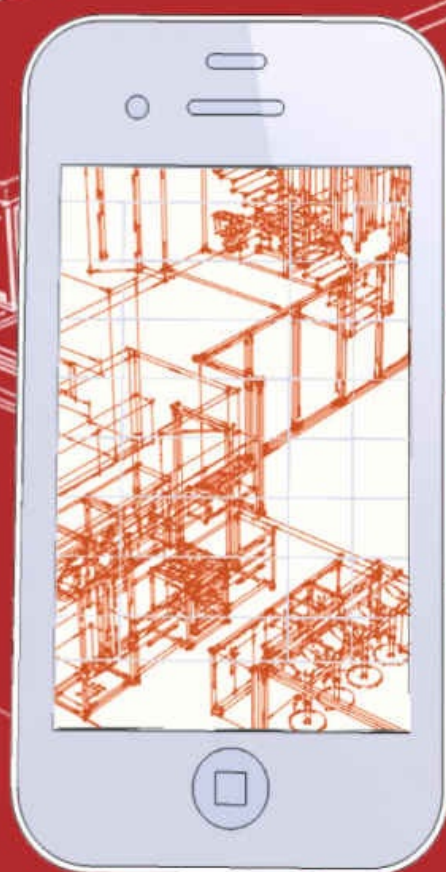


**OTHMAR KYAS**

# **HOW TO SMART HOME**



**KEY CONCEPT PRESS**

# How To Smart Home

## *A Step by Step Guide to Your Personal Internet of Things*

A Key Concept Book by  
Othmar Kyas

*3rd Edition*

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without permission in writing from the publisher.

Copyright © 2015 by KEY CONCEP PRESS

Published by Key Concept Press e.K., Wyk, Germany

[www.keyconceptpress.com](http://www.keyconceptpress.com)

Cover design: Joerg Nestle

ISBN 978-3-944980-06-5

Third Edition March 2015





## Disclaimer

Every effort has been made to make this book as accurate as possible. However, there may be typographical and or content errors. Therefore, this book should serve only as a general guide and not as the ultimate source of subject information. This book contains information that might be dated and is intended only to educate and entertain. The author and publisher shall have no liability or responsibility to any person or entity regarding any loss or damage incurred, or alleged to have incurred, directly or indirectly, by the information contained in this book. References to websites in the book are provided for informational purposes only and do not constitute endorsement of any products or services provided by these websites. Further the provided links are subject to change, expire, or be redirected without any notice.



## Bonus Material for Download

Bonus material for this book containing code and design templates can be downloaded from the book website <http://www.howtosmarthome.com>.



## Notification on Updates and New Releases

If you want to be notified when an update to this book or a new release from Key Concept Press becomes available sign up [here](#). Our mailing list is exclusively used to keep you and others informed about Key Concept Press. We do not share or sell your information to any third parties <http://www.keyconceptpress.com/newsletter.html>.



## About the Author

Othmar Kyas is an internationally renowned expert in communication technology and strategic marketing. He is author of twelve books, which have been translated into five languages.



# Table of Content

Disclaimer

Bonus Material for Download

Notification on Updates and New Releases

About the Author

Table of Content

# 1 Read Me

1.1 Who is this Book for?

1.2 What You Will NOT Find

1.3 Take no Risks

1.4 Formatting Rules

## 2 The Big Picture

### 2.1 The Potential for Energy Conservation

## 3 Key Concepts

3.1 Devices under Control

3.2 Sensors and Actuators

3.3 Control Networks

3.4 Controller

3.5 Remote Control Devices

3.6 Market Trends

3.7 Smart Homes for the Masses: Google, Apple, Samsung and more ...

3.8 Where do we go from here?

## 4 The Project

### 4.1 Overview

### 4.2 Equipment and Prerequisites

## 5 The Home Control Centre: Open Remote

5.1 OpenRemote Overview

5.2 OpenRemote Controller Installation

5.3 Installation under Mac OS X

5.4 Installation under Windows 7, 8 and Windows XP

5.5 OpenRemote Designer

5.6 The “Hello World” App

## 6 A Pretty Smart Sensor: Internet Weather

6.1 OpenRemote Control via HTTP: Retrieving Internet Weather Data

6.2 Designing the App Layout

## 7 Smartphone Based Presence Detection

7.1 Building a DHCP – MAC Address Monitor Function

7.2 Creating a Shell Script for Presence Detection

7.3 Shell What?

7.4 The Presence Detection Script under OS X / Linux

7.5 Testing it Right - Best Practice for Script Writing

7.6 Building the Script

7.7 A Log File for Presence Detection

7.8 Testing the Script

7.9 The Presence Detection Script under Windows 7 & 8

7.10 Testing it Right - Best Practice for Script Writing

7.11 Building the Script

7.12 Log File for Presence Detection

7.13 Testing the Script

7.14 Controlling Presence Detection via Smartphone



## 8 Integration of Multimedia: iTunes Remote

8.1 Script Based iTunes Control in OS X

8.2 Script Based iTunes Control on Windows XP/7/8

8.3 Creating the iTunes Smartphone Remote

8.4 Talk to Me

8.4.1 Speech Output Under OS X

## 9 A Little AI: Drools Rules

9.1 Wake me up Early if it Rains: iAlarm

9.2 Controlling iAlarm via Smartphone

9.3 The iAlarm Rule Script

9.4 Coming Home

## 10 More iDevices

10.1 Denon / Marantz Audio System Control

10.2 Device Control Using Z-Wave

10.2.1 Z-Wave Network Setup

# 11 Industry Grade Home Infrastructure Control: KNX

11.1 What is KNX?

11.2 How does KNX Work?

11.3 The KNX Software Infrastructure: ETS

11.4 Which Operating Systems does ETS5 Support?

11.5 ETS5 on a Mac

11.6 Other KNX.org Software Tools

11.7 ETS5 Installation

11.8 Importing Vendor Catalogs

11.9 ETS5 Infrastructure Configuration

11.10 ETS5: Adding the Building Infrastructure

11.11 ETS5: Configuring the KNX Elements

11.12. ETS5: Connecting Infrastructure to Controls

## 12 KNX Control via OpenRemote Designer

12.1 Background Pictures for the Smartphone and Tablet App

12.2 Configure KNX Based Heating Mode Control

12.3 Smartphone Based Heating Control

12.4 Drools Based Heating Automation

## 13 Remote Smarthome Control

13.1 Configuring a Dynamic DNS Service

13.2 Configuring a VPN

## 14 Cold Start: Launch Automation

14.1 Windows Task Scheduler

14.2 OS X launchd

# 15 Troubleshooting and Testing

15.1 Preventive Maintenance

15.2 OpenRemote Heartbeat and Watchdog



## 16 ... we proudly present: Reporting

### 16.1 A Drools Reporting Rule

## 17 Appendix

### 17.1 OpenRemote Professional Designer

# Bibliography





# 1 Read Me



## 1.1 Who is this Book for?

This book shows how to take home automation to the next level, using state of the art technologies such as tablets, smartphones, and the Internet in conjunction with the latest wireline and wireless home automation standards. It has been written for anyone who wants to use smartphone control to automate a building or a residential home. Expecting no specific know-how upfront, it is suited for both the technology loving hobbyist as well as the professional consultant. Technologies and platforms which are used in the projects described in the book are:

- Wi-Fi / WLAN
- Telnet, HTTP, TCP/IP
- Z-Wave
- ZigBee
- KNX
- Drools (an open source object oriented rule engine)
- OpenRemote (an open source building automation platform)
- Operating systems: Mac OS X / Linux / Windows

Parts of the projects integrate consumer electronics devices, such as audio equipment from Denon and Marantz. However, projects and instructions are designed so that that they can easily be adapted to other manufacturers. Be aware, however, that equipment which is more than two or three years old probably will lack the required interfaces for home automation integration at the level which is being covered in this book, such as built in WLAN, Bluetooth, Web server components, or “Wake-on-LAN” functionality.

After explaining the big picture and the key concepts of state of the art home automation, the book will walk you in a step-by-step manner through the implementation of several essential home automation and control projects. At the end of each project phase you should have a real, working solution on your desk, which can be further customized and expanded as desired. No programming skills are required as prerequisite. Scripts and configurations are explained line by line. Of course, if you have never written a short automation script or configured a DSL router, at some point your learning curve will be steeper than that of others. However, everything you learn will be open standard based, essential technologies, which you will be able to utilize in any other IT related project.



## 1.2 What You Will NOT Find

This book is not about legacy technology based home automation such as routing infrared signals around the house and controlling light switches and power outlets using outdated technologies like X10. It is also not a cookbook for plug and play type of home automation solution, which various vendors and utilities are offering based on closed and proprietary solutions with limited functionality. While popular solutions like Apple HomeKit, Google Nest or Samsung SmartThing are being discussed, they are not the focus of this book. At this point their capabilities to integrate existing building infrastructure and to build a customized smart home solution are still too limited.

## 1.3 Take no Risks

Be careful when following the step-by-step instructions. No two PC systems, consumer electronic devices, or other electronic gear are alike. If something goes wrong, you might need to reinstall the operating systems on your PC and you could lose all your data. So use a spare computer system, dedicated for testing or experimentation, unless you are absolutely sure what you are doing. I cannot take any liability for any undesired outcome of the given instructions.

## 1.4 Formatting Rules

For better readability, the following formatting rules are used throughout the book:

- *Italic*  
Email, IP, MAC addresses, file names, application names
- *Italic, blue*  
URLs
- Monospace  
Computer output, code, commands, variables
- LARGE CAPS  
Communication Protocols (DHCP, IP, etc.)
- *Italic, green*  
Sequence of GUI commands (application or operating system)

For the purpose of the exercises in this book I have created the user account *smarhome* (on both OS X and Windows XP/7/8). The prompts in the terminal window in some of the screenshots and terminal print-outs look accordingly.



## 2 The Big Picture

Home automation, at the intersection of rapidly developing technologies such as Internet, mobile communication, and renewable energies, has changed considerably over the course of the past years. The developments relate to all major aspects of a smart home, such as

- capabilities of home infrastructure and controlled device
- usability of mobile and stationary user interfaces
- motivation for investing in automation and control technologies

Up to recently, home automation was mainly focused on installing controllable power-outlets or light switches and wiring infrared (IR) controls around the house. Technologies developed in the early seventies of the past century, which from today's perspective are slow, unreliable, and insecure, were at the heart of building control.

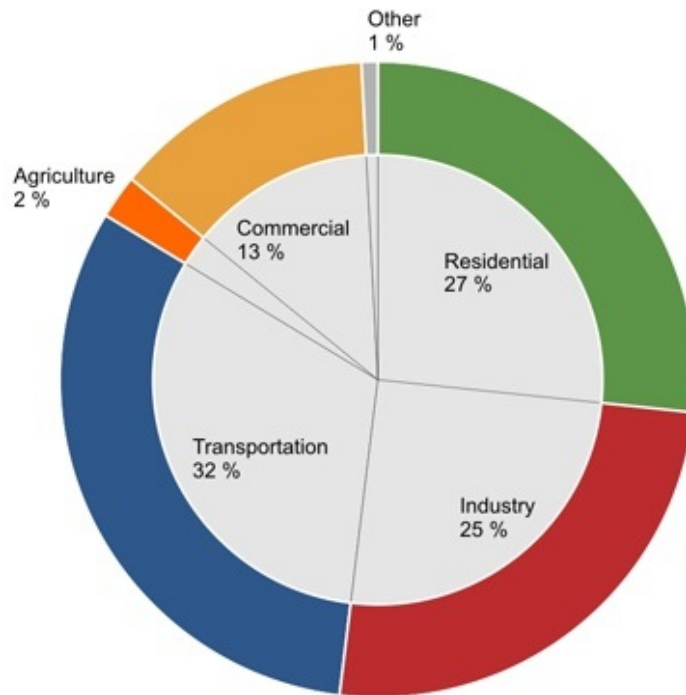
The rapid developments in mobile communications have introduced a technological leap forward in home automation. Wireless networks (3G, 4G, Wi-Fi) and smart devices, with wireless communication interfaces (Bluetooth, ZigBee, Wi-Fi), are omnipresent, and allow the user to take home control and building automation to the next level. In addition some of the largest technology companies for consumer devices and services such as Apple (HomeKit) or Google (Nest Labs) have added home automation to their portfolio. Instead of simply switching power outlets on and off, specific and meaningful functions of consumer electronics, household devices, or infrastructure components can be stirred. As a result, instead of rudimentary functionality, home automation today can deliver capabilities that have a real impact on comfort, security, and energy conservation in residential and industrial buildings.

Of similar significance to the changes in what is possible in home automation have been the advances in user interface. The smartphone and tablet revolution has finally brought the personal, universal remote control device to the home. Proprietary, stationary panels and control devices are phasing out, being replaced by apps, which are easy to operate, to maintain, and to upgrade.

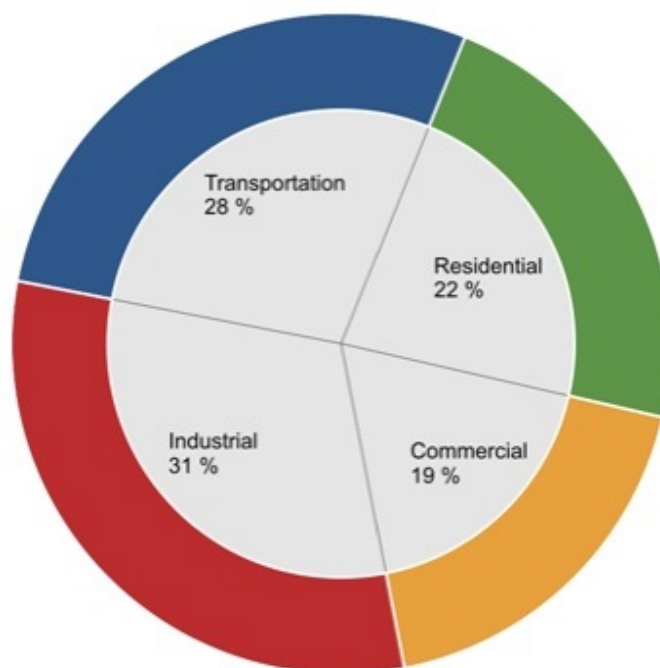
With the improved usability and capabilities, the motivations for installing home intelligence have become broader as well. The vision of a green building, capable of significantly reducing energy and water consumption, is finally becoming real. Other new applications are safety management, home automation for the elderly and disabled (assistive domotics) and remote building control.

## 2.1 The Potential for Energy Conservation

Looking at the distribution of total energy consumption, the share of the private sector is significant. In the 27 European Union countries (EU-27), in 2010, 27% of the total energy was consumed by the residential sector, in the US 22% (Figure 2.1). Thus, energy conservation in homes does move the needle even from a global perspective. And the savings potential for all energy forms used in the private sector is large. Space heating and cooling takes the largest share with between 50% and 70% of the total residential energy usage. Water heating takes second place, followed by electric appliances and lighting (Figure 2).

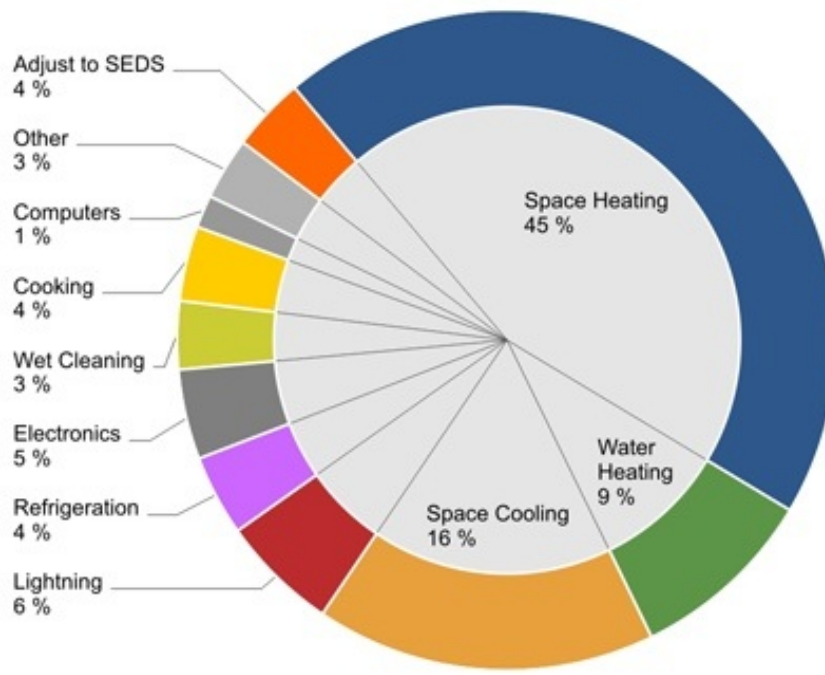


EU-27 Energy Consumption by Sector 2010

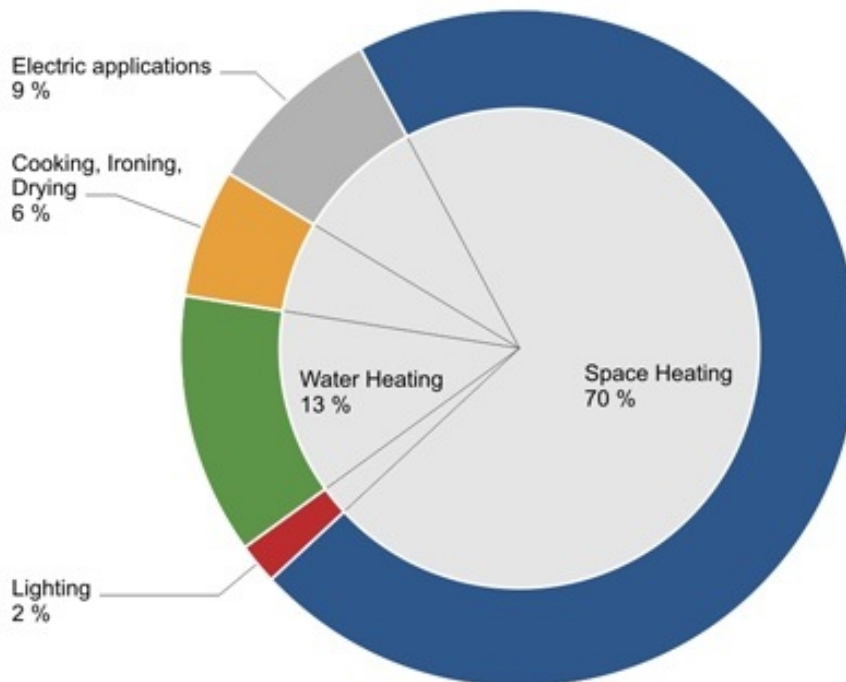


US Energy Consumption by Sector 2010

Figure 2.1 US and EU-27 Energy Consumption per Sector 2010 Source: EuroStat 2012, US Department of Energy, September 2012



2010 Residential End-Use Energy Split US



2010 Residential End-Use Energy Split Germany

Figure 2.2 Residential End-Use Energy Split US, Germany 2010 Source: Federal Statistic Bureau Wiesbaden November 2012, US Department of Energy, March 2012

There are several approaches for reducing energy consumption, all of which should be noted:

- building insulation
- state of the art appliances
- efficient water heating and space heating systems
- building automation and control

With the advances in home automation as described above, the last one in the list, building automation, has become an increasingly attractive choice, providing the opportunity for significant savings with relatively low upfront investment. Smart appliances coordinate their operation with smart meters (home gateways), reducing overall energy consumption and avoiding load peaks. Monitoring current and past power consumption and identifying load profiles provide the basis for intelligent power management with capabilities such as:

- Intelligent heating control by automatically managing room temperature based on time, outside temperature, and presence
- Smart lighting system, managing illumination based on presence detection, sunrise, or sunset timing and room function
- Intelligent, proactive blinds, keeping the interior of the building cool or warm
- Monitoring and management of electricity consumption
- Reducing water consumption through sensor faucets and intelligent plant watering management



## 2.1.1 Calculating Actual Building Automation Energy Savings

Studies report electricity savings of up to 30% using automated lighting as well as heating energy savings of 15%-20% using automated heating in residential buildings. But how much is it that you can really conserve by implementing a smart home? Answering this question was the task of a major standardization effort in Europe, which has come up with a comprehensive specification on how to measure and calculate building automation based energy savings: The European standard EN15232:“Energy performance of buildings - Impact of Building Automation, Control and Building Management”. For the first time, EN15232 specifies standardized methods to assess the impact of Building Automation and Control Systems (BACS) on the energy performance of these different building types:

- offices
- lecture halls
- education
- hospitals
- hotels
- restaurants
- wholesale & retail
- residential

The performance of building automation is categorized in four classes (A-D), A representing the highest performance building automation, D the lowest.

For each building type and each BACS Class, so called BACS Factors are given, with which the thermal and electrical energy savings can be calculated. Table 2.2 shows the description of the four BACS classes and the BACS factors for the different building types. Table 2.1 displays the percentage of thermal savings by installing building automation of efficiency class A and B in reference to the standard class C.

BACS Class	Thermal Energy				Electrical Energy			
	D	C	B	A	D	C	B	A
Offices	1,51	1,00	0,80	0,70	1,10	1,00	0,93	0,87
Lecture hall	1,24	1,00	0,75	0,50	1,06	1,00	0,94	0,89
Education	1,20	1,00	0,88	0,80	1,07	1,00	0,93	0,86
Hospitals	1,31	1,00	0,91	0,86	1,05	1,00	0,98	0,96
Hotels	1,31	1,00	0,85	0,68	1,07	1,00	0,95	0,90
Restaurants	1,23	1,00	0,77	0,68	1,04	1,00	0,96	0,92
	1,56	1,00	0,73	0,60	1,08	1,00	0,95	0,91
Residential	1,10	1,00	0,88	0,81	1,08	1,00	0,93	0,92

*Table 2.1 Thermal savings of BACS Class A and B by BACS C for office and residential buildings (1)*

<b>Class A</b>	<p><b>High energy performance BACS</b></p> <ul style="list-style-type: none"> <li>Networked room automation with automatic demand control</li> <li>Scheduled maintenance</li> <li>Energy monitoring</li> <li>Sustainable energy optimization</li> </ul>
<b>Class B</b>	<p><b>Advanced BACS and some specific TBM functions</b></p> <ul style="list-style-type: none"> <li>Networked room automation without automatic demand control</li> <li>Energy monitoring</li> </ul>
<b>Class C</b>	<p><b>Corresponds to standard BACS</b></p> <ul style="list-style-type: none"> <li>Networked building automation of primary plants</li> <li>No electronic room automation,thermostatic valves for radiators</li> <li>No energy monitoring</li> </ul>
<b>Class D</b>	<p><b>Non energy efficient BACS</b></p> <ul style="list-style-type: none"> <li>Without networked building automation functions</li> <li>No electronic room automation</li> <li>No energy monitoring</li> </ul>

*Table 2.2 BACS class definition*

(1) Angelo Baggini, Lyn Meany “Application Note Building Automation and Energy Efficiency: The EN 15232 Standard”

## 2.1.2 Smart Grids need Smart Buildings

Finally, smart homes allow for integration with smart power grids, which are in build out around the world, driven by renewable energy generation on the rise. Smart meters and smart gateways can only work if a home control and automation infrastructure is in place. This infrastructure then can interact with the supply and demand driven electricity cost in smart power grids. Wind and sun based renewable energy generation introduces significant energy level fluctuations in the utilities' power grids. Thus, for example, it can make sense to cool down the freezer two or three degrees below normal operation during times of high wind, so it can stay off longer in times of the day with lower energy supply.

By being able to continuously monitor energy levels and prices in a smart power grid, and by scheduling (delay or early start) high energy processes such as

- heating up the hot water tank
- operating the dish washer and washing machine
- cooling the freezer and refrigerator,

smart meters can contribute significant energy savings without impacting the comfort level of residents.

## 2.2. Safety Management and Assistive Domotics

Another application for state of the art home automation is remote building control and safety management with features such as

- Controlling the vacant home (temperature, energy, gas, water, smoke, wind)
- Feeding and watching pets
- Watering plants indoors and outdoors
- Presence simulation to keep out intruders
- Assistive living systems (assistive domotics), allowing elderly and handicapped people to stay home safe through reminder systems, medication dispensing, blood pressure and pulse monitoring and emergency notification.

## 2.3 Changing the World (a bit) to the Better

Smart home and building automation has come a long way. Technological advances, climate change, and demographic transition have redefined intelligent homes from a futuristic niche for geeks and luxury home owners to an integral part of the life of millions. Home automation standardization based on open Internet technologies and omnipresent smartphones, ready to control the world, have been the catalyst for manufacturers to start integrating control functionality in their products as a default. So, (finally) everything is there, that is needed for an intelligent home. We can now take a look on how to put things together and to change the world (a bit) to the better.



