Wiring scheme



In the following A and B reference the two ULN2803 chips.

The input power side (green clamp, upper right), numbered from top to bottom

1	GND
2	A +
3	GND
4	B +

The board connector of the "driver module" is connected as follows (where 1 is on the upper left on this picture).

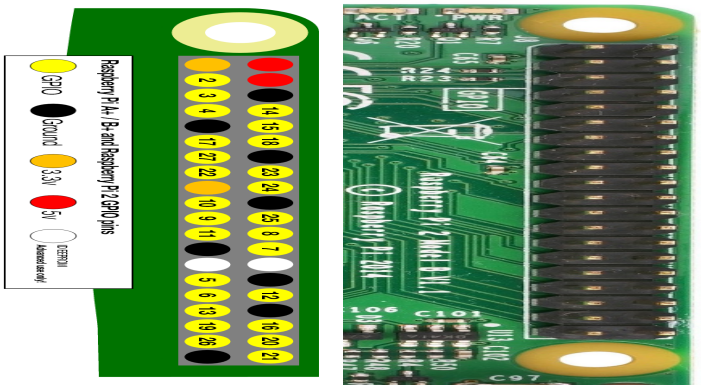
	1	2	GND
	3	4	GND
B IN 8	5	6	A IN 8
B IN 7	7	8	A IN 7
B IN 6	9	10	A IN 6
B IN 5	11	12	A IN 5
B IN 4	13	14	A IN 4
B IN 3	15	16	A IN 3
B IN 2	17	18	A IN 2
B IN 1	19	20	A IN 1

And the clamps on the output side (numbered seen from the top on the picture above)

1	GND
2	A +
3	B +
4	A OUT 8
5	A OUT 7
6	A OUT 6
7	A OUT 5
8	A OUT 4
9	A OUT 3
10	A OUT 2
11	A OUT 1
12	A OUT 8
13	B OUT 7
14	B OUT 6
15	B OUT 5
16	B OUT 4
17	B OUT 3
18	B OUT 2
19	B OUT 1

Connecting the Pi

Let's look at the Pi, this is the pinout of the connector.



	PIN	PIN	
3.3V	1	2	5V
GPIO 02 (I2C)	3	4	5V
GPIO 03 (I2C)	5	6	GND
GPIO 04 (GCLK)	7	8	GPIO 14 (TXD0)
GND	9	10	GPIO 15 (RXD0)
GPIO 17 (GEN0)	11	12	GPIO 18 (GEN1)
GPIO 27 (GEN2)	13	14	GND
GPIO 22 (GEN3)	15	16	GPIO 23 (GEN4)
3.3V	17	18	GPIO 24 (GEN5)
GPIO 10 (MOSI)	19	20	GND
GPIO 09 (MISO)	21	22	GPIO 25 (GEN6)
GPIO 11 (CLK)	23	24	GPIO 08 (CE0)
GND	25	26	GPIO 07 (CE1)
ID_SD	27	28	ID_SC
GPIO 05	29	30	GND
GPIO 06	31	32	GPIO 12
GPIO 13	33	34	GND
GPIO 19	35	36	GPIO 16
GPIO 26	37	38	GPIO 20
GND	39	40	GPIO 21

Detailed information on the Pins is available e.g. at http://elinux.org/RPi_BCM2835_GPIOs.

You can use any of the GPIO Pins, but selecting carefully may leave you the option to add other features later on. The ones selected for this project are rendered in green.

For the driver module board connector I "manufactured" a connector cable from the above GPIOs. This is the board connector wiring scheme.

Red cable	1	2	GND
	3	4	GND
GPIO 21	5	6	GPIO 20
GPIO 26	7	8	GPIO 16
GPIO 19	9	10	GPIO 13
GPIO 12	11	12	GPIO 06
GPIO 05	13	14	GPIO 25
GPIO 24	15	16	GPIO 23
GPIO 22	17	18	GPIO 27
GPIO 18	19	20	GPIO 17

The end to end connectivity on the driver module output clamps (numbered as above) with this configuration is:

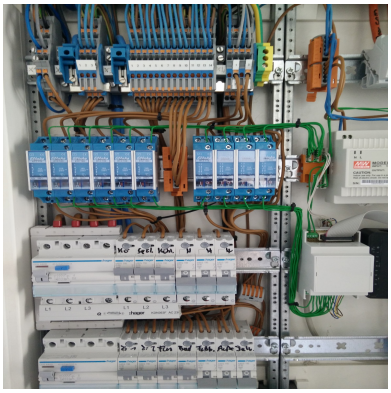
1	GND
2	A +
3	B +
4	GPIO 20
5	GPIO 16
6	GPIO 13
7	GPIO 06
8	GPIO 25
9	GPIO 23
10	GPIO 27
11	GPIO 17
12	GPIO 21
13	GPIO 26
14	GPIO 19
15	GPIO 12
16	GPIO 05
17	GPIO 24
18	GPIO 22
19	GPIO 18

Distribution Box Patching

Now we have to define a correspondence between the physical power lines and valves and the GPIOs.

The electrician was a kind one, he provided terminal blocks for every power line to the rooms. Power is distributed from the terminal blocks to the rooms in a star topology, power is switched centrally via relays.

The power line is either patched directly to a corresponding fuse (for wall outlets) or pathed to one of the relays. The relays power input is patched to the fuse for the power line. The control input (24V) is connected to the driver module.



So, even if this is not of special interest for your project i document the terminal block layout

here - I recommend to do so yourself. The day you will need this table for your reference is near...

Floor 1

Block	Terminal	Relay	Target
1	01	.	Oven
	02	.	Oven/ Oven
2	01	.	kitchen
	02	.	kitchen / washer
	03	.	kitchen / fridge
3	01	.	roomL
	02	.	roomR
	03	.	hall
	04	.	bath
	05	.	heating
	06	.	terrace
	07	R1	kitchen
	08	R2	roomL
	09	R2	roomL
	10	R3	roomR
	11	R3	roomR
	12	R4	hall
	13	R5	bath
	14	R6	terrace
4	01		shutter left N
	02	R7	shutter left grey = up
	02	R8	shutter left black=down
	03		shutter right N
	04	R9	shutter right grey = up
	04	R10	shutter right black=down

Now you can map the power lines to the driver output. In the table below you already see the technical names for the lines that are used throughout the home automation installation from now on.

The name means for example "on floor 1, the kitchen light with index 0". With this table (power line on the left and GPIO on the right) we are able to program the openHab items.

Target	Driver module	GPIO
wbf1_kitchen_light0_state	4	GPIO 20
wbf1_roomL_light0_state	5	GPIO 16
wbf1_roomR_light0_state	6	GPIO 13
wbf1_hall_light0_state	7	GPIO 06
wbf1_bath_light0_state	8	GPIO 25
wbf1_terrace_light0_state	9	GPIO 23
wbf1_roomL_shutter0_up	10	GPIO 27
wbf1_roomL_shutter0_down	11	GPIO 17
wbf1_roomR_shutter0_up	12	GPIO 21
wbf1_roomR_shutter0_down	13	GPIO 26
wbf1_roomL_valve_state	14	GPIO 19
wbf1_kitchen_valve_state	15	GPIO 12
wbf1_bath_valve_state	16	GPIO 05
wbf1_roomR_valve_state	17	GPIO 24
	18	GPIO 22
	19	GPIO 18

Floor 2

Block	Terminal	Relay	Target
1	01	.	Oven
	02	.	Oven/ Oven
2	01	.	kitchen
	02	.	kitchen / washer
	03	.	kitchen / fridge
3	01	.	roomL
	02	.	roomR
	03	.	hall
	04	.	bath
	05	.	heating
	06	.	terrace
	07	R1	terrace
	08	R2	kitchen
	09	R3	roomL
	10	R3	roomL
	11	R4	roomR
	12	R4	roomR
	13	R5	hall
	14	R6	bath
4	06		shutter left N
	07	R7	shutter left grey = up
	07	R8	shutter left black=down
	10		shutter right N
	11	R11	shutter right grey = up
	11	R12	shutter right black=down

Target	Driver module	GPIO
wbf2_kitchen_light0_state	4	GPIO 20
wbf2_roomR_light0_state	5	GPIO 16
wbf2_roomL_light0_state	6	GPIO 13
wbf2_hall_light0_state	7	GPIO 06
wbf2_bath_light0_state	8	GPIO 25
wbf2_terrace_light0_state	9	GPIO 23
wbf2_roomR_shutter0_up	10	GPIO 27
wbf2_roomR_shutter0_down	11	GPIO 17
wbf2_roomL_shutter0_up	12	GPIO 21
wbf2_roomL_shutter0_down	13	GPIO 26
wbf2_roomL_valve_state	14	GPIO 19
wbf2_kitchen_valve_state	15	GPIO 12
wbf2_bath_valve_state	16	GPIO 05
wbf2_roomR_valve_state	17	GPIO 24
	18	GPIO 22
	19	GPIO 18

Floor 3

Block	Terminal	Relay	Target
1	01	.	Oven
	02	.	Oven/ Oven
2	03	.	kitchen
	04	.	kitchen / washer
	05	.	kitchen / fridge
3	01	.	roomR
	02	.	roomL
	03	.	kitchen
	04	.	hall
	05	.	bath
	06	.	heating
	07	.	terrace
	08	R1	kitchen
	09	R2	roomR
	10	R3	roomL
	11	R1	kitchen
	12	R4	hall
	13	R5	bath

Continued on next page

Table 2.1 – continued from previous page

Block	Terminal	Relay	Target
	14	R6	terrace
4	06		shutter left N
	07	R7	shutter left grey = up
	07	R8	shutter left black=down
	08		shutter middle N
	09	R9	shutter middle grey = up
	09	R10	shutter middle black=down
	10		shutter right N
	11	R11	shutter right grey = up
	11	R12	shutter right black=down

Target	Driver module	GPIO
wbf3_kitchen_light0_state	4	GPIO 20
wbf3_roomR_light0_state	5	GPIO 16
wbf3_roomL_light0_state	6	GPIO 13
wbf3_hall_light0_state	7	GPIO 06
wbf3_bath_light0_state	8	GPIO 25
wbf3_terrace_light0_state	9	GPIO 23
wbf3_roomL_shutter0_up	10	GPIO 27
wbf3_roomL_shutter0_down	11	GPIO 17
wbf3_roomR_shutter0_up	12	GPIO 21
wbf3_roomR_shutter0_down	13	GPIO 26
wbf3_roomR_shutter1_up	14	GPIO 19
wbf3_roomR_shutter1_down	15	GPIO 12
wbf3_roomL_valve_state	16	GPIO 05
wbf3_bath_valve_state	17	GPIO 24
wbf3_kitchen_valve_state	18	GPIO 22
wbf3_roomR_valve_state	19	GPIO 18

2.3.7 Software stack

Intro

This is the software stack to run my apartment server.

Basic requirements are:

- run unattended
- run 24/7
- reboot completely after power down

- do not allow unauthorized access
- maybe some day detect failures and reboot

Raspbian

This is the most simple choice. I didn't look for an alternative to deviate as little as possible from the widespread know how on the net (and found it was wise to do so). Even with upgrading at some point from this "Debian wheezy" based image to a "Debian Jessie" distribution I had incompatibilities that I was not able to resolve with my basic Unix know how.

Most information for the Pi and most pre-built software components use this one.

There are tons of installation manuals to do this, I will not repeat the steps in detail - only some advice for the uninitiated:

- Use high quality SD cards
- Follow the basic installation steps
  - Download image
  - Copy to card
  - Launch Pi
  - Resize Image
  - Do internationalization stuff
- Now you are fine and ready to take off. But I recommend to come back after playing around for some more detail steps (see chapter beyond) - most of them quite useful
  - Clean your image from unused stuff
    - Games/Learning stuff
    - X11
    -
  - User administration
  - Set a Static IP
  - Read Only SD card
- Make a backup
- Go on...

Hmland

This installation manual is in **correct** (layered) order, so "hmland" is first. This requirement stems from:

- I want to use openHAB for control
- I want to use Homematic devices
- openHab does not have a builtin engine to control this stuff, it always needs a native device to do this. And i don't want another CCU.
- So I need homegear, an open source reimplementation of the Homematic software, running on the Pi, too.
- homegear does need a "physical" device to talk to the Homematic endpoints. This is some radio chip. Again, this may be embedded in a Homematic device like the LAN adapter, USB stick or a directly soldered CC1101 radio chip.
- I decided for a simple Homematic USB stick. At this moment it was cheap, there exists a fallback scenario (use the stick in another controller) and it was few work (I thought so..).
- This stick can not be controlled by homegear directly
- This is where hmland comes into play. Some enlightened guy has a module that controls the stick and behaves like a Homematic LAN adapter, which you can use in homegear, which is needed by openHAB. So easy.

hmland is not prebuilt for the Pi, but even me was able to do it in minutes. Follow the instructions and everything is fine. If compiled and moved all the stuff to

```
/opt/hmcfgusb/
```

Some caveats:

- hmland should start as a daemon ???, do the init.d stuff
- You can use "hmland -r" to do a fast debug
- Letting hmland serve only on the loopback (127.0.0.1) is a good idea. What happened to me one day was that the loopback interface disappeared (from /etc/network, dunno why) and hmland didn't start any longer. It takes some time to find the reason, maybe you are warned.

Homegear

In the chapter before we motivated already why we need homegear. I must say this is a nice and powerful piece of software itself. The forums are responsive and development is active.

We simply use it as a gateway from openHAB to Homematic devices - this is by far only a piece of its features. If we'd skip the requirement for Enocean compatibility (and the option to support a lot of more bindings), we would have easily stopped with our software stack at this point.

The install of the pre-built Pi package is effortless. Setup is done in /etc/homegear, log is written to /var/log/homegear/ by default.

The working data set is found in /var/lib/homegear, most importantly the device databases. Consider these when setting up backups.

One thing you may consider:

- Enhance the init.d script to require "hmland" in the startup

Java

openHAB is a Java application, we will need an up-to-date installation. I choose and recommend the Oracle JDK.

openHAB

openHAB should be the workhorse for the system. With a Pi 2 I assume there's plenty of memory and CPU for this quite big component.

This is a central hub (pun intended?) for a huge set of so called bindings. Bindings abstract the source and sink of sensor and actor communication and allow for the routing of information via rules.

This way I can get an event from an Enocean component and route it to a GPIO based relay actor, the same way with the information from a Homematic thermostat to a GPIO based driver for an electric valve.

Besides fulfilling this basic integration requirement, openHAB leaves tons of possibilities to scale the solution up own day:

- call external web services
  - calendar based control
  - notification
- internal web services
  - control any feature via web service
- integrate other actor / sensor communication protocols
- integrate system level via "exec"
- time based events

- persistence and monitoring
- web ui

One word of advice: for this installation I used version 1.7.2. Most development effort is spent on the upcoming version 2.0. With this background information and some playing around with the software you can quickly get the impression that you're on the dead end of an initial design prototype. The community has collected a lot of knowledge and ideas, that now are realized in a "completely" new product.

So the bad news is that many issues on the 1.7.2 version will never get fixed or urgent features will never be implemented. You will have to work around different issues and you will fight with suboptimal design decisions.

The good news is - I still think this a very good idea and I hope that 2.0 will bring a lot of compatibility along with solutions for all the nasty ticks of the current version.

Time Travel

At the point of this writing 2.0 is entering beta, what a pity that I'm not able to switch the installation and make my tenants beta testing... but you know, never change a running system...

The installation of 1.7.2 itself is not as straightforward as one may think. There are at least two "mainstream" installation instructions. One works by "apt-get" the installation, but this one didn't work for me. Manual installation using the "wget" approach worked fine, but you have to set up some things yourself, most importantly the init.d scripts.

2.3.8 Configuration details

Raspbian

Maybe it's not so obvious, so a little explanation: I have the following chapters on handling the Pi:

- **Installation (above)** This one is about the most basic steps to get the system running and general advice that is needed on any system. You've got a "multi purpose" Pi now.
- **Configuration (this one)** This is about special setup needed for our apartment server and the software stack we will use here. Things like user setup, GPIO stuff...
- **Advanced (beyond)** Here we **may** dig even deeper to make the Pi even more well suited for an home automation server. It's quite system specific (and you may ruin your system doing it).

Hmland

The hmland is quite simple to setup. Any change is made directly in the init.d script.

- port
- logging
- special options

Especially for achieving "low write profile" there's nothing you must do here, the defaults are fine.

Homegear

Configuration

Homegear configuration is done in two files for my scenario.

First, we look at "/etc/homegear/main.conf". Besides logging (see chapter below), for me there was no need for further changes.

The next one will be "/etc/homegear/physicalinterfaces.conf". Here you setup the real devices and communication channels.

I only set up an id and "rfKey", the base for the AES encryption with the Homematic endpoints. Everything else is default with my installation.

```
## The device family this interface is for
[HomeMaticBidCoS]

## Specify an unique id here to identify this device in Homegear
id = wbf2

## Specify a 16 byte (32 characters) long AES key here to protect your
wireless communication

## !!! IMPORTANT: Never ever lose this key. That would render your devices useless.
 ↪
## To remove the key from your devices, you need to factory reset them
# wbbase
#rfKey = 00000000000000000000000000000000
```

As the documentation states, keep this key in a safe place just in case. Loosing the key will render your Homematic equipment useless.

Important commands

To deal with homegear, you have to setup some configuration on the commandline, as this information is not stored in config files but in a database.

To enter "command mode" type

```
sudo homegear -r
```

while the homegear daemon is already running.

Now you enter the homegear commandline. On any context you can enter "help" to get directives on what you can do.

The first thing to do is selecting the Homematic family of devices (there are short forms of the respective commands, for readability reasons I give only the long forms here)

```
families select 0
```

In the Homematic context you can do some magic with "virtual" Homematic devices, but we don't need none of this. We want to access the "main" device to handle "peer" gadgets.

```
devices select 1
```

Here's where we want to work. You can

- **list all peers "peers list"** This gives you a informative list of all peers paired with the device
- **reset a paired peer "peers reset <id>"** You can unpair and remove the peer. The peer is reset to factory state. This is important if you want to add the peer to another Homematic device later, otherwise the AES key is not removed.
- **select a peer for further inspection "peers select <id>"** To go on working with a dedicated peer, you must select it.
- **enter pairing mode for new peers "pairing on"** If you want to add new peers to the device, you should do this using the pairing mode. After enabling pairing mode, for 60 seconds the device scans for peers that enter pairing mode themselves. After the pairing mode is finished, you can look up the peers detected using "peers list".

Send a list of all devices to a file

```
sudo homegear -r | tee /home/pi/homegear-peers.txt
```

In the homegear commandline enter

```
families select 0
devices select 1
peers list
```

and you will get

```
>
> families select 0
Device family "HomeMatic BidCoS" selected.
For information about the family's commands type: "help"
(Family)> devices select 1
Device selected.
For information about the device's commands type: "help"
(Device)> peers list
ID | Name | Address| Serial     | Type | Type String      | FW  | CP | Unreach
----------âĬij--------âĬij------------âĬij------âĬij------------------âĬij-----âĬij----âĬij--------
   |      |        |            |      |                  |     |    |
1  |      | 123450 | LEQxxxxxxx | 00AD | HM-TC-IT-WM-W-EU | 1.2 | No | Yes
2  |      | 123451 | MEQxxxxxxx | 00AD | HM-TC-IT-WM-W-EU | 1.2 | No | Yes
3  |      | 123452 | LEQxxxxxxx | 00AD | HM-TC-IT-WM-W-EU | 1.2 | No | Yes
4  |      | 123453 | LEQxxxxxxx | 00AD | HM-TC-IT-WM-W-EU | 1.2 | No | Yes
---âĬť------âĬť--------âĬť------------âĬť------âĬť------------------âĬť-----âĬť----âĬť--------
(Device)>
```

Pairing

Enter the pairing mode for the main Homematic device

```
families select 0
devices select 1
pairing on
```

Now enter the pairing mode for each peer you want to add.

Database

The database for the devices and peers is written to /var/lib/homegear/db.sql. For my installation i pre-built 3 different device configurations in my lab, not in the production environment.

To do this, I setup all devices for a single installation

- Change RF key
- peer all device

- backup db

Then I "reverted" by

- delete BidCos central device (via homegear commandline)
- create new one
- replay steps above

openHAB

Intro

openHAB is **big**. There's fewer documentation than there should have been, and I think because of this there's a lot of third party advice of how to do something. The bad news is that this is often incomplete or outdated.

I will give you the info I have built up so far with no guarantee that this is the best practice way or that it still holds true at the time of your reading..

Installation

Time Travel

When I first installed openHAB, it was still version 1.6.x and, as I said above, it was not clear what was the official and the best way to install. There have been 2 different installation directory recommendations and the choice between manual and automatic installation.

Today the situation is similar, but with a little more advice from openHAB itself

- The recommended installation directory is /opt/openhab for a manual installation
- The recommended way for a Pi (Linux) is using the automatic repository based installation. This has the advantage ofa utomatic software updates and automatic startup scripts. But it still installs somewhere else:
  - /usr/share/openhab holds the "binaries"
  - /etc/openhab holds the configuration

You should have this difference in mind when reading some tutorials with references to directories.

Java

But first, to run openHAB you must have installed the Java Runtime. This process is simplified, too and only requires

```
sudo apt-get install oracle-java8-jdk
```

Even if this is already installed on your system, you should check if this VM is the default to be used when running java.

```
pi@wbbase ~ $ java -version
>java version "1.8.0"
>Java(TM) SE Runtime Environment (build 1.8.0-b132)
>Java HotSpot(TM) Client VM (build 25.0-b70, mixed mode)
```

If the version is not what you expected, you may have to switch the "alternative" implementation.

```
sudo update-alternatives --config java
sudo update-alternatives --config javac
```

Dabei jeweils die neu installierte Version auswÃďhlen.

openHAB runtime

While the installation via apt-get is more straightforward, you have to add the repositories first.

This will add the public key for the signature.

```
wget -qO -'https://bintray.com/user/downloadSubjectPublicKey?username=openhab' |␣
↪sudo apt-key add -
```

And this will add a new repository source where the openHAB artifacts are found.

```
echo "deb http://dl.bintray.com/openhab/apt-repo stable main" | sudo tee /etc/apt/
↪sources.list.d/openhab.list
```

The "stable" token refers to the version you want to track in your installation. "stable" means stable releases, so this will include vaersion changes from 1.6.x to 1.7.x, too. If you don't want this, you can restrict to a versioned release (e.g. using "1.8.0" instead of "stable").

After changing the sources, you can update the apt meta information

```
sudo apt-get update
```

Now you are ready to install openHab

```
sudo apt-get install openhab-runtime
```

If you want to autostart openHAB (and you should), update the startup meta data

```
sudo update-rc.d openhab defaults
```

The installation will add an âĂđopenhab" user, but installation artifacts are owned by "root". You may want to change this.

openHAB service configuration

If you have installed and plan to use the service, you should have a look at /etc/default/openhab. This is where you find the main settings the binaries are using when starting the openHAB container. The user openHAB is running under is of particular interest - some bindings still require "root" access. At the time of installation this was the easiest way to interact with GPIO, so I switched it to root.

```
USER_AND_GROUP=openhab:openhab
```

to

```
USER_AND_GROUP=root:root
```

openHAB runtime configuration

There are other numerous configuration options around, here a short overview:

- **login.conf** You can set up authentication for accessing openHAB
- **quartz.properties** openHAB relies on quartz for scheduling tasks. This is the global quartz configuration file
- **logback.xml** The "logback" logging framework configuration
- **jetty/ directory** "jetty" is the built-in web server. You can provide detailed configuration here
- **configurations/ directory** The main openHAB site settings
- **configurations/openhab.cfg** The global settings for all bindings and the container. You should copy "openhab_default.cfg" for a fully documented configuration template. Because of the size of this file I really recommend deleting all addon configuration that is not used at your site. We will come back to this file when we are talking about concrete addons we will have to include.

openHAB runtime data

```
/var/lib/openhab
```

openHAB logging

By default you can find the openHAB log files in /var/log/openhab

openHAB addons

Now you need to add some flesh to the bones, openHAB is only the "framework". The interesting stuff is done in "extensions" and you have to carefully select which one to use. Adding all extensions will bring down your Pi.

This command will show you everything you have at hand to add to your installation (and the list is pretty long). You can add extensions at any time later on.

```
sudo apt-cache search openhab
```

For installation, enter

```
sudo apt-get install openhab-addon-${addon-type}-${addon-name}
```

For the most important tasks in this project we will use the following addons

- openhab-addon-io-gpio
- openhab-addon-binding-gpio
- openhab-addon-action-homematic
- openhab-addon-binding-homematic
- openhab-addon-binding-enocean

This will provide us with the possibility to switch Pi GPIOS for controlling our power lines and heating valves and to interact with the homematic and enocean based sensors.

We will go through some of the other extensions later.

2.3.9 openHAB Items

As openHAB is at the center of all automation here, we have this special chapter that finally puts together the concrete components.

As already said, this is not a really "complex" automation scenario and you can argue that openHAB is a bit oversized. You're right if you will stop after these initial steps - but I bet you won't.

The great plan

As always it is important to have a "great plan".

One of my most stressed points in real life is "naming". You will never get a scalable and maintainable solution if you do not strive for consistent naming. Every change or issue will send you in a nightmare of "wtf"...

One of the caveats in openHAB declarations is the namespacing (a sure indication that this is still on the hobbyist scale... imho).

Even so you can have different declaration files like "unit1.items" and "unit2.items" all of them live in the same namespace. You can't simply add a new floor to a central installation by copying the "items" file to a new name, the namespace must be explicit in the items name. There's no concept of an include with a parameter either.

So, for our project we have to duplicate the item definitions and choose a slightly different name prefix or suffix in each one.

I further recommend using names that seem reasonable compatible to a later integration in scripting environments. This means using only

- a-z, A-Z
- 0-9
- underscore

Items

I recommend using a consistent naming scheme, e.g.

```
<site>_<group>_<target>_<property>
```

Example

```
wbf1_kitchen_light1_state
```

An example for an overall namespace definition

site	wbbase | wbf1 | wbf2 | wbf3
group	kitchen | bath | hall | roomL | roomR | terrace | heating | service | <any top level group you think off>
target	switch<n> | light<n> | shutter<n> | therm<n> | valve<n>
property	state | up | down | <other item boolean properties> lowbat| rssi | <other item meta properties> tempAct | tempSet | humidity | <other item sensor data>

GPIO

Overview

GPIO - the "general purpose Input/Output" of the Pi can be used to sense or switch a digital (on/off) signal. The binding within openHAB is done based on the generic Linux GPIO framework.

For a raspberry in the current openHAB version you should not need to add anything to the openhab.cfg to use GPIO.

This is an example on how to control the pins using standard Linux commands.

```
cdÂă/sys/class/gpio
sudo sh -c 'echo 17 > export'
sudo sh -c 'echo out > gpio17/direction'
sudo sh -c 'echo 1 > gpio17/value'
sudo sh -c 'echo 0 > gpio17/value'
sudo sh -c 'echo 17 > unexport'
```

Its important that you must "reserve" the pin using "export" and finally "release" the pin using "unexport". Another application (or the same application after a restart) cannot access the pin resource otherwise. Even when in newer versions a "force" property tries to forcefully unexport/export the pin before use, I recommend adding those directives to the start/stop script instead.

Initially only "root" has access to these resources. If you do not run openHAB as <root>, you must allow access to the GPIO resources

```
sudo adduser openhab gpio
```

openHAB will really try to release its GPIO resources, but it's better not to rely on this, there are enough reasons why you may find dangling state when starting up to always set up GPIO before properly. For the setup defined in this project you should add the following code to your startup script

```
echo unexport GPIO
echo 17 > /sys/class/gpio/unexport
... // for every pin you use
```

Item definition

The basic template for GPIO item definition is

```
gpio="pin:PIN_NUMBER "
```

You can find more details in the https://github.com/openhab/openhab/wiki/GPIO-Binding page.

Defining a "Switch" item will make the pin an output, defining a "Contact" item will make it an input.

Full item example

```
Switch btnTest "TestOut" { gpio="pin:17âĂş }
Contact ctcTest "TestIn" { gpio="pin:18âĂş }
```

We use this basic pattern heavily to define the control for power lines and heating valves.

Enocean

The enocan binding allows to include Enocean push buttons in your installation easily. We use them for controlling the GPIOs only here, but you can launch anything from this event source.

To enable enocean, you have to add the serial port that is used by openHAB to communicate with the enocean hardware. I used a USB based gateway that registers itself as /dev/ttyUSB0. So you have to configure in openhab.cfg

```
##############################
# enocean
#
# USB serial port
enocean:serialPort=/dev/ttyUSB0
```

Now we need to define the items we are communicating with.

All buttons come with an id printed on them - this is the "id" parameter for the binding string in the item definition, eg. "ab:cd:f0:e3".

In addition you must declare the enocean item type "eep" (Enocean equipment profile) (which is "F6:02:01" for the common push button). Enocean is a well documented technology, you find all supported item types in the respective specification.

Depending on the item types, additional "channel" and "parameter" arguments may be given.

All in all the syntax for the binding is

```
enocean="{id=<myid>, eep=<mytype> [, channel=<mychannel>] [,parameter=<myparameter>
 ↪]}
```

Full item example

```
Switch btnTest "Test enocean" { enocean="{id=aa:bb:cc:dd, eep=F6:02:01,channel=A}
 ↪" }
```

Depending on the openHAB item type a lot of magic is done behind the scenes, you can map the button to a switch, a dimmer or a rollershutter.