## Bibliography

“Buildings Energy Data Book”. US Department of Energy, March 2012

<http://buildingsdatabook.eren.doe.gov/TableView.aspx?table=2.1.5>

“Annual Energy Review 2011”. US Department of Energy, September 2012

<http://www.eia.gov/aer>

“Energieverbrauch der privaten Haushalte für Wohnen”. Statistisches Bundesamt, Wiesbaden, November 2012

<https://www.destatis.de/DE/ZahlenFakten/GesamtwirtschaftUmwelt/Umwelt/Umweltoeko>

-

“Final energy consumption, by sector.” Eurostat European Commission, April 2012

[http://epp.eurostat.ec.europa.eu/portal/page/portal/energy/data/main\\_tables](http://epp.eurostat.ec.europa.eu/portal/page/portal/energy/data/main_tables)

Angelo Baggini, Lyn Meany. “Application Note Building Automation and Energy Efficiency: The EN 15232 Standard”, European Copper Institute, May 2012

<http://www.leonardo-energy.org/good-practice-guide/building-automation-and-energy-efficiency-en-15232-standard>





# 3 Key Concepts

From a technical perspective, Home Automation consists of five building blocks:

- devices under control (DUC)
- sensors and actuators
- the control network
- the controller
- remote control devices.

## 3.1 Devices under Control

Devices under control are all components, such as home appliances or consumer electronics, which are connected to and controlled by the home automation system. An increasing number of components come with built in functionality (Web-servers, WLAN-, Bluetooth-, Z-Wave-interfaces, etc.), which allow for direct connectivity to the control network. Other components need to be equipped with adapters in order to integrate them with the smart home infrastructure.

## 3.2 Sensors and Actuators

Sensors are the eyes and ears of the home network. There are sensors for a wide range of applications such as measuring temperature, humidity, light, liquid, and gas and detecting movement or noise.

Actuators are the hands of the home network. They are the means of how the smart network can actually do things in the real world. Depending on the type of interaction required, there are mechanical actuators such as pumps and electrical motors or electronic actuators such as electric switches and dimmers.

## 3.3 Control Networks

The control network provides the connectivity between devices under control, sensors, and actuators on the one hand and the controller along with remote control devices on the other hand. There are three main technology options for home and building automation control networks today:

- Power-line Communication
- Wireless Transmission
- Wireline Transmission

### 3.3.1 Power-line Communication

The power line communication principle uses existing electric power lines in buildings to transmit carrier wave signals from 20 kHz to 100 MHz. The long dominant, decades old, low speed power line standard X.10, while still widely installed, has been finally replaced by the high performance HomePlug standard, which became the IEEE 1901 standard in 2010. The latest version AV2 of the specification is able to achieve transmission speeds of up to 500 Mbit/s. A key advantage of power line communication is the low price for its components and the fact, that no additional wiring is required. One disadvantage of the technology is that power line distribution units can impact transmission speeds. In some cases the design of the electric wiring can even prohibit the coverage of parts of the electric power line infrastructure in a building. The manufacturers, service providers, and retailers which support the HomePlug standard have formed the HomePlug Power-line Alliance to foster the deployment of the technology. (<http://www.homeplug.org>)



EnOcean				902MHz (North America), 868MHz (Europe)	ASK	ISO/IEC 14543-3-10	N/A
DASH7 (active RFID)	2004	- 1000m	200kBit/s	433MHz	GFSK	ISO/IEC 18000-7	1m
Thread	2015	30m - 500m	250 kBit/s	2.4GHz	QPSK	IEEE 802.15.4(*)	10m(**)
HAP	2014	10m	1MBit/s	2.4GHz	GFSK	BLE	< 1m

*Table 3.1 Wireless Building Automation Standards*

(\*)LR-WPAN (Low Rate Wireless Personal Area Networks)

(\*\*) heavily depends on topology, frequency and distortion of the sensor

### 3.3.3 Wire Line Building Automation

The two main open standards for wire line based building automation are KNX and LON. KNX is a European (EN50090, 2003) and international (ISO/IEC 14543-3, 2006) standard for home and building automation. The abbreviation KNX stands for Konnex, and replaces the older European standards EIB (European Installation Bus), Batibus (primarily used in France), and EHS (European Home Systems). Today in Europe, more than 75% of industrial building automation solutions as well as upscale residential smart homes are realized using KNX. Over the past years, KNX has started to be adopted in many regions of the world outside of Europe as well. In the US KNX was approved as the US Standard ANSI/ASHRAE 135 in 2005. In 2006, CEN approved KNX as EN 13321-1 and in the same year KNX technology was approved as the International Standard ISO/IEC 14543-3. KNX technology was also approved as the Chinese Standard GB/T 20965 in 2007.

LON (Local Operating Network), originally introduced in 1990 by Echolon Corporation and an ISO/IEC 14908 standard since 2008, is the building automation solution of choice for large scale automation projects such as airports, stadiums, or street lightning. Contrary to the hierarchical KNX architecture, it uses a decentralized approach. In large installations, local information can be processed locally, without being sent to a central control node. This allows for the scalability and redundancy needed in public installations with high availability requirements.



### 3.3.4 Control Networks Summary

All three control network technologies — power-line, wireless, and wireline based — have significantly improved in transmission speed, reliability and interoperability through standardization efforts over the past ten years. In general, control networks based on power line communication and wireless transmission are dominant in residential home automation due to lower component prices and installation cost. Wire line control networks, on the other hand, are found in the premium residential segment and in industrial building control applications.

## 3.4 Controller

The controller is the computer system which acts as the brain of the building automation system. It collects information through sensors and receives commands through remote control devices. It acts based on commands or a set of predefined rules using actuators or means of communication such as loud speaker, email, or telephone. For residential home automation, the controller typically is an “always-on” standalone or embedded Linux / Windows / OS-X PC, running the control application for the house. Higher end residential and industrial buildings use dedicated high availability, redundant controller systems with uninterruptible power supplies (UPS).

## 3.5 Remote Control Devices

One of the main reasons for the increased acceptance of home automation systems in the residential segment is that, with the omnipresence of smart phones and tablets, the need for dedicated automation control devices has vanished. Within a few years, literally all home automation systems on the market have introduced smartphone and tablet based control applications. In addition, advances in voice recognition have finally brought voice based control to smart homes as well. The remote control devices act by connecting to the home automation application on the home controller. They do this either by connecting to the controller through the control network itself, or through any other interface the controller provides, such as WLAN, the Internet, or the telephone network. Thus, the use of smartphones makes the capability of remote building control via Internet or the mobile telephone network a feature which is available by default.

## 3.6 Market Trends

The traditional differentiation between expensive, proprietary building control systems and residential smart homes is blurring. Over the past ten years these two market segments have changed drastically and are increasingly overlapping. Expensive proprietary solutions have become more open standards based and less expensive. Low end solutions for residential customers have become more sophisticated and are using the same technologies as industrial systems. (This is similar to what happened when the markets for professional and home PCs blended a few decades ago.)

While the requirements for reliability, redundancy and robustness of professional building control systems have led to the development of many proprietary standards, now the pace of the digital evolution has caught up with these requirements. In addition, new requirements for smart building control are arriving at a speed which proprietary standards cannot match anymore. Recent examples are

- integration of smart grid and smart meters
- tariff based energy management
- integration of web/IP enabled home appliances
- integration of web/IP enabled consumer electronics
- integration of Internet based information and services such as supply & demand based tariffs from utilities or weather and traffic information from dedicated websites

## 3.7 Smart Homes for the Masses: Google, Apple, Samsung and more ...

While the smart home market has experienced double digit growth rates in recent years, it was the year 2014, by when it truly became mainstream. It was the year when three of the largest consumer product and service companies made bold entries into the smart home market. Apple introduced it's HomeKit architecture, Samsung spent more than 200 million US\$ for home automation startup SmartThings and Google acquired learning thermostat maker Nest Labs for 3.2 billion US\$.

### 3.7.1 Google's Nest Labs

Nest Labs was founded in 2010 by two former Apple engineers with the focus on learning thermostats. The key innovation of Nest Labs centers around the fact that most people do not program their thermostats because it is too complicated. Nest thermostats automatically create a heating (cooling) schedule based on the daily routines of the residents. Initially the residents frequently set the target room temperature by turning the Nest thermostat wheel several times a day. Storing these settings the thermostat is capable of building a temperature schedule. Nest thermostats have to be connected to the Internet to receive software updates. Since part of their function is based on their location determined by the US zip-code, international deployment is limited. At the end of 2014 Nest acquired streaming video camera maker Dropcam and has since integrated its products with Dropcam's surveillance capabilities. Dropcam recordings can now be triggered by Nest smoke detector alarms and Dropcam motion alerts are turned on when Nest thermostats are being set to „away“.

Nest devices communicate using Nest Lab's Thread protocol (<http://www.threadgroup.org>), which is based on the 6LoWPAN standard (IPv6 over IEEE 802.15.4 LR-WPAN). With that it uses the same transport protocol as ZigBee and WirelessHART. Existing 802.15.4 products like the Philips Hue lamps could be upgraded to the Thread protocol via a software update.

### 3.7.2 One More Thing ... Apple HomeKit

With its HomeKit framework Apple has made a strategic move to enter the smart home market. The large installed base of smartphones and tablets with the powerful voice assistant Siri provide the platform for a basic, easy to use, plug and play type smart home solution. The core of Apples HomeKit consists of the three components

- home configuration database
- HAP HomeKit Accessory Protocol
- API for HomeKit Apps

As transport protocol Apple has specified IP (LAN, WiFi) and BLE (Low Energy Bluetooth). Using the HomeKit API third party developers can build iOS applications, which discover HomeKit compliant accessories and add them to the home configuration database, access the database and communicate with configured accessories and services. In addition to iOS applications Apples voice assistant Siri has also access to HomeKit, allowing for voice based smart home control.

Accessories which are not HomeKit compliant can connect to the HomeKit infrastructure through bridging devices (HomeKit Bridges). However this approach is limited to accessories which

- offer no user control
- have no physical access (such as door locks)
- and which use non competing transport layer technologies such as ZigBee or Z-Wave

This basically restricts HomeKit bridging to simple sensors, which do not use WiFi or BLE. All WiFi or BLE based sensors as well as all smart home components which offer active user control (e.g. thermostat controllers, light switches, door locks) will have to implement the HAP protocol and enter the Apple MFi (Made-for-iPhone/iPad ) program. Software bridges to integrate HomeKit with wireline smart home technologies such as KNX or HomePlug are currently not on the roadmap. The Apple TV hardware is taking over the part of the smart home hub for remote access to smart home accessories. With the addition of HomeKit capabilities it functions as a relay between the local smart home accessories and a HomeKit cloud account, which in turn can be accessed by the smartphone HomeKit app from anywhere (Figure 3.1).

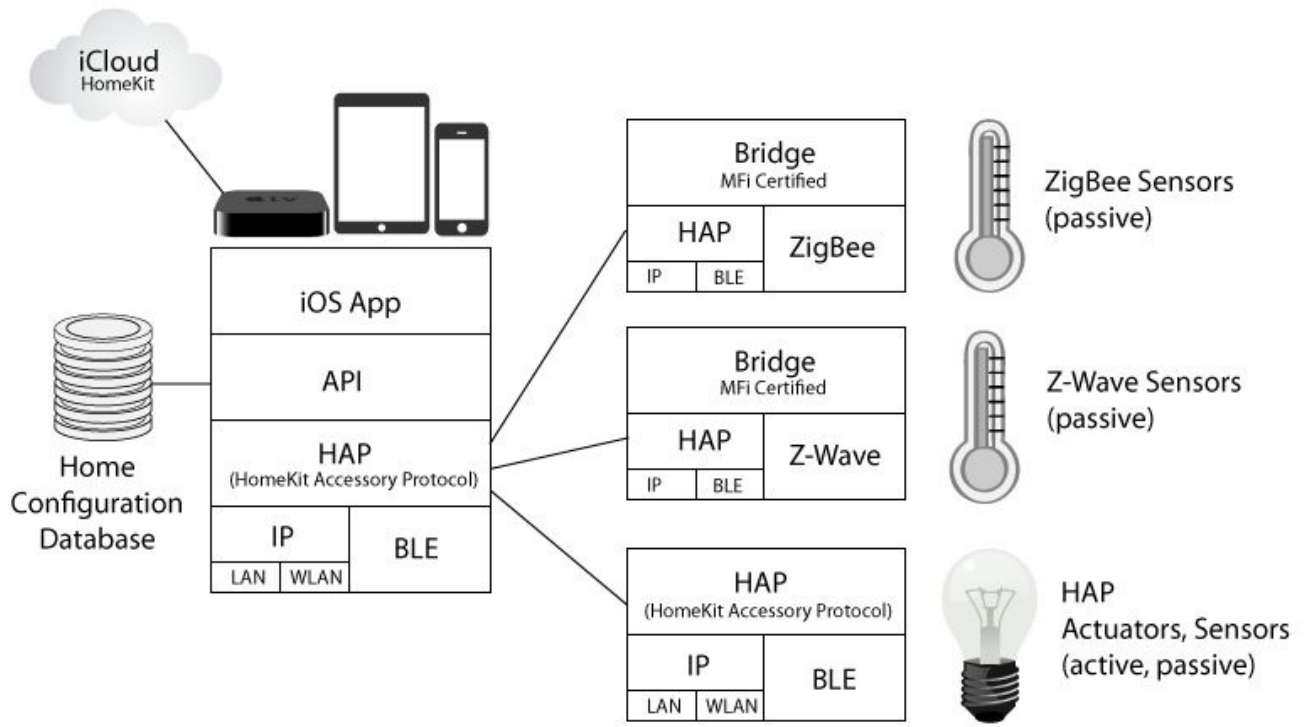


Figure 3.1 Apple's smart home framework HomeKit



### 3.7.3 Samsung's SmartThings

The third large consumer products company in 2014 to make a serious effort towards smart home technologies was Samsung with its acquisition of US startup SmartThings (<http://www.smarthings.com>). Core of the SmartThing solution is an easy to use smartphone app (iOS, Android), which communicates to the SmartThing Hub, which in turn controls Z-Wave and Zigbee compliant smart home accessories. The SmartThings Hub can directly communicate with the smartphone app as long as it is within its range. In parallel it connects to a cloud account, which serves as the communication hub when communicating with the building from away.

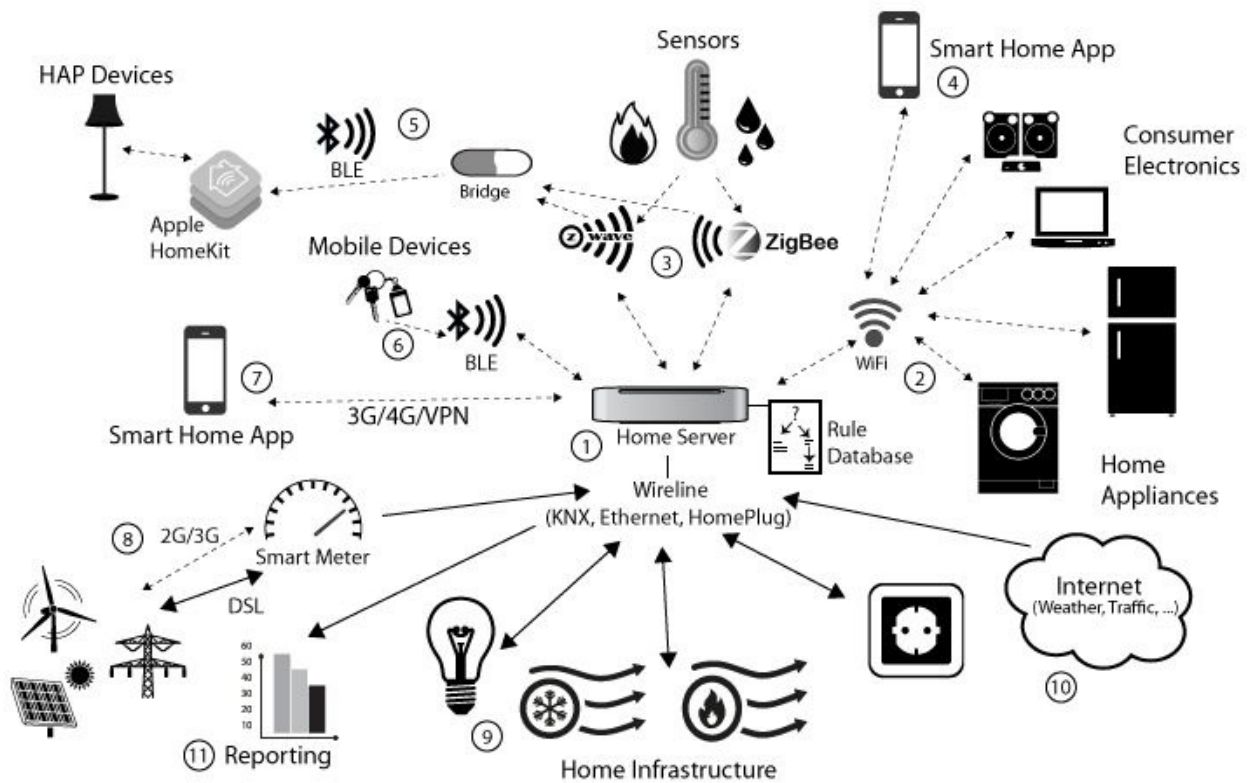
## 3.8 A Future Proof Smart Home Architecture

In spite of the trends towards open standards, for the realization of smart home projects the variety of wireline and wireless standards in combination with proprietary vendor solutions remains a challenge. Any architecture with the objective to go beyond point solutions which control garage doors or lights using a smartphone will need to be built upon a central, rule based home server, capable of connecting to devices via multiple technologies. In most homes at least part of the control infrastructure will be based on WLAN (WiFi) and wireline technologies for the foreseeable future. Examples are the latest generation of consumer electronic devices such as audio equipment, TV sets and appliances (ovens, refrigerators, dish washers, washing machines) which are all equipped with WLAN interfaces ready to be integrated in smart home infrastructures. The full line of WiFi connected appliances from General Electric (<http://www.geappliances.com/connected-home-smart-appliances/>), Samsung's series of SmartTV sets, or the Denon and Marantz music systems are just a few representatives for this market reality. In newly built residential homes as well as in commercial and public buildings for security and reliability reasons wireline technologies will continue to serve as the backbone for the control of key building infrastructure elements such as power outlets, lighting and HVAC (heating, ventilation and air conditioning). The smart home solutions of new market entrants such as Nest, Apple or the many new smart home startups typically provide point solutions for specific needs. They are capable of providing a quick and inexpensive initial step towards home automation with restricted functionality and limited customization capabilities. Most of these solutions can be at least partially integrated in server based scalable multi technology solutions such as OpenRemote. However, their reliance on wireless technology and cloud based control limits their use from a security and reliability perspective. An example is the attenuation of the popular 2.4 GHz frequency band through rain and plants. The 900 MHz band on the other hand suffers from low data rates. And last but not least, any wireless technology is prone to denial of service attacks through signal jammers, which anybody can buy on the Internet for a few dollars.

Equally critical from a security and privacy perspective is the usage of cloud accounts for smart home control and communication. A prominent example is Nest Labs, which officially passes user data stored on the thermostats of its customers to its parent company Google. And even independent of the wide spread practice of selling customer data and profiles, cloud based solutions represent a significant risk from a security and reliability perspective. It is a fact that cloud based services frequently suffer from outages caused by technical problems or hacker attacks. Just take a look at [InfoWorld's](#) annual listing of the top ten cloud service outages.

To summarize, a reliable and secure architecture needs to be based on a local smart home controller with wireline control links to the key home infrastructure components such as power outlets, lighting, HVAC, surveillance and door locks. Since most consumer electronics and appliances with integrated power supply will continue to offer connectivity using WiFi, WLAN integration is mandatory. In addition a new generation of mobile, battery powered devices are coming to the market, which provide connectivity through

low power technologies such as BLE or EnOcean. Some of these devices are based on proprietary vendor implementations such as Apple's HomeKit or Nest's Thread protocol. The degree to which they can or should be integrated in an overall smart home architecture needs to be looked at on a case per case basis. Perhaps the biggest downside of these proprietary plug and play solutions is their lack of customization capabilities. The price for being easy to install and for using proprietary technology is, that they cannot be used to build an integrated rule base, which delivers meaningful interaction between residents, environment and building infrastructure. And without that the fact that the garage door can now be opened using smart phone based voice control cannot conceal that such a smart home solution is a mere remote control. Figure 3.2 shows how an integrated smart home architecture as discussed above could look like.



- |    |   |
|----|---|
| 1  | 24/7 Home controller with rule database interfacing actuators and sensors via multiple technologies   |
| 2  | WiFi (WLAN) network interfacing to power supply based devices (consumer electronics, home appliances)   |
| 3  | 2nd generation wireless technologies (z-wave, ZigBee) interfacing to small devices and sensors  |
| 4  | Local smartphone / tablet based home control app connected to controllable devices through the home server  |
| 5  | Proprietary smart home components (e.g. Apple's HomeKit) connected to select devices and integrated to the home network using bridges   |
| 6  | 3rd generation low energy wireless technologies (e.g. Bluetooth LE, EnOcean) connected to mobile, battery powered devices providing services such as location tracking.   |
| 7  | Remote smartphone based home control app connected to the home server via 3G/Internet/VPN connections   |
| 8  | Smart meter reporting on supply & demand based tariffs and the availability of local generated power (e.g. roof based solar energy). Connectivity to utilities via 2G/3G or DSL/Internet and to the home server via wireline connections. |
| 9  | Wireline based control of the building infrastructure (Lighting, HVAC, wall outlets, door locks, alarm system)  |
| 10 | Integration of data from public and private Internet based services (calendar, sport, medical, weather, traffic, etc.) to the home automation rule base.  |
| 11 | The automatic generation of reports on key operating parameters plays a vital role in monitoring and optimizing building operation and the continued development of the automation rule base.   |

Figure 3.2 An integrated smart home architecture

## 3.9 Where do we go from here?

With the above trends and developments, the slow moving market of building automation has changed radically. New players and start-ups are taking on the opportunities, which the intersection of new technologies and new demands are offering in home automation. It looks like that finally the vision of an „Internet of Things“, which seaming less integrates with our life, handling daily routines while saving energy, is becoming reality.



## Bibliography

“IEEE Standard for Local and metropolitan area networks Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)”. IEEE Computer Society, June 2011

<http://standards.ieee.org/getieee802/download/802.15.4-2011.pdf>

“Recommendation ITU-T G.9959 Short range narrow-band digital radio communication transceivers – PHY and MAC layer specifications”. International Telecommunication Union, February 2012

<http://www.itu.int/rec/T-REC-G.9959-201202-I/en>

“EnOcean Wireless Standard ISO/IEC 14543-3-10”. EnOcean Alliance, May 2013

<http://www.enocean-alliance.org/en/home/>



# 4 The Project



## 4.1 Overview

Complex functionality in information technology can be explained best using an incremental approach, starting from simple “hello-world” type of functionality to sophisticated features in sequential steps, each of which can be tested and demonstrated individually. In software engineering terms, this approach is called a development sprint. Sprints are relatively small coding modules, which need to be designed in a way that they can be demonstrated independent of other development elements (sprint demonstration) once their implementation is finished. The advantage of the sprint approach is that functionality is continuously validated. Problems are recognized early and remain manageable. The continuous monitoring of the growing functionality avoids surprises and keeps the fun factor high. In following this philosophy, most design phases of our project are independent from each other and work stand alone. However, some components do build upon others, so it does make sense to follow the sequence as outlined below for the most part.

We will start with installing and configuring the open source building control platform OpenRemote. (Chapter 5). This will allow us to build a customized smartphone and tablet control app in less than one hour with no programming skills required. Later, the OpenRemote controller will also be used to run the automation rules, which we will build during the course of the project.

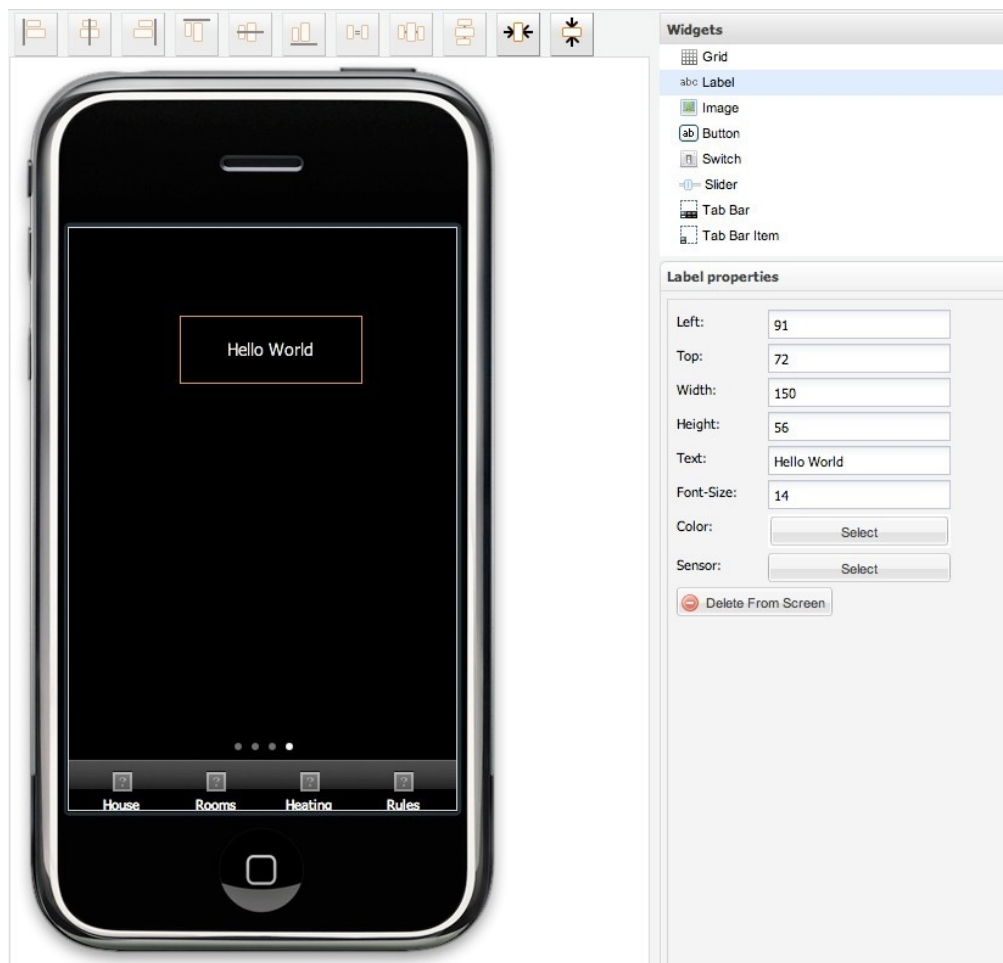
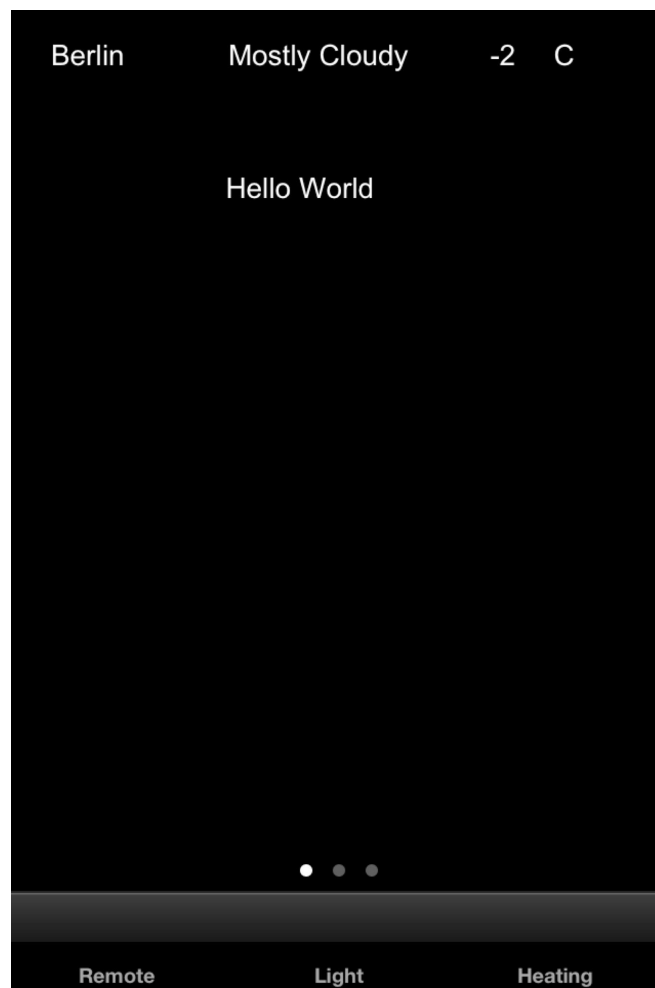


Figure 4.1 Project phase 1: “Hello World” on the custom smart home app

In chapter six we will configure our first sensor and connect it to the smart home

application. Specifically, we will be polling weather condition and temperature for a specific location from the Yahoo Internet weather service and displaying it with our smartphone and tablet app (Figure 4.2).



*Figure 4.2 Project phase 2: Retrieving and displaying weather information*

Chapter 7 adds presence control. We will configure our home network WLAN (Wi-Fi) to detect the registration of a particular smartphone, which we will use to trigger a welcome home scenario in the next phase.

In chapter 8 we will integrate the control of multimedia PC functions through iTunes (both on PCs and Macs) to our OpenRemote based smart home infrastructure.

In chapter 9 we introduce the automation rules capabilities of OpenRemote. We will use the components, we have built so far, to put together an intelligent wake up scenario: “Wake me up early in case it rains or snows”. The idea is to start a morning wake up scenario 45 minutes earlier than normal in case of nightly rain or snowfall, to avoid the potential traffic jam. For that, we will use our Internet weather sensor to poll the weather conditions during the night, and, once a wake up condition is met, our scenario will start playing music. Another scenario will be “Welcome Home”, which uses the smartphone triggered presence detection to start playing the iTunes playlist of choice for the person returning home. We will even give our smart home a voice and have it read reminders and appointments for the day to us via its Hi-Fi stereo.

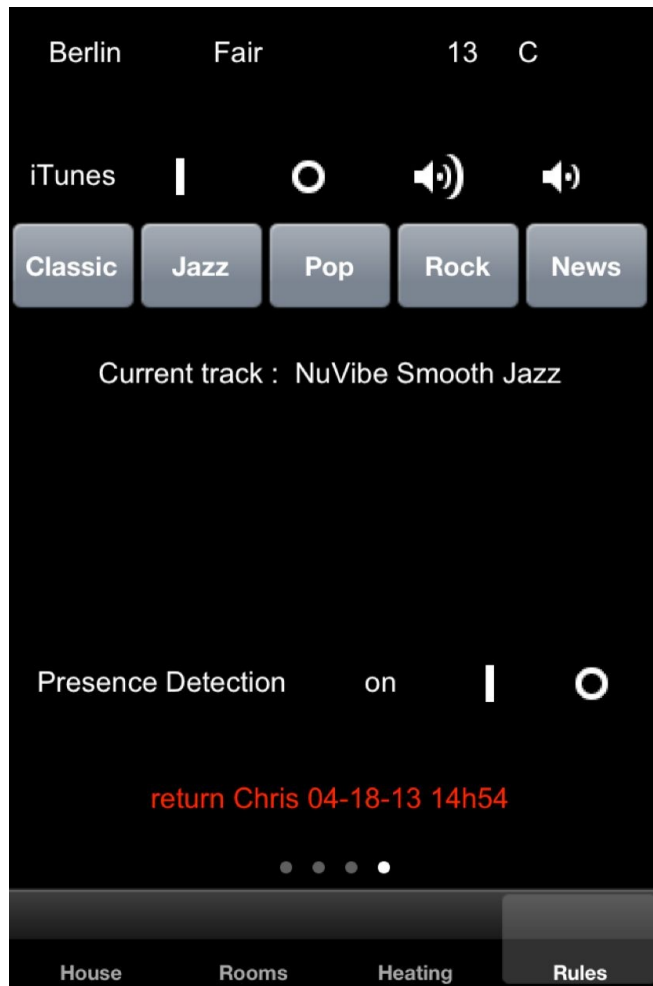


Figure 4.3 Project phase 3: Adding controls for iTunes

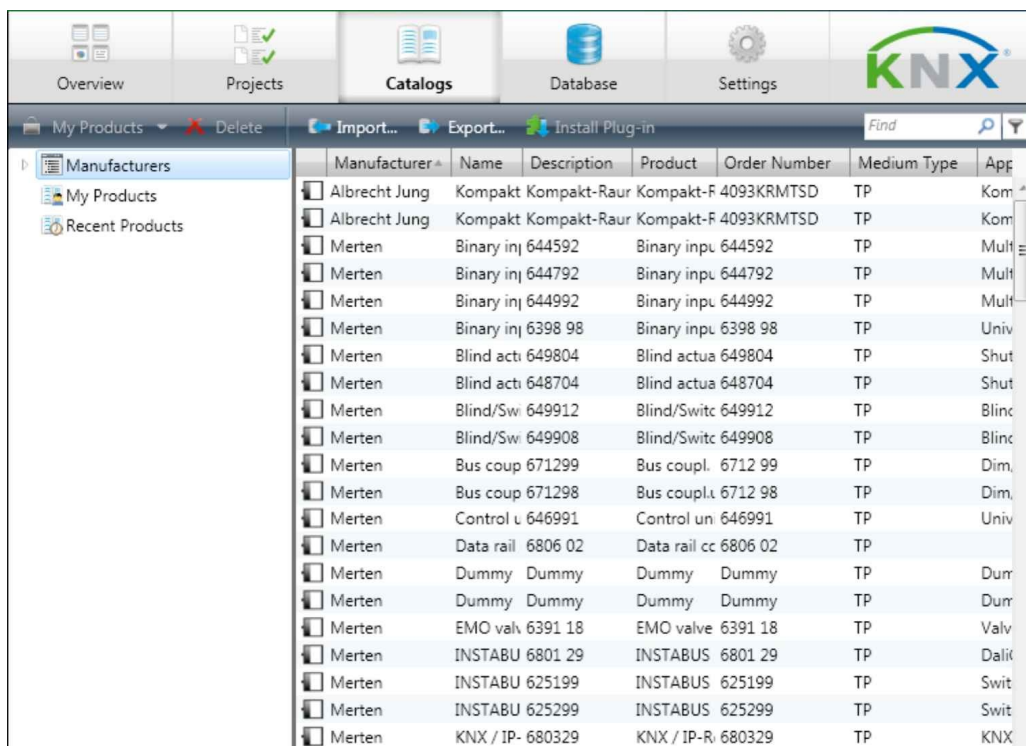


## Figure 4.4 Project phase 4: Controlling lights and power outlets

Up to this phase all you need for following and implementing the project is a Mac or PC, a Wi-Fi network and an Internet capable smartphone. From here on, you need the components of the technology you have chosen for your smart home, such as Z-Wave or KNX sensors and actuators.

In chapter 10, we add wireless light and power outlet control to our project, using the popular Z-Wave standard. In addition, we accomplish the integration of consumer electronics hardware using a Denon audio video receiver (AVR 3313), as an example.

In the final project phase (chapters 11 and 12), we integrate KNX based infrastructure components for heating and lighting. In a step-by-step fashion, we explore how to download, install and configure ETS, the official KNX association control software. Then, we fully integrate the KNX controls with our OpenRemote project.



Manufacturer	Name	Description	Product	Order Number	Medium Type	App
Albrecht Jung	Kompakt Kompakt-Raur	Kompakt-F 4093KRMTSD	TP	Komp	Komp	
Albrecht Jung	Kompakt Kompakt-Raur	Kompakt-F 4093KRMTSD	TP	Komp	Komp	
Merten	Binary inj 644592	Binary inpl. 644592	TP	Mult	Mult	
Merten	Binary inj 644792	Binary inpl. 644792	TP	Mult	Mult	
Merten	Binary inj 644992	Binary inpl. 644992	TP	Mult	Mult	
Merten	Binary inj 6398 98	Binary inpl. 6398 98	TP	Univ	Univ	
Merten	Blind acti 649804	Blind actua 649804	TP	Shut	Shut	
Merten	Blind acti 648704	Blind actua 648704	TP	Shut	Shut	
Merten	Blind/Sw 649912	Blind/Swite 649912	TP	Blin	Blin	
Merten	Blind/Sw 649908	Blind/Swite 649908	TP	Blin	Blin	
Merten	Bus coup 671299	Bus coupl. 6712 99	TP	Dim.	Dim.	
Merten	Bus coup 671298	Bus coupl. 6712 98	TP	Dim.	Dim.	
Merten	Control u 646991	Control un 646991	TP	Univ	Univ	
Merten	Data rail 6806 02	Data rail cc 6806 02	TP			
Merten	Dummy Dummy	Dummy Dummy	TP	Durr	Durr	
Merten	Dummy Dummy	Dummy Dummy	TP	Durr	Durr	
Merten	EMO valv 6391 18	EMO valve 6391 18	TP	Valv	Valv	
Merten	INSTABU 6801 29	INSTABUS 6801 29	TP	Dalit	Dalit	
Merten	INSTABU 625199	INSTABUS 625199	TP	Swit	Swit	
Merten	INSTABU 625299	INSTABUS 625299	TP	Swit	Swit	
Merten	KNX / IP- 680329	KNX / IP-R 680329	TP	KNX	KNX	

## Figure 4.5 Project phase 6: Adding KNX control

With that done, our smart home control system will be capable of:

- smartphone / tablet based display of weather and temperature
- WLAN / smartphone based presence control
- smartphone / tablet based control of lights, heating, power-outlets, consumer electronics
- smartphone / tablet based scenario control for scenarios such as Good Morning, Welcome, Good Night, Leaving Home
- operation of an audio reminder system with text-to-voice conversion of calendar items
- rule based scenario execution triggered by time, date, weather condition, temperature, WLAN/smartphone based presence detection

- automatic daily temperature report in CSV format sent via email
- heartbeat function to monitor system availability
- watchdog monitoring of the heartbeat with automatic alert email

Sensing approaches such as presence detection based on smartphones or weather condition information retrieved from the Internet provide just a glimpse of what state of the art home automation based on open standards is capable of delivering. Using the functionality of our project as a start, a vast variety of variations and add-ons can easily be implemented.

## 4.2 Equipment and Prerequisites

In general you will find that in order to implement smart home controls with functions beyond switching power outlets and lights, as we demonstrate it, you need relatively new equipment. This is true for your WLAN (Wi-Fi) router, for the appliances and consumer electronic devices you want to control as well for the mobile clients (smartphones and tablets) you plan to use. Fortunately, prices for all of the above have gone down over the past years. Thus, in many cases, you might rather want to upgrade the equipment you have to the latest generation than compromising and spending a lot of effort to integrate legacy equipment. Of course, there are always also good reasons not to upgrade. Everyone will have to make that decision on an individual base.

In order to be able to follow the project in this book, you will need the following, obviously depending on which functionality you plan to implement:

- a home network with Internet access and a WLAN/DSL router
- An iOS or Android powered smartphone or tablet
- A Mac OS X or a Windows XP/7 PC with iTunes installed
- Z-Wave components you plan to use to control power-outlets, lighting , etc.
- KNX components you plan to use
- consumer electronic devices with LAN / WLAN capability build in

Alternative to WiFi, Z-Wave or KNX, the usage of other building control standards such as 1-Wire or X10 for the projects described in the book is also possible, although not described in detail. The building automation platform OpenRemote, which we use throughout this book, supports most major building control standards. (Table 4.1).

In addition to the above equipment, some familiarity with computer and network technology is recommended. You do not have to be able to actually write code. However, if you have never heard about IP, Telnet or HTTP, and if you have never edited a batch file (.bat) or a shell script (.sh), you will probably have to go through a steeper learning curve than others. On the other hand, with the thousands of good Internet tutorials just a mouse click away, there is nothing you cannot learn within a few hours.

;-)



## 5 The Home Control Centre: Open Remote

We will start our project with the installation and configuration of OpenRemote.

OpenRemote is a state of the art open source software platform for building control and automation. While it requires little effort and only very basic programming skills, it allows to build a custom, professional smartphone app, which will serve as our mobile control centre. The OpenRemote controller, which is supported on OS X, Linux, Windows and other platforms, will run the “always on” automation rules for our project and will serve as the smart home controller.

In addition to the open source variant there is also a commercial version of OpenRemote available. Details on this version, which is called OpenRemote Professional Designer, are provided in the appendix (chapter 16). For the projects discussed in this book however, with the exception of the controller software installation procedure and the configuration of Z-Wave devices, there are no differences between working with Designer and Designer Professional.



## 5.1 OpenRemote Overview

The OpenRemote platform consists of three software components:

- The OpenRemote controller, an always-on (24/7) Linux, Windows or OS X server application, which connects the mobile control devices (smartphones, tablets) to building automation systems and devices under control. Control devices can be building infrastructure (light switches, power outlets etc.), consumer electronic devices, or home appliances. The OpenRemote controller can also run scripts, which are called rules. These rules are automation sequences, which are implemented based on the open Drools event processing language.
- The second component consists of the OpenRemote mobile clients (OpenRemote Panels) for iOS or Android. Graphical user interface and functionality of these apps can be fully customized using the third component of OpenRemote, the OpenRemote Designer.
- OpenRemote Designer is an online, cloud based application, providing a graphical user interface for crafting the mobile client interface and the related commands, sensors, and switches. Once user interface and control functions are designed, the OpenRemote Designer configuration files are synchronized with the local controller installation. The smartphone client application is updated automatically, when connecting to the controller, immediately reflecting changes or updates made in the OpenRemote Designer project.

OpenRemote supports a large variety of building automation protocol standards. In addition, it provides API's for the customization and extension of its capabilities. The current software release 2 supports the following control protocols (Table 5.1).

KNX	International standard for industry grade wireline home automation <a href="http://www.knx.org">http://www.knx.org</a>
TCP/IP, UDP, Telnet, HTTP	Internet protocols
Insteon	Home automation system based on power line and radio frequency (RF). <a href="http://www.smartlabsinc.com">http://www.smartlabsinc.com</a>
Shell execution protocol	Execution of shell scripts.
DateTime Protocol	Display of date and time, including sunrise/sunset calculation.
EnOcean	Energy harvesting wireless technology for device control <a href="http://www.iso.org">ISO/IEC</a> 14543-3-10 <a href="http://www.enocean.com">http://www.enocean.com</a>
Russound RNET Protocol	Protocol for Distributed Audio/Video solutions from Russound.
DSC IT-100	Protocol for DSC (Digital Security Controls) systems.
HSC Z-WAVE IP Gateway	Honeywell Z-Wave Gateway.
Z-Wave	Wireless communication protocol

	optimized for home automation. <a href="http://www.z-wavealliance.org">http://www.z-wavealliance.org</a>
AMX Controller	AMX Inc. proprietary device control protocol.
1-Wire Protocol	Low data rate communication bus for Maxim Integrated Products. <a href="http://www.maximintegrated.com">http://www.maximintegrated.com</a>
ISY-99	Control protocol for Universal Devices home automation solution.
panStamp lagarto	Open source protocol for PanStamp wireless modules. <a href="http://www.panstamp.com">http://www.panstamp.com</a>
Wake-On-Lan Protocol	Protocol activating networked systems in power save mode.
Lutron HomeWorks	Protocol for Lutron building control infrastructure.
Domintell	Protocol for Domintell building control infrastructure.
Denon Serial AVR Protocol	Protocol to control Denon / Marantz audio / video/ devices.
Samsung TV Remote Protocol	Protocol used to control Samsung TV systems.
X10	Legacy standard for power line based home automation.
GlobalCache	Infrared control devices by specialist GlobalCache. <a href="http://www.globalcache.com">http://www.globalcache.com</a>
XBMC	Open source media player platform. <a href="http://xbmc.org">http://xbmc.org</a>
xPL,IRTrans, VLC, FreeBox, MythTV	Commercial and OpenSource home automation solutions.
Philips Hue	Protocol to control Philips smart bulbs

*Table 5.1 Communication protocols and automation standards supported by OpenRemote*

With its intuitive user interface, OpenRemote allows for designing a fully customizable building and home control solution without the need to actually write code. This is not to say that home automation is becoming as easy as an off the shelf software installation. With components from different vendors having to play together, there will often be the need for iterations of test, troubleshooting, and fine-tuning. However, with OpenRemote, we have a powerful platform at hand, which allows for professional results and comprehensive functionality. In addition, the large and helpful OpenRemote user community of building automation professionals and home automation hobbyists provide help and support.

## 5.2 OpenRemote Controller Installation

Getting OpenRemote downloaded, installed, and running should take less than one hour. First we need to register for two accounts:

– an OpenRemote user account at the OpenRemote main site:

<http://www.openremote.org/signup.action>

– and an OpenRemote Designer account (for the online OpenRemote Designer application):

<http://composer.openremote.org/demo/login.jsp>

After registration, we download the OpenRemote Controller software from

<http://www.openremote.org/display/HOME/Download>

We uncompress the file and copy its directory tree to a folder (e.g. *shProject*) in our home directory. Under Windows, as well as under OS X, the home directory is the one we are in when opening a terminal window. The top level OpenRemote directory, which is called something like *OpenRemote-Controller-2.1.3*, we rename to the simpler name *ORC*. With that, we have the start script for the OpenRemote controller below:

`shProject/ORC/bin/openremote.sh` (the OS X start script)

`shProject/ORC/bin/openremote.bat` (the Windows start script)

Throughout the book, I will be using *shProject* as the master directory containing the OpenRemote directory tree and other files related to the project.

## 5.3 Installation under Mac OS X

On a Mac we start by opening an OS X Terminal window selecting *Applications — Utilities*. If you have not worked with the terminal application before, there are a few fundamental commands you need to know in order to be able to start:

ls -l	display listing with the option -l (l for long), displays the content of the current directory, including hidden files and permissions
ls -a	display listing with the option -a also shows hidden files (e.g. all files starting with a period such as .bash profile are hidden)
pwd	print working directory - displays the path of the current directory
cd ..	change directory followed by a space and two dots - gets us one directory hierarchy up
cd	just typing cd gets us back to our home directory
cd /target	changes to the specified directory
mkdir name	create (make) directory
man mdc	show the manual entry for a command
./	the leading dot in a directory specification means “relative to the current directory”

### 5.3.1 Java: Verification and Installation

Since OpenRemote is a Java application, as a first step we have to verify if we have a Java Runtime Environment (JRE) installed. To do this we use the command

```
java -version
```

On a new Mac, you will not find a JRE installed, and in response to the above command the system will propose to download and install a Java Runtime Environment (JRE), taking you to the Oracle website. If you continue, you will install the latest JRE version available. However this might run you into incompatibilities with existing applications on your Mac. This is also true for the Drools rules engine version 5.1.1, which is part of OpenRemote and which is not compatible with Java versions above 1.6. For maximum compatibility Apple recommends to use an updated Java 1.6 version, which can be downloaded directly from Apple:

<http://support.apple.com/kb/DL1572>

Thus at the point of this writing (early 2015) on a Mac I recommend to use this version (Java 1.6 from Apple), even if you initially do not plan to use the OpenRemote rules engine. Chances are you will use Drools at some point, and with Java 1.6 you are on the safe side. After installation you can verify the installed Java version using the `java -version` command:

```
java -version
```

```
java version "1.6.0_25"
```

```
Java(TM) SE Runtime Environment (build 1.6.0_25-b17)
```

### 5.3.2 Setting the `$JAVA_HOME` variable

There is one more thing we need to do before we can start the OpenRemote controller, which is setting the `$JAVA_HOME` variable. `$JAVA_HOME` is used by Java programs to find the path of the Java files and needs to contain the full path to our Java installation.

Under OS X and Linux, the list of locations or paths, which a program uses to search for executables is stored in the `$PATH` variable. There is a system wide and a user specific `$PATH` definition. We will just use the user specific `$PATH` variable at this point. The user specific `$PATH` definitions under OS X are contained in the file `.bash_profile` file in the user home directory. The (hidden) file `.bash_profile` might not exist yet in your home directory, which is why you probably will need to create it. (Type: `ls -a` in your home directory to list the hidden files.) We enter the following two commands in the terminal window:

```
touch ~/.bash_profile
open ~/.bash_profile
```

The command `touch` along with a filename creates an empty file and the command `open` plus a filename opens the specified file in the default text editor, which on a Mac is TextEdit. Now we just need to add the line that sets `$JAVA_HOME` to contain the directory of our Java Runtime Environment:

```
export JAVA_HOME=/System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home
```

Since we are about to set `$PATH` variables, we can at this point also add additional directory paths that we are using. This saves us from typing the full file path all the time, and it prevents scripts from being stopped because of missing path definitions. The directories we want to add are our project directory `shProject` and the OpenRemote binary directory `ORC/bin`. For this we add the line:

```
export PATH="$HOME/shProject:$HOME/shProject/ORC/bin:$PATH"
```

`$HOME` is the system variable, which contains the path to our home directory (you can try `echo $HOME` to see its content). So all paths that we enter in `.bash_profile` start with `$HOME` and then contain the complete path relative to it. The individual paths are separated by a colon. At the end of the line, we add `$PATH`, which adds the content of the global system variable `$PATH` to the definition of our local user account. We save `.bash_profile` in TextEdit, close the terminal window and open it up again (which forces the system to process the new `$PATH` definition), and test our work by entering `echo $PATH` and `echo $JAVA_HOME`. We see the default global `$PATH` definitions expanded by our local directory paths as well as the value of `$JAVA_HOME`:

```
echo $PATH
```

```
/Users/smarthome/shProject:/Users/smarthome/shProject/ORC:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin:/opt/X11/bin
```

```
echo $JAVA_HOME
```

```
/System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home
```

### 5.3.3 \$JAVA\_HOME Variable Setting and OS X Upgrades on a MAC

When upgrading the operating system on a Mac, be aware, that you will have to re-install Java, because since OS X 10.8 Java is no more part of the MAC operating systems. Even if you had Java installed under OS X 10.8, under Mavericks (OS X 10.9) and Yosemite (OS X 10.10) for security reasons every operating system upgrade will disable existing Java runtime or development environments (JRE, JDK). With that, after downloading and installing the latest Java version from [java.com](http://java.com) or the Java version Apple recommends from <http://support.apple.com/kb/DL1572> you will also have to reinstall Java and update the settings of your environment variable \$JAVA\_HOME in the \$PATH definition file *.bash\_profile*. This is why on a Mac, rather than manually setting the \$JAVA\_HOME variable you should use the `java_home` command as described below, which automatically points to the location of the Java installation.