I only use the push button in my installation, so i only dare to rant about this one.

A push button (of type "F6:02:01") as described in the Enocean Equipment Profile specification, supports an enumeration of buttons "AI", "AO", "BI"and "BO", this is one channel for each of the 4 sensor areas.

In an unknown logic this transforms to openHAB parameters

- channel "A" and "B", the left and right switch
- optional parameter "I" (an uppercase "i") and "O" (an uppercase "o"), the "up" and "down" sensors

You can map the item to a "Switch" type, with 2 different behaviors.

- **enocean="{id=00:00:00:00, eep=F6:02:01 , channel=A}"** This will result in a switch that updates to **ON** when AI is pressed, nothing happens when the button is released. The switch is turned **OFF** when AO is pressed. You need two physical button sensors to control the switch
- **enocean="{id=00:00:00:00, eep=F6:02:01 , channel=A, parameter=I}"** This will result in a switch that updates to **ON** when AI is pressed, and to **OFF** when the button is released. Nothing happens when AO is pressed.

Homematic

homematic is an established german provider of home automation appliances, dating back to times when home automation was not a hype yet.

To support homematic, you need a homematic server appliance, either in the form of a real homematic "ccu" or the already mentioned and installed **homegear**.

Basically you only need to add to openhab.cfg

```
############################## Homematic Binding
####################################
#
# Hostname / IP address of the Homematic CCU
homematic:host=127.0.0.1
```

I left all other settings untouched. This will make all devices and their properties available for use in your item definitions.

Thermometer

from https://forum.homegear.eu/viewtopic.php?f=23&t=169&p=1147&hilit=boost_state#p1147

"CONTROL_MODE" ist nur lesbar und zeigt dir den aktuellen Modus an.

"BOOST_STATE" ist ebenfalls nur lesbar und wird als Ereignis gesendet, sobald "BOOST" aktiviert wurde. Die angegebene Zeit ist die Dauer des Boostmodus ab diesem Ereignis.

AUTO_MODE, BOOST_MODE, COMFORT_MODE und LOWERING_MODE sind vom Typ "Action", d. h. sie haben eigentlich keinen Wert. ÃIJbergeben musst du aber einen "Boolean" "true". Durch setzen der Variablen setzt du den entsprechenden Modus.

MANU_MODE akzeptiert als Parameter die manuell einzustellende Temperatur als Float, d. h. es wird sowohl der Modus "manuell" aktiviert als auch die manuelle Temperatur gesetzt.

Fehlt noch PARTY_MODE. Da sollte immer PARTY_MODE_SUBMIT verwendet werden, da sonst unnÃűtig viele Pakete Ãijber den ÃĎther gesendet werden. Siehe: https://forum.homegear.eu/viewtopic.php?f=17&t=115&p=736&hilit=PARTY_MODE_SUBMIT#p736

2.3.10 openHAB Rules

Overview

Rules define the logic ffor your site. What happens when a certain event arrives. The rules are the wires between your sensors and actors.

Any file ending with ".rules" in the <openHAB>/configuration/rules folder is parsed and inserted in the rule set.

System events

Currently two system events are supported

- System started
- System shuts down

You may nearly always need "System started" to initialize the items to their state. After the system is started, the state of each item is set to "Uninitialized". This state is altered then as soon as information from associated bindings start to drop in or when previous state from persistence providers is found. All other stay "Uninitialized" (Be aware that the state is reset, too, when items are reloaded - and that in this case there is no "System started" event, so you simply have no choice to get them initialized).

Example

```
rule "init"
when System started
then
        wbf1_heating.postUpdate(ON)
end
```

Cron events

An interesting feature is triggering an event via a timer expression. The name "Cron" stems from well known unix system component that schedules scripts in the background based on time expressions.

The expression consists of 7 fields

```
<seconds> <minutes> <hours> <dayOfMonth> <month> <dayOfWeek> [<year>]
```

Supported values

Field	Values
seconds	0-59
minutes	0-59
hours	0-23
dayOfMonth	1-31
month	0-11 JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC
dayOfWeek	1-7 SUN, MON, TUE, WED, THU, FRI, SAT 1=Sunday

The wildcard '*' character stands for "every possible value".

The questionmark '?' means no specific value and can only be used in **dayOfMonth** and **dayOfWeek** to leave one the fields undefined.

The slash '/' specifies increments in the form of "base/increment" and can be used with numeric values. For example to specify "every 10 minutes, starting at 0" one could

```
0 0,10,20,30,40,50 * * * ?
```

or better

```
0 0/10 * * * ?
```

There are some other specialties, for more information you should look up some cron manual. Most of the times this syntax will suffice.

Examples

Every Monday at 12:00:00

```
0 0 12 ? * MON
```

At 0:0:0 on the 15th of april

```
0 0 0 15 APR ?
```

This cron expression can be used as the condition in a rule

```
rule "cron based rule"
when
Time cron "0 0 0 15 APR ?"
then
        // do your stuff
end
```

Item events

The "real meat" is provided by the events that are triggered by the items.

You can check for commands received of state changes. Every trigger condition can optionally be restricted to a specific command or state.

```
Item <item> received command [<command>]
Item <item> received update [<state>]
Item <item> changed [from <state>] [to <state>]
```

A typical and simple rule would be

```
rule "switch light"
when wbf1_kitchen_switch_state changed
then
        wbf1_kitchen_light_state.postUpdate()
end
```

Lighting

Heating

2.3.11 openHAB Sites

UI

For this installation a UI is not planned immediately. The user will interact using physical sensors. Nevertheless, I created sitemap for "personal use".

I bet some geeky tenant will show up soon and requesting remote access...

2.3.12 openHAB Persistence

One of the original goals of the installation was to get away "write cycle free" or at least rely only on tmpfs compatible information.

With openHAB this gets difficult for two reasons

- You may have bindings that don't send state when they are not changed
- You may have "virtual items" that only interact via computations or the UI

Initializing these items may get challenging without a persistence service.

MapDB

The simplest solution to get a system up in its previous state may be the MapDB based persistence service.

First

```
sudo apt-get install openhab-addon-persistence-mapdb
```

The plugin will save (unconfigurable) its file in /var/lib/openhab/persistence/mapbd/storage.mapdb.

To configure the persistence, create a new file "mapdb.persist" in "configurations/persistence".

```
Strategies {
}
Items {
        gRestore* : strategy = everyChange, restoreOnStartup
}
```

This is the simplest one - it declares that every item that belongs to group "gRestore" is automatically saved after a change and restored upon startup.

You can tweak the settings in "openhab.cfg", but they seem reasonable.

You can perform a simple test - add a "Switch" item, map it on a sitemap. After switching on, wait at least 5 seconds and restart openHAB. If the sitemap comes up with the switch activated, it was restored from mapDB.

Excerpt from the items definition.

```
Group gRestore
...
Switch mapdbTest (gRestore)
```

Adding a simple rule like

```
rule mapdbTestRule
when Item mapdbTest changed
then
        logInfo("demo", "mapdbTest is " + mapdbTest.state)
end
```

will demonstrate you, that the rule is executed (as expected) after the change initiated by the persistence service.

2.3.13 openHAB - more addons

Here we will have a short look at some of the more interesting and commonly used addons (imho).

Persistence

We have already looked at the mapDB plugin. Depending on your needs, there are other plugins around you may want to use.

- **openhab-addon-persistence-exec** You can perform a shell script for every item storage. You can not retrieve item state with this plugin.

- **openhab-addon-persistence-jdbc** You can use plain JDBC to store and restore item state
- **openhab-addon-persistence-mqtt** You can publish the item state via MQTT. This again is a write only service.
- **openhab-addon-persistence-rrd4j** This is one of the most often used persistence services. Item state is stored in a RRD database, allowing for the retrieval of historic data and the quick creation of complex charts. Mastering RRD is a topic of its own and maybe we will add another chapter some day.

Pushing information

Getting information about the state is one of the key features of home automation. Besides the basic building blocks (like TCP, MQTT or HTTP) to create an information source openHAB comes along with interesting plug and play components

- **openhab-addon-action-mail** You can directly address an SMTP service to send mail from your rules.
- **openhab-addon-action-pushover** A well known provider with a sophisticated (and free) infrastructure for pushing information
- **openhab-addon-action-telegram** Another provider, much like "pushover"
- **openhab-addon-action-twitter** Well, should have heard about Twitter. Here you can connect your installation to a (typically dedicated) account.

All this addons provide some simple means to tasks like

- get informed when washer is ready
- motion detection warning
- ...

Making sound

- **openhab-addon-binding-mpd** This binding can control the "Music player daemon".
- **openhab-addon-io-multimedia-marytts** A binding to a text-to-speech engine

Generic integration

- **openhab-addon-action-mqtt** Trigger MQTT events directly from rules
- **openhab-addon-binding-mqtt** An MQTT client for sending and receiving MQTT events, bound to events on the openHAB bus or from openHAB items

- **openhab-addon-binding-exec** A bi-directional connection to the platform shell. Use executables to read state and interact
- **openhab-addon-binding-http** This is a bi-directional binding to any HTTP server. Features are quite limited and you will be very fast better of using "curl" and the exec binding.
- **openhab-addon-binding-serial** A bi-directional binding of items to a serial interface
- **openhab-addon-binding-tcp** Support plain socket communication

Receiving events

- **openhab-addon-binding-bluetooth** Detect and react on available bluetooth devices
- **openhab-addon-binding-mailcontrol** Control openHAB via emails
- **openhab-addon-binding-asterisk** Get informed about calls

External conditions

- **openhab-addon-action-astro** Calculate sunrise and sunset
- **openhab-addon-binding-astro** Bind astro information to your items
- **openhab-addon-binding-networkhealth** Bind items to network health information
- **openhab-addon-binding-systeminfo** Bind items to platform information like swap space, CPI load,...
- **openhab-addon-binding-weather** Bind items to weather data retrieved from different providers for your location

Date & time based control

- **openhab-addon-io-caldav** This is the basic protocol implementation that is always needed
- **openhab-addon-binding-caldav-command** You need this to send commands via calender entries
- **openhab-addon-binding-caldav-personal** Retrieve entries for display
- **openhab-addon-persistence-caldav** Save item state in a calendar

After installation of the addons you want to use, we can start to configure our binding.

The obvious choice (for some of us) is using the Google calendar for this task. To my regret this is not really supported. The (yet) working old Google CalDav URL is officially deprecated and not an acceptable choice for a production scenario. The new URLs can only be accessed after OAuth2 authorization - which is basically a good idea but does not work out really good for a headless application like openHAB. With 1.9.0 a beta is available that supports access to a certain extent - but for authentication (and re-authentication) manual intervention is needed. I dropped this for the moment, maybe some day we will see better support with some configuration pages.

When starting to search for alternatives we will soon discover that free CalDav services are rare. The last option remaining is hosting the service yourself. Having already installed such a bunch of software for this project, this seems a viable alternative. For this project we opted to give the lightweight "radicale" server a chance. Everything you need to know you will find in the "Tools & Applications" part.

Now we have everything at hand we need. We can now send commands via calendar entries using the simple syntax in the event description.

```
<BEGIN|END>:<Item-Name>:<Command>
```

"BEGIN" will trigger the command at the beginning of the event, "END" at the end. You can have as many triggers in the description as you want, each on a line of its own.

The event summary is free to use as you like.

> **Attention:** An event is read and re-scheduled as long as it is still "active" (its start time is smaller than current time and its end time is larger than current time).

This means:

- For events that are longer than the reload time (default 10 minutes), the BEGIN command is executed multiple times.
- If multiple events are active and act on the same item, command execution sequence is undefined

Tracking

Another interesting option is "tracking" - position based events.

- **openhab-addon-binding-mqttitude** Bind OwnTracks location information to items. You can switch state depending on your position.
- **openhab-addon-binding-openpaths** This is the same idea with another provider, OpenPaths

2.3.14 openHAB - caveats

Really, I do not want to rant around blaming some guys great work. But be warned, openHAB is not only a visionary, great tool. It's a time hog when you are sitting around and the small little "features" play hide and seek with you.

Item initialization

The item values get reset every time the *.items file is reloaded. You may give them initial values in a rule that fires on startup, but this is not executed when only *.items has changed - you have to touch *.rules, too.

Another solution might be to check the item against "Uninitialized" within rules before use, but this does not help when you simply map them to a UI. If for example you map a Number item directly to a Setpoint UI element, after reloading you can no longer access, because it is undefined.

You might get along by adding some persistence service, but this seems quite far-fetched for such a basic task as an objects initial value.

The item state

Don't confuse an item and its state. While you might think in a *items file the line

```
Number foo
```

will define a numeric variable like in other scripting languages you have used, it declares an observable value holder that interacts with the system, especially the event bus that **holds** a numeric value in its **state** property.

In addition, openHAB introduces a type system of its own, so that the Number state is not a Java number or primitive type, but derived from the **State** interface. For a Number item the concrete state value will be a **DecimalType** (inherited from java.lang.Number).

So, this is how you read **rule** statements dealing with these items

```
// this is a real java.lang.Number definition
var Number value

// here we want to know what value the item really holds.
// as "state" is defined as **State**, we must cast it to a **Number**  compatible
value = (foo.state as DecimalType)

// now you can use it for computation
value = value * 2
```

```
// or squeeze the primitive value out of it
var int valuePrimitive = value.intValue
// assign a new **State** to the item
// as this method is internally overloaded, a variety of parameter  types
// are accepted, e.g. **Number** or **String** (and their literals,  for sure)
foo.postUpdate(3)
```

openHAB is **quite** underspecified at this point. For me the most irritating fact was, while we have a **Scripting** and **DSL** approach, from time to time the internals pop up. To find a solution you either copy some strange expression some clever contributor sends you in the forums (my kudos at this point for the active and supportive community) or you have to dig deep in the code to understand what's going on.

The DateTime item

While an item like all the others I think this one deserves a topic of its own as it is **very** special in all respects.

We now know that a DateItem in the *.items file defines the value holder.

```
DateTime date_item
```

We cleverly guess that the **State** it returns is of **DateTimeType.** But this implementation does not share anything with Date/Time implementations you know and as such, as said above, you must go to the plain basics to understand what's going on.

To get a value out of this item you can act upon you have only two options, **getCalendar** gets you a Java Calendar and **toString** gets you a String in format

```
"yyyy-MM-dd'T'HH:mm:ss"
```

(use **format(pattern)** to get any other pattern).

Example reading state

```
var value = (date_item.state as DateTimeType).calendar
var value = (date_item.state as DateTimeType).toString()
```

To get something into it, you only have the options of using a String (again in the above format) or a **DateTimeType** instance. The DateTimeType in return only has constructors for a Calendar or a String (besides the empty one, creating the current date).

Examples writing state

```
postUpdate(date_item, "2016-01-01T01:01:00")
```

```
var Calendar cal = Calendar::getInstance()
cal.setTime(someMillis)
postUpdate(date_item, new DateTimeType(cal))
```

To make things more complicated, much of the examples (and the built-in methods or variables like **parse** and **now**) return **org.joda.time.DateTime** instances (which have **nothing** to do with any DateTime item or DateTimeType). Joda DateTime is very useful for date handling and a good choice anyway. The bad news: there's neither an overload for **postUpdate** nor a constructor for **DateItemType**.

Example

```
var org.joda.time.DateTime dateJoda = parse("2016-01-01")
// do some computation...
// and update item via DateTimeType
date_item.postUpdate(new DateTimeType(dateJoda.toGregorianCalendar()))
// this is equivalent
date_item.postUpdate(new DateTimeType(dateJoda.toString()))
```

This is an epic failure:

```
var org.joda.time.DateTime dateJoda = parse("2016-01-01")
// do some computation...
// and update item via DateTimeType
date_item.postUpdate(new DateTimeType(dateJoda.toGregorianCalendar()))
// this is equivalent
date_item.postUpdate(dateJoda)
```

Why? Not only that it does not work - for reasons we explained above and that you will understand looking in the source code. But this is the log entry you will find to help you debugging:

```
2016-07-24 11:25:48.755 [ERROR] [o.o.c.s.ScriptExecutionThread ] - Error during␣
↪the execution of rule 'SetDate': Could not invoke method: org.openhab.model.
↪script.actions.BusEvent.postUpdate(org.openhab.core.items.Item,java.lang.
↪Number) on instance: null
```

The DateTime UI

The last (?) annoyance with DateTime is its UI element. To be more exact the fact that it does not exist. You cannot enter dates or times. There are some workarounds for this.

First you might choose to "construct" a composite UI from primitives, such as this

The items definition

```
DateTime date_item "Date [%1$td.%1$tB]"
Number date_month_item "Month [%d]"
Number date_day_item "Day [%d]"
```

The sitemap definition

```
sitemap default label="Main Menu" {
        Frame label="Date" {
                Text item=date_item
                Setpoint item=date_month_item minValue=1 maxValue=12 step=1
                Setpoint item=date_day_item minValue=1 maxValue=31 step=1
        }
}
```

The rules definition

```
import org.openhab.core.library.types.*
import org.joda.time.*

rule Startup
when System started
then
        logInfo("wbfRule", "initializing system")
        postUpdate(date_month_item,1)
        postUpdate(date_day_item,1)
end

rule SetDate
when
        Item date_month_item changed or
        Item date_day_item changed
then
        var int year = now.getYear()
        var int month = (date_month_item.state as DecimalType).intValue()
```

```
        var int day = (date_day_item.state as DecimalType).intValue()
        var org.joda.time.DateTime dateJoda = parse(year + "-" + month + "-" + day)
        date_item.postUpdate(new DateTimeType(dateJoda.toString()))
end
```

Similar things can be done with times and examples can be found in the forum.

2.3.15 openHAB special use cases

Intro

Home automation is not only the simple trigger / action stuff we are used to think in because of our "socialization". In everyday life we are used to leverage simple tools like a switch to control things like lights or other electrical devices. It seems natural because tools are not intelligent and we do all the rule processing implicitly and unconscious.

We enter a room and its dark, so we touch the light button. A phone call is pending, so we turn TV down to better understand what is said.

The most interesting home automation comes up when we integrate context information and implicit rule processing and delegate to the system. This project is far from doing exiting stuff, but even in small scale applications you will find that there are many solutions to a problem and that it's fun to inspect it from many sides.

Date based decisions

One of the tasks when controlling the heating is switching of the floor heating in summer. The most basic way may be to go to your apartment two times a year and switch it off for summer and on for winter.

After fooling around with openHAB dates for some time this may be the option of choice for some of us .

But heaving read this book and following many forum conversations you will finally see reason that openHAB behavior is nevertheless deterministic and give it a try. How many options will we find?

Using a timer based rule

If the date when the heating goes off is fix, you can create a cron based rule

```
import org.joda.time.*
import org.openhab.core.library.types.*
```

```
rule "heating off cron based"
when Time cron "0 0 0 15 APR ?"
then
        wbf1_heating.postUpdate(OFF)
end

rule "heating on cron based"
when Time cron "0 0 0 15 OCT ?"
then
        wbf1_heating.postUpdate(ON)
end

rule "initialization"
when System started
then
        var DateTime stopTime = new DateTime(now.getYear(), 4, 15, 0, 0)
        var DateTime startTime = new DateTime(now.getYear(), 10, 15, 0, 0)
        if (now.isBefore(stopTime)) {
                wbf1_heating.postUpdate(ON)
        } else if (now.isBefore(startTime)) {
                wbf1_heating.postUpdate(OFF)
        } else {
                wbf1_heating.postUpdate(ON)
        }
end
```

On April, 15th the heating is switched off, on October, 15th it is switched on. Do not forget to define an initial state - this is a pattern you will need to apply **everywhere** in an event based system. A better alternative might be the correct integration of item state persistence (see below).

The resulting addition in the code for controlling the heating valves

```
...
if (wbf1_heating.state == OFF){
        return false
}
...
```

The ugly thing about this solution:

- the relevant dates are coded redundant, in different syntax formats
- the dates are hard coded and cannot be changed via UI or REST

Using DateTime items in rules

Another solution might bring up two DateTime items. This way we could read and set the respective dates via UI or REST.

```
DateItem wbf1_heating_startDate
DateItem wbf1_heating_stopDate
```

The UI part is quite ugly (see chapters above), but, that's the way it is.

As we do not have an event in this solution, we must either define a cron rule that regularly checks if the heating should be on or off and set **wbf1_heating** accordingly or we must include the check if the heating valves should be triggered within the heating valve rules itself (I don't know if setting a timer instance for such a one shot event in the startup rule is a viable solution).

A possible implementation in the code for controlling the heating valves

```
...
var DateTime startTime = new DateTime((wbf1_heating_startDate.state as␣
↪DateTimeType).calendar)
var DateTime startTimeThisYear = new DateTime(now.getYear(), startTime.
↪getMonthOfYear(), startTime.getDayOfMonth(), 0 , 0 );
var DateTime stopTime = new DateTime((wbf1_heating_stopDate.state as DateTimeType).
↪calendar)
var DateTime stopTimeThisYear = new DateTime(now.getYear(), stopTime.
↪getMonthOfYear(), stopTime.getDayOfMonth(), 0 , 0 );
if (now.isBefore(stopTimeThisYear)) {
        // go on
} else if (now.isBefore(startTimeThisYear)) {
        return false
} else {
        // go on
}
...
```

Using a calendar binding

A quite elegant solution would be the use of a CalDav calendar. Instead of having hardcoded values, ugly user interfaces or polluted code we stay with our

```
Switch wbf1_heating
```

and control the state via some calendar entries.

```
BEGIN:wbf1_heating:OFF
```

The caveat is again the item initialization after restart, you should use a persistence service like MapDB here.

2.4 Basement server

2.4.1 Intro

As previously said, this one mainly handles door communication. In the more advanced parts this component will receive additional services (WAN, network file system) for the complete installation.

In these times I simply found it anachronistic to install an expensive proprietary solution that finally will provide less features than a plain telephony based infrastructure.

2.4.2 Hardware

Server

Well, here the same reasoning as in chapter "apartment server" applies. To no surprise we choose a raspberry here.

Switch

As we will have a bunch of components attached to the basement server, we will need a little switch. I thought about being clever and take an "oversized" one, having 8 ports at hand.

I decided for no special reason for a EDI ES-3308PV3. Be warned: they are always to small...

At the moment these components are connected:

- The three apartment satellite Pis
- The indoor DECT gateway
- The outdoor VOIP adapter
- The heating

Doorphone outdoor

The outdoor part was quite problematic to decide.

There are interesting attempts in the community to create the station completely from scratch ("Doorberry"). But I think there are a some problems to overcome that should not be underestimated

- An outdoor installation has heavy requirements regarding humidity and temperature

- Some complain about poor speech quality, both on the hearing and speaking side.
- The risk of needing "maintenance" on the doorphone (reboot, change SDcard,..) is higher than with a standard product.
- The price difference to a prebuilt module is not that big to justify the risk in my opinion

In short, I wanted a shrink wrapped product.

A proprietary solution was not viable, leaving only an analog or IP based component.

While my initial intention was to go for a completely IP based system, I abandoned the plan simply for pricing reasons. The cheapest solution, based on an EntryCom Vario IP is around 300€ more expensive than a plain analog solution.

The alternative is a plain a/b outdoor telephony station with an ATA adapter. The devices of choice were an "Auerswald Dialog 304" and a "GrandStream HandyTone 701". This sums up to a total of ~270 €. In my opinion this is a valid fallback.

The connections are straightforward, the Auerswald is very good documented.

Doorphone indoor

While "open" outdoor solutions still seem a little bit exotic and expensive, indoor ones are less problematic.

I decided to go for a complete phone instead of a slimmed dedicated doorphone. The main reason: I didn't find a dedicated indoor doorstation based on a/b or IP. But the whole wide world of standard phones is available here...

One possibility is to use a simple (20€) analog phone and an ATA adapter to attach to asterisk. Here i disliked the idea of a stationary phone for this new installation, something cordless would be fine...

At least in Germany a "Fritz!Box" is an interesting component to look at. It has

- Router
- Switch
- DECT base

- Analog phone input
- Telephony software

built in. As you may not need the WAN feature you can buy very cheap ones used around 40€. There's no way to pay less...

The solution I finally selected was DECT based, too. I liked the idea of taking the door bell, door phone and door opener simply with you... With a new product "Gigaset Go Box" I found an interesting solution to do it all digital. It has VOIP support for 6 channels on the incoming side, 6 DECT channels on the handset side. This is enough and even has a small buffer to play around one day.

Along with some "cheap" handsets you are very flexible with your new "door opener".

One should also consider the drawbacks of the telephony approach:

- You can accept the "call" only as long the phone is ringing
- You need a time based automatic hangup (as the visitor will not hangup himself)
- After timeout, the indoor can no longer simply off-hook the phone and talk to the outside - you have to dial a number to reconnect. The outside phone must support auto-accept.
- You have to enter a code combination to open the door instead of a simple push button (at least in a basic installation).
- Some people may see handling this "many buttons" a bit scary in comparison to a plain doorphone.

Time Travel

I have had no user acceptance problem with the "doorbell" so far.

2.4.3 Software

Raspbian

The same basic rules apply to the Raspbian used in this installation.

Freeswitch

This is a short story. While having tested successfully freeswitch on my PC and decided to go for it, I didn't manage a clean install on the Pi. I liked Freeswitch very much for its design and ideas, but hey, it must solve a problem, not create one.

There's an old installer for the 1.4 version and maybe its working.

For the 1.6

- There's no Pi package
- You have to switch to Debian Jessie
- You have to compile yourself
- There's no dedicated help for this setup

I tried the above and failed. Especially switching the distribution did some harm to the already existing installations of my home automation stack.

If you are more of a unix geek you may get it up and running and I assume that a running freeswitch is just fine...

Asterisk

After some failed attempts to stick with the freeswitch know how I gathered on the PC platform, for the sake of eventually coming to a working end I finally installed asterisk.

Installing asterisk was a breeze (even if not the newest release, but my requirements are not on the edge of technology). With a little know how on the tasks and architecture of a telephony software, getting a basic understanding and setting up a working environment was a matter of hours.

```
sudo apt-get install asterisk
```

2.4.4 Configuration

Asterisk

Again, there are a lot of tutorials available. One particular important resource is the (free) "Asterisk-the definitive resource".

The first step in this environment is a relatively small one. We have to go for some configuration files. You will find them in /etc/asterisk.

For our system we only need to look at

- modules.conf
- sip.conf
- extensions.conf

It's best to leave all other files untouched.

modules

Asterisk has a hell of extension modules and I do not claim to know what they do. I can again recommend the excellent book and internet resources.

For me it was important that I have a small footprint, all modules that are immediately needed and a bunch that I plan to use in the near future.

An important point is the "autoload=no" in the header. We do not want asterisk to load anything it stumbles upon. For the rest I started with a listing of all available modules and moved backwards...

Most important is the support of SIP. The rest are dependencies or "future feature" modules (musiconhold, playback).

This is my /etc/asterisk/modules..conf

```
[modules]
autoload=no

load => res_cliorginate.so
load => res_crypto.so
load => res_curl.so
load => res_http_post.so
load => res_http_websocket.so
load => res_musiconhold.so
load => res_rtp_asterisk.so
load => res_speech.so
load => res_srtp.so
load => res_stun_monitor.so
load => res_timing_timerfd.so
load => app_authenticate.so
load => app_confbridge.so
load => app_dial.so
load => app_echo.so
load => app_exec.so
load => app_macro.so
load => app_mp3.so
load => app_originate.so
load => app_page.so
load => app_playback.so
load => app_playtones.so
load => app_read.so
load => app_saycountpl.so
load => app_senddtmf.so
load => app_sendtext.so
load => app_softhangup.so
load => app_system.so
load => app_transfer.so
load => app_verbose.so
load => chan_local.so
load => chan_sip.so
load => codec_alaw.so
load => codec_gsm.so
load => codec_ulaw.so
load => format_gsm.so
load => format_wav_gsm.so
load => format_wav.so
load => func_base64.so
load => func_curl.so
load => pbx_config.so
```

sip

SIP (session initiation protocol) is the most important component here. It's the heart of VOIP connections.

The outdoor and indoor phones are finally terminated on the asterisk machine, dispatching door bell and internal calls using the SIP protocol - just as if they were on different corners of the world.

Every participant is mentioned in the sip.conf. I have added an account for each appartment, the door itself and an "administration" account.

```
[wbf901]
type=friend
host=dynamic
secret=password
context=wb

[wbf101]
type=friend
host=dynamic
secret=password
context=wb

[wbf201]
type=friend
host=dynamic
secret=password
context=wb

[wbf301]
type=friend
host=dynamic
secret=password
context=wb

[admin]
type=friend
host=dynamic
secret=password
context=wb
```

extensions

Extensions finally define what asterisk should do when a certain number is dialed, it is the "dialplan". The dial codes are the default codes used by the Auerswald outdoor phone (except

the 901).

Here we have our minimal rule set. If for example the first button is pressed, a "101" is dialed. This in turn is sent (through the Grandstream) to Asterisk which will activate the wbf101 participant. This participant is attached to Asterisk in the form on one of the IP slots of the Gigaset Go Box.

This is /etc/asterisk/extensions.conf

```
[wb]
exten => 101,1,Dial(SIP/wbf101,20)
exten => 201,1,Dial(SIP/wbf201,20)
exten => 301,1,Dial(SIP/wbf301,20)
exten => 901,1,Dial(SIP/wbf901,20)
```

Well, from here everyone should be able to discover that maybe our software installation is not really challenged with what we do. But then, there are a lot of bells and whistles we can attach using asterisk, especially in combination with the other home automation components.

How about some interactive voice response implementation, sending your annoying visitor to the neighbor next door, a mailbox registering a short message for you or forwarding the visitor to your cellphone.

Dialing special numbers (initiating a call or even while talking to somebody) can trigger scripts or enter the openHAB scene...

Run in commandine mode

If you want to connect to an asterisk running in the background, run

```
asterisk -r
# or verbose, the more v's the more output
asterisk -vvr
```

Once in the console, you can quit with.

Asking for help

```
*CLI> help
```

Exit the commandline

```
*CLI> exit

If you are tweaking around the configuration, you can force reloading the files
```

```
*CLI> reload
# or for the extensions only
*CLI> dialplan reload
```

Connect a softphone

Connecting a softphone for testing purposes is a breeze. I use Zoiper for Android, i think any other (PC based, too) will do.

You only need the server address (port 5060 is SIP default), the user "admin" and its password from your sip.conf and you are ready to call.

Attaching a softphone is especially useful when we are enhancing our asterisk with more control features. Via softphone, the "System" call and a shell script you can do anything...

Calling the system

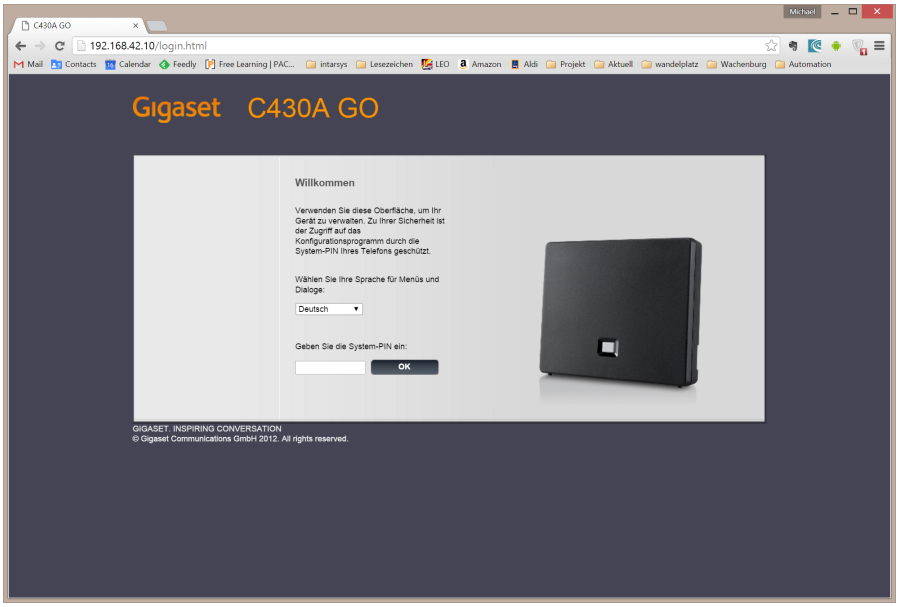
Callouts to scripts from asterisk is a great feature that allows integration in nearly every scenario. This dialplan entry triggers the script "trigger.sh" upon calling 88. For sure the user "asterisk" must have execution rights.

```
exten => 88,1,System(/opt/admin/trigger.sh)
exten => 88,n,Hangup()
```

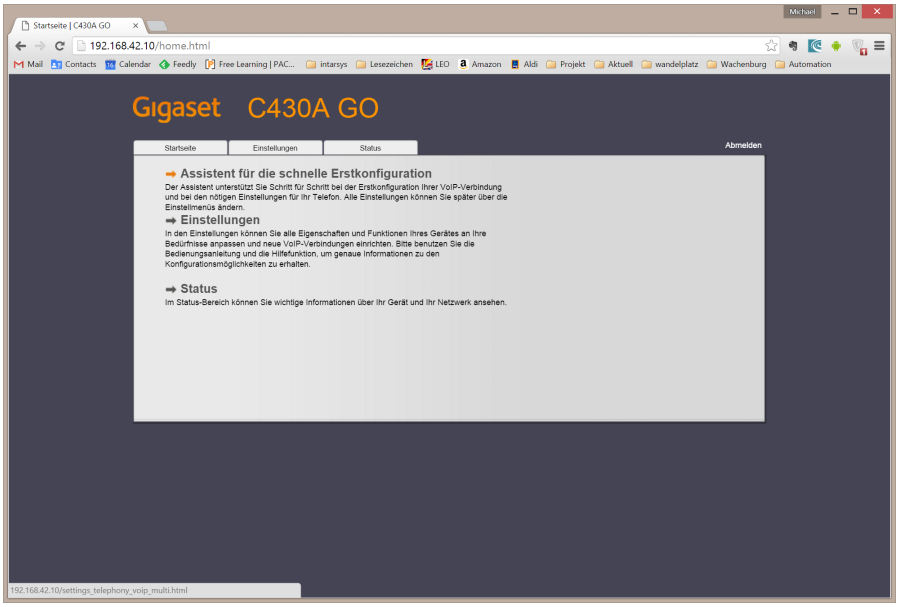
GO/Box 100

This small box is the in-door part of the door com.

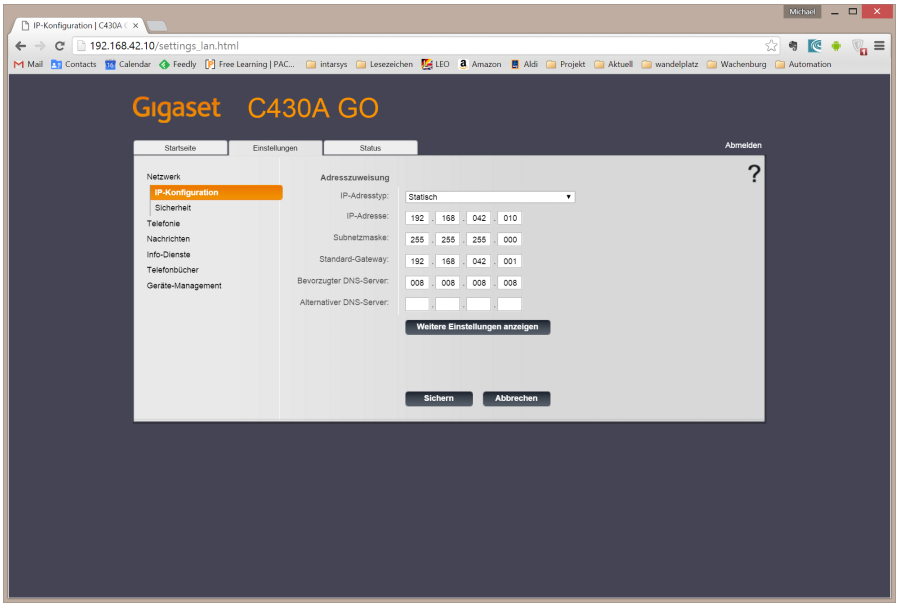
We will go through the configuration and connection to Asterisk shortly.



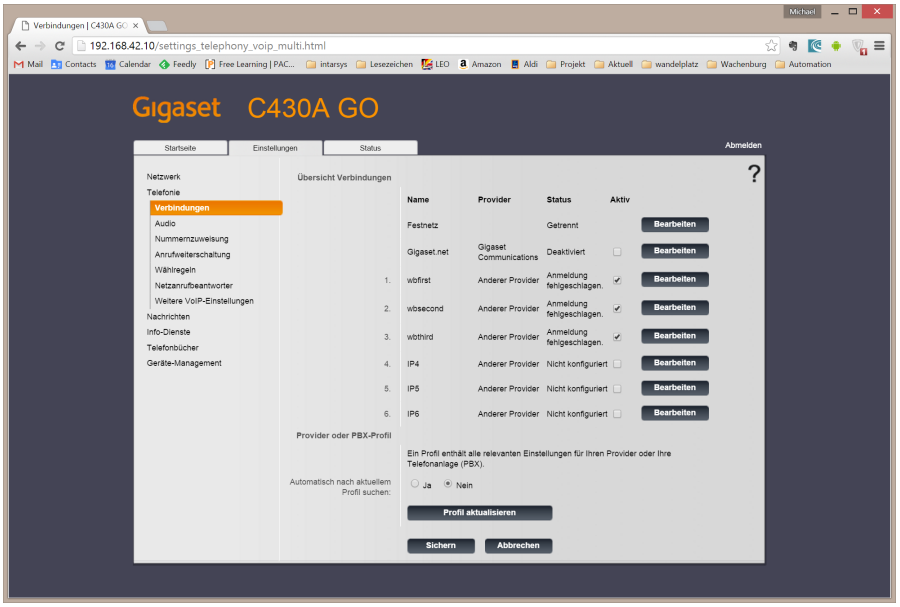
I really recommend changing the PIN - you surely don't want some geeky nerd in the house reconfigure your system.



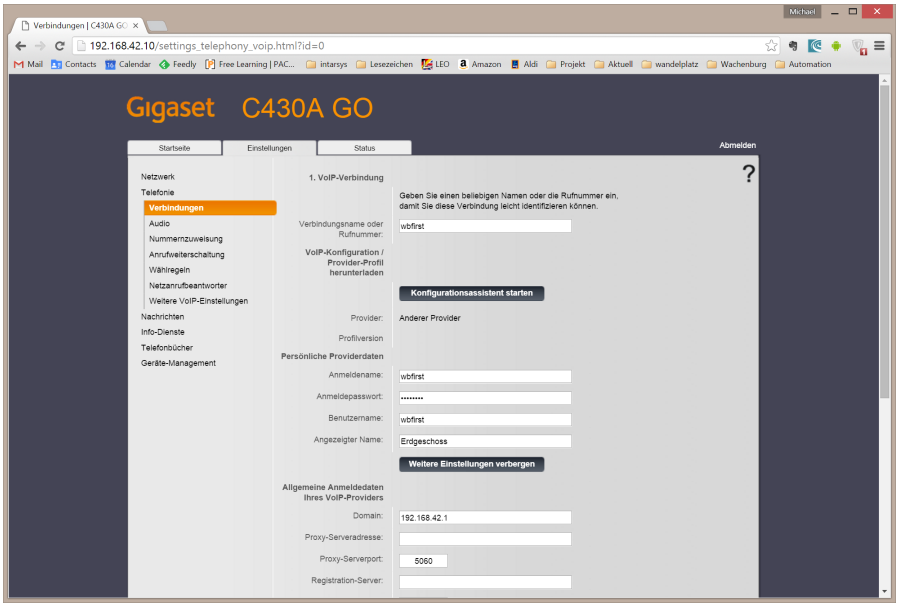
From the start page, we go immediately to the settings.



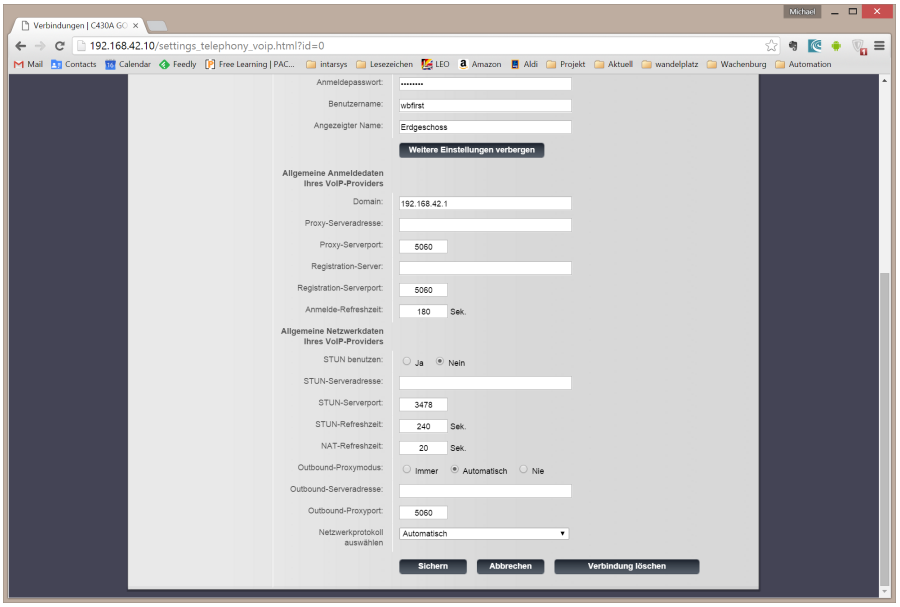
I setup every HA component with a static address - you may go for DHCP, but I recommend setting sticky address assignment.

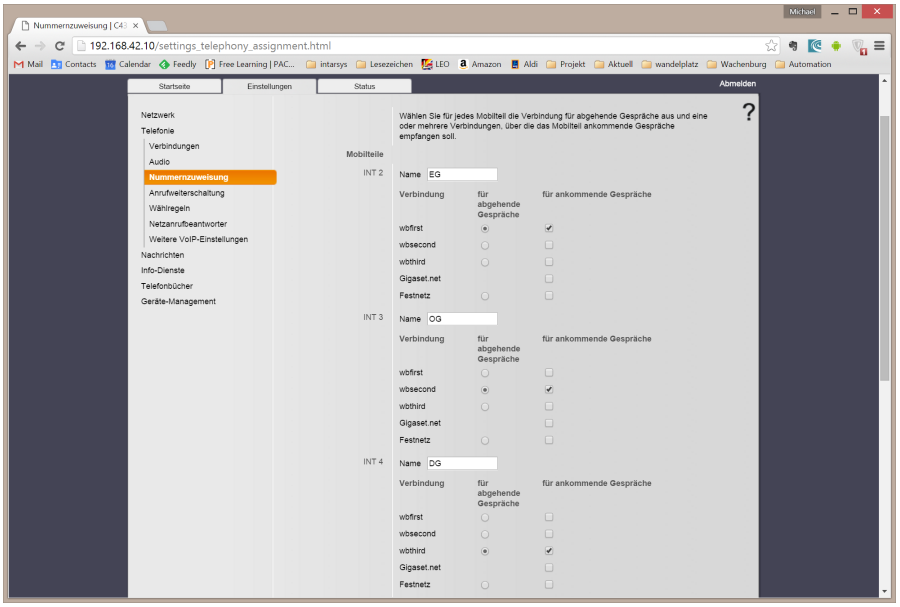


I set up 3 independent IP accounts as clients to the Asterisk VOIP server.

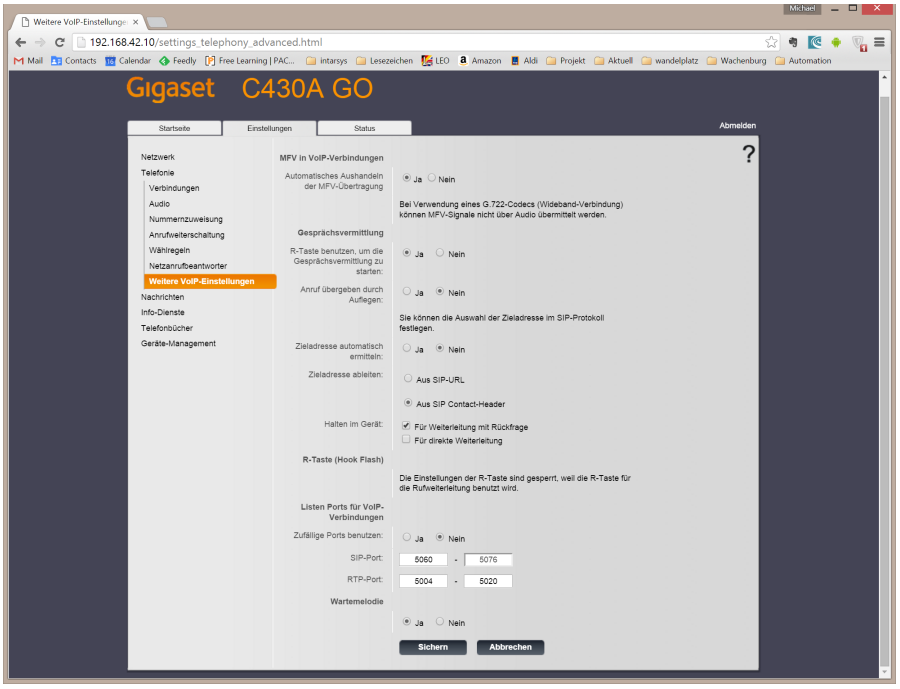


As Asterisk is running on the "gateway" itself, the server IP is 192.168.42.1, the rest is default.





Next, we setup the three handsets and assign each an IP account of its own. This does not mean that we cannot make in-house calls using the handsets.

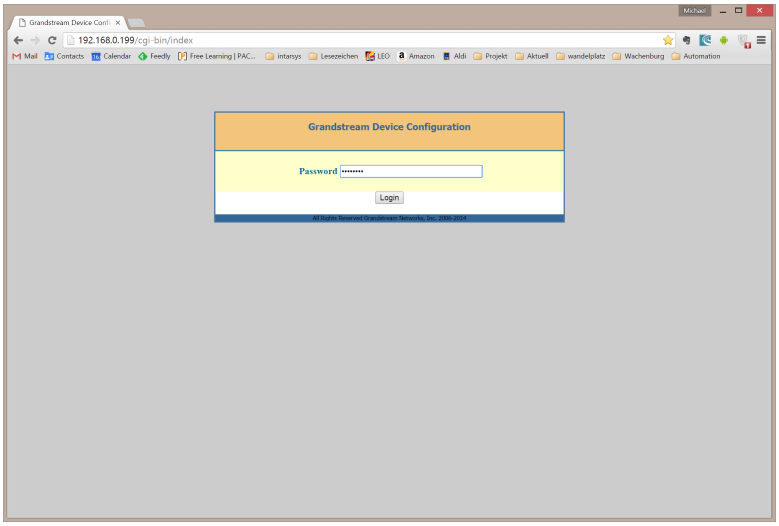


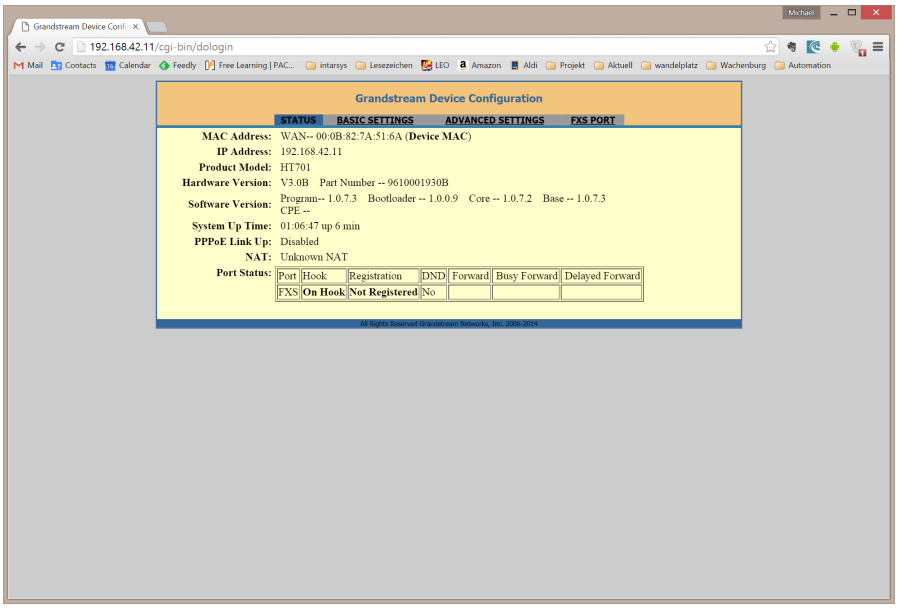
The rest is standard again.

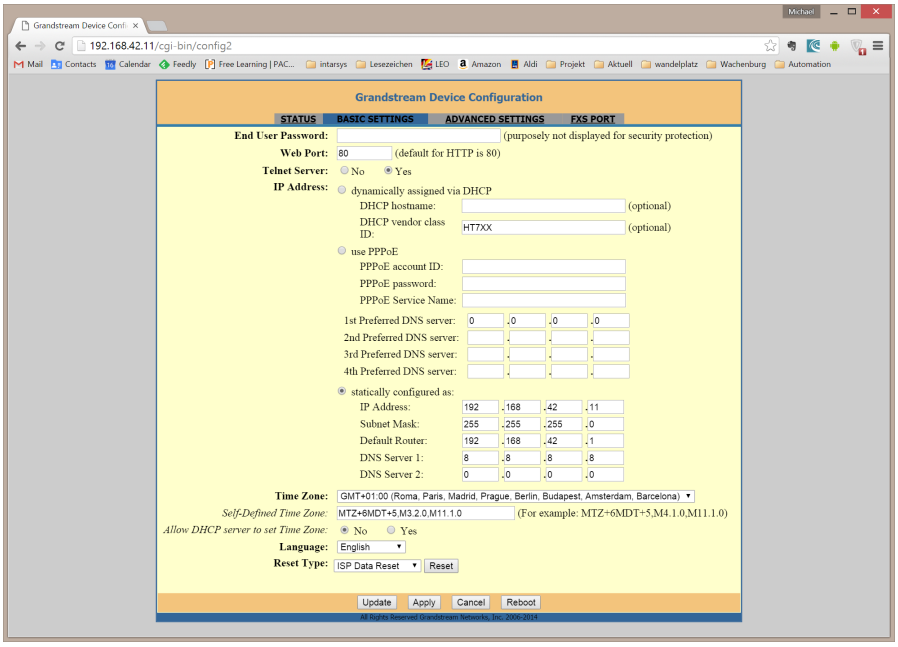
No you should be able to contact the handsets via Asterisk.

Grandstream HT701

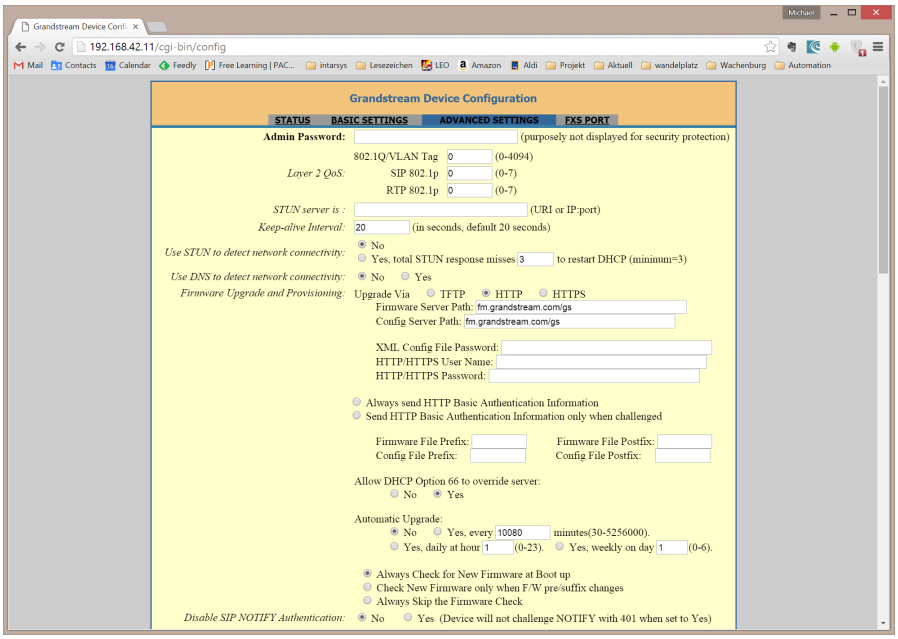
This component is the interface between the analog outside phone (Auerswald) and the VOIP based internals.

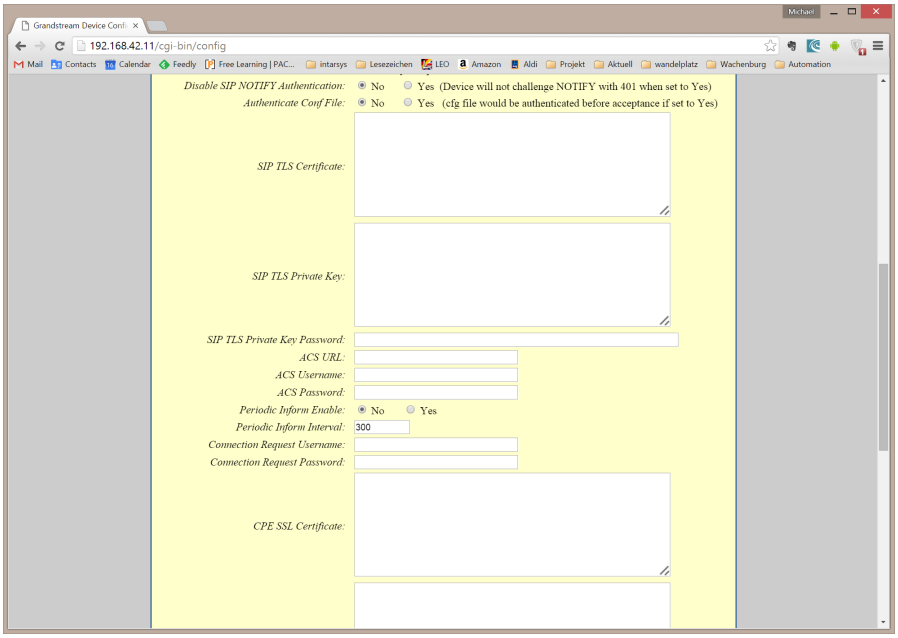


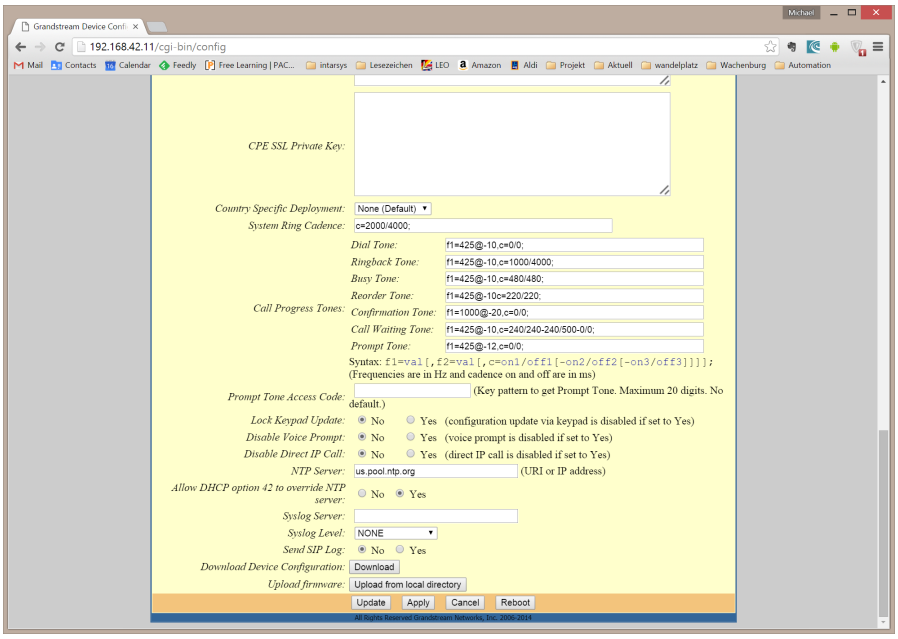




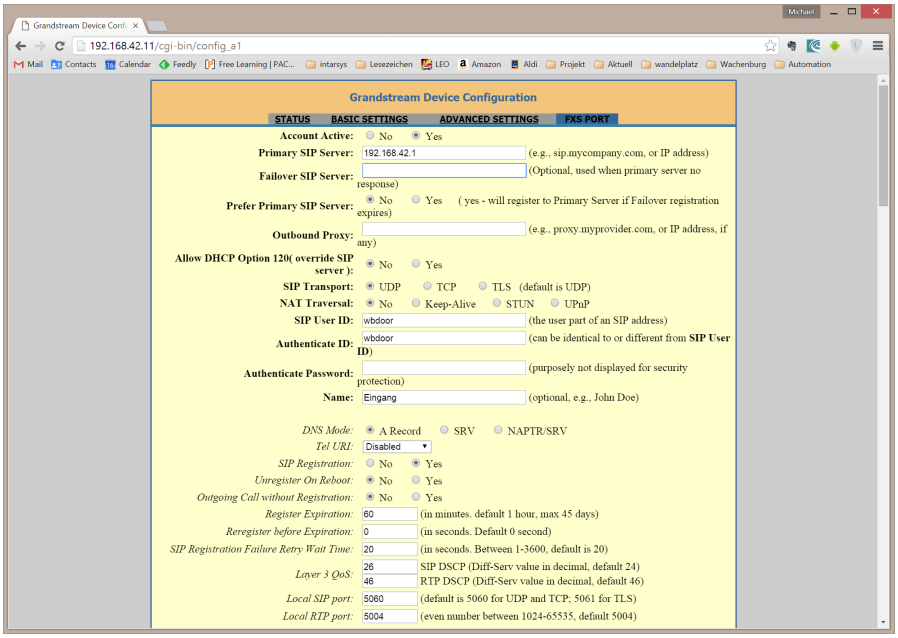
Once again I use static IP.