### 5.3.4 \$JAVA\_HOME on a Mac: The java\_home command

Alternative to manually setting the path for \$JAVA\_HOME as described above, Apple provides the command `java_home`, which automatically returns a path suitable for the \$JAVA\_HOME environment variable. The command can be found under `/usr/libexec/`. With that the entry of the file `.bash_profile` would look like the following, if we display it with the `echo $PATH` command:

```
export
PATH="$HOME/shProject:$HOME/shProject/ORC/bin:$HOME/shProject/ORC/webapps/controller/rules:$PATH"
export JAVA_HOME=$(/usr/libexec/java_home)
```



### 5.3.5 Permission setting and startup of openremote.sh

The start script for the OpenRemote controller is *openremote.sh* in */shProject/ORC/bin*. With

```
cd shProject/ORC/bin
```

we change to the working directory of the OpenRemote start script. When we do a long listing (*ls -l*) of the files in the *ORC/bin* directory we can see that we do not yet have the execution right for the startup file yet. In case you are new to file permissions: File permissions in OS X and Linux are set in three groups (owner, group, everyone) with three symbols for each group receiving permissions. The symbols can contain:

-	no permission
r	read permission
w	write permission
x	execute permission

The first letter of the file listing is not a permission, but shows whether the line entry references a file (-) or a directory (d). So the permissions start actually at the second letter of the below listing:

```
ls -l ./shProject/ORC/bin
total 48
-rw-r--r--@ 9854 10 Mar 2012 openremote.bat
-rw-r--r--@ 12039 10 Mar 2012 openremote.sh
```

We see that we have only read and write rights for *openremote.sh*. With the command *chmod +x* we set the execution rights:

```
chmod +x ./shProject/ORC/bin/openremote.sh
```

and can now start the OpenRemote controller by entering:

```
./openremote.sh run
```

In the terminal window, you now see a lot of text running by until it stops with a line, displaying something like

```
INFO: Server startup in 3159 ms
```

Our OpenRemote controller is now up and running. We validate our installation by accessing the controller web interface at the URL

<http://localhost:8080/controller>

The below screen (Figure 5.1) will show up. Before we synchronize our local controller installation for the first time with a mobile client design, we need to configure the OpenRemote Online Designer as described further down.

### 5.3.6 Hint: Clean directory management

I recommend to place all custom scripts and files, which we will develop throughout this project, in the root directory of *shProject*, rather than in the OpenRemote directory (which in our project we call *ORC*) or one of its subdirectories. Otherwise you will have to manually move your custom scripts and files from your old to your new OpenRemote installation in case of a software upgrade.

## 5.4 Installation under Windows 7, 8 and Windows XP

Under Windows (XP/7/8) we need to make sure we have the Java Development Kit (JDK) installed. To find out if we have a version installed, we click on *Start — Control — Panel — Add or Remove Programs*. If you find you do not have Java installed (or if you only have the JRE installed but not the JDK), you need to go to <http://www.oracle.com/technetwork/java/index.html> and download and install the Java Development Kit (JDK) for your Windows version.

Since the Drools rule engine version 5.1.1, which is part of OpenRemote, is not compatible with JDK versions higher than 6.x, at the point of this writing (early 2015) I recommend to use the latest JDK 6.x version (JDK 6.45) for your OpenRemote installation, even if you initially do not plan to use the OpenRemote rules engine. Chances are you will use Drools at some point, and with Java 1.6 you are on the safe side. You can download 6.x JDK versions from the [Oracle Java download archive](#).

(While you will not do any Java development work, the OpenRemote install routine at this point requires the JDK rather than the JRE (Java Runtime Environment). Before the next step, determine the exact directory path of your Java installation. It will be something like

```
C:\Program Files\Java\jdk1.6.0_45\
```

We now need to go to the Windows menu for environment variables. We open *Control Panel — System&Security — System — Advanced System Settings — Environment Variables*. We select *New* to add a new variable name and value. We enter `JAVA_HOME` as variable name and the path to our Java directory. Be very careful to make no mistakes when setting the environment variable. The controller will not start up if the environment variable does not point to the correct directory. With the command `set` in the terminal window (*Start — run* and enter `CMD`), we can validate the setting of our Environment variable:

```
set J
```

```
JAVA_HOME=C:\Program Files\Java\jdk1.6.0_45
```

(The command `set` will output all kinds of other settings as well, while `set J` will only output the settings of variables starting with `J`).

We now download the OpenRemote controller software, uncompress the files into a directory in our home directory (e.g. `shProject`), avoiding a folder name with spaces. As mentioned above, I rename the top level OpenRemote directory, which per default is called something like `OpenRemote-Controller-2.1.3` to the simpler name `ORC`, and I move the OpenRemote software directory tree to our project directory `shProject`. With that, the controller start script is located under

```
shProject/ORC/bin/openremote.bat
```

We can now open the command line interface, change to the `bin` directory of our OpenRemote files structure and enter

```
openremote.bat run
```

In the terminal window we will now see a lot of text running by until it stops at a line,

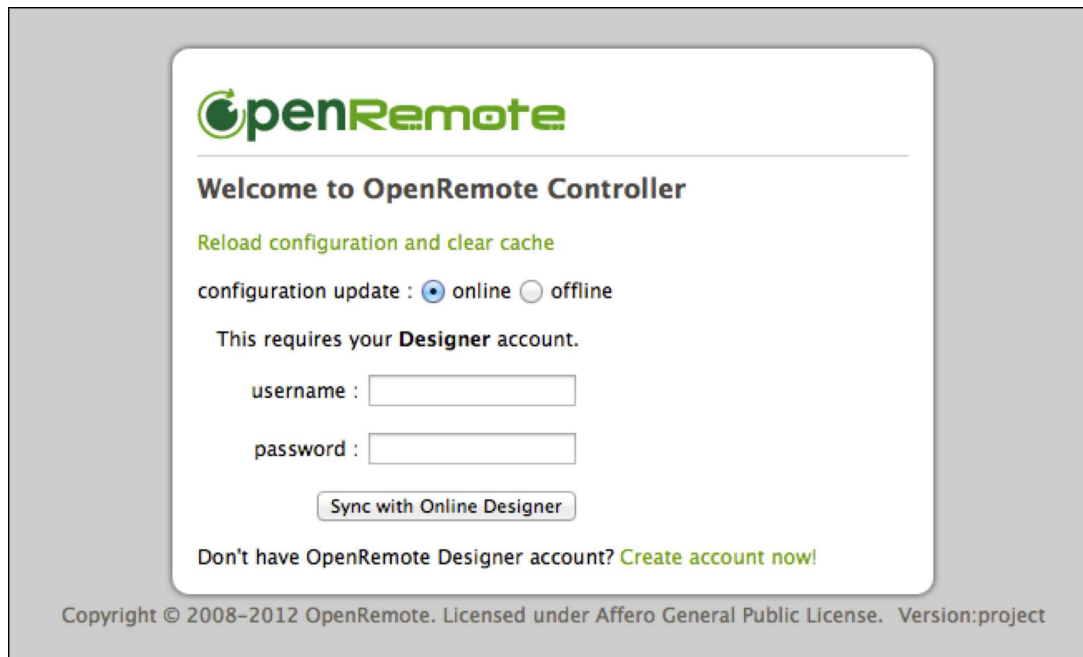
displaying something like

INFO: Server startup in 3159 ms

Our OpenRemote controller is now up and running. We validate our installation by accessing the controller web interface at the URL

<http://localhost:8080/controller>

The below screen (Fig. 5.1) will show up. Before we synchronize our local controller installation for the first time with a mobile client design, we need to configure the OpenRemote Online Designer as described further down.



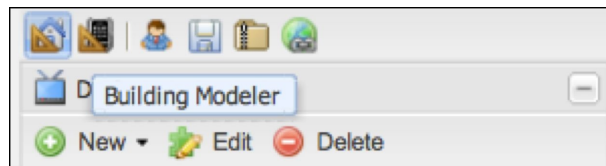
*Figure 5.1 Open Remote Controller login screen*

## 5.5 OpenRemote Designer

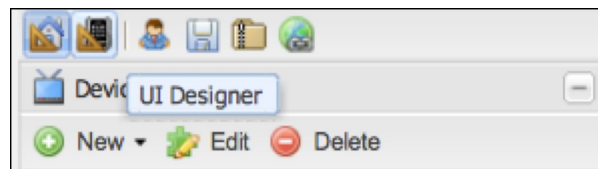
We now go to the online OpenRemote Designer site and log in with our OpenRemote Designer account credentials:

<http://composer.openremote.org/demo/login.jsp>

The OpenRemote designer GUI consists of two main elements. The Building Modeler and the UI Designer. The Building Modeler (Figure 5.2) is used for defining commands, sensors, switches and sliders, and for selecting and configuring the associated protocols. In the UI Designer, we create the graphical user interface for our smartphone or tablet control app, linking its functional elements to the commands in the Building Modeler (Figure 5.3).



*Figure 5.2 Selecting the Building Modeler Screen in Open Remote Designer*



*Fig. 5.3 Selecting the UI Designer Screen in Open Remote Designer*

## 5.6 The “Hello World” App

As a first step to understanding the workflow of OpenRemote Designer we simply want to display “Hello World” on our smartphone or tablet. In the UI designer window, we click on *New — New Panel*, select our mobile device (Android, iPhone, iPad) and enter our project name (e.g. smart home), as the name under panel property on the right hand menu bar. The default name for the screen, which is Starting Screen, we rename to Remote. Next, from the right hand menu, we drag an *abc label* element onto the smartphone panel and enter Hello World in its text field. As the last step, we click on the *save* button in the upper left corner. Our OpenRemote UI Designer screen should now look similar to the one in Figure 5.4.

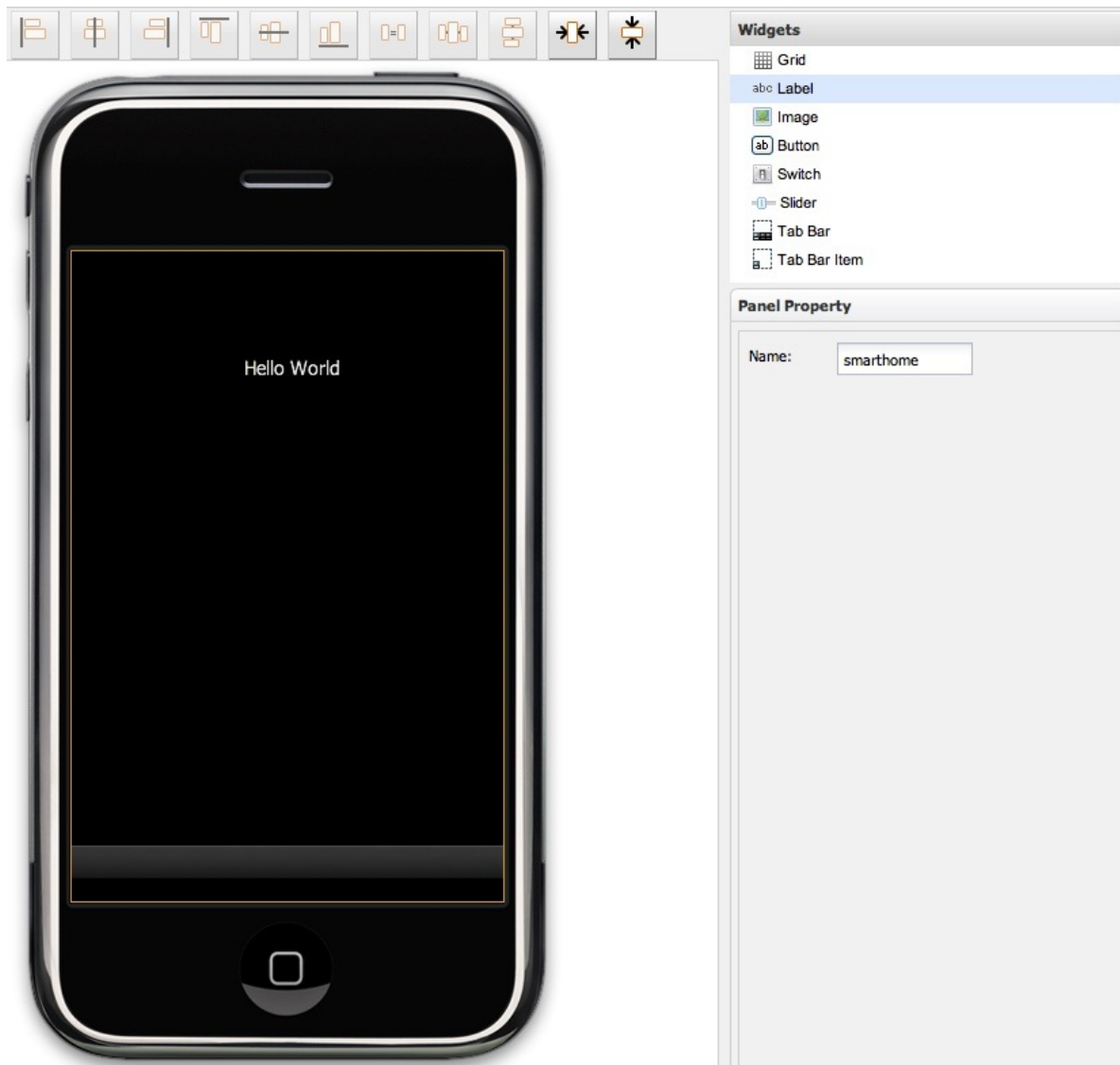
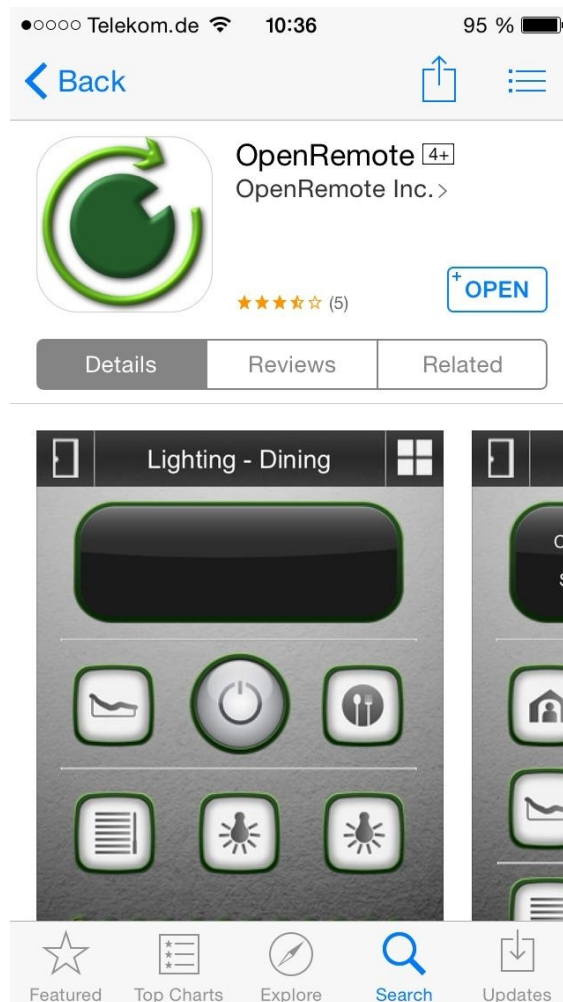


Figure 5.4 First steps with OpenRemote Designer



*Figure 5.5 Downloading the OpenRemote client app for iPhone*

As the final step in setting up our control platform, we download the mobile client app, the OpenRemote Panel, onto our tablet or smartphone. Do not get confused by multiple OpenRemote client versions. You need the one which is described as: “Universal version of the OpenRemote 2 console”(Figure 5.5).

When we start the app on our smartphone or tablet for the first time, it requests the IP-address of our controller hardware. To find your PC’s IP address under OS X 8 (Mountain Lion), we start the Network Utility by selecting *Applications — Utility — Network Utility*. Since OS-X 10.9 (Mavericks) the network utility can be found in the folder `/System/Library/CoreServices/Applications/`. Open Finder, select *Go - Go to Folder...*, enter `/System/Library/CoreServices/Applications/` and select *Network Utility*. Alternatively you can open the Mac search function Spotlight and search for Network Utility.

Under Windows (XP / 7 / 8) we open the terminal window and type

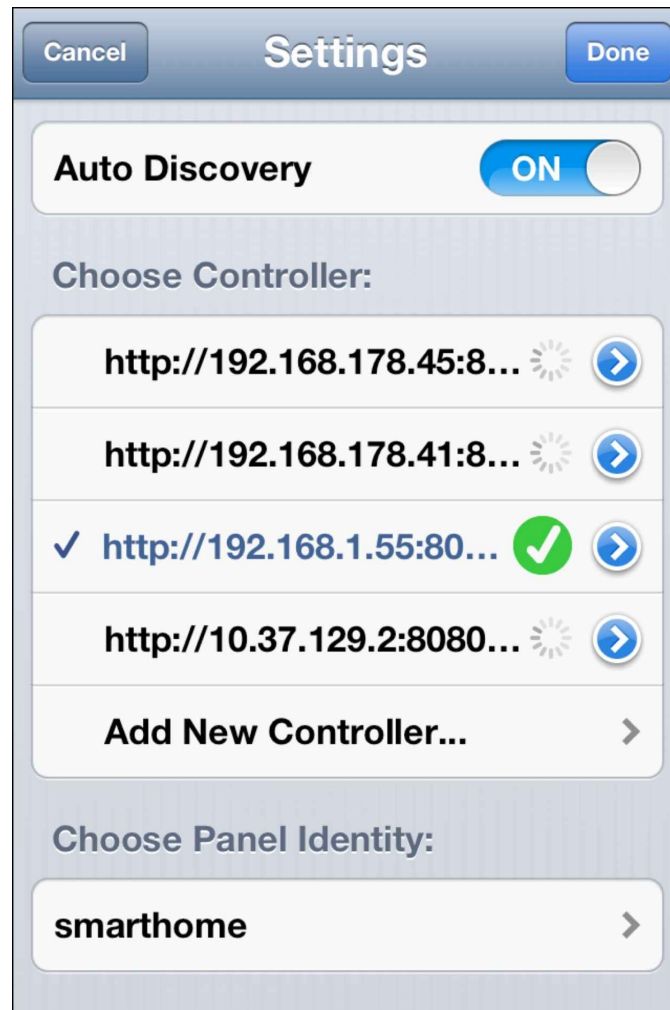
```
ipconfig /all
```

Once we have the controller IP address we enter it followed by `:8080/controller` in the configuration screen of our OpenRemote panel app, e.g.:

```
http://192.168.174.23:8080/controller
```

When you start the OpenRemote app for the first time, it will automatically open the configuration screen. Once the app has loaded, access to the configuration screen is gained

by shaking the phone (Figure 5.6).



*Figure 5.6 The OpenRemote App's configuration screen (shake phone for access)*

As the final step, we select the panel identity of our online-design (Figure 5.5), which in our case is `smarthome`. The app now automatically connects to our local controller and retrieves the application we have designed with OpenRemote Designer. Before we are able to see our “Hello World” OpenRemote design on our smartphone or tablet, we need to synchronize our local controller with our OpenRemote Designer. In case you see an error message from the client app or the OpenRemote controller such as `controller.xml not found` or `panel.xml not found`, don't panic. These files are only downloaded to controller and client after the first synchronization with OpenRemote Online Designer. This is why, before we are able to see our “Hello World” OpenRemote design on our smartphone or tablet, we need to synchronize our local controller with OpenRemote Designer. In order to do that we go to <http://localhost:8080/controller> and click on Sync with Online Designer. The controller will now connect to our online OpenRemote Designer account and synchronize with our project. The local copy of our design will be stored in `panel.xml`. After a restart of our smartphone app, we see “Hello World” on the screen of our smart home app. We have successfully installed our control platform and can get started with our first project.





## 6 A Pretty Smart Sensor: Internet Weather

One of the key aspects of OpenRemote is its ability to fully integrate open standard protocols like TCP/IP, HTTP and Telnet. This allows for correlation and interaction of the building infrastructure with devices that use Internet protocols, such as consumer electronics, home appliances, smartphones or information services. As an example we will design a smart weather sensor for our building automation project, which retrieves weather information from the Internet.

## 6.1 OpenRemote Control via HTTP: Retrieving Internet Weather Data

Our objective for this section is to retrieve precipitation and temperature information for a particular geographic location from the Yahoo Internet weather service, import it into an OpenRemote sensor and display it on our smartphone or tablet control panel.

We start by going to OpenRemote Designer and, creating a new device, which we call HTTP. Under this device we create a command which we will call Weather Berlin by selecting *New — New command*. As the protocol we select HTTP.

The corresponding HTTP URL for retrieving the weather data from the Yahoo website can easily be crafted following the Yahoo format instructions. The location needs to be specified using the Yahoo WOEID (Where On Earth Identifier), which for our example Berlin, Germany turns out to be 638242. (There are many sites on the Internet that offer WOEID searches). As the unit *u* for the temperature we can choose between *c* for Celsius and *f* for Fahrenheit. With that we have our URL which we enter into our command window:

```
http://weather.yahooapis.com/forecastrss?w=638242&u=c
```

(Do not try to call up the above URL yet, since this is an RSS feed. You will see in a minute what we do.) As *HTTP method* we specify *GET* and as polling interval *15s* with *s* for seconds. What is left is the slightly tricky part of extracting the temperature value from the data the Yahoo site sends back triggered by our HTTP GET command.

For data extraction from the HTTP response OpenRemote offers two possibilities: XPath or regular expressions. If the data comes back in XML format, which would be the preferred way, it is easiest to use an XPath expression for the data filter.

For those who have not worked with XML before - it is a really simple, mostly self-explanatory specification for data containers. It is also widely used on the Internet as the data transport format of choice, allowing for easy data exchange between different entities (websites, databases, software, etc.). XPath is nothing more than a structured definition language (similar to regular expressions) to extract and filter data out of XML data structures. (A good tutorial to start with XPath, in case you have not worked with it before, is available at [http://www.w3schools.com/xpath/xpath\\_syntax.asp](http://www.w3schools.com/xpath/xpath_syntax.asp)). W3 also provides an excellent XML tutorial at <http://www.w3schools.com/xml/>.

Since this Yahoo response comes back as a RSS service (with a XML data structure inside) standard browsers will try to activate their RSS plug-in, when the HTTP response arrives. Since at first we just want to take a look at the source of the data being sent back, we use a little trick, which works with Firefox browsers. We precede our URL entry with “view-source:” and enter:

```
view-source:http://weather.yahooapis.com/forecastrss?w=638242&u=c
```

This will get us the source of the Yahoo response displayed:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
  <rss version="2.0" xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0"
  xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#">
    <channel>

<title>Yahoo! Weather - Berlin, DE</title>
<link>http://us.rd.yahoo.com/dailynews/rss/weather/Berlin__DE/*http://weather.yahoo.com/forecast/GMXX1273_c.htm
</link>
<description>Yahoo! Weather for Berlin, DE</description>
<language>en-us</language>
<lastBuildDate>Sun, 28 Oct 2012 6:50 am CEST</lastBuildDate>
<ttl>60</ttl>
<yweather:location city="Berlin" region="BE" country="Germany"/>
<yweather:units temperature="C" distance="km" pressure="mb" speed="km/h"/>
<yweather:wind chill="-6" direction="250" speed="8.05" />
<yweather:atmosphere humidity="100" visibility="9.99" pressure="982.05" rising="1" />
<yweather:astronomy sunrise="6:54 am" sunset="4:43 pm"/>
<image>
<title>Yahoo! Weather</title>
<width>142</width>
<height>18</height>
<link>http://weather.yahoo.com</link>
<url>http://l.yimg.com/a/i/brand/purplelogo/uh/us/news-wea.gif</url>
</image>
<item>
<title>Conditions for Berlin, DE at 6:50 am CEST</title>
<geo:lat>52.52</geo:lat>
<geo:long>13.38</geo:long>
<link>http://us.rd.yahoo.com/dailynews/rss/weather/Berlin__DE/*http://weather.yahoo.com/forecast/GMXX1273_c.htm
</link>
<pubDate>Sun, 28 Oct 2012 6:50 am CEST</pubDate>
<yweather:condition text="Partly Cloudy" code="29" temp="-3" date="Sun, 28 Oct 2012 6:50 am CEST" />
<description><![CDATA[
<br />
<b>Current Conditions:</b><br />
Partly Cloudy, -3 C<BR />
<BR /><b>Forecast:</b><BR />
Sun - Sunny. High: 7 Low: -3<br />
Mon - Partly Cloudy. High: 6 Low: 1<br />
```

```

<br />
<a
href="http://us.rd.yahoo.com/dailynews/rss/weather/Berlin__DE/*http://weather.yahoo.com/forecast/GMXX1273_c.html
Forecast at Yahoo! Weather</a><BR/><BR/>
(provided by <a href="http://www.weather.com" >The Weather Channel</a><br/>
]]></description>
<yweather:forecast day="Sun" date="28 Oct 2012" low="-3" high="7" text="Sunny" code="32" />
<yweather:forecast day="Mon" date="29 Oct 2012" low="1" high="6" text="Partly Cloudy" code="30" />
<guid isPermaLink="false">GMXX1273_2012_10_29_7_00_CEST</guid>
</item>
</channel>
</rss>
<!-- api7.weather.ch1.yahoo.com Sun Oct 28 06:31:55 PST 2012 -->

```

What looks scary at first sight is actually not all that bad, since it is XML, and thus well structured. Recognizing the XML node structure, what we really want to look at is quickly reduced to

```

<channel>
.....
<item>
.....
<yweather:condition text="Fair" code="33" temp="13" date="Wed, 17 Oct 2012 8:49 pm CEST" />
.....
</item>
.....
</channel>

```

What we are looking for are the values of the attributes `temp` (13) and `text` (Fair) inside the XML element `<condition>` right inside its parent `<item>`.

There is one more thing we need to know before we can design the XPATH definition for our search string. The element we are looking for starts with a name, followed by a colon:

```
<yweather:Conditio ...
```

This is the syntax for a so called XML namespace. XML namespaces are local additions to element names. They are used to avoid element conflict when adding together documents that use the same element names. A namespace is defined by the so called `xmlns` attribute at the start tag of the element where they are used or in the XML root element. In our case the name space `yweather` is defined at the very beginning of the XML document:

```
xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0"
```

With a double backslash // XPath can jump right to the XML node regardless of where it is, and @temp references the value of the attribute temp. Without the namespace our XML definition would be as simple as

```
//condition//@temp
```

In our case we need to tell XPath, that our target element is part of a local namespace. With that, our XPath expressions read as follows:

```
//*[local-name() = 'condition']//@temp
```

```
//*[local-name() = 'condition']//@text
```

To validate our assumption we copy the source of the Yahoo response and our XPath expression into the online XPath query checker under

[http://emdin.info/r/xpath\\_checker/](http://emdin.info/r/xpath_checker/)

and see that our assumption was correct (Figure 6.1).

The screenshot shows the 'Online XPath Query Checker' interface. At the top, there is a 'Query:' label and a text input field containing the XPath expression: `//*[local-name() = 'condition']//@text`. To the right of the input field is a 'check' button. Below the input field, there are two main sections: 'XML code:' and 'Query result:'. The 'XML code:' section contains a large block of XML code, including elements like `<lastBuildDate>`, `<weather:location>`, `<weather:units>`, `<weather:wind>`, `<weather:atmosphere>`, `<weather:astronomy>`, `<image>`, `<title>`, `<geo:lat>`, `<geo:long>`, `<link>`, `<pubDate>`, `<weather:condition>`, and `<description>`. The 'Query result:' section shows the output of the query: `<undefined>Partly Cloudy...</undefined>`.

Figure 6.1 XPath expression validation with Online XPath Checker

We can now copy the XPath expressions in the OpenRemote command windows of our two commands, which we call Temp Berlin and Weather Condition Berlin. In order to use the commands in our OpenRemote app we need, as a final step, to embed them in a sensor definition. This is easily done by selecting *New — New Sensor* and adding the command we want to associate with the sensor. We simply name the two sensors after their commands Temp Berlin and Weather Condition Berlin and select for each of them their respective command (Figure 6.2).

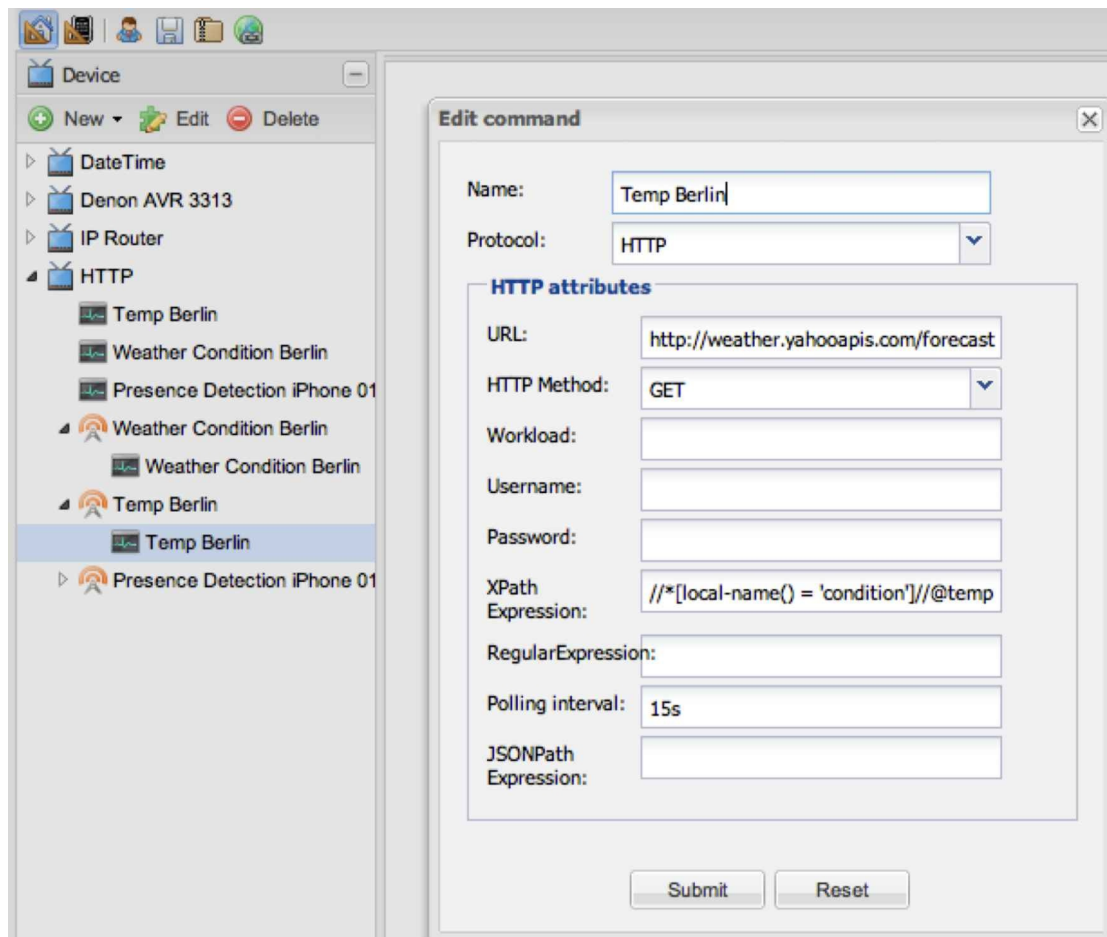


Figure 6.2 Temperature extraction from Yahoo weather using XPath

Alternatively we could have also used a regular expression to filter on one, two or three digits following the string temp=". An according regular expression syntax could have been `(?<=temp\=)"d{1,3}`

and would have led to the same result. For those who have not worked with regular expressions before: They are a structured specification language used to specify string pattern matches. You find a good reference under

[http://en.wikipedia.org/wiki/Wikipedia:AutoWikiBrowser/Regular\\_expression](http://en.wikipedia.org/wiki/Wikipedia:AutoWikiBrowser/Regular_expression)

A good online tool to validate regex expressions can be found under

<http://www.regexplanet.com/advanced/java/index.html>

## 6.2 Designing the App Layout

As the final step, we want to display the sensor output with our smart home app. Before we do that, we need to plan the layout for our GUI. Our plan is to have three screens called *Remote*, *Lighting* and *Heating*. In the UI designer window, we drag a *Tab Bar* item from the widget menu on the right to the bottom of our design and add three *Tab Bar Item* elements to it, which we call Remote, Heating, and Lighting. Then we add two new screens, in addition to the existing Remote screen, which we call Lighting and Heating. We will make use of these two screens later. (Fig. 6.3, Fig . 6.4)

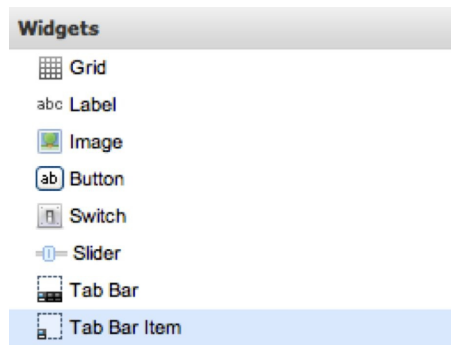


Fig. 6.3 Adding a *Tab Bar Item* to the OpenRemote GUI design

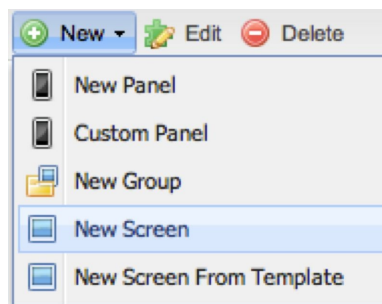


Fig 6.4 Adding a New Screen to the Openremote GUI design

For that, we add a grid on our remote screen for the display of our weather sensor. On our Remote screen we add a *Grid* element by dragging it from the menu into the design setting the parameters to:

Row Count: 1, Col Count: 1, Left:0, Top:0, Width:75, Weight:45

This positions a grid with one cell of the size of 75x45 to the upper left corner of our design. We add three additional cells to the right:

Row Count: 1, Col Count: 1, Left:75, Top:0, Width:150, Weight:45

Row Count: 1, Col Count: 1, Left:225, Top:0, Width:30, Weight:45

Row Count: 1, Col Count: 1, Left:255, Top:0, Width:30, Weight:45

We now drag an *abc label* symbol into to the upper left cell and enter the location name for our weather report in the *Label Properties* field to the right, which is in our case Berlin. We do the same for the second cell, entering Weather Condition in the text field, but here we also add the sensor *Weather Condition*, which can be selected from the sensor drop down menu in Label properties. In the next field, we enter T for temperature and select the sensor *Temperature Berlin*. And the final cell just contains the unit for our temperature display, in our case c for Celsius (Figure 6.5). Now we save our design by clicking on the *disc* symbol.



We now go to the controller window in our web browser and click on *Synch with Online Designer*. After restarting our OpenRemote *smarthome* app on our smartphone we should now see in addition to our Hello World message weather condition and temperature for Berlin (Figure 6.6).

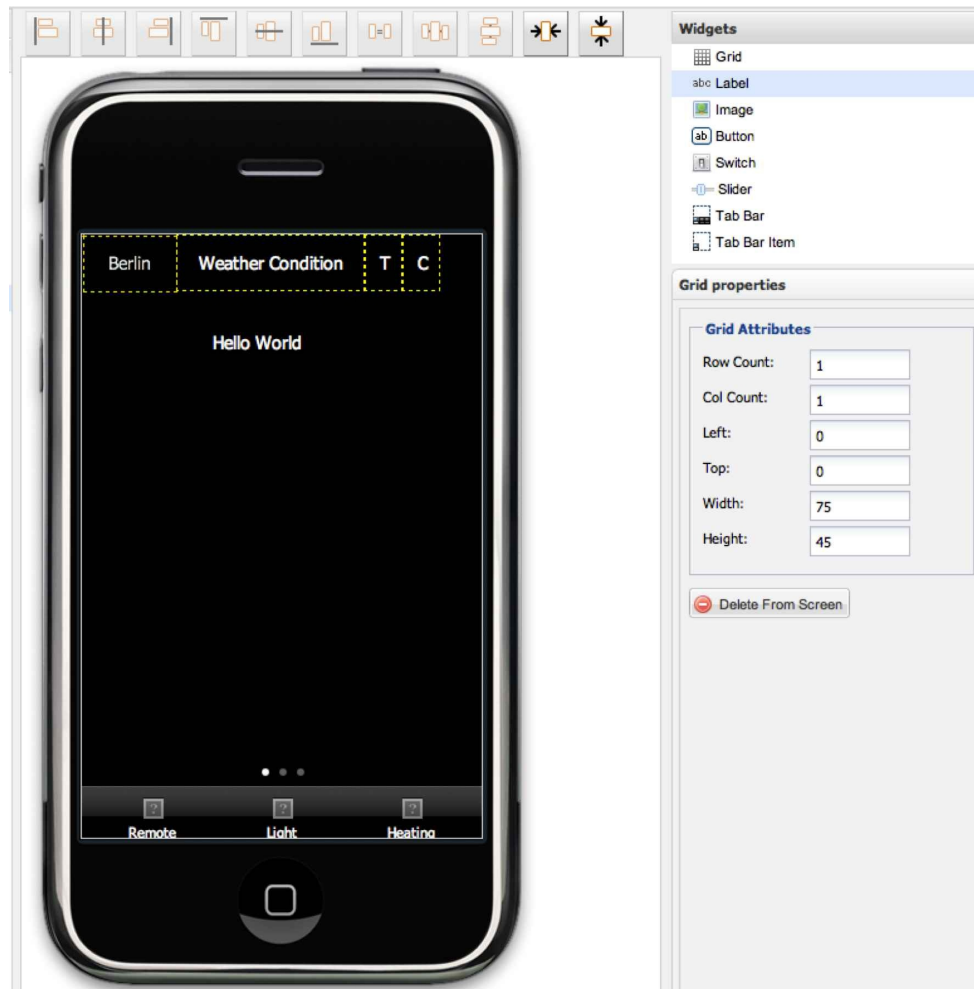
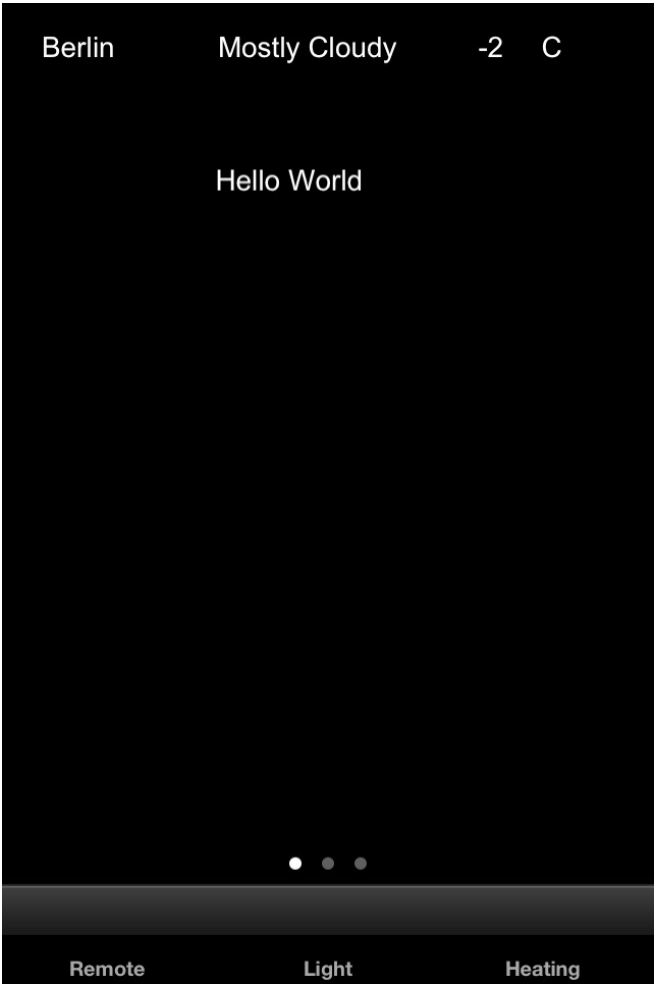


Figure 6.5 The OpenRemote GUI design for the Internet based weather sensor



*Figure 6.6 The weather sensor display in operation*



# 7 Smartphone Based Presence Detection

As we have seen, we can use any data source from the Internet as a (soft) sensor for adding information to our smart home. In this chapter we will see how we can use our Wi-Fi home network itself in conjunction with smartphone Wi-Fi hardware as a sensor for presence detection. Compared to traditional motion and light detector based presence monitoring, this solution has an important advantage. By detecting a particular smartphone in its Wi-Fi network, our smart home not only recognizes that someone is present, but also who is there. (Of course this assumes that the smartphone moves around with its owner.) Traditional motion detectors don't even differentiate if whatever moves is person, an animal, a twig, or some other object. While this does not sound much like a big deal, for an intelligent home and its capabilities, it actually is. With our advanced presence detection functionality, we will be able to design personalized smart home scenarios based on who is entering the building.

So, while from a technical perspective this chapter is probably the most difficult one, it delivers the base for some functionality, which we will be building upon in later chapters. However, if you think you will not need presence detection, or want to try out some of the other - easier to implement - project functionalities first, go ahead and skip this chapter.

What we need is the capability to send an alert to our OpenRemote controller when a specific smartphone is logging into the Wi-Fi network of our smart home. In network terms, the process of logging into a WLAN network is called DHCP registration, during which the smartphone requests the assignment of an IP address from the home network. With that, we need to monitor our home WLAN for DHCP commands sent from our target smartphone. DHCP protocol messages in general are sent within UDP packets, on ports 67 and 68. This means we need to monitor all traffic in our home network, waiting for UDP packets sent to or from ports 67 and 68 with the source MAC address of our target. (Since smartphones enter a sleep mode typically 30 seconds after their screen goes black, alternative methods like continuous Ping requests until the smartphone responds would not work as a reliable detection method).

We will build our solution in two steps:

- First we configure a function which monitors our home network and triggers on the condition as described above
- Then we embed the function in a shell script, so it can run continuously in the background of our smart home control PC.

## 7.1 Building a DHCP – MAC Address Monitor Function

To get a rough idea what we are looking at when monitoring our network, we initially work with Wireshark. Wireshark is a popular open source network sniffer that is easy to use and setup. For the final implementation of our detection feature, we will then use the much leaner tool tcpdump (OS X/Linux), or WinDump (Windows). The reason is that Wireshark, as well as its command line version Tshark, like other more complex sniffing tools, are not designed for long term 24/7 type tracing tasks as we need it for our presence detection feature. They potentially become unstable after hours of tracing, due to the number of concurrent real time tasks they need to manage. In any case, however, it is a good idea to install Wireshark on your system and to become familiar with its basic functions. In a network environment like a smart home, with network nodes from different vendors running different protocols, odds are high that you will have to do some type of network troubleshooting sooner or later. Documentation and installation instructions for Wireshark (OS X, Linux, Windows) are available on <http://www.wireshark.org> and do not need to be repeated here.

After installing Wireshark, we do some first test runs. We select *Capture — Interfaces* from the top menu, tick the interface we want to monitor, and select *Start*. After stopping our first capture (*Capture — Stop* and *File — Quit*), we begin setting up a capture filter on Wireshark that filters out everything but the traffic on the two DHCP ports 67 and 68. In the simple Wireshark filter language, the necessary expression is

```
port 67 or port 68
```

This time we start out from the top menu by selecting *Capture — Options*. We open the capture filter menu by double clicking on our active interface in the top window and insert our filter definition in the capture filter field (Figure 7.1). We select *Start* and then switch WLAN on our smartphone off and back on. You should see something like what is in Figure 7.2, which shows the DHCP request of the smartphone and the response of the DHCP server, containing the IP address lease.

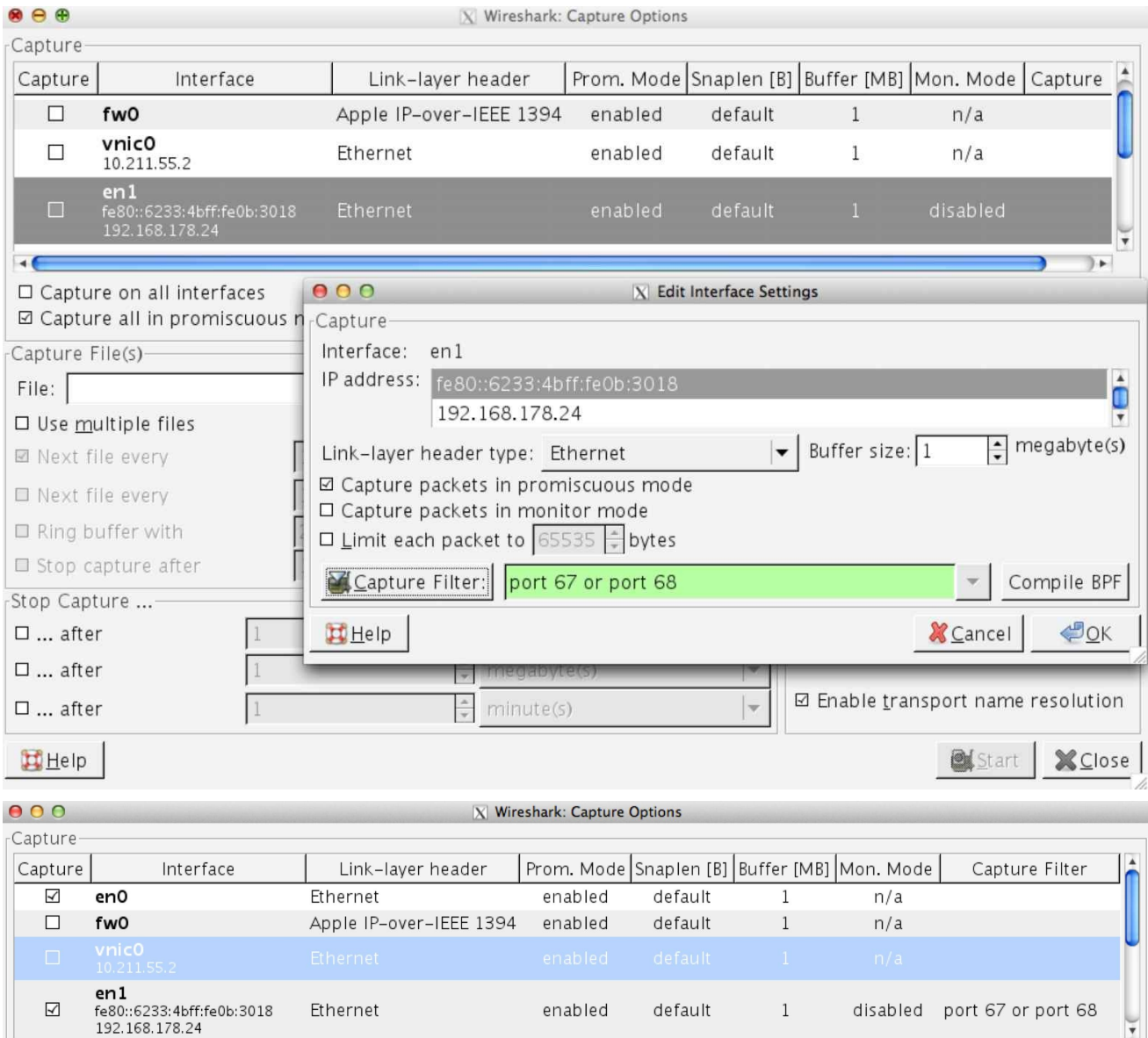
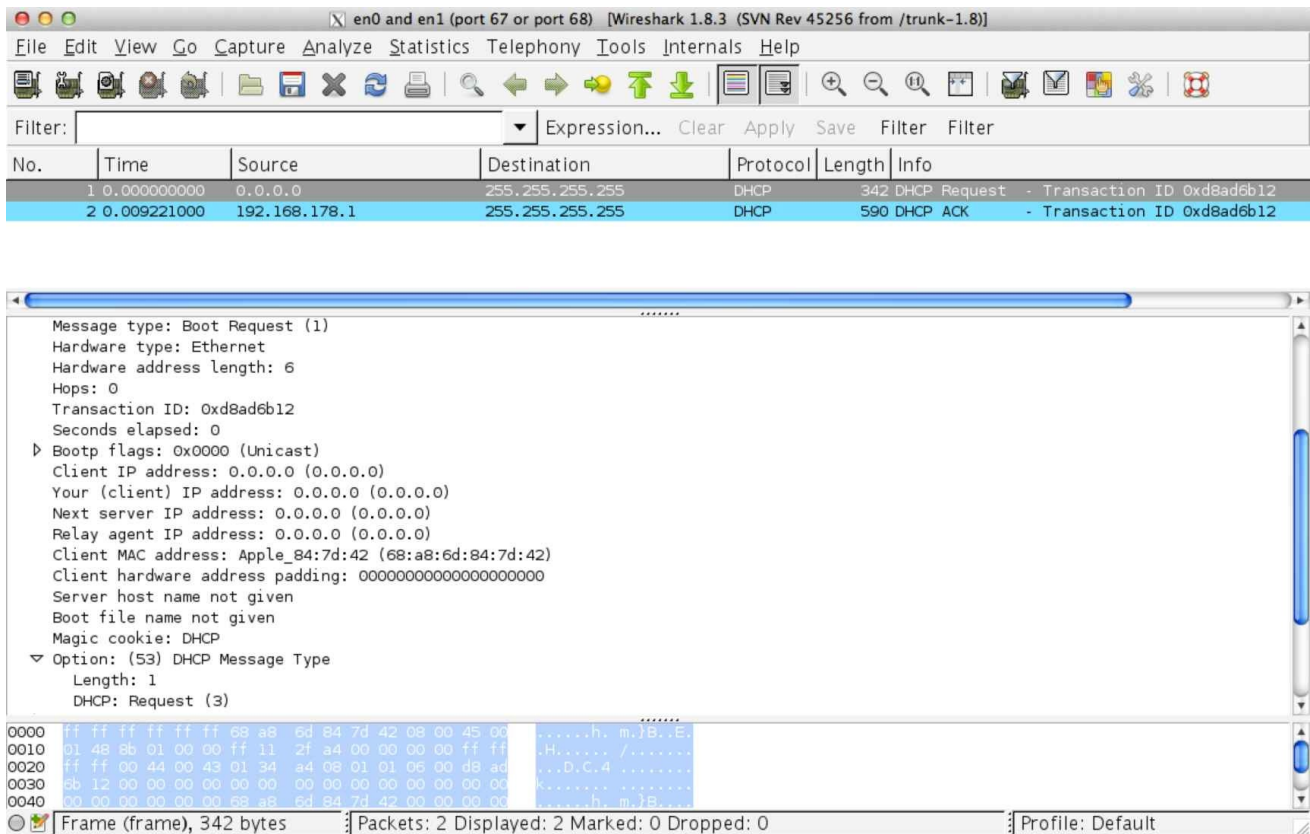


Figure 7.1 Wireshark capture filter setting “port 67 or port 68”



*Figure 7.2 Wireshark DHCP packet decode of the smartphone DHCP request message*

We see that the DHCP request message of the mobile device is sent from UDP port 68 to UDP port 67, the DHCP response from UDP port 67 to port 68.

We have learned enough about the network procedure and can now move on to TcpDump, a small command-line trace utility, which is part of every Unix or OS X operating system. (<http://www.tcpdump.org>) The Windows version is called WinDump and can be obtained from <http://www.winpcap.org>. WinDump installation is a simple two-step process. You first download and install the WinPcap driver and library, then WinDump itself, which is just a single file ready to run.

On the OS X / Linux side, for security reasons per default TcpDump is hidden and its usage requires the administrator password. If you simply type `tcpdump` in the terminal window, you will get

```
tcpdump: no suitable device found
```

as a response. However, if you type

```
sudo tcpdump
```

the program will start and you will see something like the following:

```
tcpdump: WARNING: en0: no IPv4 address assigned
```

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
```

```
listening on en0, link-type EN10MB (Ethernet), capture size 65535 bytes
```

Stop the TcpDump execution by typing `ctrl-C`. In order to be able to use the `tcpdump` command inside a script, we need to add executions rights to it using the `chmod` command.