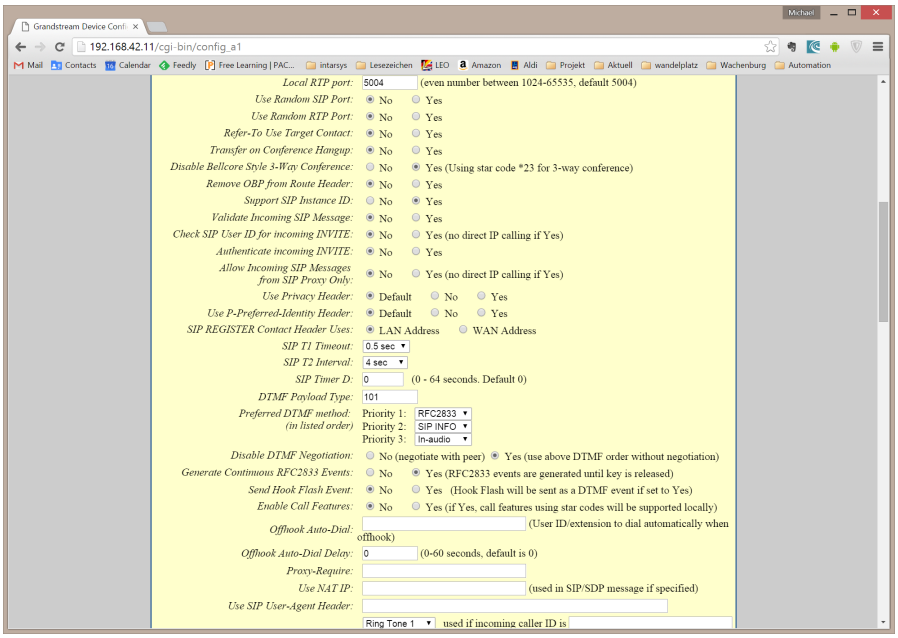


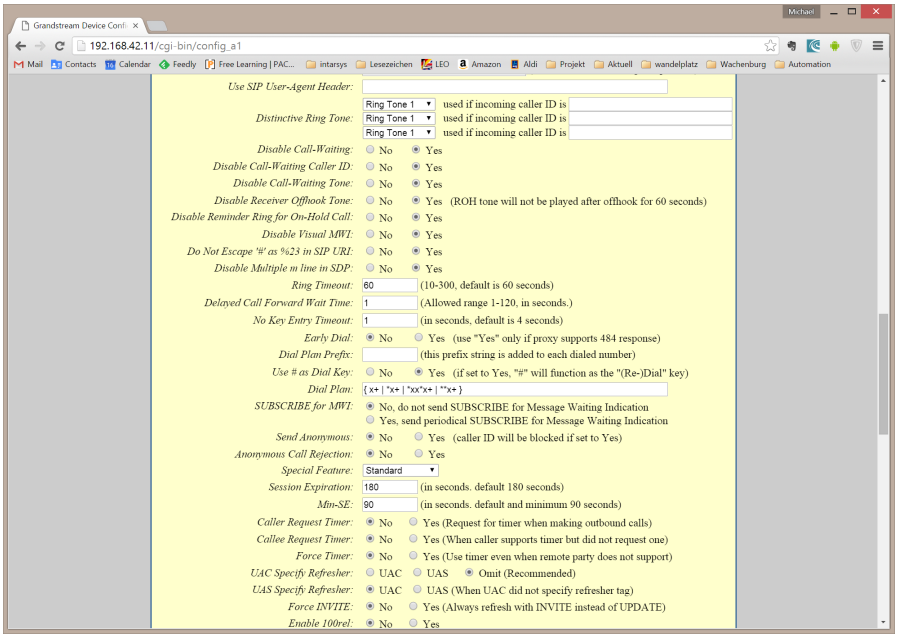


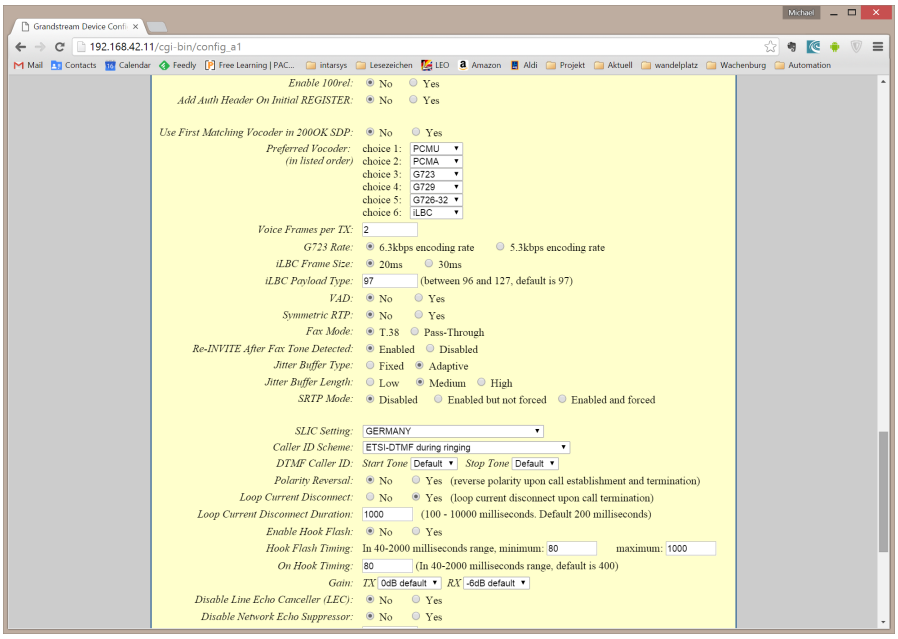
The call progress tone generator settings were trial and error and I really did not get it as smooth as I thought it should be: The analog outdoor phone for example was not able to detect the tone when hung up.

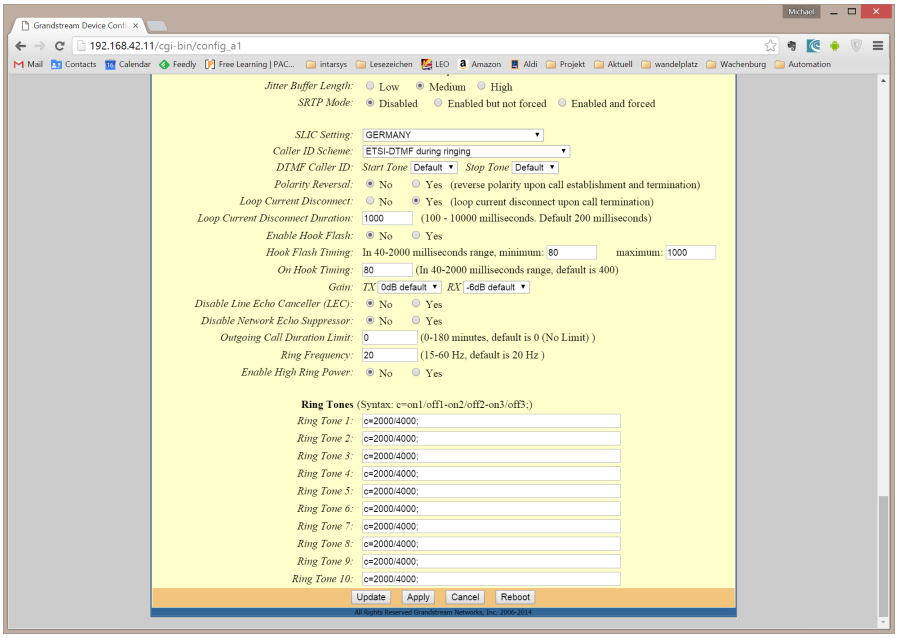


The SIP configuration is mostly standard.




Grandstream Device Confi
192.168.42.11/cgi-bin/config_a1
Use SIP User-Agent Header:
Distinctive Ring Tone:
Ring Tone 1 used if incoming caller ID is
Disable Call-Waiting:
Disable Call-Waiting Caller ID:
Disable Call-Waiting Tone:
Disable Receiver Offhook Tone: (ROH tone will not be played after offhook for 60 seconds)
Disable Reminder Ring for On-Hold Call:
Disable Visual MWI:
Do Not Escape '#' as %23 in SIP URI:
Disable Multiple m line in SDP:
Ring Timeout: 60 (10-300, default is 60 seconds)
Delayed Call Forward Wait Time: 1 (Allowed range 1-120, in seconds.)
No Key Entry Timeout: 1 (in seconds, default is 4 seconds)
Early Dial: Yes (use "Yes" only if proxy supports 484 response)
Dial Plan Prefix: (this prefix string is added to each dialed number)
Use # as Dial Key: Yes (if set to Yes, "#" will function as the "(Re-)Dial" key)
Dial Plan: { x+ | *x+ | *xx*x+ | **x+ }
SUBSCRIBE for MWI: No, do not send SUBSCRIBE for Message Waiting Indication
Yes, send periodical SUBSCRIBE for Message Waiting Indication
Send Anonymous: Yes (caller ID will be blocked if set to Yes)
Anonymous Call Rejection:
Special Feature: Standard
Session Expiration: 180 (in seconds. default 180 seconds)
Min-SE: 90 (in seconds. default and minimum 90 seconds)
Caller Request Timer: Yes (Request for timer when making outbound calls)
Callee Request Timer: Yes (When caller supports timer but did not request one)
Force Timer: Yes (Use timer even when remote party does not support)
UAC Specify Refresher: UAC UAS Omit (Recommended)
UAS Specify Refresher: UAC UAS (When UAC did not specify refresher tag)
Force INVITE: Yes (Always refresh with INVITE instead of UPDATE)
Enable 100rel:


Grandstream Device Confi
192.168.42.11/cgi-bin/config_a1
Enable 100rel:
Add Auth Header On Initial REGISTER:
Use First Matching Vocoder in 200OK SDP:
Preferred Vocoder: (in listed order)
choice 1: PCMU
choice 2: PCMA
choice 3: G723
choice 4: G729
choice 5: G726-32
choice 6: iLBC
Voice Frames per TX: 2
G723 Rate: 6.3kbps encoding rate 5.3kbps encoding rate
iLBC Frame Size: 20ms 30ms
iLBC Payload Type: 97 (between 96 and 127, default is 97)
VAD:
Symmetric RTP:
Fax Mode: T.38 Pass-Through
Re-INVITE After Fax Tone Detected: Enabled Disabled
Jitter Buffer Type: Fixed Adaptive
Jitter Buffer Length: Low Medium High
SRTP Mode: Disabled Enabled but not forced Enabled and forced
SLIC Setting: GERMANY
Caller ID Scheme: ETSI-DTMF during ringing
DTMF Caller ID: Start Tone Default Stop Tone Default
Polarity Reversal: Yes (reverse polarity upon call establishment and termination)
Loop Current Disconnect: Yes (loop current disconnect upon call termination)
Loop Current Disconnect Duration: 1000 (100 - 10000 milliseconds. Default 200 milliseconds)
Enable Hook Flash:
Hook Flash Timing: In 40-2000 milliseconds range, minimum: 80 maximum: 1000
On Hook Timing: 80 (In 40-2000 milliseconds range, default is 400)
Gain: TX 0dB default RX -6dB default
Disable Line Echo Canceller (LEC):
Disable Network Echo Suppressor:



With most of the settings you may play around until you're satisfied with voice quality (along with the settings of the outside phone). You can easily see that the installation is not really a corner case for the Grandstream.

One thing that was important with me: To get the Auerswald reliably "hooked" after calling, I had to enable "Look current Disconnect".

CHAPTER

THREE

PART II

Now we have some island components - already with nice functionality, but we can do better. Here are the basic steps to provide interoperability.

3.1 Network setup

3.1.1 Static IP

To set up a static IP address, you have to edit the /etc/network/interfaces in your Pi. A typical configuration would look like this, important settings are bold.

```
# Ethernet
auto etho
allow-hotplug eth0
iface eth0 inet static
address 192.168.42.210
netmask 255.255.255.0
gateway 192.168.42.1
dns-nameservers 8.8.8.8
```

For sure you have to take care to use an address that is unique in your network and that does not lie in the dynamic address range of your DHCP server.

If your Pi should be able to resolve logical host names (as "google.com"), you must add a "dns-nameservers" entry.

It may be necessary to disable the DHCP daemon on your Pi to make things work

```
sudo service dhcpcd stop
sudo update-rc.d -f dhcpcd remove
```

To activate your new settings, you must restart the network

```
sudo /etc/init.d/networking restart
```

3.1.2 Dynamic IP

If you want to run a Pi with dynamic IP address, you can use the following configuration in /etc/network/interfaces

```
# Ethernet
auto etho
allow-hotplug eth0
iface eth0 inet dhcp
```

You should assure that somewhere in the network a DHCP server is up and running...

3.1.3 IP address setup

To administrate your network, it is important to have convenient access to the components. While the system has a DHCP server, I found it the easiest way to give each of the components a static IP.

Hostname	IP	services
wbbase	192.168.42.1	isc-dhcp-server, asterisk, samba
wbf1	192.168.42.210	hmland, homegear, openhab
wbf2	192.168.42.220	hmland, homegear, openhab
wbf3	192.168.42.230	hmland, homegear, openhab
wbgo100	192.168.42.10	VOIP Client(s)
wbht701	192.168.42.11	VOIP Adapter
wbpellematic	192.168.42.2	Heating
DHCP Clients	192.168.42.1xx	... all the rest

3.1.4 Hostname and hosts

If you want your Pi to have a more meaningful hostname (or as in this case you don't want your Pi's to have all the same name), you should change /etc/hostname

```
wbf1
```

This information is simply required on every Linux and used in many places where the identifying the host is required (mail,...).

In addition you should edit the /etc/hosts, the poor mans DNS, so that accessing the component using its hostname will resolve to the correct component. And if you are already about to edit the file, you should add all other components in your network as we do not set up a local DNS. This allows to access the components via their logical names instead of the IP addresses.

Example for /etc/hosts

```
127.0.0.1 localhost
127.0.1.1 raspberry

192.168.42.1 wbbase
192.168.42.210 wbf1
192.168.42.220 wbf2
192.168.42.230 wbf3
192.168.42.2 wbpellematic
192.168.42.10 wbgo100
192.168.42.11 wbht701
```

3.1.5 Provide DHCP services

If you want one of your Pi's to provide a DHCP server (mmmh, your gateway maybe..) you should install and configure the DHCP server.

For me, using the ISC server simply worked.

```
sudo apt-get install isc-dhcp-server
```

An error message at this point is OK, as the default configuration is not working yet. You must edit the configuration at /etc/dhcp/dhcpd.conf

This is what is working for me - the important pieces are bold. You have to mention the gateway ("option routers"), the DNS servers ("option domain-name-servers", here a Google server), a flag that this is the one and only valid DHCP source ("authoritative") and then the definition of the subrange of addresses that are available for dynamic clients.

```
# The ddns-updates-style parameter controls whether or not the server will
# attempt to do a DNS update when a lease is confirmed. We default to the
# behavior of the version 2 packages ('none', since DHCP v2 didn't
# have support for DDNS.)
ddns-update-style none;

# option definitions common to all supported networks...
option domain-name "wb.local";

default-lease-time 86400; # 24 hours
max-lease-time 172800; # 48 hours
```

```
option routers 192.168.42.1;
option domain-name-servers 8.8.8.8;
authoritative;

subnet 192.168.42.0 netmask 255.255.255.0 {
        range 192.168.42.100 192.168.42.199;
}
```

Restart the DHCP server after configuration changes

```
sudo service isc-dhcp-server restart
```

3.2 Mass Storage

Storing data on the SD card has some serious drawbacks

- Many operations will wear out the SD card
- SD card capacity is quite restricted
- Each unit has its own card, central storage is not available

The solution is to add some mass storage as a service.

3.2.1 USB Hub

While you may get away with connecting smaller gadgets directly to the Pi's USB, sucking all of the power it has to give and pushing it to the limit of getting unstable (well, maybe this sounds a little dramatic...) - attaching a USB powered disk is definitely **not** what you want to do.

I bought a small USB hub that can neatly be hidden beneath the Pi in the cabinet, in this case a i-tec 4-Port USB 2.0 component.

3.2.2 USB Disk

Checking the available disks for the Pi can be done using

```
sudo fdisk -l
```

You will see something similar to

```
Disk /dev/mmcblk0: 7948 MB, 7948206080 bytes
4 heads, 16 sectors/track, 242560 cylinders, total 15523840 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xa6202af7
Device Boot Start End Blocks Id System
/dev/mmcblk0p1 8192 122879 57344 c W95 FAT32 (LBA)
/dev/mmcblk0p2 122880 15523839 7700480 83 Linux
```

This is the SD card "disk" and its partitioning.

Plugging in a USB disk will change the matter - you will see an additional disk now:

```
Disk /dev/sda: 500.1 GB, 500107861504 bytes
255 heads, 63 sectors/track, 60801 cylinders, total 976773167 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x386abe69
Device Boot Start End Blocks Id System
/dev/sda1 2048 976768064 488383008+ 7 HPFS/NTFS/exFAT
```

This one is preformatted NTFS on a Windows system.

To get this far you must have installed the NTFS support on the PI. This is easy, simply

```
sudo apt-get -y install ntfs-3g
```

Now we need a "mount point" on the Pi, this is a location in the "main file system" where the USB disk is "mounted", i.e. the USB disk filesystem is seen as if it is underneath this mount point.

```
sudo mkdir /media/wbdisk
```

Furthermore we need the UUID of the disk, as the name device name "/dev/sda1" is assigned dynamically and may change in other circumstances. To get the UUID enter

```
sudo blkid
```

This will show you something like

```
/dev/mmcblk0p1: SEC_TYPE="msdos" LABEL="boot" UUID="15CD-3B79" TYPE="vfat"
/dev/mmcblk0p2: UUID="13d368bf-6dbf-4751-8ba1-88bed06bef77" TYPE="ext4"
/dev/sda1: LABEL="wbdisk" UUID="340CC0590CC017AE" TYPE="ntfs"
```

Now we have everything we need to mount the disk for further use. Manually you can mount the disk using

```
sudo mount -t ntfs-3g -o utf8,noatime /dev/sda1 /media/wbdisk
```

and unmount

```
sudo umount /media/wbdisk
```

As we don't want to do this anytime we launch the Pi, we make this a permanent feature. To do this edit the /etc/fstab file and add a line like this. Do not forget to use your correct UUID and file system driver.

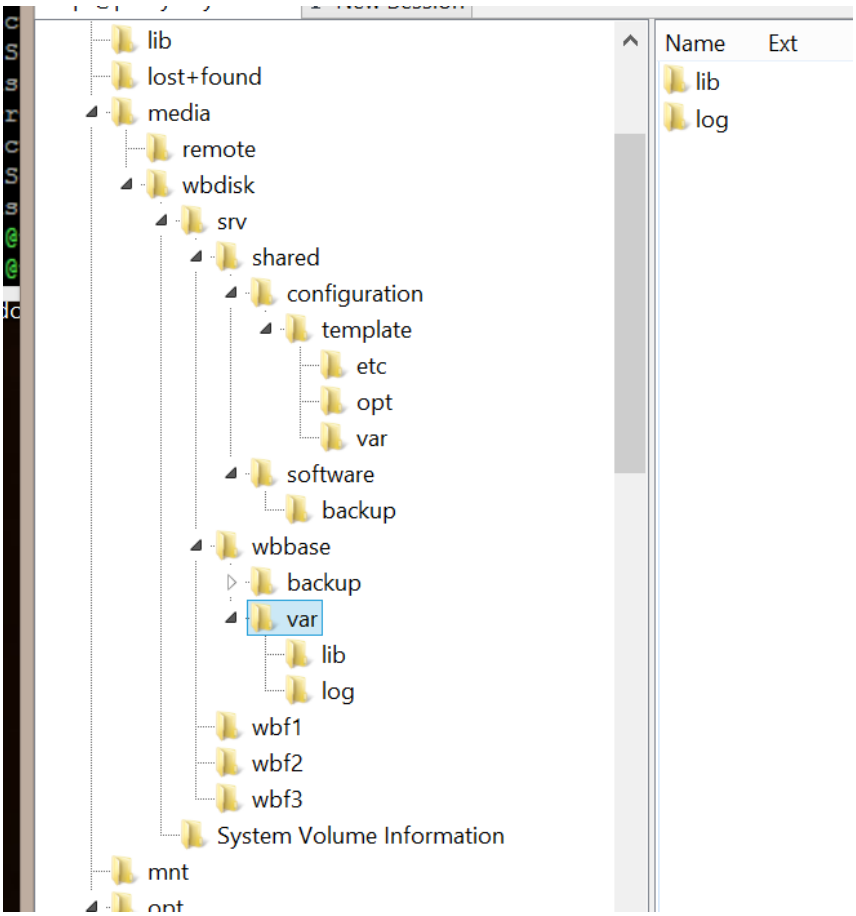
```
# here's the USB disk
UUID=340CC0590CC017AE /media/wbdisk/ ntfs-3g utf8,noatime 0
```

3.2.3 Directory layout

Now lets design the layout in the USB disk itself.

We already know that storage devices are mounted at /media and that we called our device "wbdisk".

On this device I'd like to mark a section as reserved for server storage, "srv". Below we make a directory for "shared" data and one for each active participant, this makes up for "wbf1", "wbf2", "wbf3" and "wbbase".



If you make extensions to this layout, ensure that you create directories upfront that you want to reference from mounts, they are not created automatically for you.

In the client directories i have created a layout mirroring the original file system layout whenever I wanted to switch from the original data location to a server based one.

3.2.4 NFS

There must be a reason we added a USB disk and here it is: we add a network file system (NFS) and have plenty of space for writing logs, database and whatsoever...

So far as theory is concerned. NFS may be an easy to install, performant solution, but it's an island. For me ease of access from other platforms (ok, lets give it a name: Windows) is crucial and that's where NFS simply fails.

I decided quickly (after having nearly havocked my Windows installation in the attempt to access the shares) to migrate to Samba.

3.2.5 Samba

Misc

https://wiki.samba.org/index.php/Mounting_samba_shares_from_a_unix_client

Pi server

Working with NFS when we also want to include a Windows PC in the network is a pain. For the sake of interoperability we switch to Samba here.

Install

```
apt-get install samba samba-common-bin
```

Configuration

```
sudo nano /etc/samba/smb.conf
```

Here we will setup a share for every client and a shared one, using the already defined directories on the mounted USB disk. As you can see, we do not care much about privacy at this point - this may or may not be appropriate for you.

At the end of the file, below all other sections add:

```
[wbf1]
path = /media/wbdisk/srv/wbf1
writeable = yes
guest ok = yes

[wbf2]
path = /media/wbdisk/srv/wbf2
writeable = yes
guest ok = yes

[wbf3]
path = /media/wbdisk/srv/wbf3
writeable = yes
guest ok = yes
```

```
[wbbase]
path = /media/wbdisk/srv/wbbase
writeable = yes
guest ok = yes

[shared]
path = /media/wbdisk/srv/shared
writeable = yes
guest ok = yes
```

Most probably, at least to support hardlinks, you will want to add this in the [global] section. Be sure to add this near the beginning of the configuration file.

```
unix extensions = yes
follow symlinks = yes
```

After a change to the configuration, you must restart the samba service.

Pi client

To access a Samba share, the client must install

```
sudo apt-get install cifs-utils
```

As this is available by default on the standard Raspbian, you should be able to omit this.

Now you can mount the network share on your client.

We create a mount point in each client at

```
mkdir /media/remote/wbdisk/private
mkdir /media/remote/wbdisk/shared
```

The mount point must belong to the root user. It is a very clever idea to name the mount points locally using the same name (they will be mapped to a different server location later). This gives you the ability to reuse the same scripts on every component (e.g. backup scripts) without any change.

Now you can mount the shares, for example for "wbf1" with the commands below. This will work only if you have guests allowed on the server. Omitting "-o guest" will nag you with a password prompt.

Note how each component will map its private share to a dedicated directory on the server.

mount -t cifs //wbbase/shared /media/remote/wbdisk/shared -o guest

mount -t cifs //wbbase/wbf1 /media/remote/wbdisk/private -o guest

Here we use the network aliases defined in /etc/hosts, you can use the IP address if you want.

to manually unmount you can call

```
umount /media/remote/wbdisk/shared
umount /media/remote/wbdisk/private
```

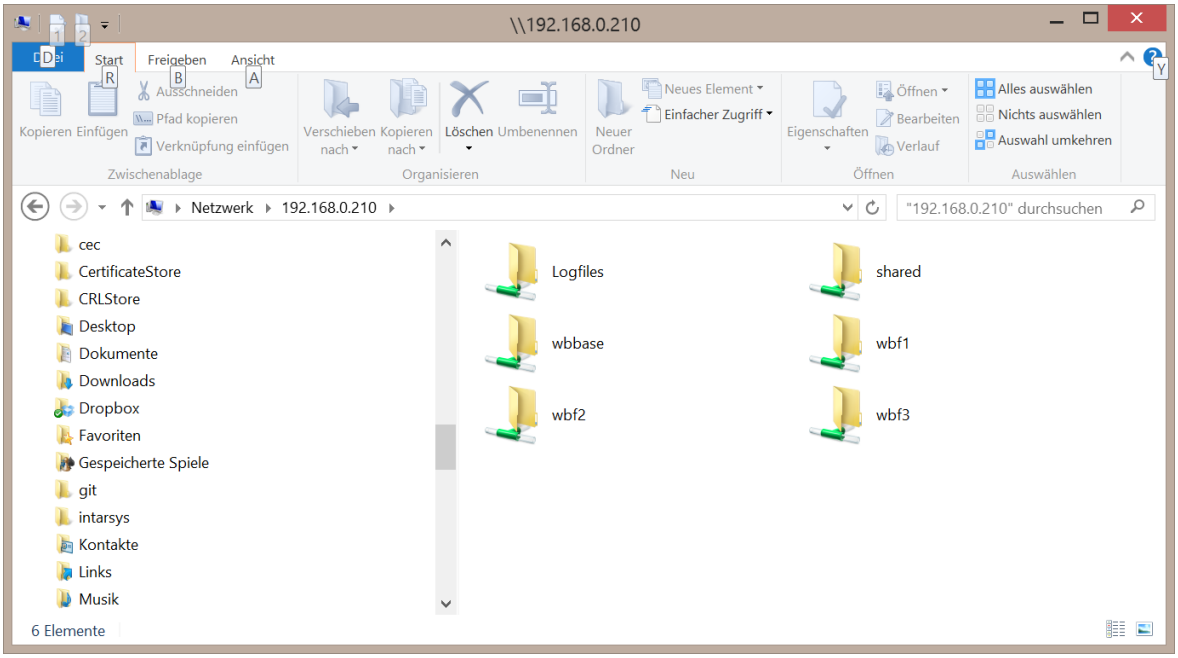
Eventually we want to make this bindings permanent, so we change the /etc/fstab and add

```
//wbbase/shared /media/remote/wbdisk/shared cifs auto,guest,_netdev 0 0
//wbbase/wbf1 /media/remote/wbdisk/private cifs auto,guest,_netdev 0 0
```

Windows client

Well, a windows client should be able to access the share without effort, you only have to use the IP address as in the standard installation the Pi names are not propagated to Windows (this can be done, though).

Open an explorer and type "\\<address>" in the address field (don't forget the backslashes".



More tweaks

Samba provides more services than file sharing only. If it happens that you don't need the print services and you encounter messages like this in you log files

```
[2016/08/19 04:44:39.680867,  0] printing/print_cups.c:487(cups_async_callback)
  failed to retrieve printer list: NT_STATUS_UNSUCCESSFUL
```

you may want to disable printing completely. Add these settings in /etc/samba/smb.conf

```
load printers = no
printing = bsd
printcap name = /dev/null
```

Restart with

```
sudo service samba restart
```

Now your logs should be clean.

3.3 Backup

3.3.1 So why?

Even if it seems to be unnecessary (well, it isn't) - consider setting up a backup.

In the beginning my intention was to setup each station and then keep copies of the respective SD cards. On failure I'd simply plug in the SD copy and things are all right. So while in theory this was a good idea, in practice it's not really feasible.

Already after 5 days of "production" the image of the respective clients was outdated because of little tweaks in the settings, added operating system features and so on.

Having a live backup possibility is the best you can do (IMHO). So after we already decided to go for a fully networked system, added an USB disk and installed Samba, why wouldn't we provide a simple backup... Simply starting the script after some non trivial changes is a very simple task and gives you a quite good feeling when going to sleep...

3.3.2 rsync

rsync is a classical tool for synchronizing two file system hierarchies. It is widespread and has a very efficient implementation. Because of its many options it may seem intimidating at first, but you have come this far, so give it a try.

First, I created a directory at /opt/backup for all backup related scripts.

Then we create the following shell script. While there exist much more sophisticated scripts and full fledged programs, it will do the job. The skeleton to this script and the parameters used are borrowed from a plethora of rsync related blog entries, blame all errors on me and all fame on them.