Chmod (change mode) is used to set the permissions (r read, w write, e execute, s setuid - set user ID upon execution) for files to user groups (a all, u user, g group o other) in OS X / Linux environments. Since tcpdump is a file that has been given the setuid attribute, we need to add the s permission for our account u with the command

```
chmod u+s
```

Under OS X tcpdump is located in the directory /usr/sbin/. From our home directory in the terminal, we take a look at the current permissions of tcpdump with the ls -l command:

```
ls -l //usr/sbin/tcpdump
```

```
-rwxr-xr-x@ 1 root wheel 692720 2 Nov 00:01 //usr/sbin/tcpdump
```

The third letter from the left, x, indicates that the setuid bit is not set. We now type (as superuser sudo)

```
sudo chmod u+s //usr/sbin/tcpdump
```

and, after entering our administrator password, we can validate the change with another ls -l command. The third bit from the right has changed from x to s:

```
ls -l //usr/sbin/tcpdump
```

```
-rwsr-xr-x@ 1 root wheel 692720 2 Nov 00:01 //usr/sbin/tcpdump
```

In general, setting the sudo bit can have significant impact on the security of your system. So always be very careful if you make changes like the above. On the Windows side there are no security settings active for WinDump and we can start the configuration for our monitoring script.

The various functions of tcpdump and WinDump are identical and invoked using options following the start command. Option -D displays a list of all available network interfaces:

```
tcpdump -D (or windump -D)
```

```
1.en0
```

```
2.fw0
```

```
3.vnic0
```

```
4.en1
```

```
5.vnic1
```

```
6.p2p0
```

```
7.lo0
```

With the -i option we specify the capture interface that we want to select, using the interface name or the number of the -D output, in our case the en1 interface or number 4:

```
tcpdump -i 4
```

We now add the appropriate options for tcpdump to filter on the DHCP requests of our target smartphone with the MAC address 68:a8:6d:84:7d:42:

```
tcpdump -i 4 -c 1 -n -v ether host 68:a8:6d:84:7d:42 and dst port 67 > dhcp_capture.txt
```

-c sets the number of packets to capture (in our case 1)

-n suppresses the name resolution which assigns names to IP addresses and ports. This



process can slow down the capture process and we do not need it for our purposes.

`-v` sets the degree of decode detail (`-v` some decode, `-vv` full decode)

`ether host <filter MAC address>` sets the MAC address filter (in our case the smartphone MAC address `68:a8:6d:84:7d:42`).

and `dst port 67` further restricts our filter condition to packets with destination port 67 only.

`> dhcp_capture.txt` writes the `tcpdump` output to the file `dhcp_capture.txt`

The appropriate WinDump command is identical and reads:

```
windump -i 1 -c 1 -n -v ether host 68:a8:6d:84:7d:42 and dst port 67 > dhcp_capture.txt
```

Table 7.1 shows the terminal output of the above WinDump command (not storing the output to a file). We can see (in bold) the protocol (UDP), the port number (67), the command type (BOOTP/DHCP Request) and the MAC source address of the packet (`68:a8:6d:84:7d:42`) that identifies our smartphone, which we have turned on after starting the capture.

```
C:\Users\smarhome>windump -i 1 -c 1 -n -v ether host 68:a8:6d:84:7d:42 and dst port 67
```

```
windump: listening on \Device\NPF_{7CA38408-739A-488E-8BA1-D9685DC6FD1B}
```

```
17:37:50.449481 IP (tos 0x0, ttl 255, id 35576, offset 0, flags [none], proto: UDP (17), length: 328) 0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP, Request from 68:a8:6d:84:7d:42, length 300, xid 0xd8ad6b0a, secs 1, Flags [ none ]
```

```
Client-Ethernet-Address 68:a8:6d:84:7d:42 [[bootp]
```

```
1 packets captured
```

```
266 packets received by filter
```

```
0 packets dropped by kernel
```

*Table 7.1 WinDump terminal output for the presence detection script*

After testing the above commands, ensuring the expected results are returned, we proceed to write a shell script to handle our presence detection.

## 7.2 Creating a Shell Script for Presence Detection

Shell scripts are short programs mostly used to automate sequences of command-line operations for recurring tasks. This is exactly what we need in order to grow our command line presence detection command into a continuously operating presence detection function. In order to do that, we want to add three main functions

- we want to be able to switch the function on and off via a html page called *presencefunction.html*, to which we assign the content `on` or `off`. We will later use an OpenRemote HTTP Command and Sensor to monitor and set this file, using it as an on/off switch for our presence detection function.
- as long as *presencefunction.html* reads `on` we want to monitor for presence events. In case the function triggers the specified MAC address we write the text string `return` to *presence.html* for 60 seconds, after which the content of *presence.html* is again set back to `idle`. Later in the chapter we will set up a OpenRemote HTTP sensor, which polls *presence.html* every 15 seconds in order to detect a return event, which then will allow to trigger the various welcome home actions.
- And finally we want to be able to set a dead time for our function, providing the possibility to set a minimum delay between two presence detection events. This ensures that a restart of the smartphone does not trigger a return event.

As you can see, we are using two HTML files (*presencefunction.html* and *presence.html*) to communicate between our shell script based presence detection function and the OpenRemote environment.

## 7.3 Shell What?

A shell is nothing but a command interpreter in Unix type systems. With Unix Version 7 an implementation called Bourne shell became popular, in later years its open source implementation Bash shell was chosen as the default command line interpreter environment for Linux and OS X. Thus when opening the terminal window on a Mac, you are using its Bash shell environment.

The Windows equivalent to Unix shells is the Command.exe environment, which was introduced with Windows NT in 1993. Previous to NT the Windows scripting application was Command.com, with an even less function rich scripting language. In 2006 Microsoft released PowerShell, a command line scripting environment with a feature set, which is comparable with Unix shells, and which uses many of the same commands. It ships as part of Windows 7 and 8 and can be downloaded free for previous Windows versions. We will be using PowerShell for the Windows version of our script. Mac OS X users continue here, Windows users can skip the next section and move right on to the Windows instructions.

## 7.4 The Presence Detection Script under OS X / Linux

For writing our shell script we can use any text editor. On a Mac it is easiest to use TextEdit. Open TextEdit and choose *File — New* and then *Format — Make Plain Text*. Save the file using the extension `.sh` for shell. Execution of the script is done by typing `./` if we reside in the same directory as the script, followed by the name of the script, in the Mac terminal window, e.g.:

```
./presence.sh
```

## 7.5 Testing it Right - Best Practice for Script Writing

While scripts are really short pieces of code (e.g. our presence detection function counts less than twenty lines), they are very sensitive to syntax errors, and their functioning depends heavily on your system settings and your file structure. This means, that you probably will need to make a few modifications to the source we describe in order to get it to run on your system.

So I would advise you NOT to copy and paste the code of the script as a whole onto your machine and execute to see what's happening.

I recommend to test the script line by line (by copying each line (or in case of a loop just the portion with the loop - as long it can run on its own) in the terminal window. If needed make necessary modifications, execute the code, and watch if the outcome is as expected. Once the line is tested and works as desired, you add it to the already cleared code and execute the script from there. While doing this, you probably want to add a few temporary `echo` commands to the code, to be able to better observe what's going on. I promise this approach will get you to a working and stable script very fast.

## 7.6 Building the Script

For the purpose of this example I have created the directory *shProject* in my home directory. In it I save the script *presence.sh* as well as the OpenRemote directory tree *ORC*. If you have never done a script before, and you want to follow exactly the instructions in the book, also create the directory *shProject* in your home directory by opening up a terminal window ([Applications — Utilities — Terminal](#)) by typing

```
mkdir shProject
```

Then enter the following command line in TextEdit

```
echo "Hello World"
```

and save the file to

```
/shProject/presence.sh.
```

In order to give the file execution permission you need to set the required permission for the script by typing:

```
chmod a+x presence.sh
```

Now run the script from the terminal window:

```
./shProject/presence.sh
```

If you see the `Hello World` output in the terminal window, you have successfully run your first script. Be aware, that you always have to enter the full path to the script in order to start it. Even if you reside in the directory of the script itself, you need - for security reasons - to precede the file name with `./` as below:

```
./presence.sh
```

In order to avoid typing the full file path all the time (and for scripts to avoid being stopped because of missing path definitions), we can add the directories, where our scripts (*presence.sh* and *openremote.sh*) are located to the `$PATH` variable. Here the path definitions, which are searched by the system per default, when executing a command, are stored. The `$PATH` definitions on a MAC are contained in the file *.bash\_profile* in the user home directory. (An alternative file you might have heard of is *.profile*, however we go with the Apple recommended one *.bash\_profile* here). It might not exist yet in your home directory, which is why you probably will need to create it. Type the following two commands in the terminal window:

```
touch ~/.bash_profile
```

```
open ~/.bash_profile
```

The command `touch` along with a filename creates an empty file and the command `open` plus a filename opens the specified file in the default text editor, which on a Mac is TextEdit.

You can now add the directories you want to add to your `PATH` definition by entering:

```
export PATH="$HOME/shProject:$HOME/shProject/ORC:$PATH"
```

`$HOME` is the system variable, which contains the path to your home directory (you can try `echo $HOME`). So all paths you enter start with `$HOME` and from there contain the complete

path relative to your home directory. The individual paths are separated by a colon. At the end we add `$PATH`, which adds the content of the global system `PATH` variable to the `PATH` definition of our local user account. This last entry `:$PATH` is important for the functioning of your terminal and must not be removed. We then save `.bash_profile`, close the terminal window and open it up again (which forces the system to process the new `PATH` definition) and test our work by entering `echo $PATH`. We see the global `PATH` definitions expanded by your local directories:

```
echo $PATH
```

```
/Users/smarthome/shProject:/Users/smarthome/shProject/ORC:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin:/opt/X11/bin
```

We are now able to run `presence.sh` from our home directory, without specifying the file path to `/shProject` by just entering:

```
presence.sh
```

We now can get started with the design of our script. In order to be able to switch our presence function on and off, as outlined above, we will create the file `presencefunction.html` with the content `on`. In order to have the option to access the file through a web browser or later through the OpenRemote HTTP protocol function, we save the file in the root directory of the OpenRemote web server:

```
echo "on" > ./shProject/ORC/webapps/controller/presencefunction.html
```

This command writes the string `on` to the file `presencefunction.html`. In case the file exists, previous content is overwritten, in case it does not exist, the file is created. To validate we have done everything right we can use the `cat` command to read out the content of `presencefunction.html`:

```
cat ./shProject/ORC/webapps/controller/presencefunction.html
```

```
on
```

Later we will implement the above step in our startup script and use the content of `presencefunction.html` as the condition for our shell script to run. In order to switch our presence detection function on and off we create two simple batch files, using TextEdit as described above, which we call `turnPresenceOff.sh` and `turnPresenceOn.sh`. Each of the files just contain the one line, which writes `on` or `off` to `presencefunction.html`.

```
echo "off" > ./shProject/ORC/webapps/controller/presencefunction.html
```

```
echo "on" > ./shProject/ORC/webapps/controller/presencefunction.html
```

After the creation of the files we add execution rights to our two batch files and do a quick test, if everything works as expected:

```
chmod a+x turnPresenceOff.sh
```

```
chmod a+x turnPresenceOn.sh
```

```
./turnPresenceOff.sh
```

```
cat presencefunction.html
```

```
off
```

```
./turnPresenceOn.sh
```

cat presencefunction.html

on

At the end of the chapter we will integrate the two batch files with our smart home app for activation and deactivation of our presence script.

For now we build the timer, which defines the dead-time of our function. As explained above, we do not want the presence scenario to become active, if the smartphone is just switched off and back on. Thus we want the capability to define a minimum required delay between two subsequent presence trigger events. For that we store the absolute time in seconds at the startup of the script in the variable `current_time_in_seconds` and the event trigger time, which is calculated by adding the dead time to current time, in

`trigger_time_in_seconds:`

```
timeout_in_seconds=10;
```

```
current_time_in_seconds=$(date +%s);
```

```
((trigger_time_in_seconds = $current_time_in_seconds + $timeout_in_seconds));
```

```
capture="empty"
```

A few words of explanation to the above code lines: The `date` function provides date and time. Using the option `+%s` formats the output of `date` to a Unix timestamp, which is nothing but the number of seconds elapsed since 1.1.1970. Working with timestamps is much easier than dealing with human readable date-time representations, especially if you want to compare, add or subtract dates and times as we do. (Just for reference, another example for date formatting would be `date +%Y-%m-%d`, which produces the format `YYY MM DD`). Also note, that in shell scripting a space separates arguments from commands. Strings, which contain spaces, therefore need to be surrounded with double quote marks (e.g. `echo "Hello world"`). This is why you need to be really careful when typing shell code, since scripts are very sensitive to spaces on the wrong place.

Assigning a value to a variable is simply done by using the variable name followed by an equal sign and the value you want to assign.

```
timeout_in_seconds=10
```

If you want to access the value of the variable, you need to precede the variable name with a dollar sign (`$`). The shell then inserts the contents of the variable at that point in the script, e.g.:

```
echo $timeout_in_seconds
```

Numeric calculations are implemented by wrapping the entire expression in double parenthesis. (There are also other methods for doing shell script maths like `expr` or `let`, but they are less intuitive to use). For testing purposes we choose the small value of 10 seconds for the dead time, so we can easily test the various states of the function.

Next we use the `tcpdump` command from above and send its output to the file `capture.txt`:

```
tcpdump -i en1 -c 1 -v ether host 68:a8:6d:84:7d:42 and dst port 67 | > capture.txt
```

(We will not do anything with the trace in `capture.txt`, we just store it for potential troubleshooting purposes). Now let's add the above lines to our script in TextEdit, which



by now stores the current time in form of a Unix timestamp, calculates the dead time of our function and starts tcpdump with our filter condition, waiting for a match event. Only if a match occurs, the script will move on to the next set of commands. All we need to do now is, in case a presence match occurs, to check if the dead time has passed. We use an `if` statement to compare the current time (which is the time of the tcpdump pattern match) with the value in our variable `trigger_time_in_seconds`. The general syntax of the `if` statement in our shell environment is:

```
if [ condition ] ; then
    echo "test "
fi
```

With that, the `if` statement to compare current time with `$trigger_time_in_seconds` looks as follows:

```
if [ "$(date +%s)" -le "$trigger_time_in_seconds" ] ; then
```

`-le` stands for the compare command less than or equal, other compare options would be `-eq`, `-gt`, or `-ge`. If we are beyond the dead time, we want to write the string return followed by the MAC address of the device which as triggered the function into our result file `presence.html` in the web server root directory of the OpenRemote controller. After sixty-seconds (command `sleep 60`), which is sufficient for the polling interval of our OpenRemote sensor as we will see further down, we want to set the content of `presence.html` back to idle. If the dead-time has not passed yet, we do nothing and simply print to the terminal window: return detection blocked:

```
if [ "$(date +%s)" -le "$trigger_time_in_seconds" ] ; then
    echo "return detection blocked"
else
    echo "return_Chris" > ./shProject/ORC/webapps/controller/presence.html
    cat ./presence.html
    sleep 60
    echo "idle" > ./shProject/ORC/webapps/controller/presence.html
    cat ./presence.html
fi
```

The two `cat` statements in between are just for the purpose of observing the proceeding of the script in the terminal window. We are almost done by now. What is left, is to repeat the entire process within a `while` loop, which does not terminate, until our control file `presencefunction.html` contains the value `off`. The `while` statement has the following general shell syntax:

```
while [ condition ] ; do
    echo "loop";
done
```

Note that in general in shell scripts semicolons at the end of a line are superfluous, since newline is also interpreted as a command separator. However if, as in the above case, for

better readability, the `then` respectively the `do` statement is written in the same line as the `if` / `while` statements, it is required to separate those commands.

At the very beginning of our script we now add the interpreter identifier `#!/bin/sh`, which tells the kernel which shell to use, and a line of comment, which always starts with `#`, and we are done (Table 7.2)

```
#!/bin/sh
#Presence detection using the smartphone DHCP request when booking into a Wi-Fi (WLAN) network
while [ $(cat ./shProject/ORC/webapps/controller/presencefunction.html) = "on" ]; do
# set timers for return detection timeout and variable for dhcp packet capture
    echo "idle" > ./shProject/ORC/webapps/controller/presence.html
    timeout_in_seconds=10;
    current_time_in_seconds=$(date +%s);
    ((trigger_time_in_seconds = $current_time_in_seconds + $timeout_in_seconds));
# capture until a dhcp request from MAC address 68:a8:6d:84:7d:42 is detected, then store output in capture.txt
tcpdump -i en1 -c 1 -v ether host 68:a8:6d:84:7d:42 and dst port 67 > capture.txt
# if the dhcp request is received before the timeout, do nothing, else set presence.html to "return_Chris" for 60 seconds,
then back to "idle"
    if [ "$(date +%s)" -le "$trigger_time_in_seconds" ]; then
        echo "return detection blocked"
    else
        echo "return_Chris" > ./shProject/ORC/webapps/controller/presence.html
        cat ./shProject/ORC/webapps/controller/presence.html
        sleep 60
        echo "idle" > ./shProject/ORC/webapps/controller/presence.html
        cat ./shProject/ORC/webapps/controller/presence.html
    fi
done
echo "presence detection off"
```

*Table 7.2 Shell script for presence detection (OS X / Linux)*

The execution of the script, while switching our smartphone Wi-Fi function on and off, should output something like the below in the terminal window:

```
tcpdump: listening on en1, link-type EN10MB (Ethernet), capture size 65535 bytes
1 packets captured
113 packets received by filter
0 packets dropped by kernel
return_Chris
```

When we enter

```
echo "off" > ./presencefunction.html
```

in a second terminal window, after the next return event the function will turn off reporting presence detection off

Finally, in order to make our script a bit more flexible, we will replace the values for dead-time and MAC address with the special variables \$1 and \$2. System variables \$1 ... \$9 are reserved to be used for passing arguments at start up to the script. We replace the value 10, which is our test dead time in seconds, by \$1 and the MAC address (*68:a8:6d:84:7d:42*) by \$2.

```
timeout_in_seconds=$1;
```

```
.....
```

```
tcpdump -i en1 -c 1 -v ether host $2 and dst port 67 > capture.txt
```

```
.....
```

The script now needs to be started by typing the script name followed by the two arguments for dead-time and MAC address:

```
presence.sh 10 68:a8:6d:84:7d:42
```

While testing, for the dead time we use a value of 10 seconds. This allows us to assess if the blocking conditions are recognized, without a long wait. In operation we will set this value to something like 3000 or 5000, depending on the desired behavior of the script.

To round our script off we add some basic error handling at the beginning, which makes sure the correct number of arguments are being specified:

```
if [[ $# -lt 2 || $# -gt 2 ]];then
    echo "$0: Argument error: presence.sh [dead time in seconds] [MAC address of target]"
    exit 2
fi
```

We use the variable \$#, which contains the number of arguments which are passed on to the script. The if statement validates, if the number of arguments provided is exactly 2. In case less than two or more than two arguments are passed on, an error message is displayed and the script is terminated. For the script termination we use the command `exit` and its option 2: unexpected error. The special variable in the error message `echo` command, \$0, references the script itself.

```
#!/bin/sh
```

```
#Presence detection using the smartphone DHCP request when booking ton a Wi-Fi (WLAN) network
```

```
if [[ $# -lt 2 || $# -gt 2 ]];then
    echo "$0: Argument error: presence.sh [dead time in seconds] [target MAC address]"
    exit 2
fi
```

```
while [ $(cat /Users/smarthome/shProject/ORC/webapps/controller/presencefunction.html) = "on" ] ; do
```

```
# set timers for return detection timeout and variable for dhcp packet capture
```

```
    echo "idle" > /Users/smarthome/shProject/ORC/webapps/controller/presence.html
    timeout_in_seconds=$1;
```

```

current_time_in_seconds=$(date +%s);
((trigger_time_in_seconds = $current_time_in_seconds + $timeout_in_seconds));
# capture until a dhcp request from MAC address 68:a8:6d:84:7d:42 is detected
tcpdump -i en1 -c 1 -v ether host $2 and dst port 67 > capture.txt
# if dhcp request is received before the timeout, do nothing, else set presence.html to "return_Chris"
if [ "$(date +%s)" -le "$trigger_time_in_seconds" ]; then
    echo "return detection blocked"
else
    cat /Users/smarthome/shProject/ORC/webapps/controller/presence.html
    sleep 60
    echo "idle" > /Users/smarthome/shProject/ORC/webapps/controller/presence.html
    cat /Users/smarthome/shProject/ORC/webapps/controller/presence.html
fi
done
echo "presence detection off"

```

### *Table 7.3 Final shell script for presence detection (OS X / Linux)*

As a final improvement of our code we enhance our two presence detection control files *turnPresenceOff.sh* and *turnPresenceOn.sh*. In order to start presence detection in addition to setting *presencefunction.html* to on we need to start the batch file *presence.sh*. This gets us to the following two lines in *turnPresenceOn.sh*:

```

echo "on" > /Users/smarthome/shProject/ORC/webapps/controller/presencefunction.html
sh ./presence.sh 10 68:a8:6d:84:7d:42

```

And for stopping presence detection in addition to setting *presencefunction.html* to off we will stop the three processes *turnPresenceOn.sh*, *presence.sh* and *tcpdump*, which are initiated by our script *turnPresenceOn.sh*. Otherwise our presence detection would only stop after one more presence occurrence, which could take a long time. To stop a process we can use the `kill` command along with the process id. To list the processes which are currently active we use the command `ps`.

```

PID TTY    TIME CMD
53735 ttys000 0:00.27 -bash
1058  ttys001 0:00.01 -bash
54459 ttys001 0:00.00 sh /Users/smarthome/shProject/turnPresenceOn.sh
54460 ttys001 0:00.00 /bin/sh /Users/smarthome/shProject/presence.sh 10 68:a8:6d:84:7d:42
54463 ttys001 0:00.03 tcpdump -i en1 -c 1 -v ether host 68:a8:6d:84:7d:42 and dst port 67

```

In order to retrieve the process id (PID) for our presence script we use the filter command `egrep`:

```
ps | egrep 'resence' | awk '{print $1}'
```

(We avoid the `p` in the filter expression to have a match for both variants of presence:

presence.sh and turnPresenceOn.sh). The command `ps` lists all active processes, the pipe command `|` routes the output to `egrep`, which searches for the string 'resence', in order to capture `turnPresenceOn.sh` and `presence.sh`. Then the output of `grep` is routed to `awk '{print $1}'`, which gives us the first field of every matching line, which is the process id (PID) of the processes we are looking for. After starting our presence function you can try

```
ps | egrep 'resence' | awk '{print $1}'
```

which will get you something like

```
54512
```

```
54459
```

```
54460
```

We can now construct our kill command for all processes containing the strings 'resence' and 'tcpdump'. The term `$(x)` means to execute `x`, then take its output and put it on the command line. Since our `egrep` command outputs a process id, the below commands would actually execute something like `kill 54512`. To match two or more strings with `egrep` you need to separate the strings with a vertical bar `|`. With that our command reads:

```
kill $(ps | egrep 'resence|tcpdump' | awk '{print $1}')
```

We now have the following two lines in the `turnPresenceOff.sh` script:

```
echo "off" > /Users/smarthome/shProject/ORC/webapps/controller/presencefunction.html
```

```
kill $(ps | egrep 'resence|tcpdump' | awk '{print $1}')
```

For the many options of `kill`, `ps`, `egrep` in the terminal window simply type `man` followed by the according command.