This is my /opt/backup/backup.sh (don't you forget to make it executable).

```
#!/bin/sh
SOURCE=/
DEST=/media/remote/wbdisk/private/backup/
TODAY=$(date +%Y-%m-%d)

rsync -avqRH -stats --delete-during --exclude-from=/opt/backup/exclude.txt \\
  "${SOURCE}" "${DEST}${TODAY}/" --link-dest="${DEST}last/" > \\
  /var/log/rsync.log
ln -nsf "${DEST}${TODAY}" "${DEST}last"

exit 0
```

First, we have some variable definitions to ease script changes. With

```
SOURCE=/
```

we backup everything starting from root (exceptions see later).

```
DEST=/media/remote/wbdisk/private/backup/
```

The destination is our private share on the Samba server. Notice that you have to create the target "backup" directory by hand.

```
TODAY=$(date +%Y-%m-%d)
```

This is a helper variable that holds a date string with year, month and day. We will use this for incremental backups.

```
rsync -aqRH
```

Option	Description
-a	combines -rlptgoD
-r	Copy subdirectories
-l	Copy symbolic links
-p	keep permissions
-t	keep timestamps
-g	keep group settings
-o	keep owner settings
-D	keep device files
-q	suppress non error messages
-R	use relative path names
-H	preserve hard links

```
rsync ... -stats
```

Create statistical output at the end of the backup

```
rsync ... --delete-during
```

I have a incremental backup where I delete files in the destination when they are deleted in the source. "delete-during" keeps time and memory consumption low.

```
rsync ... --exclude-from=/opt/backup/exclude.txt \\
```

Now this is important: we create a file with a list of exclusions I do not want to be backed up, starting from my SOURCE.

For me this is /opt/backup/exclude.txt

```
/proc/*
/sys/*
/dev/*
/boot/*
/tmp/*
/run/*
/mnt/*
/media/*
/var/log/*
/var/run/*
/var/lock/*
```

Most (all?) dynamic or transient parts of the system are ignored.

```
rsync ... "${SOURCE}" "${DEST}${TODAY}/" --link-dest="${DEST}last/"
```

This is the copy itself, from SOURCE to DEST/TODAY, so we create a new directory every day we create a backup. Together with the symbolic link feature "âĂŤlink-dest" we ensure that files that are unchanged exist only once in the target. As this needs the underlying "symbolic link" feature of the file system, be sure your target does support this (e.g. by activating it in your samba configuration).

```
rsync ... > /var/log/rsync.log
```

For everything we have done so far, a detailed log file is written.

After rsync is ready we create a new symbolic link to the new backup named "last". This is in turn the reference for the next incremental backup.

```
ln -nsf "${DEST}${TODAY}" "${DEST}last"
```

Now you can start the backup manually or add it to crontab. I decided against a scheduled backup as I do not expect extensive working on the configuration or content.

CHAPTER

FOUR

PART III

We have a basic system up and running. Now we can start with the bells and whistles that will improve the design and bring the project from lab to real life.

4.1 Access Point

wbf1, wbf2 and wbf3 should provide Access Point features. The eth0 and wlan0 interfaces need to be bridged only.

This is a very special chapter and not everything I did here is really understood.

First, I made the mistake of getting one of the much praised Edimax WLAN sticks (EDIMAX-7612UAn V2). While these are plug and play with regard to WLAN client mode building an access point is not. The problem with them is that they need a driver that is not available by default for the "hostapd" package.

- Switch hostapd
- Configure hostapd
- Switch of power saving
- Run hostapd as daemon
- Install dhcp server
- Configure dhcp server

4.1.1 /etc/network/interfaces

The following network configuration describes two network interfaces, eth0 and wlan0 that "bridge" the traffic between them

```
##############################
# The loopback network interface
auto lo
iface lo inet loopback

##############################
# The cable network interface
auto eth0
allow-hotplug eth0
iface eth0 inet manual

########################
# The WLAN interface
auto wlan0
allow-hotplug wlan0
iface wlan0 inet manual

##############################
# The bridges
#
auto br0
iface br0 inet static
address 192.168.42.1
netmask 255.255.255.0

bridge_fd 0
bridge_stp off
bridge_ports eth0 wlan0
```

4.1.2 Disable power saving

If you want your access point to be available reliably, you should also disable power management for the WLAN.

```
sudo nano /etc/modprobe.d/8192cu.conf
```

You should enter the following content

```
# Disable power management
options 8192cu rtw_power_mgnt=0 rtw_enusbss=0
```

4.1.3 Switch hostapd

"hostapd" is a piece of software that allows your computer to act as an access point. The only problem is: the driver it relies on is not available for the chip on the Edimax wlan component. Now you have two options: use another WLAN component (providing a NL80211 driver) or use another hostapd package. Using forum KnowHow is tried the second approach.

There are different tutorials regarding how to work around the hostapd issue

[hostapd original] http://www.tacticalcode.de/2013/02/raspberry-pi-als-accesspoint-oder-wlan-bridge.html

[hostapd 2.0] http://www.jenssegers.be/43/Realtek-RTL8188-based-access-point-on-Raspberry-Pi%29

[hostapd 2.2] http://lispm.blogspot.de/2015/03/edimax-ew-7811un-als-wlan-access-point.html

Well, first I tried the original Pi configuration - which **can** not work because of the "nl80211" driver needed and not available for the Realtek chip.

Then I looked around and found I need to build the hostapd from scratch from Realtek sources (that are quite outdated). This failed for me, I still don't know why.

Some helping hand offered a prebuilt version (still hostapd v0.8). I installed it and really was lucky, hostapd started... but this time I could not connect to the WLAN it offered, even so I saw the SSID in my clients.

The next step was to update to a newer precompiled version, v2.2. This time when starting, hostapd terminated with a failure (that even was not identifiable in the log, at least for me). As a last try I played around with the hostapd settings and found out that after disabling 802.11n (HT capabilities) completely - the daemon finally started. I could connect to the Pi on my new WLAN...

Unfortunately I can not say what were the exact reasons for the different failures.

4.1.4 Run hostapd as daemon

Now we want the access point to be available after every reboot. You can achieve this by changing the hostapd configuration.

```
sudo nano /etc/default/hostapd
```

Remove the hashmark from the following line

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

4.1.5 Experience

While I do not really need AP features for my project, I wanted it to be available for later extensions (anything from tablet UIs to MQTT components).

But: I didn't get it to run reliably. After a lot of spurious failures where connection to the PI no longer was possible, I stopped this experiment and opted for stable production of the basic components.

Maybe using a plug and play AP component, added to the network and providing the house with WLAN may be a better idea anyway.

4.2 WAN access

This is one of the most interesting features, unfortunately this has been one of the worst experiences when setting up. But, one info at a time...

The HA system as it is planned is an "island", we do not have internet access as the network is physically and logically detached from any internet service that a resident may have. A full fledged internet account is oversized for this installation and as such much to expensive. The only remaining alternative is GPRS/UMTS - a very interesting one from the nerd point of view and as such the solution of choice.

The driving forces to install this feature are easy to guess

- **Internet access for easy system maintenance (local -> WAN)** For me this means update and installation of software packages. There are solutions for "offline maintenance", but they are too complex imho.
- **Push Notifications (local -> WAN)** For exceptional states the system should be able to proactively inform the operator (yes, that's me). The channels available are eMail, SMS and even phone calls, as we have set up Asterisk.
- **Remote monitoring and control of the HA system (WAN -> local)** This makes the HA much more operating friendly - I do not have to visit the building every time a user claims to have a problem, some scenarios can be handled in the home office now.

Time Travel

I must admit, I underestimated the use of the remote access feature. This has proven one of the most useful "whistle" I've added so far.

This results in the following technical requirements:

- Low bandwith

- Low volume
- High availability

This may be completely different for your installation, so carefully check the requirements.

The initial steps are easy enough:

4.2.1 Surfstick

To cling to a known name like "HUAWEI" seemed a good approach here, too. I found an unlocked HUAWEI E353 stick for a reasonable price.

Important: whatever stick you have, chances are good that your Pi can not provide enough current for the stick. Plan to buy a powered USB hub anyway.

Inserting the stick may be not so simple, too - in my case the central server is in the basement, connectivity is bad. Enclosing the stick within the distribution box will make things even worse.

In my special case, the stick is attached via a 5m USB cable and resides in the hallway on the ground floor.

Another possible issue: there are surfsticks around that are **not** modems, they identify themselves as complete USB Ethernet devices. Configuration is different for these.

4.2.2 SIM card

There's a myriad of providers and sub-providers, I can't advise any special of them. What you should take into account in your selection:

- Do I have network coverage where the stick is going to be installed
- **Do I need access from outside to the HA system** This is an important point. Some (if not all) providers and some plans completely disable external access (your stick/your Pi is not visible from the internet side of the world). This means that e.g. a DYNDNS account is not possible. For me this was not a point of concern, as (heavy) external access is not planned and (see chapters below) for special cases there are workarounds.
- **Data volume** You should make some assumptions for the data volume to be expected based on your "driving forces" defined above. For me a low volume account is more than enough.

4.2.3 Configuration

Here's where trouble begins.

There are a lot of tutorials around - but, at least for me, the information was not really enough to solve the problems that came up. One of the reasons for this is the very complex infrastructure when it comes to networking in linux (at least from my noob point of view). When it comes to "merging" information from different tutorials, be sure all of them base on the same initial networking tools. As far as my basic understanding goes, you will find the following approaches.

- Old fashioned networking is configured via /etc/network - modern approach is via NetworkManager
- Old fashioned WAN access is done via modems (wvdial) and pppd - modern approach is to use QMI
- You can have a wvdial first or pppd first approach
- There are "magic scripts" like the famous "sakis3g" that will fiddle around with the basic infrastructure behind the scenes.

I choose to go by "classic networking via modems, wvdial first", again having many information and well tested settings around. The steps from here seem quite clear from a high level view:

- The stick is inserted and recognized as a modem
- the wvdial software is instructed how to talk to the modem
- when wvdial has setup the modem and initiated a connection, control is handed to pppd

- pppd negotiates the session with the provider and sets up a new network adapter (ppp0).

Now lets look this up in detail.

4.2.4 Stick configuration

After inserting the stick, try to look it up via *lsusb*

```
pi@wbbase ~ $ lsusb
Bus 001 Device 002: ID 0424:9514 Standard Microsystems Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp.
Bus 001 Device 004: ID 1a40:0101 Terminus Technology Inc. 4-Port HUB
Bus 001 Device 007: ID 12d1:1506 Huawei... E398 LTE/UMTS/GSM Modem/Networkcard
```

With a little luck you find something like this and you're done. Your stick is found and recognized as a modem. You can see this on one hand in the name part or as the nerd that you are you recognize the **ID 12d1:1506.** The first part of the ID is a vendor specific number, here HUAWEI, the second is the device type, here a modem.

Many sticks are recognized as a mass storage device. I found a clever article on this with the following explanation:

"This is a clever default behavior on windows. On the mass storage part we have windows drivers that enable access to the stick, after access the stick is automatically switched to modem mode. Plug and play..."

On the Pi things are different. Here you have to switch the stick yourself. The software doing this is called "usb-modeswitch" (what a surprise). So, if you have only mass storage so far, install it..

```
sudo apt-get install usb-modeswitch
```

After installation and stick insertion you should finally see the modem variant. If not, bad luck. You should really have bought a stick that is already in the standard configuration or that does not need a switch at all. But not all hope is lost, you can add the information yourself - but this is beyond this chapter, look here http://wiki.ubuntuusers.de/USB_ModeSwitch/ for example.

When the stick is recognized as a modem, you should lookup the device name associated with it, you will need this later for the dialer or PPP connection. Having a look in the dmesg after inserting the stick you will find something like this.

```
[2934611.713325] usb 1-1.5.2: new high-speed USB device number 6 using dwc\_otg
[2934611.814306] usb 1-1.5.2: New USB device found, idVendor=12d1, idProduct=14fe
[2934611.814331] usb 1-1.5.2: New USB device: Mfr=2, Product=1, SerialNumber=0
```

```
[2934611.814349] usb 1-1.5.2: Product: HUAWEI Mobile
[2934611.814365] usb 1-1.5.2: Manufacturer: HUAWEI
[2934611.816789] usb-storage 1-1.5.2:1.0: USB Mass Storage device detected
[2934611.818962] scsi host1: usb-storage 1-1.5.2:1.0
[2934611.819840] usb-storage 1-1.5.2:1.1: USB Mass Storage device detected
[2934611.820235] scsi host2: usb-storage 1-1.5.2:1.1
[2934612.497513] usb 1-1.5.2: USB disconnect, device number 6
[2934618.113327] usb 1-1.5.2: new high-speed USB device number 7 using dwc\_otg
[2934618.214538] usb 1-1.5.2: New USB device found, idVendor=12d1, idProduct=1506
[2934618.214564] usb 1-1.5.2: New USB device strings: Mfr=3, Product=2 ...
[2934618.214582] usb 1-1.5.2: Product: HUAWEI Mobile
[2934618.214597] usb 1-1.5.2: Manufacturer: HUAWEI
[2934618.218759] usb-storage 1-1.5.2:1.4: USB Mass Storage device detected
[2934618.219301] scsi host3: usb-storage 1-1.5.2:1.4
[2934618.220157] usb-storage 1-1.5.2:1.5: USB Mass Storage device detected
[2934618.220558] scsi host4: usb-storage 1-1.5.2:1.5
[2934618.265855] usbcore: registered new interface driver usbserial
[2934618.266054] usbcore: registered new interface driver usbserial\_generic
[2934618.266242] usbserial: USB Serial support registered for generic
[2934618.269213] usbcore: registered new interface driver cdc\_ncm
[2934618.275659] usbcore: registered new interface driver cdc\_wdm
[2934618.286789] usbcore: registered new interface driver option
[2934618.286983] usbserial: USB Serial support registered for GSM modem (1-port)
[2934618.287328] huawei\_cdc\_ncm 1-1.5.2:1.1: MAC-Address: 58:2c:80:13:92:63
[2934618.287357] huawei\_cdc\_ncm 1-1.5.2:1.1: setting rx\_max = 16384
[2934618.287755] huawei\_cdc\_ncm 1-1.5.2:1.1: setting tx\_max = 16384
[2934618.288436] huawei\_cdc\_ncm 1-1.5.2:1.1: cdc-wdm0: USB WDM device
[2934618.290042] huawei\_cdc\_ncm 1-1.5.2:1.1 wwan0: register 'huawei\_cdc\_ncm'␣
↪at \\usb-3f980000.usb-1.5.2, Huawei CDC NCM device, 58:2c:80:13:92:63
[2934618.290335] option 1-1.5.2:1.0: GSM modem (1-port) converter detected
[2934618.290402] usbcore: registered new interface driver huawei\_cdc\_ncm
[2934618.290895] usb 1-1.5.2: GSM modem (1-port) converter attached to ttyUSB0
[2934618.291214] option 1-1.5.2:1.2: GSM modem (1-port) converter detected
[2934618.292108] usb 1-1.5.2: GSM modem (1-port) converter attached to ttyUSB1
[2934618.292291] option 1-1.5.2:1.3: GSM modem (1-port) converter detected
[2934618.292824] usb 1-1.5.2: GSM modem (1-port) converter attached to ttyUSB2
[2934618.414538] huawei\_cdc\_ncm 1-1.5.2:1.1 wwan0: kevent 12 ..dropped
[2934619.214405] scsi 4:0:0:0: Direct-Access HUAWEI SD Storage \\ 2.31 PQ: 0 ANSI:
↪ 2
[2934619.215258] scsi 3:0:0:0: CD-ROM HUAWEI Mass Storage \\ 2.31 PQ: 0 ANSI: 2
```

4.2.5 wvdial

The next step in the old fashioned way is setting up wvdial.

```
sudo apt-get install wvdial
```

There are other alternatives to access the modem, but his one just works fine for me. If you don't like it, maybe have a look at http://www.sakis3g.com/.

wvdial is responsible for communicating with the modem hardware. After connection, work is handed over to pppd.

wvdial uses the configuration at /etc/wvdial.conf. It has also a builtin feature to create a basic default configuration for the currently detected modem. So, as a first step insert the stick and run

```
sudo wvdialconf /etc/wvdial.conf
```

No edit the /etc/wvdial.conf and add a section for the provider you want to dial up at the end. Here's my example

```
[Dialer Defaults]
Init1 = ATZ
Init2 = ATQ0 V1 E1 S0=0 &C1 &D2
Modem Type = Analog Modem
ISDN = 0
Baud = 9600

[Dialer pin]
Modem = /dev/gsmmodem
Init3 = AT+CFUN=1,1
Init4 = AT+CPIN="1234"

[Dialer myprovider]
Modem = /dev/gsmmodem
Init4 = AT+CGDCONT=1,"IP","internet.partner"
Phone = *99#
Username = " "
Password = " "
```

The first section was created initially by the wvdialconf call, I have just removed garbage.

The next section defines the things wvdial should do when I want to set the pin or talk to "unitymedia".

Parameter	Description
Modem	The device we are talking to. You can derive this from the messages that you get via dmesg after inserting the stick. Here you see that a ttyUSB0, ttyUSB1 and ttyUSB2 is automatically created. The interesting information is that usb-modeswitch additionally installs automatically a device link /dev/gsmmodem So unless this link to the correct tty* is not available, use /dev/gsmmodem. If for some reason this does not work (there seem to be strange conditions in the USB jungle causing this), just try ttyUSB0.
Init<n>	The init strings that are sent to the modem. The first section one unlocks the SIM - you can omit this if your SIM is not PIN protected. While in theory you can include this command within the dial sequence itself, for me it was not possible because a second attempt to authenticate the PIN (when already authenticated) resulted in a modem error. As the PIN is not reset until power off, it was not possible to bounce a connection. Setting the PIN within the wvdial init sequence is one of many options, you can also • remove the PIN from the card • set the pin via a pre-up script. This is a hook within the network configuration that allows you to call scripts before a network adapter is started. The second one identifies and initialized the APN. Setting the correct APN is crucial. Setting the wrong APN not only can lead to malfunction, but can select an endpoint that is not compatible to your data plan and lead to extreme costs!
Username	The username for authentication with your provider. If you need no username it's a little tricky: wvdial does not accept an empty entry. Just adding a dummy entry " " (including the blank) should work.
Password	The password for authentication with your provider. If you need no password, set to " " (including the blank).
Phone	The number to dial. In most cases this will be *99#. See your respective provider manuals.

Setting "Baud" is unnecessary, as the real network speed is negotiated on the PPP level later on.

Depending on your provider you may need to set still other parameters. See the wvdial man page for more.

Running (the "&" is important - the console is not available otherwise after calling)

```
sudo wvdial myprovider &
```

will hopefully result in

```
[ pi@wbbase ~ $ --> WvDial: Internet dialer version 1.61
--> Initializing modem.
--> Sending: ATZ
ATZ
OK
--> Sending: ATQ0 V1 E1 S0=0 &C1 &D2
ATQ0 V1 E1 S0=0 &C1 &D2
OK
--> Sending: AT+CGDCONT=1,"IP","<APN>"
AT+CGDCONT=1,"IP","<APN>"
OK
--> Modem initialized.
--> Sending: ATDT*99#
--> Waiting for carrier.
ATDT*99#
CONNECT
--> Carrier detected. Waiting for prompt.
--> Don't know what to do! Starting pppd and hoping for the best.
--> Starting pppd at Sun Jan 10 16:22:57 2016
--> Pid of pppd: 2808
--> Using interface ppp0
--> local IP address 10.148.172.74
--> remote IP address 10.64.64.64
--> primary DNS address 193.189.244.225
--> secondary DNS address 193.189.244.206
```

Don't worry if you have lines like

```
--> pppd: Ãř±ÃťvPÃŘ[01][08]Ãŕ[01][03]
--> pppd: Ãř±ÃťvPÃŘ[01][08]Ãŕ[01][03]
```

between, just ignore them.

If everything goes well, ifconfig will show after this a network interface ppp0.

```
ppp0 Link encap:Punkt-zu-Punkt-Verbindung
        inet Adresse:x.x.x.x P-z-P:x.x.x.x Maske:255.255.255.255
```

```
        UP PUNKTZUPUNKT RUNNING NOARP MULTICAST MTU:1440 Metrik:1
        RX packets:14 errors:0 dropped:0 overruns:0 frame:0
        TX packets:15 errors:0 dropped:0 overruns:0 carrier:0
        Kollisionen:0 SendewarteschlangenlÃČÂďnge:3
        RX bytes:458 (458.0 B) TX bytes:521 (521.0 B)
```

This interface is created by the pppd after getting control from wvdial and negotiating the parameters with the provider.

Do not forget to

```
sudo killall wvdial
```

to end the connection.

If you fail, you will get an error, look it up for further investigation. A typical error (well, hmm, my error) is "error 11" - the SIM is PIN protected. Read the next chapter.

```
--> Sending: AT+CGDCONT=1,"IP","internet.partner"
AT+CGDCONT=1,"IP","internet.partner"
+CME ERROR: 11
--> Bad init string.
```

4.2.6 SIM PIN

Keep the PIN

If you absolutely want to keep the SIM PIN you have to go the extra mile.

On one hand, I didn't manage adding the modem PIN command to the wvdial.conf. The second time the PIN authentication is executed the modem will return an error and wvdial will fail. As there is no such thing as an optional wvdial entry, you must enter the PIN separately. The best bet is here to add it in the /etc/network/interfaces as an pre-up hook script, saved in /opt/admin/setupPIN.sh.

```
#!/bin/bash
echo "AT+CPIN=1234\\n\\r" > /dev/gsmmodem
```

Then you add to the /etc/network/interfaces in the ppp0 section the last line. Before uping the adapter the PIN will be set.

```
auto ppp0
iface ppp0 inet wvdial
provider myprovider
pre-up /opt/admin/setPIN.sh
```

To ease manual PIN unlocking add this section to the wvdial.conf

```
[Dialer pin]
Modem = /dev/gsmmodem
Init1 = AT+CPIN="1234"
```

Now you can manually unlock with

```
sudo wvdial pin
```

Unlock the SIM

The easiest way is to put the SIM in a handy and interactively remove the PIN.

As an alternative you can use "minicom" and send

```
// check
AT+CPIN?
+CPIN: PIN
// unlock
AT+CPIN="<your PIN>"
OK
// remove PIN
AT+CLCK="SC", 0, "<your PIN>"
OK
```

4.2.7 pppd

PPP (Point to Point Protocol) is a high level protocol between peers (your Pi and the provider endpoint). It has mechanisms to negotiate connection parameters and the IP address.

At the heart of the protocol implementation is pppd - this is called after the modem is connected to the provider by wvdial. wvdial is more of a wrapper for pppd here.

You can configure pppd options when started via wvdial in the /etc/ppp/peers/wvdial config file.

Important options

- **defaultroute** Add a default route for this connection
- **replacedefaultroute** Option to "defaultroute", replace an already existing defaultroute
- **persist** Do not exit after a connection is terminated; instead try to reopen the connection.
- **usepeerdns** Use the DNS information from the connected server

- **holdoff 10** How many seconds to wait before re-initiating the connection

4.2.8 Network config

Now we integrate our stick in the standard network configuration.

/etc/network/interfaces

```
auto ppp0
iface ppp0 inet wvdial
provider myprovider
pre-up /bin/sleep 30
pre-up /opt/admin/setPIN.sh
```

This configuration launches the "myprovider" entry via wdial upon startup. Before starting the adapter, we wait 30 seconds, then the script setPIN.sh is executed.

Manual shutdown and startup

To manually startup ppp0

```
sudo ifup ppp0
```

An here the shutdown

```
sudo ifdown ppp0
```

The routing table after successful connection

Now lets have a look at the current routing information with "netstat -r"

```
Kernel-IP-Routentabelle
Ziel Router Genmask Flags MSS Fenster irtt Iface
default * 0.0.0.0 U 0 0 0 ppp0
10.64.64.64 * 255.255.255.255 UH 0 0 0 ppp0
192.168.42.0 * 255.255.255.0 U 0 0 0 eth0
```

As far as i understood the network mechanics this means that traffic to the network segment 192.18.42.* is routed via the Ethernet card, the rest is sent to the WAN interface. That's it.

Provide internet access for all other nodes

Now that our basement server is online, we'd like to share this access throughout our network. It will become a breeze to maintain and update the other network nodes, we will be able to access internet services (let our heating send mails, then...) - pretty much features regarding that this is none of an effort now.

Two things must happen to provide seamless internet access:

- Defining a default gateway for all nodes
- Support NATing the requests from other nodes on the default gateway

Setting the default gateway

For all other nodes in your network that should have internet access you must add the GSM enabled station as the default gateway, ensuring you have internet everywhere.

This is an additional entry in the /etc/network/interfaces

```
gateway 192.168.41.1
dns-nameservers 8.8.8.8
```

The routing table on the satellites should look like this.

```
Kernel-IP-Routentabelle
Ziel Router Genmask Flags Metric Ref Use Iface
default wbbase 0.0.0.0 UG 0 0 0 eth0
192.168.42.0 * 255.255.255.0 U 0 0 0 eth0
```

Configuring NAT on the default gateway

Now that the other clients have a default gateway, they simply send all IP packets with addresses that do not lie in the local network to the default gateway. And we must tell the gateway what it should do with all the IP packets it will receive - forward to its own default gateway at the internet provider.

First, we have to enable ip forwarding for the system at all. This is done by adding the following line to /etc/sysctl.conf

```
# Uncomment the next line to enable packet forwarding for IPv4
#net.ipv4.ip_forward=1
net.ipv4.ip_forward=1
```

Then we must add routing rules with the "iptables" command.

```
# add NAT masquerading to output via ppp0
sudo iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
# allow explicitly packet forwarding from eth0 to ppp0
sudo iptables -A FORWARD -i eth0 -o ppp0 -j ACCEPT
# allow packet forwarding from ppp0 to eth0 only if originated from here
sudo iptables -A FORWARD -i ppp0 -o eth0 -m state --state RELATED,ESTABLISHED -j␣
↪ACCEPT
```

These rules are transient, you have to reenter them after reboot. To make them permanent you should follow these steps

```
sudo iptables-save > ./iptables.dump
sudo mv iptables.dump /opt/admin/iptables.dump
```

This saves the current (all!) settings to a dump file. To restore these settings we create a script that will be executed every time the network comes up.

```
sudo nano /etc/network/if-pre-up.d/iptables-restore
```

This creates a script file "iptables-restore". Add the following content

```
#!/bin/bash
iptables-restore < /opt/admin/iptables.dump
exit 0
```

This will restore the content of the backup. In order to make it an executable script, you have to make it executable for sure.

```
sudo chmod 755 /etc/network/if-pre-up.d/iptables-restore
```

4.2.9 Keep alive

While ppd has builtin features to monitor the connection, there seem to be enough scenarios where you should have another check. The following script monitors the existence of an internet connection purely from a client point of view.

Edit and save as /opt/admin/internetMonitor.sh. As always, make the script file executable..

```
#!/bin/bash
#
# Restart network interfaces
# if ppp0 connection is down

PING="/bin/ping -q -c1 -W 10 -I ppp0"
HOST=8.8.8.8
```

```
${PING} ${HOST}
if [ $? -ne 0 ]; then
        echo "ppp0 is down, try reconnect..."
        /sbin/ifdown --force ppp0
        sleep 10
        /sbin/ifup --force ppp0
        sleep 10
fi
```

Use crontab to run this script on a regular base - you can lookup the chapter below to find more information on crontab itself.

```
sudo crontab -e
```

This line will run the script every 10 minutes

```
*/10 * * * * /opt/admin/internetMonitor.sh
```

4.2.10 Auto disconnect

Some providers disconnect your modem on a regular base (every 24h). It may be helpful to automatically restart the connection using a cron script.

4.2.11 Security

For some providers the Pi now gets a (temporary) public IP address. Be prepared to be visible from the outside world immediately. There are strange guys out there that scan for open *ssh* ports to login. With a "standard" Pi configuration with active *ssh*, user "pi" and password "raspberry" it's a matter of time until you fall prey to one of these - and this is only one of the many attack scenarios.

Securing a server in the internet is really out of scope for this book, only the most important tips:

- Disable all unneeded servers
- Change the default password to strong ones
- If you need *ssh*, use a public/private key scenario
- Set up a firewall on the Pi. The iptables rules from above are a start, but I don't know if its enough!

If you see an address in the private IP ranges (e.g. 10.*.*.* like mine), you are not visible from the "outside" (At least for the guys from outside your providers private network. But

the bad guys may be nearby...). This reduces the possibilities of an attack, but now you have serious challenges when you want to remote access your system. More on this later.

4.2.12 Issues

Modem not detected

Some bloggers have experienced timing problems, so after a reboot the device may not be available for the USB modeswitch. For me this was not a problem so I cannot give advice. These guys tried with a script that restart all devices.

For example (from https://nicovddussen.wordpress.com/2014/11/12/setting-up-your-raspberry-pi-to-work-with-a-3g-dongle/)

```
#!/bin/bash
sleep 1m
udevadm trigger
sleep 1m
wvdial
```

Network to fast

Another type of problem is the network is up before the device is provided by the kernel (different from above where modeswitch was not able to reconfigure the device). pppd will fail if the modem device is not available. Solution is in this case to wait for the device to appear.

The "wait" script

```
#!/bin/bash
#INTERFACE="/dev/$(head -1 /etc/ppp/options-mobile)"
INTERFACE="/dev/$1"
MAX_SECONDS_TIMEOUT=$2

dsec=$((${MAX_SECONDS_TIMEOUT} * 10))
for ((retry=0; retry < ${dsec}; retry++))
do
        if [ -c ${INTERFACE} ]; then
                echo "$0: OK existing required device ${INTERFACE} (in $((retry /␣
↪10)).$((100 * (retry % 10) / 10)) seconds)"
                logger "$0: OK existing required device ${INTERFACE} (in $((retry /
↪ 10)).$((100 * (retry % 10) / 10)) seconds)"
                break
        else
                sleep 0.1
```

```
        fi
done
if [ ! -c ${INTERFACE} ]; then
        echo "$0: ERROR timeout waiting for required device ${INTERFACE}"
        logger "$0: ERROR timeout waiting for required device ${INTERFACE}"
        exit 1
fi
```

The /etc/network/interfaces

```
auto ppp0
iface ppp0 inet wvdial
        provider vodafone
        pre-up /etc/ppp/wait-dialup-hardware gsmmodem 20
```

4.2.13 More ...

The following links have more detailed information on the highly complex (subtopic) of WAN networking.

https://www.debian.org/doc/manuals/debian-reference/ch05.en.html

http://www.tldp.org/HOWTO/PPP-HOWTO/index.html

4.3 Remote Control

One particularly important topic is remote control. At least in my project, the installation is headless and you have to work with a remote console (if you don't want to run through the house all day and connect your keyboard to another Pi).

So, even when not remote accessing from the internet, access from the LAN is highly recommended. The workhorse of all this is "*ssh*".

4.3.1 ssh

Install ssh

Before you can remote connect to your Pi, you must ensure that the *ssh* daemon is up and running, and correctly configured. In current distributions, *ssh* should be installed and only has to be activated in the Raspberry initial configuration menu. To restart the configuration use

```
sudo raspi-config
```

A manual install is not hard, though

```
sudo apt-get install ssh
```

Now you can start the *ssh* server with

```
sudo /etc/init.d/ssh start
```

It is important to add the *ssh* server to the autostart to have it available after reboot.

```
sudo update-rc.d ssh defaults
```

Basic usage

ssh is the single most important tool to control a standalone, console-less Pi.

You use it primarily to work on the commandline of the "remote" systems. To connect remotely to your Pi you type

```
ssh <host>
```

which will connect you to the target host using the same user as locally.

If you do this for the first time, a prompt will ask you to approve the connection to a still unknown machine. The "fingerprint" is then saved, so you won't be prompted again. Take the prompt for serious, especially if you think that you have established the connection before but suddenly the fingerprint has changed - someone may play dirty tricks on you.

Finally you will have to enter the password for the user on the remote machine.

Important options

Option	Description
-l (lowercase L)	The user on the target host
-p	The port on the target host

Public/Private key

The best way to use *ssh* is based on public/private key authentication instead of password based authentication. While the procedure seems awkward at first, you will get through it very fast.

First, we create a key pair on (each) Pi and each client that wants to access remotely.

```
ssh-keygen
```

If you answer all questions with a return, this will create a default 2048 bit RSA based key pair in the file /home/<user>/.ssh/id_rsa (which is the private key) and /home/<user>/.ssh/id_rsa.pub (the public key).

Now, whatever way you prefer to do so, you must add the public key you just created for the client to the list of authorized keys in the *ssh* configuration on the server. This will enable future passwordless logins of this "authorized" client on the server machine.

- lookup the .ssh directory in the home directory of the target user on the server (in this case /home/pi/.ssh). If it does not exist, create it (it **must** have the permission set to 0700, owner **must** be the user to which the directory belongs).

```
mkdir .ssh
chmod 0700 .ssh
```

- Here you open the file "authorized_keys" (or create it, permissions **must** be 0600, owner **must** be the user to which the directory belongs)

```
touch authorized_keys
chmod 0600 authorized_keys
```

and simply add the content of the public key file from the client that should be allowed to access to the end.

I've done this for every account in this matrix - this eases life considerably.

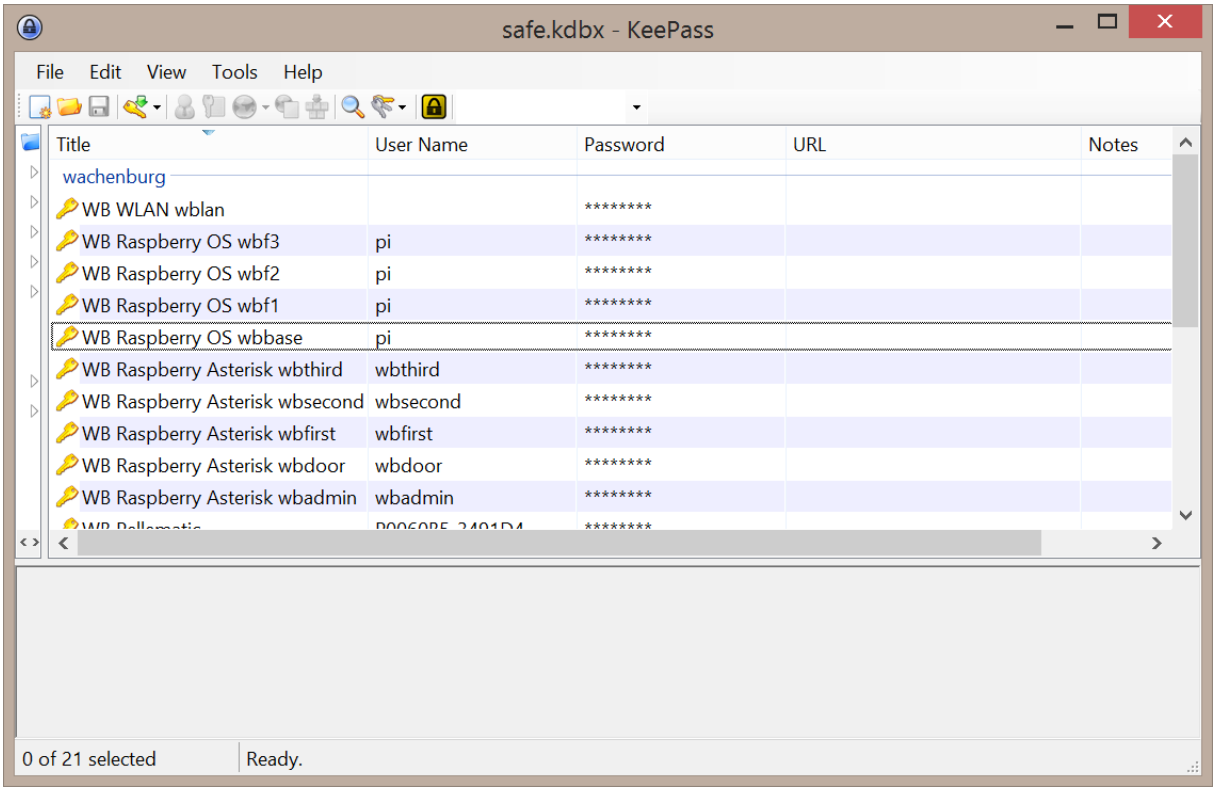
Local \ Remote	wbbase	wbf1	wbf2	wbf3
WinSCP	*	*	*	*
Connect Bot	*	*	*	*
wbbase		*	*	*

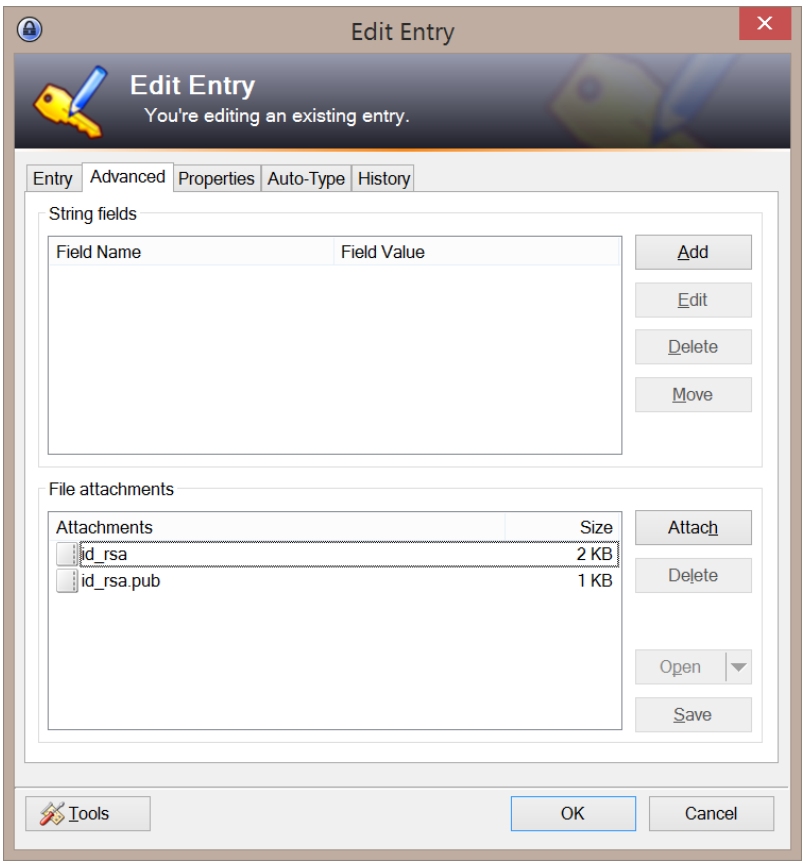
This way I can access any Pi in the house from a console (*ssh* client) on the machines listed on the left.

To ease later administration, I copy the public key to the samba share for each machine

```
cp /home/pi/.ssh/id_rsa.pub /media/remote/wbdisk/private/admin
```

I recommend to copy the private key, too, and save it a secure place like the Keepass vault. The current version has support for storing files along with an entry.


safe.kdbx - KeePass
File
Edit
View
Tools
Help
Title
User Name
Password
URL
Notes
wachenburg
WB WLAN wblan
WB Raspberry OS wbf3
pi
WB Raspberry OS wbf2
pi
WB Raspberry OS wbf1
pi
WB Raspberry OS wbbase
pi
WB Raspberry Asterisk wbthird
wbthird
WB Raspberry Asterisk wbsecond
wbsecond
WB Raspberry Asterisk wbfirst
wbfirst
WB Raspberry Asterisk wbdoor
wbdoor
WB Raspberry Asterisk wbadmin
wbadmin
0 of 21 selected
Ready.


Edit Entry
Edit Entry
You're editing an existing entry.
Entry
Advanced
Properties
Auto-Type
History
String fields
Field Name
Field Value
Add
Edit
Delete
Move
File attachments
Attachments
Size
id_rsa
2 KB
id_rsa.pub
1 KB
Attach
Delete
Open
Save
Tools
OK
Cancel

ssh daemon

On the server side, you may make some tweaks to the *ssh* configuration, but it's not really that important first. To configure the *ssh* daemon, you have to edit /etc/ssh/sshd_config

Its considered best practice to disable the *ssh* login for user "root".

```
PermitRootLogin no
```

Some guys recommend to change the *ssh* port from the default 22 to something an intruder might not try to connect to.

```
Port 222
```

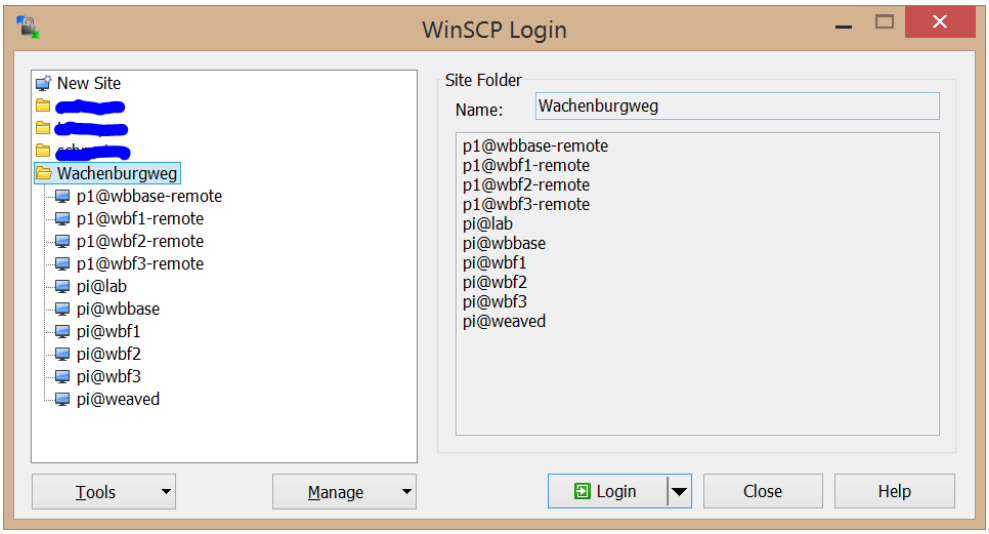
Upon saving you must restart the server.

```
sudo service ssh restart
```

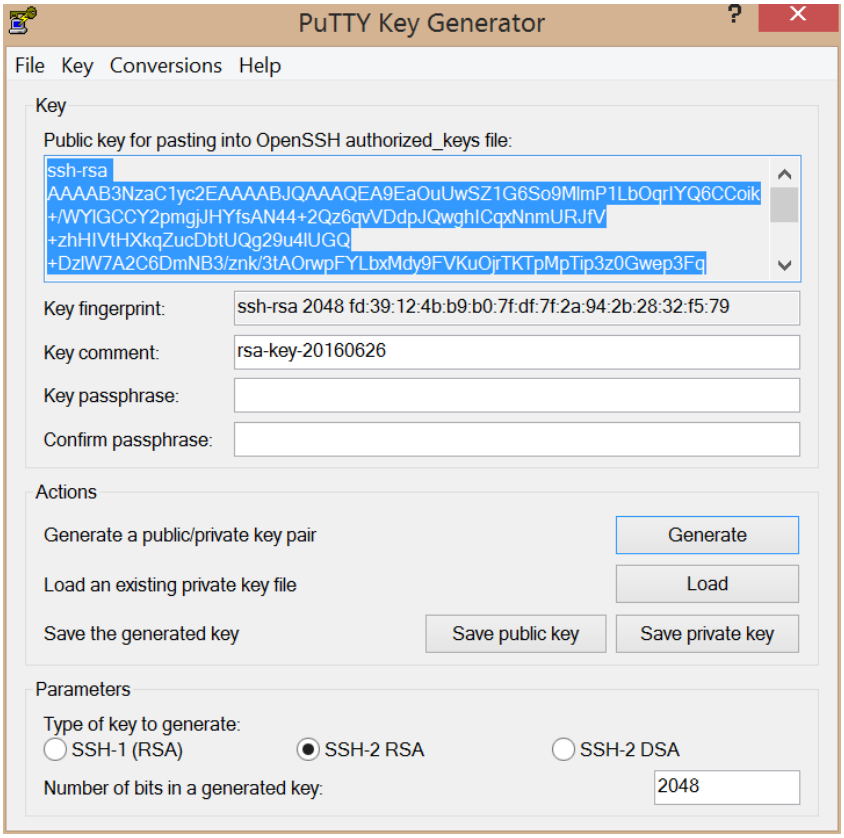
WinSCP

If you're using *ssh* and Windows I can recommend very much Win SCP, a remote "explorer" onto your machine. Combined with "putty", a remote console you have everything you need to efficiently manage your Pi's.

Included with WinSCP is a simple way to create a key pair for installation on your target machines.

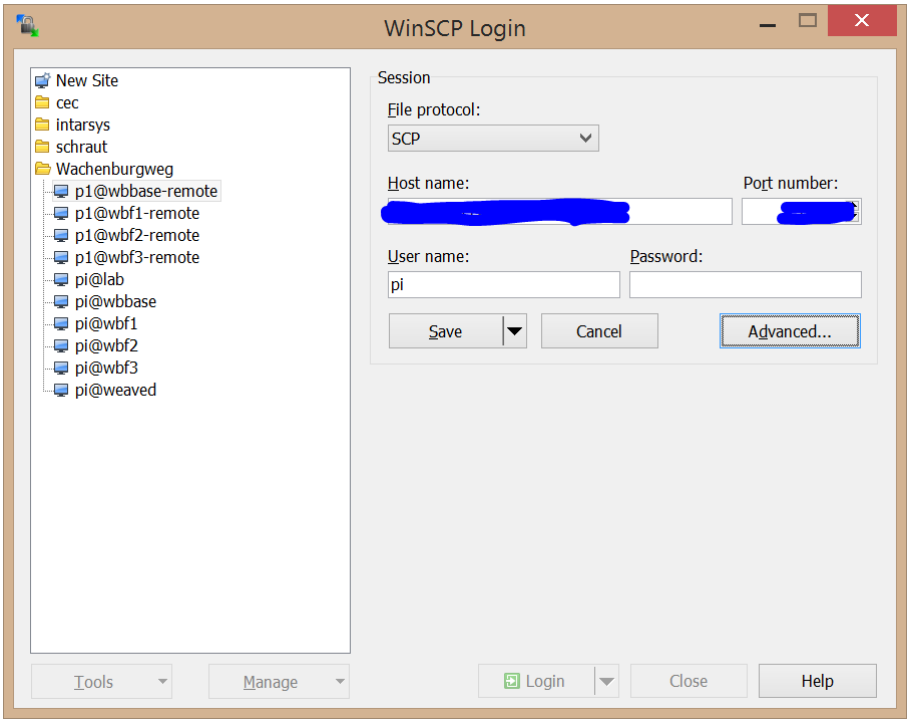


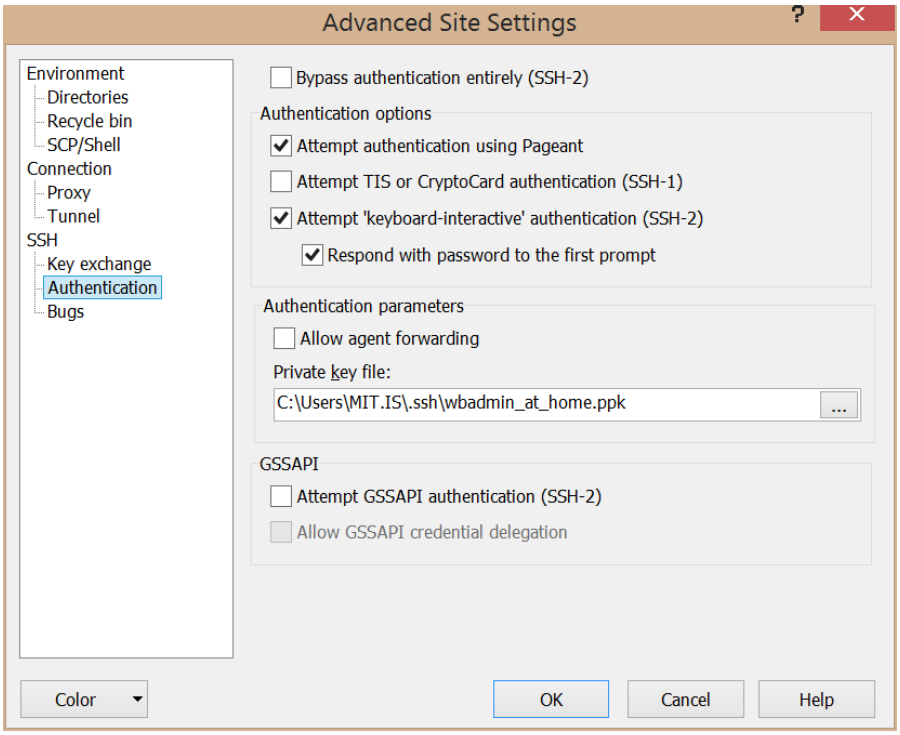
Using the button "Tools", you can start "Puttygen", a simple tools for key generation.



I'd recommend using 2048 bit keys. To protect your private key from unauthorized use you can enter a password here. You will then be prompted for this password before every use of the key. Press "Generate". Save the keys and transport the public keys to the respective servers.

You can configure a configuration by adding a "New Site". Under "Advanced..." you can enter your private key.





If everything is correct, the next attempt to login from WinSCP to the server where you copied the public key to will not ask for your password - you're simply connected.

ConnectBot

A great additional feature is provided by "ConnectBot", an *ssh* client for Android.

Following this easy steps you will get remote control in the palm of your hands

- Install ConnectBot
- Create a key with a meaningful name
- Copy the public key and send it via mail to an account where you can manage the Pi's authorized keys
- Add the public key to each /home/pi/.ssh/authorized_keys
- Connect via Android...

fail2ban

If your account is in any way visible in the internet you will most certainly encounter malicious login attempts. A very good and simple tool is "fail2ban" in addition to strong passwords and/or *ssh* via PPK.

Fail2ban will effectively slow down machines trying to break into your account using brute force.

4.3.2 DynDNS

The next problem you will encounter is the fact that the IP address of your gateway is not static - every time you provider considers a change to be useful, your client receives a new address.

In this case, you can setup "dynamic DNS". At a DNS service you can register a name (like "www.mydomain.de") along with an IP address, so any client can find your server using the name only. Dynamic DNS in addition has an API to change the IP address for the name, well, dynamically. Old bindings are not cached. There are a lot of such services around, the classical one being "dyndns.org". As they serve no longer accounts for free, you may have to look around for some other. I'm actually using "changeip.com".

In addition you will have to install a client that will call the service upon a change of the IP address. I cannot recommend any special as I have not installed any.

The problem is: In the case of our WAN connection, life is even more restrictive. Most surfsticks will never ever have a "real" IP address. They have addresses local in a private address range setup by the provider (such as "10.10.10.*").

You simply cannot reach this components from the internet! To circumvent this, you have to setup more delicate techniques.

4.3.3 Reverse tunneling

Intro

"Reverse tunneling" is the option in cases like our "invisible" WAN connection because of NAT (network address translation) or restrictive firewalls around our service. What you need is a gateway computer under your control, living in the internet.

As no one can see the target from outside, the target itself calls out to a well known server (your gateway) and initiates a forwarding service. When we want to talk with our service on the Pi, instead of calling in the network to the target service, we call the external gateways forwarding service. The call is transparently forwarded to the target, tunneled through the already existing connection initiated by the target. For the client the gateway is not really participating in the communication, so I use for example the same PPK credentials for the target like I'd use in a direct connection.

We have a look at two different "reverse scenarios".

Weaved

I stumbled over "weaved" by hazard. If you're not into heavy configuration or do not have a server at hand that is under your control and visible in the internet, this is a really useful service.

You can register an account for free and after installation of the Weaved client on your raspberry, you can setup a *ssh* remote console very easily.

I will not go into details here, but this is really fine for initial steps or under the above mentioned conditions. The service was stable - to be honest, more stable than my own reverse tunnel configuration in the beginning. Even today i wonder how they manage to resurrect availability after downs of all kind this fast. I do not know what the additional mile is they did go for it... and i admit i have still installed Weaved support, just in case...

ssh Tunneling

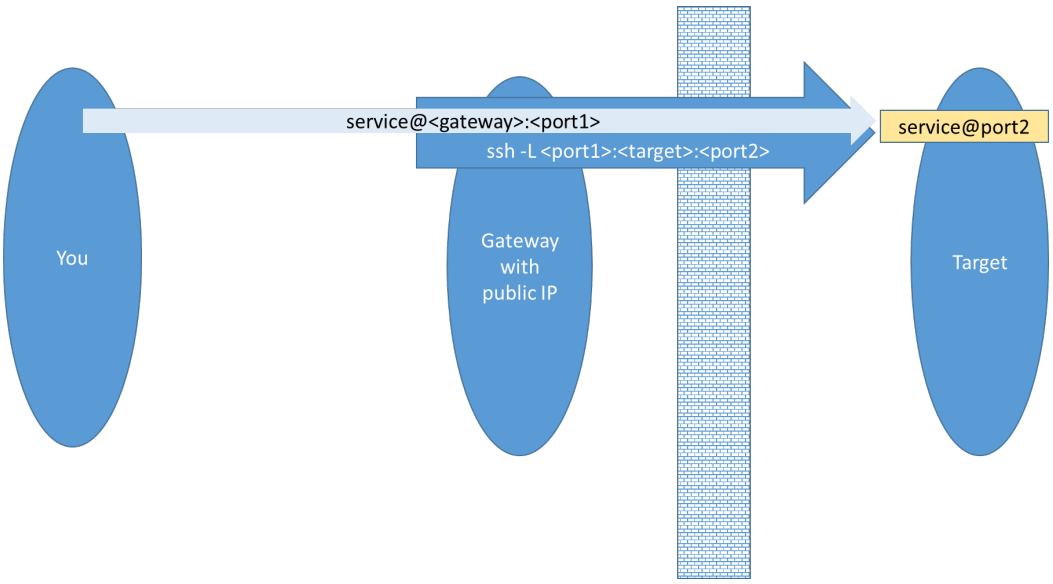
Plain *ssh* (reverse) tunneling is available to anyone that has access to a machine on the internet - for me this is a "JiffyBox", a flexible and scalable way of hosting my own server.

As understanding and configuring the different steps may be a bit complex, i moved this to a chapter of its own - right below.

4.3.4 ssh Tunneling

Straight tunnel

You use a "straight" tunnel if you have some services behind a known, reachable gateway that only provides *ssh* services. The services themselves are not visible from the outside because of NATs, firewalls or DMZs. The gateway itself **can** see the target, e.g. it is connected with a second network adapter with a private subnet. Using an *ssh* tunnel you can access the target and its service through the *ssh* tunnel, even if the target itself is not visible.



In this example you request a service from "gateway" at "port1" - not knowing that the real service is implemented at "target", "port2". The *ssh* process you started at the gateway simply relays the content through the tunnel. The *ssh* command is issued on the gateway in this case.

Option	Description
-f	Go into the background
-N	Don't execute command on the target machine
-L <local-port>:<host>:<port>	when connecting to localport on the local machine *ssh* will create a tunnel to remote host:port service. host is the name of the target known on the *ssh* server side!

Using this feature you could for example forward a HTTP protocol through the *ssh* service of the gateway to reach some HTTP server (your heating!) in the network behind.

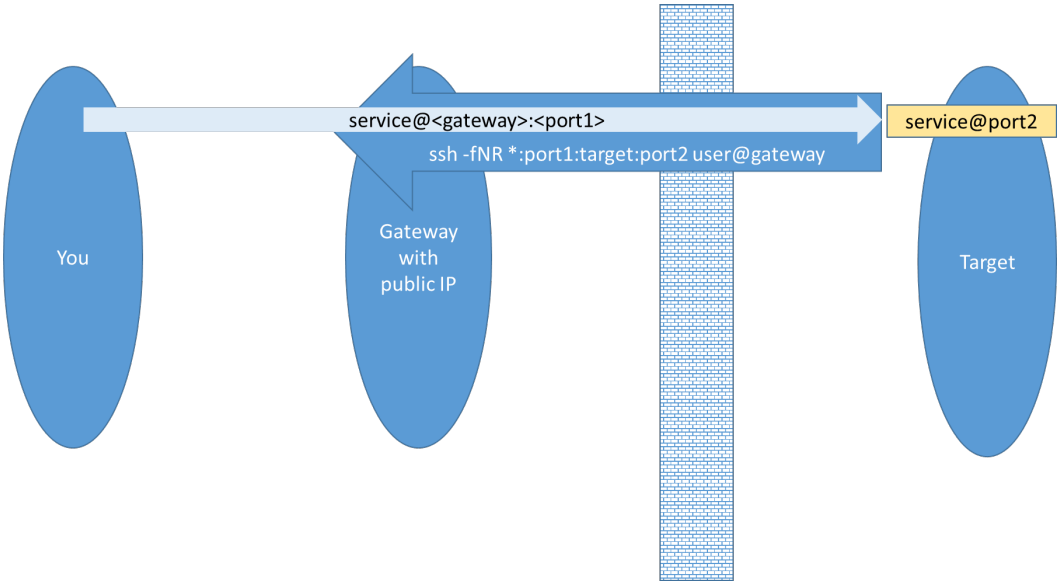
Example (this command is issued on the gateway)

```
ssh -fNL 80:wbf1:80
```

To sad that this scenario is of no use for us, as there is no way to see the target for any server under our control - they are all owned by your WAN provider.

Reverse ssh

In some occasions we want to connect to a service behind some firewall or (in our case of surfstick based WAN connection) behind some NATing box. The target is not visible on the outside for no machine in this scenario.



In this case we issue the *ssh* command on a machine in our local subnet (where target is situated, too), connecting to the gateway as "user". We instruct the gateway to open a listener on port1 and relay all traffic to our port1. Note that the target machine does not necessarily need to be the machine issuing the command, you can relay still further through the issuing machine to the target.

The leading "*" means that the gateway will bind its listener to all network addresses on the gateway (if you omit it, it will be reachable from localhost only).

Example

```
ssh -fNR *:10122:wbf1:22 tunnel@myserver.com
```

This command will create a port 10122 on the machine myserver.com. When connecting to this port, the connection is forwarded to wbf1:22 as seen from the machine issuing the command.

Option	Description
-f	Go into the background
-N	Don't execute command on the target machine
-R [bind:]<remoteport>:<host>:<port>	when connecting to remoteport on the remote machine *ssh* will create a tunnel to a local host:port service. host is the name of the machine known on the local *ssh* side!

This is a really powerful technique and I recommend trying it at least once.

To avoid prompting user and password, especially as we want to automate connection setup, it is very recommended to combine this with a public/private key protocol. You should add the public key of any user/machine that wants to create a *ssh* tunnel to the "tunnel" user on the gateway.