## 7.7 A Log File for Presence Detection

In addition to writing the `return` message to the file `presence.html` we want to log every return event along with date and time to the file `presencelog.html`. While `presence.html` serves as the real time sensor output, which reports a return for 60 seconds, and then goes back to idle, waiting for the next return event, `presencelog.html` shall contain a history of all return events, of which the last entry we want to display via the GUI of our smartphone app. For this purpose we write another brief bash script, which uses the arguments `$1` as log entry and `$2` as log filename. We further use the command `-f filename`, which returns `true` if a particular file is a regular file and exists. Now we can create our presence logging script `preslog.sh` which either creates the log file, in case it does not exist yet, and writes the first log entry or, in case it does exist, appends the log entry at the end of the file. As a presence logging file we again use a html file located in the root directory of the OpenRemote webserver under `ORC/webapps/controller/`

in order to have the option to access its content through a web browser or through the OpenRemote HTTP protocol function. At the beginning of the script we again validate if the correct number of arguments are provided, when the script is called:

```
#!/bin/sh

#Logging script appends a log message to the end of a log file in case it exists, or creates the log file and writes the log
message as the first line in case the log file does not exist.

if [[ $# -lt 2 || $# -gt 2 ]];then
    echo "$0: Argument error: preslog.sh [log file] [log message]"
    exit 2
fi

if [ -f "$1" ]
then
    date "+$2 %m-%d-%y %Hh%M" >> /Users/smarthome/shProject/ORC/webapps/controller/$1
    echo "$1 found."
else
    date "+$2 %m-%d-%y %Hh%M" > /Users/smarthome/shProject/ORC/webapps/controller/$1
    echo "$1 not found."
fi
```

All what is left is to add the code line to `presence.sh`, which calls the above logging script, right after the code line which writes the return event to the file `presence.html`:

```
echo "return $2" > /Users/smarthome/shProject/ORC/webapps/controller/presence.html
sh /Users/smarthome/shProject/preslog.sh presencelog.html "return Chris"
```

Keep in mind, that in case a shell argument (as in our case "return Chris") contains spaces, it needs to be surrounded by quotation marks. Or you bridge the space with an underscore: "return\_Chris". For better readability instead of logging the MAC address of the phone we use the name of the smartphone owner. For each smartphone you want to monitor, you now can easily set up the monitoring and logging function in parallel processes.

## 7.8 Testing the Script

In order to test our work we open a terminal window (*Applications — Utilities — Terminal*) and start the presence function typing:

```
./turnPresenceOn.sh
```

We open the presence status file *presence.html* and the presence log file *presencelog.html* in our web browser window:

```
file:///Users/smarthome/shProject/ORC/webapps/controller/presencefunction.html
```

```
file:///Users/smarthome/shProject/ORC/webapps/controller/presencelog.html
```

After waiting until the dead time since the start of our presence script has passed, we switch on the Wi-Fi function of our smartphone. Refreshing the two tabs in our web browser should display *on* in *presencefunction.html* and the return message along with date and time in *presencelog.html*.

We now open a new terminal window (*Applications — Utilities — Terminal*) and stop our presence function by typing:

```
./turnPresenceOff.sh
```

After refreshing the tab for *presencefunction.html* in our web browser, its content should read off

.

If the above works as desired, we can move on to the last step, which is to configure our presence function in OpenRemote Designer.

```
presence.sh
```

```
#!/bin/sh
```

```
#Presence detection using the smartphone DHCP request when booking ton a Wi-Fi (WLAN) network
```

```
if [[ $# -lt 2 || $# -gt 2 ]];then
```

```
    echo "$0: Argument error: presence.sh [dead time in seconds] [target MAC address]"
```

```
    exit 2
```

```
fi
```

```
while [ $(cat /Users/smarthome/shProject/ORC/webapps/controller/presencefunction.html) = "on" ]; do
```

```
# set timers for return detection timeout and variable for dhcp packet capture
```

```
    echo "idle" > /Users/smarthome/shProject/ORC/webapps/controller/presence.html
```

```
    timeout_in_seconds=$1;
```

```
    current_time_in_seconds=$(date +%s);
```

```
    ((trigger_time_in_seconds = $current_time_in_seconds + $timeout_in_seconds));
```

```
# capture until a dhcp request from MAC address 68:a8:6d:84:7d:42 is detected
```

```
tcpdump -i en1 -c 1 -v ether host $2 and dst port 67 > capture.txt
```

```
# if dhcp request is received before the timeout, do nothing, else set presence.html to "return_Chris"
```

```
    if [ "$(date +%s)" -le "$trigger_time_in_seconds" ]; then
```

```

    echo "return detection blocked"
else
    echo "return $2" > /Users/smarthome/shProject/ORC/webapps/controller/presence.html
    sh /Users/smarthome/shProject/preslog.sh prescelog.html "return Chris"
    cat /Users/smarthome/shProject/ORC/webapps/controller/presence.html
    sleep 60
    echo "idle" > /Users/smarthome/shProject/ORC/webapps/controller/presence.html
    cat /Users/smarthome/shProject/ORC/webapps/controller/presence.html
fi
done
echo "presence detection off"

```

*preslog.sh*

```
#!/bin/sh
```

#Logging script appends a log message to the end of a log file in case it exists, or creates the log file and writes the log message as the first line in case the log file does not exist.

```
if [[ $# -lt 2 || $# -gt 2 ]];then
```

```
    echo "$0: Argument error: preslog.sh [log file] [log message]"
```

```
    exit 2
```

```
fi
```

```
if [ -f "/Users/smarthome/shProject/ORC/webapps/controller/$1" ]
```

```
then
```

```
    date "+$2 %m-%d-%y %Hh%M" >> /Users/smarthome/shProject/ORC/webapps/controller/$1
```

```
    echo "$1 found."
```

```
else
```

```
    date "+$2 %m-%d-%y %Hh%M" > /Users/smarthome/shProject/ORC/webapps/controller/$1
```

```
    echo "$1 not found."
```

```
fi
```

turnPresenceOn.sh (dead time: 10 seconds)

```
#!/bin/sh
```

```
echo "on" > /Users/smarthome/shProject/ORC/webapps/controller/presencefunction.html
```

```
sh ./presence.sh 10 68:a8:6d:84:7d:42
```

*turnPresenceOff.sh*

```
#!/bin/sh
```

```
echo "off" > /Users/smarthome/shProject/ORC/webapps/controller/presencefunction.html
```

```
kill $(ps | egrep 'resence|tspdump' | awk '{print $1}')
```



*Table 7.5 Summary of presence detection scripts under OS X*

## 7.9 The Presence Detection Script under Windows 7 & 8

In Windows 7/8/XP we start PowerShell by selecting *Start — Accessories — Windows — PowerShell Windows — PowerShell ISE*. There are actually two versions of PowerShell: PowerShell ISE, which provides a graphical user interface with editor and terminal window integrated, and the standard PowerShell, which just gets you the terminal window with the PS prompt for PowerShell. Since it is convenient to use, we choose PowerShell ISE, which consists of three windows. These are the script editor in the upper window, the terminal input window (command pane) for command input in the middle, and the terminal output window at the bottom. You can actually switch the position of the command pane from the middle to the bottom and back up to the middle, using the green arrow at the very right side of its window. Just to avoid confusion, I will use it in the middle position in this book (Figure 7.3).

In case you have not worked with a command line interface before, there are a few commands you need at the very beginning in order to survive:

ls	listing - display the content of the current directory
pwd	print working directory - display the path of the current directory
cd ..	change directory followed by a space and two dots - move one directory hierarchy up
cd /target	change to the specified directory
cd	

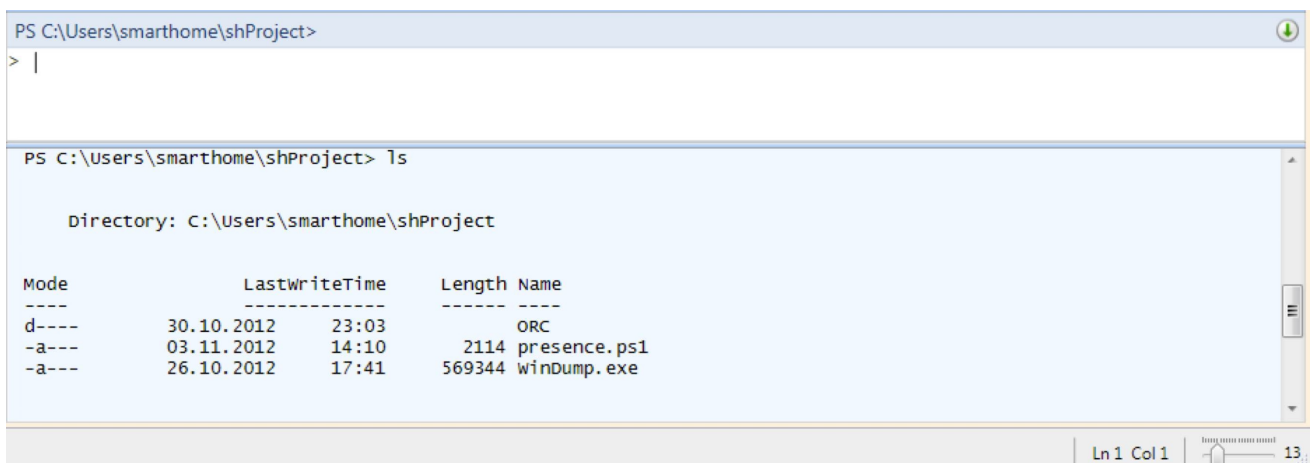
mkdir name	create (make) directory
man cmd	show the manual entry for a command

For the purpose of this book I have created the directory *shProject* in my home directory. In it I have saved *WindDump.exe* as well as the OpenRemote directory tree *ORC*. If you have never done a script before, and you want to follow exactly the instructions in the book, you may also create the directory *shProject* in your home directory by entering in the command pane:

```
mkdir shProject
```

```
ls
```

You can of course also use the Microsoft file manager application Explorer to create the directory. We enter `echo "Hello World"` in the editing window of our PowerShell and save the file as *presence.ps1* in our directory *\shProject*.



```
PS C:\Users\smarthome\shProject>
> |

PS C:\Users\smarthome\shProject> ls

Directory: C:\Users\smarthome\shProject

Mode                LastWriteTime         Length Name
----                -
d----              30.10.2012   23:03         ORC
-a---              03.11.2012   14:10        2114 presence.ps1
-a---              26.10.2012   17:41   569344 windump.exe
```

### *Figure 7.3 First steps in PowerShell*

Before we are able to run the script we need to change the PowerShell security settings, which per default prevent any script from execution. Typing `get-executionpolicy` gets us the following response:

```
PS C:\Users\smarthome\shProject> get-executionpolicy  
Restricted
```

To enable all of our own scripts to run and to restrict remote scripts to those which are digitally signed from a trusted source, we set the execution policy (for the user account we are using) to Remote Signed:

```
Set-ExecutionPolicy -Scope CurrentUser RemoteSigned
```

We now can execute our first PowerShell script by typing

`C:\Users\smarthome\shProject\presence.ps1` or alternatively, if we reside in the directory where the script is stored, just `.\presence.ps1` to get the below response.

```
.\presence.ps1
```

```
PS C:\Users\smarthome\shProject> .\presence.ps1
```

```
Hello World
```

Be aware that for security reasons you always need to precede the script name with `.\` or even the full path if the script is not in the current working directory.

## 7.10 Testing it Right - Best Practice for Script Writing

While scripts are really short pieces of code (e.g. our presence detection function counts less than twenty lines), they are very sensitive to syntax errors, and their functioning depends heavily on your system settings and your file structure. This means that you probably will need to make a few modifications to the source we provide in order to get it to run on your system.

Thus I would advise you NOT to copy and paste the code of the script as a whole onto your machine and execute to see what's happening.

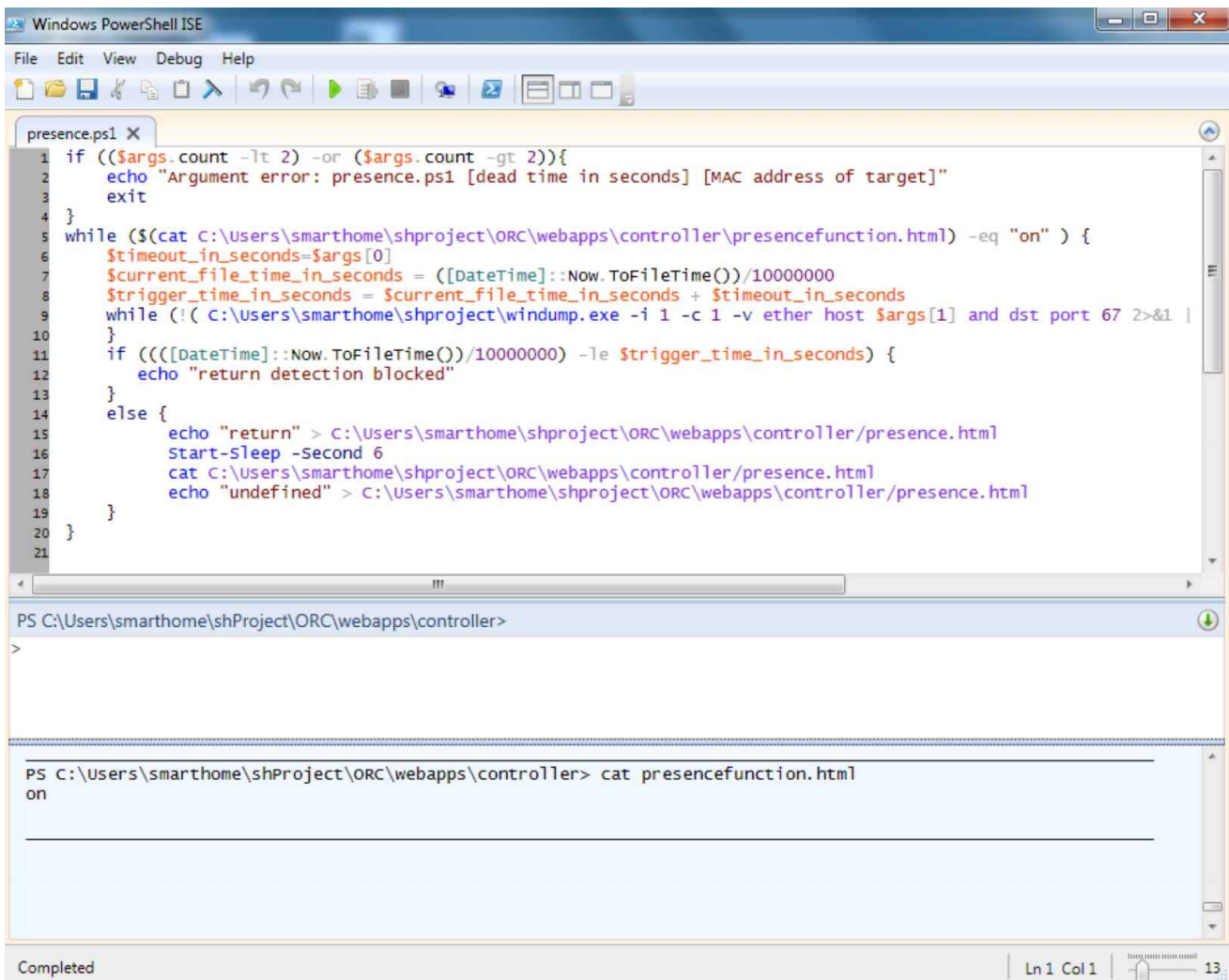
I recommend testing the script line by line, by copying each line, (or in case of a loop just the portion with the loop - as long it can run on its own) into the command input window. Execute it, observe if it works, and watch if the outcome is as expected. Once the line is tested and works as desired, you add it to the already cleared code in the editor window of the PowerShell ISE GUI and execute the script from there (push the green button). While doing this, you probably want to add a few temporary `echo` commands to the code, to be able to better observe what's going on. I promise this approach will get you to a working and stable script fast.

## 7.11 Building the Script

We can now get started writing our presence detection script. In order to be able to switch our presence function on and off, we create the file *presencefunction.html* with the content on and save it in the OpenRemote controller directory *ORC\webapps\controller\*, which is the directory where our OpenRemote project files are stored. It is also the root directory of the OpenRemote webserver, which gives us the option to access the file through a web browser or through the OpenRemote HTTP protocol function. We type the following two lines in the terminal window portion of the PowerShell GUI:

```
cd ORC\webapps\controller\  
echo "on" > presencefunction.html
```

The first command changes the directory to the OpenRemote controller directory *ORC\webapps\controller\*. If you have installed OpenRemote in another directory, you need to alter the path accordingly. The second command writes the string on to the file *presencefunction.html*. In case the file exists, previous content is overwritten, if it does not exist, the file is created. To validate if we have done everything right, we can use the `cat` command, typing it into the middle section of the PowerShell ISE GUI and hitting return, to read out the content of *presencefunction.html* (Figure 7.4).



```
Windows PowerShell ISE  
File Edit View Debug Help  
presence.ps1 X  
1 if (($args.count -lt 2) -or ($args.count -gt 2)){  
2   echo "Argument error: presence.ps1 [dead time in seconds] [MAC address of target]"  
3   exit  
4 }  
5 while (($cat C:\Users\smarthome\shproject\ORC\webapps\controller\presencefunction.html) -eq "on" ) {  
6   $timeout_in_seconds=$args[0]  
7   $current_file_time_in_seconds = ([DateTime]::Now.ToFileTime())/10000000  
8   $trigger_time_in_seconds = $current_file_time_in_seconds + $timeout_in_seconds  
9   while (!(C:\Users\smarthome\shproject\windump.exe -i 1 -c 1 -v ether host $args[1] and dst port 67 2>&1 |  
10  })  
11   if ((([DateTime]::Now.ToFileTime())/10000000) -le $trigger_time_in_seconds) {  
12     echo "return detection blocked"  
13   }  
14   else {  
15     echo "return" > C:\Users\smarthome\shproject\ORC\webapps\controller\presence.html  
16     Start-Sleep -Second 6  
17     cat C:\Users\smarthome\shproject\ORC\webapps\controller\presence.html  
18     echo "undefined" > C:\Users\smarthome\shproject\ORC\webapps\controller\presence.html  
19   }  
20 }  
21  
PS C:\Users\smarthome\shProject\ORC\webapps\controller>  
>  
PS C:\Users\smarthome\shProject\ORC\webapps\controller> cat presencefunction.html  
on  
Completed | Ln 1 Col 1 | 13
```

Figure 7.4 Creation of the control file *presencefunction.html*

Later we will implement the above step in our startup script and use the content of *presencefunction.html* as the condition for our shell script to run. In order to switch our presence detection function on and off, we also create two simple batch files, which we call *turnPresenceOff.ps1* and *turnPresenceOn.ps1*. Each of the files contain just the one line, which writes on or off to *presencefunction.html*:

```
echo "off" > .\shProject\ORC\webapps\controller\presencefunction.html
```

```
echo "on" > .\shProject\ORC\webapps\controller\presencefunction.html
```

Like *presence.html* both files need to be located in the root directory of the OpenRemote webserver:

```
ORC\webapps\controller\
```

Next we build the timer that defines the dead time of our function. As explained above, we do not want the presence scenario to become active if the smartphone is just switched off and back on. Thus we want the capability to define a minimum required delay between two subsequent presence trigger events. In order to do this, we store the absolute time in seconds at the startup of the script in the variable `current_file_time_in_seconds` and the event trigger time in `trigger_time_in_seconds`:

```
$timeout_in_seconds=10
```

```
$current_file_time_in_seconds = ([DateTime]::Now.ToFileTime())/10000000
```

```
$trigger_time_in_seconds = $current_file_time_in_seconds + $timeout_in_seconds
```

A few words of explanation to the three code lines above:

In PowerShell variables are always preceded with the \$ sign:

```
$timeout_in_seconds=10
```

To get the current timestamp we use the static method `NOW` of the class `[DateTime]`. The specifier `ToFileTime()` formats the output in Windows file time format. The Windows file time is a 64-bit value that represents the number of 100-nanosecond intervals that have elapsed since 1.1.1601. (The year 1601 is the beginning of the 400-year Gregorian leap-year cycle. For comparison, the Unix timestamp represents the number of seconds elapsed since 1.1.1970). In order to convert the Windows file time to seconds we divide it by 10,000,000. Timestamps are easier to work with than human readable time and date formats, especially when comparing, adding or subtracting times or dates. Numerical operations in PowerShell are simply conducted using `+` `-` `*` and `/`, as well as parentheses to manage the order of the operations.

We now move on to the next two lines of code:

```
C:\Users\smarthome\shProject\windump.exe -i 1 -c 1 -v ether host 68:a8:6d:84:7d:42 and dst port 67 2>&1 | select-string -pattern "DHCP"
```

In order to use our `windump` statement from above in PowerShell, we add the full path of the *windump.exe* location to our command. (In general in PowerShell we need to provide the complete file path in order to execute or manipulate a file. That's true regardless of our

location within the file system). We further use the pipe command `|` to route the `windump` output to the `select-string` Cmdlet with the argument `-pattern`. In PowerShell `Select-string` is the function for string search, which will return a logical `True`, if the string that is fed to it contains the specified pattern. In our case we just want to see the subset of the `WinDump` output that contains the string DHCP. If we did not want any output, we could further add the option `-quiet` at the end of our statement. (We will not further use anything from the trace output of our `WinDump` function, unless for troubleshooting.)

You might have noticed that before the pipe command we have added another cryptic looking statement:

```
2>&1
```

This statement is due to the fact that `windump.exe` writes its output to the PowerShell standard error variable `stderr`, which stands for standard error. The standard output definitions are important and you will frequently run across them in scripts. PowerShell (like all shells) defines three pre-defined output descriptors:

descriptor 0 references standard input (`stdin`)

descriptor 1 references standard output (`stdout`), and

descriptor 2 references standard error (`stderr`).

By stating `2>&1` we simply redirect `stderr` output (descriptor 2) to `stdout` ( descriptor 1), using the redirect command `>&`.