In our case, any Pi in the network opens a tunnel and as such we add all the keys of the Pi user on wbbase, wbf1, wbf2 and wbf3 to the authorized keys for user "tunnel" on the gateway. As we need distinct ports on the gateway we need the following four commands, **all** issued on "wbbase" our local gateway.

```
ssh -fNR *:10022:wbbase:22 tunnel@myserver.com
ssh -fNR *:10122:wbf1:22 tunnel@myserver.com
ssh -fNR *:10222:wbf2:22 tunnel@myserver.com
ssh -fNR *:10322:wbf3:22 tunnel@myserver.com
```

Or, more shortly (and with less overhead)

```
ssh -fN -R *:10022:wbbase:22 -R *:10122:wbf1:22 -R *:10222:wbf2:22 -R *:10322:wbf3:
↪22 tunnel@myserver.com
```

ssh daemon

By default a forwarded port can only be accessed via "localhost". This means if you have a reverse tunnel opened to your gateway, you can access the service on your gateway, but not from another machine via your gateway. To allow *ssh* to publish the port to all interfaces, you need to set GatewayPorts=yes in your /etc/ssh/sshd_config on the gateway server.

```
GatewayPorts yes
```

The *ssh* server automatically disconnects a client if no data is transmitted. In the case of our tunnels this is quite annoying - you must restart the tunnels from the console. But you can't, as you can no longer access the console... In addition, there may be network components around in your environment that "forget" to support a connection if there's no traffic for a while (misconfigured firewalls..). Best countermeasure: you should simply instruct the server to send keep alive messages to the client to ensure it's still there.

```
ClientAliveInterval 60
ClientAliveCountMax 2
```

This will send messages to your client every 60 seconds with a maximum attempt of 2 times. If connection is down, the server disconnects the ssh session at least afte r2 minutes. This setting will become particularly important in addition with using autossh and the **ExitOnForwardFailure** flag later on.

After any changes on the sshd_config file, you should restart the service

```
sudo service ssh restart
```

There's a similar option for the client, too. In this case you must change the *ssh* config file /etc/ssh/ssh_config

```
ServerAliveInterval 60
ServerAliveCountMax 2
```

autossh

One problem with the above approach is that in real world connections, especially via a surfstick, are not stable. In practice you have to check the connection repeatedly and restart if it is lost.

There's a wrapper implementation for *ssh* that will do this for you. First, install autossh

```
sudo apt-get install autossh
```

The "autossh" is a drop in replacement for "ssh", so this should work

```
autossh -fN -R *:10022:wbbase:22 -R *:10122:wbf1:22 -R *:10222:wbf2:22 -R *:10322:
↪wbf3:22 tunnel@myserver.com
```

For *autossh* to be "fully functional" there is a special flag you should add: **ExitOnForwardFailure**. If for example you reboot your target (the one that will issue the autossh) for some reason and on reboot you will resurrect the ssh connections you will find the server ports for forwarding still in use - the server is not aware that the connection was dropped from the client. But without the ExitOnForwardFailure flag, ssh will not exit with a failure code and autossh will not retry... So, what you need is the combination of the ServerAliveInterval/ServerAliveCountMax, ensuring that the server will find the connection is dropped at some time **and** the **ExitOnForwardFailure** flag on the autossh side to force a reconnect attempt upon initial failure.

So, working code in all its beauty:

```
autossh -o "ExitOnForwardFailure=yes" -fN -R *:10022:wbbase:22 -R *:10122:wbf1:22 -
↪R *:10222:wbf2:22 -R *:10322:wbf3:22 tunnel@myserver.com
```

Add to crontab

Most probably you want to open the tunnels upon every restart, so best is to put the call in the crontab and execute on every reboot.

Create a script /opt/admin/ssh-tunnel.sh (make it executable!!) containing only the code for launching the ssh connections above, then add this script to crontab - personally I'd add a small delay to to give the server side a better chance to clean up ssh ports.

```
crontab -e
```

Be sure to use the correct crontab - depending on the user you setup your keys for (pi or root) you must execute the script (with "pi" or "root" respectively) which again implies "crontab" or "sudo crontab".

```
@reboot sleep 150 && /opt/admin/ssh-tunnel-ssh.sh > /dev/null
```

Experience

We have a really complex scenario here. Our USB stick needs to be detected as a modem, wvdial and ppp will take care of creating a connection to your provider then. If this is successful, a ppp0 network device will appear, acting as the default route for our "wbbase". Over this connection we build a tunnel to a gateway server out there in the cloud...

There are a lot of dependencies and points of failures. I hope, the combination of commands and options is optimal for reducing downtime and enforce reconnection every time. Until now, my machines have reliably been available, at least after a short period.If you have some idea to improve in this section, you are really welcome. E.g. my experience tells me that "weaved" is able to keep the downtime even smaller!

All in all, having the option is great and really worth the effort. It has been more than once that I had to take "a quick look" or bounce some service.

4.4 Thou shalt not write...

A weak spot of the raspberry (at least regarding stability) is the SD card. Chances are good that after a certain time the card is simply no longer working. From spurious failures to non booting Pi's - everything is possible. This is called "wear out".

The best bet is to write as few as possible in everyday work, prolonging SD card life.

4.4.1 Use high quality cards

This is one of the more common tipps on the net. While it seems reasonable, I do not know how to identify "high quality cards". I actually use only "SanDisk" cards and had no drop out so far.

4.4.2 Use bigger card

Another tipp is to use cards that are bigger than the reuired size, as the card firmware will identify corrupt areas and exclude from use. So the time until failure may be prolonged. Personally I do not know if this hold true.

4.4.3 Do not write at all

If you can avoid writing data, it is the best way to improve SD card life. Especially logging is an area where data can be avoided.

Double check the log level of each components installed. Some of them produce incredible amount of data.

Especially with openHAB check the decision to support historic data.

4.4.4 Use temporary file system

If you do not want to write to SD, well either do not write at all or do write only to ram.

To do so, you can add entries in /etc/fstab. This entry for example creates a "redirection" for /var/log to the temporary file system of maximum size 1 MB.

Again, consider disabling logging completely if your data will be lost anyway for post mortem analysis. Information on the topic of "sustainable SD card use" you will find in the respective configuration sections of the software components.

And consider that a tmpfs may run out of space, too.

```
none /var/log tmpfs size=1M,noatime,nodiratime 00
none /var/run tmpfs size=1M,noatime,nodiratime 00
```

Be aware that logging to RAM will lose information from subsystems like apt, too, that record the installation history from your system.

4.4.5 File system flags

Your file system keeps track of file access. While this is sometimes nice it produces a great amount of persistent changed that at some point will be written to the card. I recommend switching off this option for every file system, editing /etc/fstab

"noatime" (and "nodiratime") simply says "do not record time of last file access".

4.4.6 Disable swap

A Linux system may use non RAM memory to emulate more RAM. As this swapping operation creates SD card access, it is recommended disabling it. If you overload the PI in such a way that you predictably **need** more Ram than physically available, you may have a problem anyway.

```
sudo dphys-swapfile swapoff
sudo dphys-swapfile uninstall
sudo update-rc.d dphys-swapfile remove
```

4.4.7 Reduce system logging

- untested feature -

Replace log management with busybox one

```
sudo apt-get install busybox-syslogd; dpkg --purge rsyslog
```

This will put log into circular memory buffer, you will able to see log using logread command

4.4.8 Boot from another device

You may go this far as to no longer use the SD card at all. You need a small bootloader and switch as early as possible to an USB stick or USB drive (change the root file system).

I did not test this feature.

4.5 Tweak installation

4.5.1 Memory

On the Pi, memory is shared between CPU and GPU. As we do not need any graphic processing, I recommend giving the CPU the biggest chunk we can afford.

The simples way is to use

```
sudo raspi-config
```

and enter the new value for GPU memory under "Advanced options", "Memory split". The smallest possible value is 16 MB.

The Pi must be restarted afterwards.

4.5.2 Watchdog

The Pi CPU has a builtin "watchdog". You can use this feature to automatically reboot the Pi if chances are good that it is stale anyway.

To understand the feature and its implications you must be aware that we talk of different parts

- **The watchdog itself** This is a hardware feature on the chip, accessible via /dev/watchdog after installation of the kernel module
- **The watchdog activation** Writing to /dev/watchdog wakes the dog and it will wait for more food on this device. If it gets none, it will reset. On the Pi the maximum hardware delay is about 16 sec. I did not find any information on how to set the hardware timer, so I just use the defaults.
- **The watchdog daemon** The watchdog is not particularly picky about who is feeding him. You can write a tool that satisfies your logic and do it yourself. Good news is that there's a generic watchdog daemon available that should handle most of the tasks. It follows your instructions and writes to /dev/watchdog

In conclusion take care of the following before activating the watchdog or its daemon:

- Be sure you can feed the dog in a timely manner after the first write. If e.g. launching the "feeder" takes too long, your Pi may stuck in reboot.
- **Be sure the conditions in the watchdog daemon can be fulfilled** This is especially true for network devices (that may not yet be up at this moment) and file monitoring (because the owning processes are not yet up). Take care of your launch sequence.
- Be sure you can handle the watchdog if your Pi is under service. If your watchdog is on and you stop and start e.g. "openHAB" after configuration change, it's easy to see your Pi restart because of log inactivity.

Activate watchdog and install daemon

```
sudo modprobe bcm2708_wdog
echo "bcm2708_wdog" | sudo tee -a /etc/modules
sudo apt-get install watchdog
sudo update-rc.d watchdog defaults
```

Change config in /etc/watchdog.conf, remove at least "#" hashmark from the following entries

```
watchdog-device = /dev/watchdog
max-load-1 = 24
```

If the load goes over "24 raspberrys" over 1 minute (this means that you would have needed more than 24 Raspberry Pis to complete that task in 1 minute) the watchdog will reset the

Pi.

For more information see

```
http://linux.die.net/man/5/watchdog.conf
```

The first time start

```
sudo /etc/init.d/watchdog restart
```

You can do a lot of things more with the watchdog daemon, especially it can have a look at files that get touched within a certain period. Tuning the dog is beyond this paper.

4.6 What time is it?

4.6.1 Overview

You may not be aware of, but the correct time is sometimes really essential.

On the Pi you have no realtime clock, meaning that the response you get after "date" is more or less accidental.

One of the more important advantages of having a correct time is the integration in home automation scenarios. Currently I do not have such an integration, but time will come.

A little less obvious is the correct behavior of cron - if you do not know in which universe the Pi appears after a reboot, cron schedule will become just guessing.

Last not least there may be some smaller or bigger bugs in some software package that stem from time problems. openHAB for example supports hot editing its configuration. After detecting a change, the system reloads. The problem is: after reload, all items are reset to some initial state. If light is on, it gets off. Not yet a problem, but: the timestamp for the next check is not reset to the time of the file found, but to the time we reloaded. If now the difference in timestamp between a file you changed and uploaded and local Pi time is big enough, you can easily guess how often you can switch on the light...

Best solution: correct time!

4.6.2 Network Time

The simplest solution is using the network time. If the Pi is connected to some outside world, it will get its time automatically via the internet. There's nothing you have to do. There are still settings in the /etc/ntp.conf that might be of use for you, but in practice you can leave it alone.

One particular useful configuration in our example would be to allow only one component to check for network time and all others use this local time in turn. To do this we configure NTP on the gateway to serve a "local time". By default NTP should be installed on the Pi, if not

```
sudo apt-get install ntp
update-rc.d ntp defaults
```

On the gateway we add the option to be polled from the local network for current time in /etc/ntp.conf

```
...
# By default, exchange time with everybody, but don't allow     configuration.
restrict default nomodify noquery

# lan may query
restrict 192.168.42.0 mask 255.255.255.0 nomodify notrap

# Local users may interrogate the ntp server more closely.
restrict 127.0.0.1
restrict ::1
...
```

Now we configure NTP on each satellite to use the gateway. You need to edit /etc/ntp.conf. You simply set the gateway machine as the new time server.

```
...
server wbbase
...
```

All other servers should be disabled.

4.6.3 Real time clock

More comfortable and in case of a disconnected device is a "real time clock". These are little battery buffered devices that can "serve" correct time to your Pi.

Integration is most often done via I2C, there are many tutorials around.

I did not integrate a RTC yet.

4.7 Improve logging

4.7.1 Understand (r)syslog

When you come to *nix, everybody will talk about and use (r)syslog. On a Pi (Debian system), "rsyslog" will be installed so we will talk about this flavor here only. You can find information on he "rsyslog vs. syslog" topic in the internet.

"rsyslog" is a daemon managing nearly all log output from standard *nix programs. An mor

you can find e.g. http://blog.scphillips.com/posts/2015/05/raspberry-pi-system-logging-and-loggly/.

"rsyslog" takes input from localhost:514 or "/dev/log". This is the preferred way to use logging, even if alog server is installed anywhere. You use "rsyslog" as a local indirection that will forward but provides other features in addition.

4.7.2 Log server

Having said that you should always use the local logging facilities to have a standard configuration and setup does not mean that server logging is bad. It is rather a practical feature (that can be used e.g. for "read only configurations").

A particulary interesting feature could be the "Loggly" cloud service - providing a cloud based log for free in the basic version. If you have a "multi node" installation as this one, you could opt for a local log server, logging to the USB disk right from the start. All this would be simply a matter of installing a log server and some local node configuration.

4.7.3 Tweak logrotate

Logrotate is the little tool that monitors log files on a regular base and rotates them according to the rules defined in /etc/logrotate.conf and the included rule sets in /etc/logrotate.d/*. You can define the rotation frequency and add scripts to handle logs after being rotated - e.g. copying them to a non tmpfs share on the network.

As rsyslog uses logrotate, too, you can manage all your system log file behavior here.

For this project the constraints are:

- I want to log on a temporary file system to avoid SD wear out.
- To avoid overflow on the tmpfs, we must logrotate early
- Rotated files are moved immediately to an attic folder, again to preserver tmpfs space
- Where possible, logs are created on a Samba share from the start

From the requirements results this default configuration

```
daily
rotate 1
dateext
dateformat .%Y%m%d.back.log
ifemtpy
missingok
create
```

After rotating I want to transfer the now rotated files to the attic on the Samba share. This could be done using a cron job, but I include it in the "postrotate" section to have the log related config in one place. "postrotate" must be defined on a per log file definition, not in the default. So I include with every log file I want to backup

```
postrotate
mv /var/log/<optional path>/*.back.log /var/remote/log/<optional path>
endscript
```

You should then go through /etc/logrotate.d/* and adapt any setting that should follow this convention.

In addition I removed most of the redundant logs from rsyslog.conf. Having everything in syslog is fine with me

```
#
# First some standard log files. Log by facility.
#
auth,authpriv.* /var/log/auth.log
*.*;auth,authpriv.none -/var/log/syslog
#cron.* /var/log/cron.log
#daemon.* -/var/log/daemon.log
#kern.* -/var/log/kern.log
#lpr.* -/var/log/lpr.log
#mail.* -/var/log/mail.log
#user.* -/var/log/user.log
```

To force logrotate to evaluate the definitions immediately you can call

```
logrotate --force /etc/logrotate.conf
```

which will rotate every log file (regardless of rule evaluation).

4.7.4 Log to temporary file system

A simple (already mentioned) trick is to simply map /var/log to a temporary file system.

```
tmpfs /var/log tmpfs size=1M,noatime,nodiratime,mode=0755 00
```

There are caveats, though:

- Some process log **really** much. You should check logging behavior.
- Logs are lost in case of failure. You could combine with a cron based copy to some hard disk.
- Log fails (and as such some processes) when tmpfs is full.

4.7.5 Log to external disk

Now we have some prerequisites to improve the logging still further and we move the important logs simply to the private share on the hard disk.

To be flexible we provide a complete directory on the private Samba share that will receive "working stuff" from the clients. Add this to /etc/fstab after creating the directory /var/remote locally.

```
# mount a directory for working stuff that can be moved to the server
/media/remote/wbdisk/private/var /var/remote none _netdev,bind 0 0
```

When the Samba share is available, all data should be written to the hard disk. It's important to add the "_netdev" flag in order to wait for the network is up.

Now add another subdirectory "log" to /var/remote after mounting. We will use this as a new base directory for all subsystems whose log is large and/or in some way interesting.

To check if everything was alright, you can

```
touch /var/remote/log/test
```

The file "test" should have been created on the harddisk at "/media/wbdisk/srv/wbf1/var/log" (if you have made this experiment from the wbf1 machine).

Any service that wants to log to samba share must add dependency to $remote_fs in its init script.

4.7.6 Move logs

rsyslog

We tweak the postrotate a little bit here to get all rotated logs to /var/remote

```
/var/log/syslog
{
        postrotate
                invoke-rc.d rsyslog rotate > /dev/null
                mv /var/log/*.back.log /var/remote/log
        endscript
}
/var/log/mail.info
/var/log/mail.warn
/var/log/mail.err
/var/log/mail.log
/var/log/daemon.log
/var/log/kern.log
/var/log/auth.log
/var/log/user.log
/var/log/lpr.log
/var/log/cron.log
/var/log/debug
/var/log/messages
{
        sharedscripts
        postrotate
                invoke-rc.d rsyslog rotate > /dev/null
                mv /var/log/*.back.log /var/remote/log
        endscript
}
```

apt

```
/var/log/apt/*.log {
        postrotate
                mv /var/log/apt/*.back.log /var/remote/log/apt
        endscript
}
```

Don't forget to create /var/remote/log/apt

dpkg

```
/var/log/dpkg.log {
        create 644 root root
        postrotate
                mv /var/log/*.back.log /var/remote/log
        endscript
```

```
}
/var/log/alternatives.log {
        create 644 root root
        postrotate
                mv /var/log/*.back.log /var/remote/log
        endscript
}
```

ppp

```
/var/log/ppp-connect-errors {
        nocreate
        postrotate
                mv /var/log/*.back.log /var/remote/log
        endscript
}
```

samba

```
/var/log/samba/log.smbd {
        postrotate
                /etc/init.d/samba reload > /dev/null
                mv /var/log/samba/*.back.log /var/remote/log/samba
        endscript
}
/var/log/samba/log.nmbd {
        postrotate
                [ ! -f /var/run/samba/nmbd.pid ] || kill -HUP \`cat /var/run/samba/
 ↪nmbd.pid\`
                mv /var/log/samba/*.back.log /var/remote/log/samba
        endscript
}
```

Don't forget to create /var/remote/log/samba

???does not work ???

msmtp

First we move the logfile to the remote location right from the start and edit /etc/msmtprc

```
logfile /var/remote/log/msmtp.log
```

Then we change the logrotate.d configuration

```
/var/remote/log/msmtp.log {
        su root root
        rotate 7
        copytruncate
}
```

asterisk

Asterisk logging is quite complex and offers a lot of alternatives. To stick with a simple one use the same look&feel as our other components, we simply switch the asterisk log to the samba share and have a look at the logrotate instructions.

Asterisk's log directory is determined in /etc/asterisk/asterisk.conf. We simply change the astlogdir

```
...
astlogdir => /var/remote/log/asterisk
...
```

We want the log to reside on the samba share from the start.

B.t.w. log contents are configured in /etc/asterisk/logger.conf. We leave it alone.

```
/var/remote/log/asterisk/debug /var/remote/log/asterisk/messages
/var/remote/log/asterisk/full /var/remote/log/asterisk/*_log {
        su root root
        daily
        missingok
        rotate 7
        sharedscripts
        postrotate
                /usr/sbin/invoke-rc.d asterisk logger-reload > /dev/null 2> /dev/
 ↪null
        endscript
}
```

The logrotate instructions should look like this. As we are already on the samba share, you may only want to switch to, lets say, 1 week of logs. Note the "remote/" subdirectory!

hmland

We tweak the init-d script to switch the log directory

```
chrt 50 /opt/hmcfgusb/hmland -r 0 -d -I -P -l 127.0.0.1 -p $port 2>&1 | perl -ne '
↪$|=1; print localtime . ": [hmland] $_"' >> /var/remote/log/hmland.log &
```

logrotate??

homegear

To keep the log files in production small, we reduce the log level. It's not directly documented, but according to some forum threads, lowering the level means less detail.

Changes to /etc/homegear/main.conf

```
# Default: debugLevel = 3
# higher values means more details
debugLevel = 2
...
logfilePath = /var/remote/log/homegear
```

A nasty detail with homegear: in the init.d script there's also a reference to the log file path, so you have to change it twice.

Changes to /etc/init.d/homegear

```
CONFIGPATH=/etc/homegear
DAEMON=/usr/bin/homegear
PIDPATH=/var/run/homegear
PIDFILE=$PIDPATH/homegear.pid
DBPATH=/var/lib/homegear
LOGPATH=/var/remote/log/homegear
```

And, be sure to try it twice: homegear does not start when the log directory is not present, create the "homegear" subdirectory in /var/remote/log !!! My tenant knows, he's rooms have been getting quite cold..

Now have a look at the logrotate instructions at /etc/logrotate.d/homegear. Again, as we are already on the samba share, we do not need to think about space restrictions as much, so we stick with a 1 week rolling behavior, too. Keep in mind that all defaults are in place.

```
/var/remote/log/homegear/*.log /var/remote/log/homegear/*.err {
        su root root
        rotate 7
        sharedscripts
        postrotate
                /etc/init.d/homegear reload > /dev/null
        endscript
}
```

Take care of the "remote/" subdirectory. Be sure to have "homegear" created.

openHAB

openHAB is (as always?) the exception to the rule. With its platform independent character it comes along with its own builtin rotation strategy.

Here, too we want to minimize log file writing.

First step is to edit "<openhab>/configurations/logback.xml. For me I will start a single, simple log at "INFO" level, rotating every day and keeping a history of 2 days. It looks like this

```
<configuration scan="true">
        <appender name="FILE" class="ch.qos.logback.core.rolling.
↪RollingFileAppender">
                <file>/var/log/remote/openhab/openhab.log</file>
                <rollingPolicy class="ch.qos.logback.core.rolling.
↪TimeBasedRollingPolicy">
                <!-- weekly rollover and archiving -->
                <fileNamePattern>/var/log/remote/openhab/openhab-%d{yyyy-MM-dd}.
↪log.zip</fileNamePattern>
                <!-- maximum number of archive files to keep -->
                <maxHistory>2</maxHistory>
                </rollingPolicy>
                <encoder>
                        <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%-5level] [%-20.
↪20logger] - %msg%n</pattern>
                </encoder>
        </appender>
        <logger name="javax.jmdns" level="OFF"/>
        <logger name="OSGi" level="OFF" />
        <logger name="org.atmosphere.cpr.AtmosphereFramework" level="OFF" />
        <logger name="org.atmosphere.cpr.DefaultAnnotationProcessor" level="OFF"/>
        <root level="INFO">
                <appender-ref ref="FILE" />
        </root>
</configuration>
```

You can do a lot of additional things with logback that may be interesting in this use case, e.g.

- Change log level
- Change log file
- Write to syslog
- Network logging

- Change rolling strategy and history

You should refer to the logback web site for detailed information.

The second source of logging events stems from jetty. It logs each and every request to a web resource (which may be much...). I don't need this in production, so I switch it off completely.

See <openhab>/etc/jetty.xml

```
...
<Set name="handler">
        <New id="Handlers"      class="org.eclipse.jetty.server.handler.
 ↪HandlerCollection">
                <Set name="handlers">
                        <Array type="org.eclipse.jetty.server.Handler">
                                <Item>
                                        <Ref id="Rewrite" />
                                </Item>
                                <Item>
                                        <New id="Contexts"      class="org.eclipse.
 ↪jetty.server.handler.ContextHandlerCollection"/>
                                </Item>
                                <Item>
                                        <New id="DefaultHandler"        class="org.
 ↪eclipse.jetty.server.handler.DefaultHandler"/>
                                </Item>
                                <!-- disable request log
                                <Item>
                                        <New id="RequestLog"    class="org.eclipse.
 ↪jetty.server.handler.RequestLogHandler"/>
                                </Item>
                                -->
                        </Array>
                </Set>
        </New>
</Set>
...
```

In addition you **must** comment out the section with the "Ref" to "RequestLog", as this is now no longer a valid reference.

Changes to logrotate

```
/var/remote/log/openhab.log {
        su root root
        rotate 7
        copytruncate
}
```

4.7.7 Add logging to your python scripts

If you have serious features implemented in scripts of your own, you most certainly will want to have some kind of log to check the scripts working and outcome.

Assuming you will use python as the Pi's nearly "native" language, we summarize the python best practices for creating debugging and operating output.

First, you should use the well documented python module "logging". Don't "print" to the console, don't invent some log file management of you own. We will not go into the countless reasons why - simply adopt best practice!

A simple example:

```
#!/usr/bin/python

import logging

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger("demo")
logger.info('Hello world!')
```

If you run the script you will hopefully see:

```
pi@wbbase ~ $ /opt/admin/demo_logging.py
INFO:demo:Hello world!
```

An important feature for post-mortem analysis is a detailed stacktrace of the exception. This is how to do it:

```
try:
        // some code of yours
except Exception, e:
        logger.error('oops', exc_info=True)
```

Now there's a million ways to configure the logging environment programatically, but I recommend not to do so. If you just externalize configuration from the start and have some boilerplate configuration files around you will be very flexible and future requirements are a no brainer.

There is a standard log configuration file format for python around, but newer version have replaced this with a more modern and flexible syntax, I recommend using this from the start. I prefer using a JSON format so a basic implementation would be

```
{
        "version": 1,
        "disable_existing_loggers": false,
        "formatters": {
```

```
                "simple": {
                        "format": "[%(asctime)s][%(levelname)s] %(message)s"
                }
        },
        "handlers": {
                "console": {
                        "class": "logging.StreamHandler",
                        "level": "DEBUG",
                        "formatter": "simple",
                        "stream": "ext://sys.stdout"
                },
                "file": {
                        "class": "logging.handlers.RotatingFileHandler",
                        "level": "DEBUG",
                        "formatter": "simple",
                        "filename": "/var/log/foo.log",
                        "maxBytes": 1000000,
                        "backupCount": 2,
                        "encoding": "utf8"
                }
        },
        "loggers": {
                "foo": {
                        "level": "DEBUG"
                }
        },
        "root": {
                "level": "INFO",
                "handlers": ["console", "file"]
        }
}
```

Some standard initialization code. This will read the log file from where the python source code is situated.

```
#!/usr/bin/python

import os
import json
import logging.config

def initLogging(
        path = os.path.dirname(os.path.realpath(__file__)) + '/logcfg.json'
):
        if os.path.exists(path):
                with open(path, 'rt') as f:
                        config = json.load(f)
```

```
                logging.config.dictConfig(config)
        else:
                logging.basicConfig(level=logging.INFO)

initLogging()

logger = logging.getLogger(__name__)
logger.info('Hello World!')
```

Switching to

```
path = 'logcfg.json'
```

Will read configuration from the current working dir instead.

Detailed information on logging and configuration options you will find here https://docs.python.org/3/howto/logging.html.

4.8 Network monitoring

Now, having attached a surfstick and doing all kind of traffic, you may be interested in some details regarding the connection. The tool of choice is here "vnstat", a simple console tool that allows to monitor your network interfaces.

If you're into sophisticated user interfaces, charts and the like, you should go for tools like "ntop". For my project vnstat is just the right size.

As always install from the repo

```
sudo apt-get install vnstat
```

With regard to the "SD less" system, you should tweak the configuration file at /etc/vnstat.conf. Change the following entries

```
DatabaseDir "/var/remote/lib/vnstat"
???
```

Do not forget that you must create the directory you referenced above. After this you bounce the service

```
sudo service vnstat restart
```

Now you can create a new database for collecting the information

```
sudo vnstat -u -i ppp0
```

As we want to monitor the WAN traffic, ppp0 is used as the target interface.

4.8.1 Show statistics

After a while you can start reading useful output with daily, weekly and monthly statistics using the -d, -w and -m option

```
vnstat -i ppp0 -m
```

There are some other options, so play around. For the visual types you can even create images using "vnstati".

4.8.2 Send statistics

A interesting option is an automatic notification mail about the traffic at the end of the month. Just combine the "vnstat" command and "mail" in a crontab entry.

This is the script you create that is finally called by crontab ((opt/admin/notifyVnstat.sh)

```
#!/bin/bash
(echo "Subject: ppp0 network statistics"; vnstat -i ppp0 -w) | msmtp pi
```

The output of "echo" and "vnstat" is combined to a valid SMTP message content and piped to msmtp, which we installed earlier.

Now we add to crontab, calling

```
sudo crontab -e
```

This line will call the notify once a week.

```
0 0 * * 0 /opt/admin/notifyVnstat.sh > /dev/null
```

4.9 Data logging

4.9.1 Data sources

Pi data

openHAB data

openHAB has a well documented REST interface. It allows reading and writing item data and sending commands. While a push interface is available, this is out of scope here.

The main url or the rest API is

```
http://<host>:8080/rest
```

where host is one of your openHAB instrumented Pi's.

You can select the content, I'm using JSON, so I'm adding the header

```
Accept: application/json
```

Service	
	/rest/items
Method	
	GET
Request Data	
Response Data	
	Array of items • type • name • state

Service	
	/rest/items/<item name>/state
Method	
	GET
Request Data	
Response Data	
	item state string

Service	
	/rest/items/<item name>/state
Method	
	PUT
Request Data	
	New item state as string
Response Data	

This is a simple curl call to read some openHAB item state. curls default is HTTP GET.

```
curl
        http://wbf1:8080/rest/items/wbf1_kitchen_therm0_tempAct/state
        --verbose
```

Now you will see something like

```
* About to connect() to wbf1 port 8080 (#0)
*   Trying 192.168.42.210...
* connected
* Connected to wbf1 (192.168.42.210) port 8080 (#0)
> GET /rest/items/wbf1_kitchen_therm0_tempAct/state HTTP/1.1
> User-Agent: curl/7.26.0
> Host: wbf1:8080
> Accept: */*
>
* additional stuff not fine transfer.c:1037: 0 0
* HTTP 1.1 or later with persistent connection, pipelining supported
< HTTP/1.1 200 OK
< Date: Sun, 21 Aug 2016 13:38:09 GMT
< X-Atmosphere-first-request: true
< X-Atmosphere-tracking-id: a193ef22-e92c-4d09-a80d-ebc0b6ddf155
< Expires: -1
< Cache-Control: no-store, no-cache, must-revalidate
< Pragma: no-cache
< Content-Type: text/plain
< Transfer-Encoding: chunked
< Server: Jetty(8.1.3.v20120522)
<
* Connection #0 to host wbf1 left intact
25.20
* Closing connection #0
```

The temperature is 25,2 degrees in the kitchen, first floor.

Integration is a whole lot easier when we do it in python.

Putting data with curl

```
curl
        -X PUT
        -H "Content-Type: text/plain"
        -d 'ON'
        http://wbf1:8080/rest/items/wbbase_heating_state/state
```

Posting data with curl

```
curl
        -X POST
        -H "Content-Type: text/plain"
        -d 'ON'
        http://wbf1:8080/rest/items/wbbase_heating_state
```

Heating data (Oekofen pellematic)

A very interesting feature is integration of heating sensor data. Most modern systems have some kind of API for home automation integration, most of them plain Web forms or REST services.

My system, an Oekofen pellematic, has a "hybrid" web interface that combines a web page for login and Ajax based services for providing the data.

Such a system can be very easily integrated by emulating the browser protocol, the browser protocol can easily be reverse engineered by using a modern browser, in my case Chrome. Open the developer tools and record the requests. With Chrome a very interesting feature is to save a request in "curl" format. curl is a commandline HTTP client and is the perfect tool for this task.

Lets look at the examples first. We add the -verbose flag, so you can insepct the protocol information in more detail. Here's the logon, saving the session cookies in a jar using the "âĂŤcookie-jar" argument...

```
curl
        http://192.168.42.2:8080/index.cgi
        --verbose
        --cookie-jar pellematic
        --data "username=<user>&password=<password>&language=en&submit="
```

Now we are authenticated, the cookies are injected in the following requests using the "âĂŤ-cookie" argument. Now we can request the sensor information. You request data points using a list of their names in JSON format (lets ignore the fact, that the Content-type is wrong).

```
curl
        'http://192.168.42.2:8080/?action=get&attr=1'
        --verbose
        --cookie pellematic
        -H 'Content-Type: application/x-www-form-urlencoded'
        -H 'Accept: application/json, text/javascript, */*; q=0.01'
        -H 'Connection: keep-alive'
        --data '[
        "CAPPL:LOCAL.L_aussentemperatur_ist",
        ...,
        "CAPPL:LOCAL.L_zaehler_fehler"
        ]'
```

And this is the result.

```
[
{
"formatTexts":"",
```

```
"shortText":"???",
"unitText":"ÂřC",
"lowerLimit":"???",
"upperLimit":"???",
"name":"CAPPL:LOCAL.L_aussentemperatur_ist",
"value":"16",
"divisor":"10",
"status":"OK",
"unitTextImperial":"ÂřF"
},
{
"formatTexts":"Dauerlauf|Start|Zuendung|...|Aus",
"shortText":"???",
"unitText":"???",
"lowerLimit":"",
"upperLimit":"",
"name":"CAPPL:FA[0].L_kesselstatus",
"value":"99",
"divisor":"",
"status":"OK",
"unitTextImperial":"???"
},
...
]
```


Let's analyze this in detail. "name" contains the name of the datapoint, "value" a scaled value and "divisor", if set the scale so that "realValue = value / divisor".

Service	
	/?action=get&attr=1
Method	
	POST
Request Data	
	Array of data points names in JSON format
Response Data	
	Array of data point result objects. We will read the following object properties • name • value • divisor • status

You can set data points, too.

Service	
	/?action=set
Method	
	POST
Request Data	
	Array of data points names in JSON format Example: {"CAPPL:LOCAL.hk[0].betriebsart[1]":"0"}
Response Data	
	Array of data point result objects. We will read the following object properties Example: [{status: "OK", name: "CAPPL:LOCAL.hk[0].betriebsart[1]", value: "0"}]

To integrate this data later it is much easier to use a real programming language instead of commandline scripts (except for those fluent in shellspeak..). For programming on the Pi you will end with Python very soon. To make HTTP request you should first install the "requests" package.

```
pip install requests
```

Now we can script a simple python program that does the equivalent of the curl invocations from above.

```
import requests
server = 'http://192.168.42.2:8080/'

# login ##########################################
headers = {
}
data = {
        'username' : '<user>',
        'password' : '<password>',
        'language' : 'en'
}
r = requests.post(server + 'index.cgi', headers = headers, data = data)
pksession = r.cookies['pksession']

# get datapoints #####################################
cookies = {
        'pksession' : pksession,
        'language' : 'en'
}
headers = {
        'Content-Type' : 'application/x-www-form-urlencoded',
```

```
        'Accept' : 'application/json',
        'Accept-Language' : 'en'
}
data = '["CAPPL:LOCAL.L_aussentemperatur_ist","CAPPL:FA[0].L_kesselstatus"]'
r = requests.post(server + '?action=get&attr=1', headers = headers, data = data,␣
 ↪cookies = cookies)
print r.json()
```

We will combine this to a complete script later, dumping the result values in a RRD database for visualization.

Internet data

The internet provides a lot of free services you can integrate in you home automation - to only name a few

- **weather conditions**
  - temperature
  - rain
  - wind
- **astro conditions**
  - sunset
  - sunrise
- **calendars**
  - holidays

4.9.2 Data sinks

Plain RRD

rrdTool is a de facto standard when it comes to collection data, so it's a piece of cake to get it up and running. Just kidding...

To be honest, as always in the hackers world the tools are fragmented and you have to collect the pieces together... at least in the windows world.

Get the epoch

http://www.epochconverter.com/

perl -e "print time"

If you want to estimate the size of the resulting database, you may use the online calculator at http://eccentric.slavery.cx/misc/rrdcalc.html.

install rrd

sudo apt-get install rrdtool python-rrdtool

check last update

rrdtool lastupdate filename

show database info

rrdtool info filename

fetch information

rrdtool fetch filename

Outside temperature

For the outside, I store one value per hour, only the average over 6 data points. I keep this for 2 years. I plan to push a new value every 10 minutes and allow for one missing data point.

```
rrdtool create /media/remote/wbdisk/shared/rrd/environment.rrd
        --start now-1h
        --step 600
        DS:temp_outside:GAUGE:1200:-30:50
        RRA:AVERAGE:0.5:6:17520
```

Heating

The heating provides many data points to control its effectiveness.

4320 = 1 measure/[10 min] * 6 * 24 *30

17520 = 1 measure/[1h] * 24 * 365 * 2

```
rrdtool create /media/remote/wbdisk/shared/rrd/heating.rrd
        --start now-1h
        --step 600
        DS:count_burner_start:COUNTER:1200:U:U
        DS:temp_burner_act:GAUGE:1200:-30:500
        DS:temp_floor_act:GAUGE:1200:-30:500
        DS:temp_floor_nom:GAUGE:1200:-30:500
        DS:temp_water_act:GAUGE:1200:-30:500
        DS:temp_pump_off_act:GAUGE:1200:-30:500
        DS:temp_pump_on_act:GAUGE:1200:-30:500
```

```
        RRA:AVERAGE:0.5:1:4320
        RRA:AVERAGE:0.5:6:17520
```

Room data

10080 = 1 measure/[1 min] * 60 * 24 *7

17520 = 1 measure/[1h] * 24 * 365 * 2

```
rrdtool create /media/remote/wbdisk/shared/rrd/wbf1_roomR.rrd \\
        --start now-1h \\
        --step 60 \\
        DS:humidity:GAUGE:300:0:100 \\
        DS:temp_act:GAUGE:300:-30:500 \\
        DS:temp_set:GAUGE:300:-30:500 \\
        DS:valve:GAUGE:300:0:1 \\
        RRA:AVERAGE:0.5:1:10080 \\
        RRA:AVERAGE:0.5:60:17520
```

rrdtool graphing

```
#!/bin/bash
#
rrdtool graph environment.png \\
        -w 600 -h 400 -a PNG \\
        --slope-mode \\
        --start -86400 --end now \\
        --font DEFAULT:7: \\
        --title "Heating" \\
        --vertical-label "Temperature" \\
        --lower-limit -10 \\
        --x-grid MINUTE:10:HOUR:1:MINUTE:120:0:%R \\
        --alt-y-grid --rigid \\
        DEF:temp_outside=/media/remote/wbdisk/shared/rrd/environment.rrd:temp_
↪outside:AVERAGE LINE1:temp_outside#0000FF:"Aussen"
```

openHAB persistence

Hosted cloud services

Own services

4.9.3 Data visualization

4.10 Event notifications

4.10.1 Sending eMail

Still the most important channel (well, for me) is email, so we set up this one first.

As always in this project, mailing is a really complex component and you have many software packages around. To give a rough (and maybe a bit simplifying overview):

- you create a mail using a mail user agent (MUA)
- the mail is sent to a mail submission agent (MSA)
- the MSA calls the mail transfer agent (MTA) and asks him to send the mail to its final or intermediate destination at a mail server (a remote MTA)
- the mail server acts either as a relay and still forwards to another server or is the final destination. In this case the mail is put in the target box by the mail delivery agent (MDA)
- now a user can use her mail client to access the post box.

The protocol used for sending the mail is SMTP, the protocols to read mail via the mail client are usually POP3 or IMAP. The different roles in this game can be implemented by only a single large software component or by different specialized ones.

In theory you could connect to your target mail host directly and "push" your mail. Too bad that in the meantime sending spam is a real plague and most servers will no longer accept mail from you directly. It may get quite complicated to setup a MTA with direct internet access and keep it online - believe me, I've been there.

I don't need server or relay features and no local postboxes. I just want to

- send mails from the Pi world to the outside
- receive the Pi "local" mails (e.g. mail to from crontab to root) in my regular mail infrastructure

To achieve this you don't need to install full blown MTA like Postfix or Exim, you need a tool that can send all your mail messages to a friendly server that takes care of forwarding them to whomever may be concerned. These servers or relays are often called smarthosts.

One such smarthost is provided by Google and in the next steps a Google account is provided as example. The same scheme applies to other providers.

Most tutorials I found install "ssmtp" a quite small tool that worked without much hassle. Here are the basic steps to do so.

Install the MTA "light"

```
sudo apt-get install ssmtp
```

Install the MUA

```
sudo apt-get install mailutils
```

Now we edit the configuration file at /etc/ssmtp/ssmtp.conf

```
root=<mail address to receive system mails>
mailhub=smtp.gmail.com:587
hostname=wbbase
AuthUser=<yourname>@gmail.com
AuthPass=<yourpassword>
UseSTARTTLS=YES
```

If you use a Gmail account as a smarthost, be sure to enable the setting "Allow less secure applications" in your account setting, your mail will not be delivered otherwise.

Again, there are some additional features that you can explore in some spare time...

```
echo "Test text" | mail -s "Test Mail" targetperson@example.com
```

Change the 'From' text by editing /etc/passwd to receive mail from 'root at myhost' instead of just 'root'.

```
sudo chfn -f 'root at myhost' root
sudo chfn -f 'mainuser at myhost' mainuser
```

Now that you have mail set up on your Pi, you may find out that there's a mail for each cron job.

```
>/dev/null 2>&1
```

If you want to change the name of the sender when mailing on behalf of a system user, you can do this by setting the user's name:

```
sudo chfn -f root.homeserver root
sudo chfn -f myuser.homeserver myuser
```

Failed messages are placed in dead.letter in the sender's home directory.

msmtp

Changes to /etc/msmtprc

```
defaults
auth on
tls on
tls_trust_file /etc/ssl/certs/ca-certificates.crt
logfile /var/remote/log/msmtp.log
aliases /etc/aliases

# gmail
account gmail
host smtp.gmail.com
port 587
from foo@gmail.com
user foo@gmail.com
password 1234

account default : gmail
```

CHAPTER

FIVE

PART IV

This is the part about dreaming. What could we add to make this a real reference installation. I hope many of the following topics will make it to part III one fine day.

5.1 More event notifications

5.1.1 Pushover

Pushover (https://pushover.net/) is currently one of the better known platforms for simple push messages. You can find an overview of the services provided on their platform. While not exactly free, you must only purchase a cheap client app to use the service without regular fees.

Operating a client is very simple, even for using the shell with curl an example is provided.

Registration is simple. On the website go to API, Signup. Enter eMail address and a password and you will receive a confirmation mail immediately.

After downloading and installing a client, e.g. the Android client you sign on using user and password. After this you have to connect the client to your account and that's it.

Now you are ready to receive notifications from pushover. On the main web page, when you are logged in, you can send notifications directly using a form.

To get active you need one more step - you need to register an application to send messages yourself, e.g. using curl. Once again, on the website there's a page to register an application. You enter a name, accept conditions and are ready to push messages with the application token you receive after this step.

This is all you need to send your first message from the command line.

```
curl --form-string "token=<app-token>" --form-string "user=<user-token>" --form-
↪string "message=from curl with love" https://api.pushover.net/1/messages.json
```

5.1.2 Sending SMS

5.1.3 Twitter

5.1.4 Other push message platforms

5.2 Adding a UI device

5.3 Alternate actor technology

5.4 Access control

5.5 Irrigation control

5.6 RTC

CHAPTER

SIX

PART V

This section is dedicated to tools & applications you can or should install on your Pi. As many of them have no direct relationship to the home automation topic but nonetheless are so complex and powerful themselves that squeezing a description into the other chapters would be obfuscating the whole book, I simply pulled them out into this chapter.

6.1 Radicale

6.1.1 Overview

Radicale is a lightweight CalDav & CardDav server that seems to be perfectly sized for this project. Other options may have been DaviCal, Baikal or even ownCloud (only to name a few), but all have a much larger footprint, bot in code and persistence requirements.

Radicale is a no database, no UI, no bells, no whistles implementation. I will give this a try.

6.1.2 Installation

Radicale should be available from the repository. You may install from hand using the python installer, but be prepared that settings are different.

```
sudo apt-get install radicale
```

I had to uncomment the line

```
ENABLE_RADICALE=yes
```

in /etc/default/radicale to enable automatic startup.

That's all for the first steps. Start radicale.

```
sudo service radicale
```

The server is now running on

```
http://<machine>:5232
```

6.1.3 Test

Now you can simply add a calendar and some events using your favorite CalDav client. Normally this is quite simple for example with your smartphone CalDav client application. The default calendar location is

```
http://<machine>:5232/<user>/calendar.ics
```

6.1.4 Update

If you have installed via "apt-get" (wich will give you v0.7), this may be the best moment to udpate to the latest version 1.1.1. At least for me the authentication part did either not work or my configuration file structure was not compatible with the version i tried.

But good news, update is simple. Just download the current tar

```
wget http://pypi.python.org/packages/source/R/Radicale/Radicale-1.1.1.tar.gz
```

and unpack

```
tar xzvf Radicale-*.tar.gz
```

Switch to the unpacked directory

```
cd Radicale-1.1.1
```

and start installation

```
sudo python setup.py install
```

6.1.5 Authentication

If you plan to use this in a production environment or even make the service accessible from the outside world you really must add authentication! Radicale comes with a bunch of possibilities, the simplest (and least secure) one being htpasswd. As my personal needs are not this advanced, i may get away with using this approach.

To activate htpasswd authentication for radicale you have to change the /etc/radicale/config file. Be warned: for my installation, the config file template was not up to date, i had to copy the version from the radicale web site.

```
[auth]
type = htpasswd

[rights]
type = authenticated
```

The other defaults are fine. Be sure to restart the radicale service after changing the config.

Now we need to add a htpasswd compatible file at /etc/radicale/users. There are some tools around that allow you to create such a file "by hand" - i found the easiest way to use the original apache htpasswd utility. First ensure that the apache utilities are installed.

```
sudo apt-get install apache2-utils
```

Now *htpasswd* is available. With

```
sudo htpasswd -cd /etc/radicale/users foo
> New password:
> Re-type new password:
> Adding password for user foo
```

you create the file "/etc/radicale/users" and add the user *foo* with the password you entered at the console (omit the "c" option when the file is already present). When you configure a client now you must use a valid user/password combination for authentication - be aware that this does not need to be the same user as you use in the calendar URL.

6.1.6 More security

Above are the most basic settings you can provide to secure your system. If you want to go the extra mile, you may add

- bcrypt support
- PAM authentication
- SSL support

6.1.7 Clients

Radical is not designed to have the best standard support, but most clients will do.

For my requirements i have used two options

- **CalDAV Synch for Android** This is a great and very versatile tool that neatlessly worked with Radicale
- **CalDav Zap** This is a pure browser based implementation and a very nice calendar app.

These two make up for a very simple remote date based control.

6.2 minicom

In some sceanrios it is quite useful to have a terminal around to make some low level conversation with attached devices.

A very common tool is "minicom".

```
sudo apt-get install minicom
```

6.3 screen

"screen" is a very useful tool if you need to work at the console, especially via *ssh*. In many cases you will wish to have another terminal at hand to do some related task, look up a certain file... This is where "screen" comes into play. It is a "virtual" console switcher, so you can have multiple sessions at once.

As it is really powerful, I recommend looking up some tutorial.

```
sudo apt-get install screen
```

CHAPTER

SEVEN

PART VI

These are the appendices, a bunch of information added for reference purposes.

7.1 Troubleshooting

- have a backup around
- fstab tmp too small
- update-rc.d -f ntp remove
- touch definition files

7.2 Oekofen Pellematic Datapoints

These are the datappoints reverse engineered from the Oekofen Pellematic heating, date of manufacture in 2015.

CAPPL:FA[0].asche_externe_aschebox	
CAPPL:FA[0].ausgang_motor[0]	
CAPPL:FA[0].ausgang_motor[1]	
CAPPL:FA[0].ausgang_motor[11]	
CAPPL:FA[0].ausgang_motor[2]	
CAPPL:FA[0].ausgang_motor[3]	
CAPPL:FA[0].ausgang_motor[4]	
CAPPL:FA[0].ausgang_motor[5]	
CAPPL:FA[0].ausgang_motor[7]	
CAPPL:FA[0].ausgang_motor[8]	
CAPPL:FA[0].ausgang_stoermelderelais	
CAPPL:FA[0].betriebsart_fa	
CAPPL:FA[0].frischwasser_soll_temp	

Continued on next page

Table 7.1 – continued from previous page

CAPPL:FA[0].L_abgastemperatur	
CAPPL:FA[0].L_abgastemperatur_vorhanden	
CAPPL:FA[0].L_agt_zuend_flammueb	
CAPPL:FA[0].L_alterkessel	
CAPPL:FA[0].L_anzahl_zuendung	
CAPPL:FA[0].L_aussentemperatur	
CAPPL:FA[0].L_aussentemperatur_vorhanden	
CAPPL:FA[0].L_br1	
CAPPL:FA[0].L_brennerlaufzeit_anzeige	
CAPPL:FA[0].L_brennerstarts	
CAPPL:FA[0].L_bsk_status	
CAPPL:FA[0].L_drehzahl_ascheschnecke_ist	
CAPPL:FA[0].L_drehzahl_uw_ist	
CAPPL:FA[0].L_einschublaufzeit	
CAPPL:FA[0].L_feuerraumtemperatur	
CAPPL:FA[0].L_feuerraumtemperatur_soll	
CAPPL:FA[0].L_feuerraumtemperatur_vorhanden	
CAPPL:FA[0].L_fuellstand_aktuell	
CAPPL:FA[0].L_kap_sensor_raumentnahme	
CAPPL:FA[0].L_kap_sensor_zwischenbehaelter	
CAPPL:FA[0].L_kesselstatus	
CAPPL:FA[0].L_kesseltemperatur	
CAPPL:FA[0].L_kesseltemperatur_soll_anzeige	
CAPPL:FA[0].L_luefterdrehzahl	
CAPPL:FA[0].L_mittlere_laufzeit	
CAPPL:FA[0].L_pausenzeit	
CAPPL:FA[0].L_pe_schnecke_sauganlage	
CAPPL:FA[0].L_pelletsfuellstand	
CAPPL:FA[0].L_reserve_temp1	
CAPPL:FA[0].L_saugintervall	
CAPPL:FA[0].L_saugzugdrehzahl	
CAPPL:FA[0].L_sillstandszeit	
CAPPL:FA[0].L_status_saugsonde1	
CAPPL:FA[0].L_status_saugsonde2	
CAPPL:FA[0].L_status_saugsonde3	
CAPPL:FA[0].L_unterdruck	
CAPPL:FA[0].L_unterdruck_soll_anzeige	
CAPPL:FA[0].L_unterdruck_vorhanden	
CAPPL:FA[0].L_zwischenbehaelter_aktuell	
CAPPL:FA[0].leistung_brennstoffkorrektur	
CAPPL:FA[0].saugsonden_modus	

Continued on next page

Table 7.1 – continued from previous page

CAPPL:FA[0].unterdruck_modus	
CAPPL:FA[0].wiegezelle_fuellstandserkennung	
CAPPL:FA[0].wiegezelle_fuellstandserkennung_gk	
CAPPL:LOCAL.anlage_betriebsart	
CAPPL:LOCAL.aussentemperatur_vorhanden	
CAPPL:LOCAL.externe_anforderung_aktiviert	
CAPPL:LOCAL.fa_hkr_mode	
CAPPL:LOCAL.fernwartung_einheit	
CAPPL:LOCAL.hk[0].aktives_zeitprogramm	
CAPPL:LOCAL.hk[0].alias	
CAPPL:LOCAL.hk[0].auto_extern	
CAPPL:LOCAL.hk[0].betriebsart[0]	
CAPPL:LOCAL.hk[0].betriebsart[1]	Aus = 0 Auto = 1 Heizen = 2 Absenken = 3
CAPPL:LOCAL.hk[0].betriebsart[2]	
CAPPL:LOCAL.hk[0].quelle	
CAPPL:LOCAL.hk[0].raumfuehler_zuweisung	
CAPPL:LOCAL.hk[0].raumtemp_absenken	
CAPPL:LOCAL.hk[0].raumtemp_heizen	
CAPPL:LOCAL.hk[0].raumtempabweichung_fernb	
CAPPL:LOCAL.hk[0].solarheiz_betriebsart[0]	
CAPPL:LOCAL.hk[0].solarheiz_betriebsart[1]	
CAPPL:LOCAL.hk[0].solarheiz_betriebsart[2]	
CAPPL:LOCAL.hk[0].vorhanden	
CAPPL:LOCAL.L_aussentemperatur_ist	
CAPPL:LOCAL.L_fernwartung_datum_zeit_sek	
CAPPL:LOCAL.L_hk[0].pumpe	
CAPPL:LOCAL.L_hk[0].raumtemp_ist	
CAPPL:LOCAL.L_hk[0].raumtemp_soll	
CAPPL:LOCAL.L_hk[0].status	
CAPPL:LOCAL.L_hk[0].vorlauftemp_ist	
CAPPL:LOCAL.L_hk[0].vorlauftemp_soll	
CAPPL:LOCAL.L_ke_brennerkontakt_1	
CAPPL:LOCAL.L_pu[0].ausschaltfuehler_ist	
CAPPL:LOCAL.L_pu[0].ausschaltfuehler_soll	
CAPPL:LOCAL.L_pu[0].einschaltfuehler_ist	
CAPPL:LOCAL.L_pu[0].einschaltfuehler_soll	
CAPPL:LOCAL.L_pu[0].pumpe	
CAPPL:LOCAL.L_pu[1].ausschaltfuehler_ist	
CAPPL:LOCAL.L_pu[1].ausschaltfuehler_soll	
CAPPL:LOCAL.L_pu[1].einschaltfuehler_ist	
CAPPL:LOCAL.L_pu[1].einschaltfuehler_soll	

Continued on next page

Table 7.1 – continued from previous page

CAPPL:LOCAL.L_pu[1].pumpe	
CAPPL:LOCAL.L_pu[2].ausschaltfuehler_ist	
CAPPL:LOCAL.L_pu[2].ausschaltfuehler_soll	
CAPPL:LOCAL.L_pu[2].einschaltfuehler_ist	
CAPPL:LOCAL.L_pu[2].einschaltfuehler_soll	
CAPPL:LOCAL.L_pu[2].pumpe	
CAPPL:LOCAL.L_weather[0]	
CAPPL:LOCAL.L_weather_clouds	
CAPPL:LOCAL.L_ww[0].ausschaltfuehler_ist	
CAPPL:LOCAL.L_ww[0].einschaltfuehler_ist	
CAPPL:LOCAL.L_ww[0].temp_soll	
CAPPL:LOCAL.L_zaehler_fehler	
CAPPL:LOCAL.pellematic_vorhanden[0]	
CAPPL:LOCAL.pu[0].vorhanden	
CAPPL:LOCAL.pu[1].vorhanden	
CAPPL:LOCAL.pu[2].vorhanden	
CAPPL:LOCAL.weather_config	
CAPPL:LOCAL.ww[0].betriebsart[1]	
CAPPL:LOCAL.ww[0].temp_absenken	
CAPPL:LOCAL.ww[0].temp_heizen	

7.3 Abbreviations

Abbreviation	Meaning
HA	Home Automation
SBC	Single Board Computer

7.4 *nix 101

7.4.1 Package management

Show installed packages & versions

```
dpkg-query -l [filter]
```

7.4.2 mount

To reload after changes

```
mount -a
```

7.4.3 shebang

The first line of a script file can be used as special marker to identify the interpreter context to execute the file content.

```
#!/bin/sh
```

This will for example execute as a plain shell script.

History explains this as a concatenation of sharp (the âĂŸ#' sign) and bang (the unix slang for the '!').

7.4.4 python

sudo apt-get install python-pip

sudo pip install requests

```
#!/usr/bin/python
```

make executable!

7.4.5 Users & Passwords

Change password

You can change your password using

```
passwd
```

You'll be prompted to enter your current password, then asked for a new passwordand a confirmation. Once your new password is verified, you'll be shown a success message and it will be immediately in effect.

Remove password

In certain occasions you may want to remove a users password (wich means he can not login), especially you may want to remove the possibility for the user "root" to login interactively.

```
sudo passwd root -d
```

Create a user

To create an additional user "foo" you can

```
sudo adduser foo
```

This will add a new user after prompting for a new password.

Remove a user

If you want to get rid of a user, type

```
sudo userdel -r foo
```

The "-r" option will remove the home folder, too.

7.4.6 Service Management

Deactivate

```
sudo service {dienst} stop
sudo update-rc.d -f {DIENST} remove
```

Activate

```
sudo update-rc.d {DIENST} defaults
```

7.4.7 Send command to background

```
&
```

7.4.8 Concatenating output

```
( command1 ; command2 ; command3 ) | cat
```

7.4.9 whoami

7.4.10 groups

7.4.11 rm

rm removes a file or directory

Safety belt

```
rm -I
```

Prompt once if you're about to delete recursively or a lot of files.

-r recursive delete. To delete directories you must always explicitly use -r

7.4.12 File system magic

List of open files

```
lsof
```

Show users of file

```
fuser -v <file>
```

Show file system of a file

```
df -T /else/where
```

7.4.13 Networking

netstat

Now lets have a look at the current routing information with "netstat -r"

```
Kernel-IP-Routentabelle
Ziel Router Genmask Flags MSS Fenster irtt Iface
default * 0.0.0.0 U 0 0 0 ppp0
10.64.64.64 * 255.255.255.255 UH 0 0 0 ppp0
192.168.42.0 * 255.255.255.0 U 0 0 0 eth0
```

7.4.14 crontab

To edit a crontab entry call

```
sudo crontab -e
```

or

```
crontab -e
```

depending on if you want to edit the root or current user specific crontab. The crontab definitions are stored in /var/spool/cron/crontabs, but you should not edit the crontab definitions directly.

Here a short summary of crontab syntax.

```
* * * * * <user> <command>
| | | | |
| | | | +----- day of week (0 is sunday)
| | | +------- month (1 - 12)
| | +--------- day (1 - 31)
| +----------- hour (0 - 23)
+------------- minute (0 - 59),

| */n every n unit
| x-y from unit x to unit y
```

Short forms

Short form	Meaning	cron synax
@reboot	once upon start	
@daily	once per day	0 0 * * *
@hourly	one per hour	0 * * * *
@weekly	once per week	0 0 * * 0
@monthly	once per month	0 0 1 * *
@yearly	once per year	0 0 1 1 *

To list the crontab for the current user

```
crontab -l
```

7.5 Tools

7.5.1 Connecting

WinSCP

Putty

7.5.2 Some network tools

wget

commandline HTTP client.

NMAP

Simple host discovery

```
sudo apt-get install nmap
```

```
nmap -sP 192.168.2.1/24
```

netcat

The net swiss army knife

```
sudo apt-get install ncat
```

7.6 File listings

7.6.1 /etc/fstab

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search