We can now add the `windump` command to our script in the PowerShell editor window, which by now stores the current time in form of a Windows file timestamp, calculates the dead time of our function and starts `WinDump` with our filter condition, waiting for a match event. Only if a match occurs, the script will move on to the next set of commands. We can test the script in the PowerShell ISE GUI by clicking on the green button. The script starts and waits until our `WinDump` filter finds a match. We switch our smartphone to airplane mode and back and observe the output of `WinDump` in the terminal output window (Figure 7.5).

```

presence.ps1 X
1 $timeout_in_seconds=1000
2 $current_time_in_seconds = get-date -uformat %s
3 $trigger_time_in_seconds = $current_time_in_seconds + $timeout_in_seconds
4 while (!(C:\Users\smarthome\shproject\windump.exe -i 1 -c 1 -v ether host 68:a8:6d:84:7d:42 and dst port 67 2>&1 | select-string
5 }

PS C:\Users\smarthome>
>

PS C:\Users\smarthome> C:\Users\smarthome\shProject\presence.ps1

Completed | Ln 5 Col 2 | 12

```

Figure 7.5 The core of the presence detection script: WinDump while loop

All we need to do now is, to check whether the dead time has passed if a presence match has occurred. For this we will use an if statement. The general format of the if statement in PowerShell is:

```

if (condition) {
    echo "Test1"
}
else {
    echo "Test2"
}

```

As the condition for the if statement we compare the current time, for which we again use the command for the Windows file time, with our variable `$trigger_time_in_seconds`:

```

if ((([DateTime]::Now.ToFileTime())/10000000) -le $trigger_time_in_seconds) {

```

The compare command `-le` stands for less than or equal, other compare options in PowerShell are `-eq`, `-gt`, or `-ge`). If the timer has run out, we write `return` followed by the MAC address of the device which has triggered our function into our result file `presence.html`. After sixty seconds (command `Start-Sleep -Second 60`), which is sufficient for the polling interval of our OpenRemote sensor, as we will see further down, we set the content of `presence.html` back to idle. If the timer has not run out, we do nothing and simply print to the terminal window: `echo "return detection blocked"`:

```

if ((([DateTime]::Now.ToFileTime())/10000000) -le $trigger_time_in_seconds) {
    echo "return detection blocked"
}
else {
    echo "return_Chris" > C:\Users\smarthome\shProject\ORC\webapps\controller\presence.html

```



```
Start-Sleep -Second 60
```

```
cat C:\Users\smarthome\shProject\ORC\webapps\controller\presence.html
```

```
echo "idle" > C:\Users\smarthome\shProject\ORC\webapps\controller\presence.html
```

```
}
```

We are now almost done. What is left is to run the above process within a `while` loop, which terminates only if the content of our switch file `presencefunction.html` is set to `off`. The `while` statement in PowerShell has the following general syntax:

```
$var = 0
```

```
while ($var -lt 10) {
```

```
    echo $var
```

```
    $var++
```

```
}
```

Our `while` loop will resemble the following:

```
while ($(cat C:\Users\smarthome\shProject\ORC\webapps\controller\presencefunction.html) -eq "on" ) {
```

```
    .....
```

```
}
```

Table 7.4 shows the complete listing of our script. During the test phase you will probably set `$timeout_in_seconds` to something like 10 seconds, and `Start-Sleep` to something like 5 seconds. This will allow you to quickly test the various conditions of the script by switching your smartphone's Wi-Fi function on and off.

```
while ($(cat C:\Users\smarthome\shProject\ORC\webapps\controller\presencefunction.html) -eq "on" ) {
```

```
    $timeout_in_seconds=10
```

```
    $current_file_time_in_seconds = ([DateTime]::Now.ToFileTime())/10000000
```

```
    $trigger_time_in_seconds = $current_file_time_in_seconds + $timeout_in_seconds
```

```
    while (!( C:\Users\smarthome\shProject\windump.exe -i 1 -c 1 -v ether host 68:a8:6d:84:7d:42 and dst port 67  
2>&1 | select-string -pattern "68:a8:6d:84:7d:42" -quiet )) {
```

```
    }
```

```
    if ((([DateTime]::Now.ToFileTime())/10000000) -le $trigger_time_in_seconds){
```

```
        echo "return detection blocked"
```

```
    }
```

```
    else {
```

```
        echo "return_Chris" > C:\Users\smarthome\shProject\ORC\webapps\controller\presence.html
```

```
        Start-Sleep -Second 60
```

```
        cat C:\Users\smarthome\shProject\ORC\webapps\controller\presence.html
```

```
        echo "idle" > C:\Users\smarthome\shProject\ORC\webapps\controller\presence.html
```

```
    }
```

```
}
```

## Table 7.4 Shell script for presence detection (MS Windows PowerShell)

Finally, in order to make our script a bit more flexible, we will replace the values for dead time and MAC address with the special PowerShell variable `$args`, which contains parameters, passed to a function in form of an array. Using `$args` we can further determine the number of arguments passed to a script using the `.count` specifier:

```
echo $($args.count)
```

And we can access the value of each parameter by referencing the individual array elements:

```
echo $args[0]
```

```
echo $args[1]
```

We replace the value 10, which is our the dead time in seconds, with `echo $args[0]` and the MAC address (68:a8:6d:84:7d:42) with `echo $args[1]`.

```
$timeout_in_seconds=$args[0]
```

```
.....
```

```
while (!( C:\Users\smarthome\shProject\windump.exe -i 1 -c 1 -v ether host $args[1] and dst port 67 2>&1 | select-  
string -pattern $args[1] -quiet )) {  
    }
```

```
.....
```

Our script now needs to be started by typing the script name followed by the two arguments for dead-time and MAC address, which the script now expects:

```
.\presence.ps1 10 68:a8:6d:84:7d:42
```

If we are not in the directory, where our script resides, we have to specify the entire path:

```
C:\Users\smarthome\shPProject\presence.ps1 10 68:a8:6d:84:7d:42
```

To round our script off, we add some basic error handling at the beginning, which makes sure that the correct number of arguments that are being specified:

```
if (($args.count -lt 2) -or ($args.count -gt 2)){  
    echo "Argument error: presence.ps1 [dead time in seconds] [MAC address of target]"  
    exit  
}
```

We use the special variable `$args` with the specifier `.count` to retrieve the number of arguments that are passed on to the script. The `if` statement validates whether the number of arguments provided is exactly 2. If less than two or more than two arguments are passed, an error message is displayed, and the script is terminated using the `exit` command.

```
if (($args.count -lt 2) -or ($args.count -gt 2)){  
    echo "Argument error: presence.ps1 [dead time in seconds] [MAC address of target]"  
    exit  
}  
  
while (($cat C:\Users\smarthome\shProject\ORC\webapps\controller\presencefunction.html) -eq "on" ) {
```

```

$timeout_in_seconds=$args[0]
$current_file_time_in_seconds = ([DateTime]::Now.ToFileTime())/10000000
$strigger_time_in_seconds = $current_file_time_in_seconds + $timeout_in_seconds
while (!( C:\Users\smarthome\shProject\windump.exe -i 1 -c 1 -v ether host $args[1] and dst port 67 2>&1 | select-
string -pattern $args[1] -quiet )) {
}
if ((([DateTime]::Now.ToFileTime())/10000000) -le $trigger_time_in_seconds) {
    echo "return detection blocked"
}
else {
    echo "return $args[1]" > C:\Users\smarthome\shProject\ORC\webapps\controller\presence.html
    Start-Sleep -Second 60
    cat C:\Users\smarthome\shProject\ORC\webapps\controller\presence.html
    echo "idle" > C:\Users\smarthome\shProject\ORC\webapps\controller\ presence.html
}
}
echo "presence detection stopped"

```

*Table 7.5 The final shell script for presence detection in MS Windows PowerShell*

Keep in mind that you will need to update the path definitions for the script if you change your OpenRemote installation directory.

We now complete our script *turnPresenceOn.ps1* by adding the command to start the presence detection function with the desired MAC address and dead time. With that *turnPresenceOn.ps1* reads:

```

echo "on" > C:\Users\smarthome\shProject\ORC\webapps\controller\presencefunction.html
C:\Users\smarthome\shProject\presence.ps1 1000 68:a8:6d:84:7d:42

```

There is one addition we need to make to *turnPresenceOff.ps1*, which is to stop the WinDump process. Otherwise presence detection would only stop after the next presence event, which could take some time. Therefore we add the line `kill -Name Windump` to our script, which now reads:

```

echo "off" > C:\Users\smarthome\shProject\ORC\webapps\controller\presencefunction.html
kill -name WinDump

```

The command `kill -name` followed by a process terminates all processes with that name. The command for displaying background processes is

```
ps
```

The option `-WhatIf` used with the `kill` command simulates the command without doing anything and can be used to test the command before actually invoking it. It just outputs what it would do without the `-WhatIf` option. Adding the option `-confirm` asks for

confirmation before executing the actual `kill` command:

```
kill -Name WinDump -confirm
```

```
kill -Name WinDump -WhatIf
```

## 7.12 Log File for Presence Detection

In addition to writing the return message to the file *presence.html* we want to log every return event along with date and time to the file *presencelog.html*. While *presence.html* serves as the real time sensor output, which reports a return for 60 seconds and then goes back to idle, waiting for the next return event, *presencelog.html* shall contain a history of all return events of which the last entry of which we want to display via the GUI of our smartphone app. For this purpose we write another brief Powershell script, which uses the arguments `$args[0]` as log entry and `$args[0]` as log filename.

We further use the `Test-Path Cmdlet` to verify whether the logfile already exists. It returns `True` if the file exists, and returns `False` if the file does not exist:

```
Test-Path c:\scripts\test.txt
```

Now we can create our presence logging script *preslog.ps1* which either creates the log file if it does not exist yet and writes the first log entry or, if it does exist, appends the log entry at the end of the file. As a presence logging file we again use an html file located in the root directory of the OpenRemote webserver under *ORC\webapps\controller\* in order to have the option to access its content through a web browser or through the OpenRemote HTTP protocol function. At the beginning of the script we validate whether the correct number of arguments are provided, when the script is called:

```
if (($args.count -lt 2) -or ($args.count -gt 2)){  
    echo "Argument error: preslog.ps1 [log file] [log message]"  
    exit  
}
```

The first argument `$args[0]` shall be the name of the log file, the second argument `$args[1]` the log message. If the log file exists, we append (`>>`) date and time (which we store in the variable `$now`) along with the log message to the existing file content. If it does not exist, we create the file (`>`) and write date, time and message to it:

```
if (($args.count -lt 2) -or ($args.count -gt 2)){  
    echo "Argument error: preslog.ps1 [log file] [log message]"  
    exit  
}  
  
$path="C:\Users\smarthome\shProject\ORC\webapps\controller\" + $args[0]  
  
if (Test-Path $path){  
    $now = [DateTime]::Now  
    echo $args[1] $now >> $path  
}  
  
else {  
    $now = [DateTime]::Now  
    echo $args[1] $now > $path  
    echo $args[1] not found
```

```
}
```

We can now test our script typing:

```
.\preslog.ps1 presencelog.html "return Chris"
```

All what is left is to add the code line to *presence.ps1*, which calls the above logging script, right after the code line, which writes the return event to the file *presence.html*:

```
echo "return $args[1]" > C:\Users\smarthome\shProject\ORC\webapps\controller\presence.html
```

```
.\preslog.ps1 presencelog.html "return Chris"
```

Keep in mind that if a shell argument (as in our case "return Chris") contains spaces, it needs to be surrounded by quotation marks. Or you can bridge the space with an underscore:

*return\_Chris*. For better readability instead of logging the MAC address of the phone we use the name of the smartphone owner. For each smartphone you want to monitor, you now can easily set up the monitoring and logging function in parallel processes.

In order to control our Powershell scripts from OpenRemote, we will start them from the command line without opening Powershell, which can be done by typing *powershell.exe* followed by the full path to the script and (if required) associated script parameters, for example:

```
powershell.exe C:\Users\smarthome\shProject\turnPresenceOn.ps1
```

## 7.13 Testing the Script

In order to test our work we open the terminal window (*Start — CMD*) and start the presence function by typing:

```
powershell.exe C:\Users\smarhome\shProject\turnPresenceOn.ps1
```

(In order to validate that *windump.exe* has started you can open a second terminal window and type the command *tasklist*. Among the processes listed you should see a new one called *windump.exe*).

We now open in our web browser the presence status file *presencefunction.html* and the presence log file *presencelog.html*:

```
file:///Users/smarhome/shProject/ORC/webapps/controller/presencefunction.html
```

```
file:///Users/smarhome/shProject/ORC/webapps/controller/presencelog.html
```

After waiting until the dead time since the start of our presence script has passed, we switch the Wi-Fi function of our smartphone on. Refreshing the two tabs in our web browser should now display *on* in *presencefunction.html*, and the return message in *presencelog.html*.

We open a new terminal window (*Start — CMD*) and stop our presence function by typing:

```
powershell.exe C:\Users\smarhome\shProject\turnPresenceOff.ps1
```

Refreshing the tab for *presencefunction.html* in our web browser should now display *off* .

If the above works as desired, we can move on to the last step, which is to configure our presence function in OpenRemote Designer.

```
presence.ps1
```

```
if (($args.count -lt 2) -or ($args.count -gt 2)){
    echo "Argument error: presence.ps1 [dead time in seconds] [MAC address of target]"
    exit
}
while ($(cat C:\Users\smarhome\shProject\ORC\webapps\controller\presencefunction.html) -eq "on" ) {
    $timeout_in_seconds=$args[0]
    $current_file_time_in_seconds = ([DateTime]::Now.ToFileTime())/10000000
    $trigger_time_in_seconds = $current_file_time_in_seconds + $timeout_in_seconds
    while (!( C:\Users\smarhome\shProject\windump.exe -i 1 -c 1 -v ether host $args[1] and dst port 67 2>&1 | select-string -pattern $args[1] -quiet )) {
    }
    if ((([DateTime]::Now.ToFileTime())/10000000) -le $trigger_time_in_seconds) {
        echo "return detection blocked"
    }
    else {
```

```

    echo "return $args[1]" > C:\Users\smarthome\shProject\ORC\webapps\controller\presence.html
.\preslog.ps1 prescelog.html "return Chris"
    cat C:\Users\smarthome\shProject\ORC\webapps\controller\presence.html
    Start-Sleep -Second 60
    echo "idle" > C:\Users\smarthome\shProject\ORC\webapps\controller\presence.html
}
}
echo "presence detection stopped"

```

*preslog.ps1*

```

if (($args.count -lt 2) -or ($args.count -gt 2)){
    echo "Argument error: preslog.ps1 [log file] [log message]"
    exit
}
$path="C:\Users\smarthome\shProject\ORC\webapps\controller\" + $args[0]
if (Test-Path $path){
    $now = [DateTime]::Now
    echo $args[1] $now >> $path
}
else {
    $now = [DateTime]::Now
    echo $args[1] $now > $path
    echo $args[1] not found
}

```

*turnPresenceOn.ps1 (dead time: 10 seconds)*

```

echo "on" > C:\Users\smarthome\shProject\ORC\webapps\controller\presencefunction.html
C:\Users\smarthome\shProject\presence.ps1 10 68:a8:6d:84:7d:42

```

*turnPresenceOff.ps1*

```

echo "off" > C:\Users\smarthome\shProject\ORC\webapps\controller\presencefunction.html
kill -Name Windump

```

*Table 7.6 Summary of Powershell presence detection scripts*



## 7.14 Controlling Presence Detection via Smartphone

As the last step, we want to add handling and a status display of the presence detection function to our OpenRemote based universal smartphone remote app. In OpenRemote Designer we create a new device called Presence Detection and define the three commands Turn Presence On, Turn Presence Off and Presence Status. For Turn Presence On and Turn Presence Off we select *Shell Execution Protocol*. The *OpenRemote Shell Execution Protocol* supports multiple parameters, which simply have to be separated by spaces. This is what we use under Windows now, since the command lines we need to configure read:

```
powershell.exe C:\Users\smarthome\shProject\turnPresenceOff.ps1
```

```
powershell.exe C:\Users\smarthome\shProject\turnPresenceOn.ps1
```

We enter `powershell.exe` in the *Path* field and the actual path to our script as the first parameter in the field *Command parameter* (Figure 7.6).

Under OS-X we simply enter

```
/Users/smarthome/shProject/turnPresenceOn.sh
```

and

```
/Users/smarthome/shProject/turnPresenceOff.sh
```

in the *Path* field of the command definition window and can leave the *Command parameter* field empty.

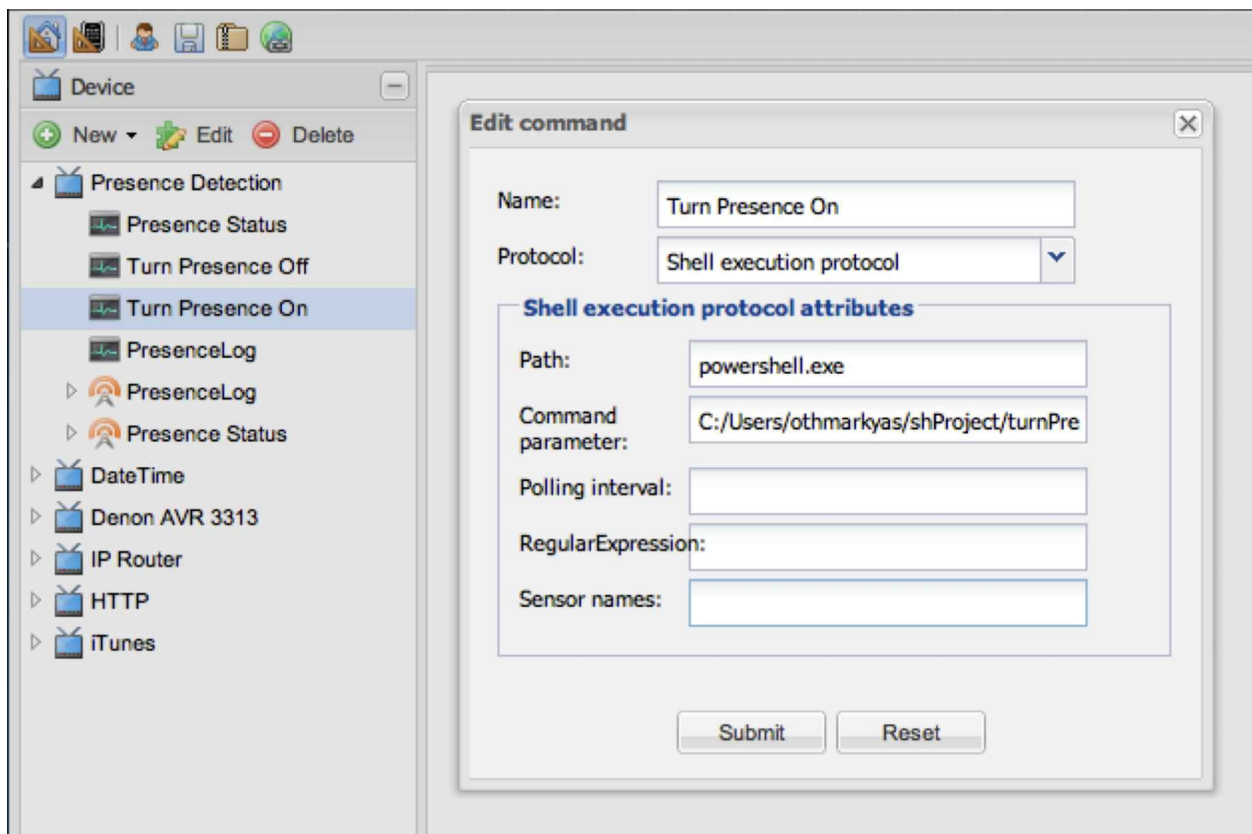
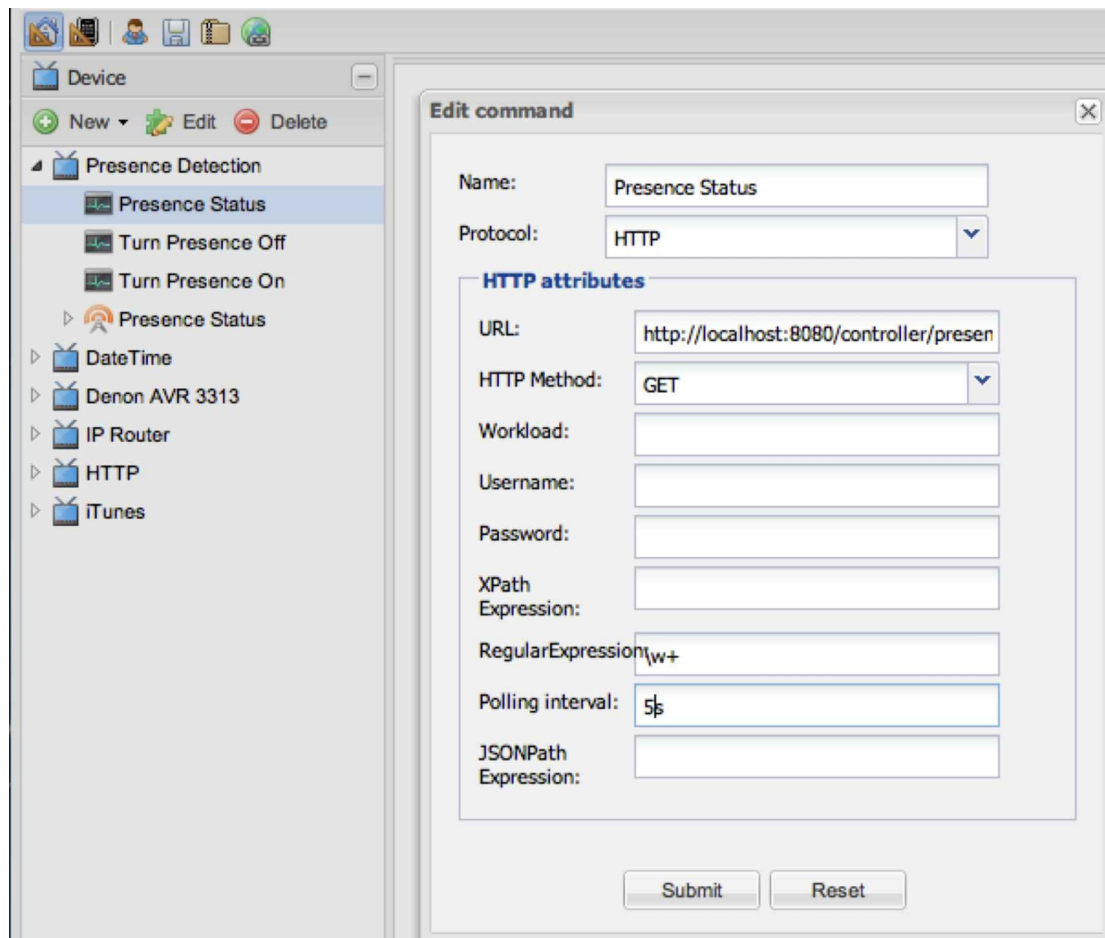


Figure 7.6 OpenRemote Command Definition Turn Presence On for MS Windows Powershell

For the command Presence Status we select the HTTP protocol, provide the URL to the local OpenRemote web server for the file `presencefunction.html` in the **URL** field

`http://localhost:8080/controller/presencefunction.html`

and select GET for the field **HTTP method**. As a regular expression we select `+\\w`, which matches multiple times (+) to any alphanumeric character (\\w) (Figure 7.7).



*Figure 7.7 OpenRemote Command Definition for Presence Status using HTTP GET*

Now we can create the sensor **Presence Status** using the command Presence Status which we just defined (Figure 7.8).

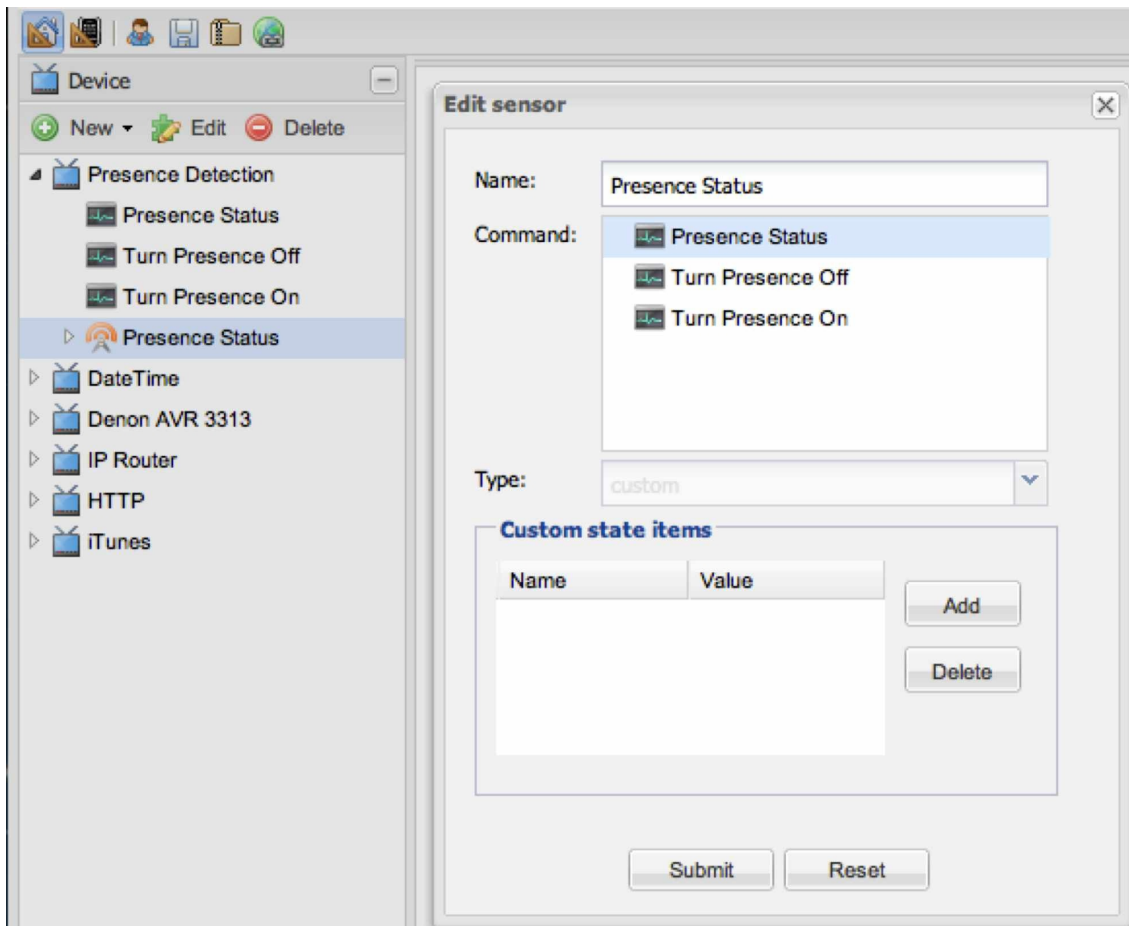


Figure 7.8 OpenRemote Sensor Definition for Presence Status

As the last step we add the GUI controls in the OpenRemote UI Designer window. We add the four grid elements, name field (*Presence Detection*), status display sensor (*Presence Status*) and the two switch commands (*Turn Presence On*, *Turn Presence Off*) for the presence function (Figure 7.9). After synchronizing our local controller with the updated design, we can toggle our presence detection function on and off, and get an updated display of the function status on our smart home control app. (A more detailed description on how to create GUI elements in OpenRemote Designer can be found in chapter 6.

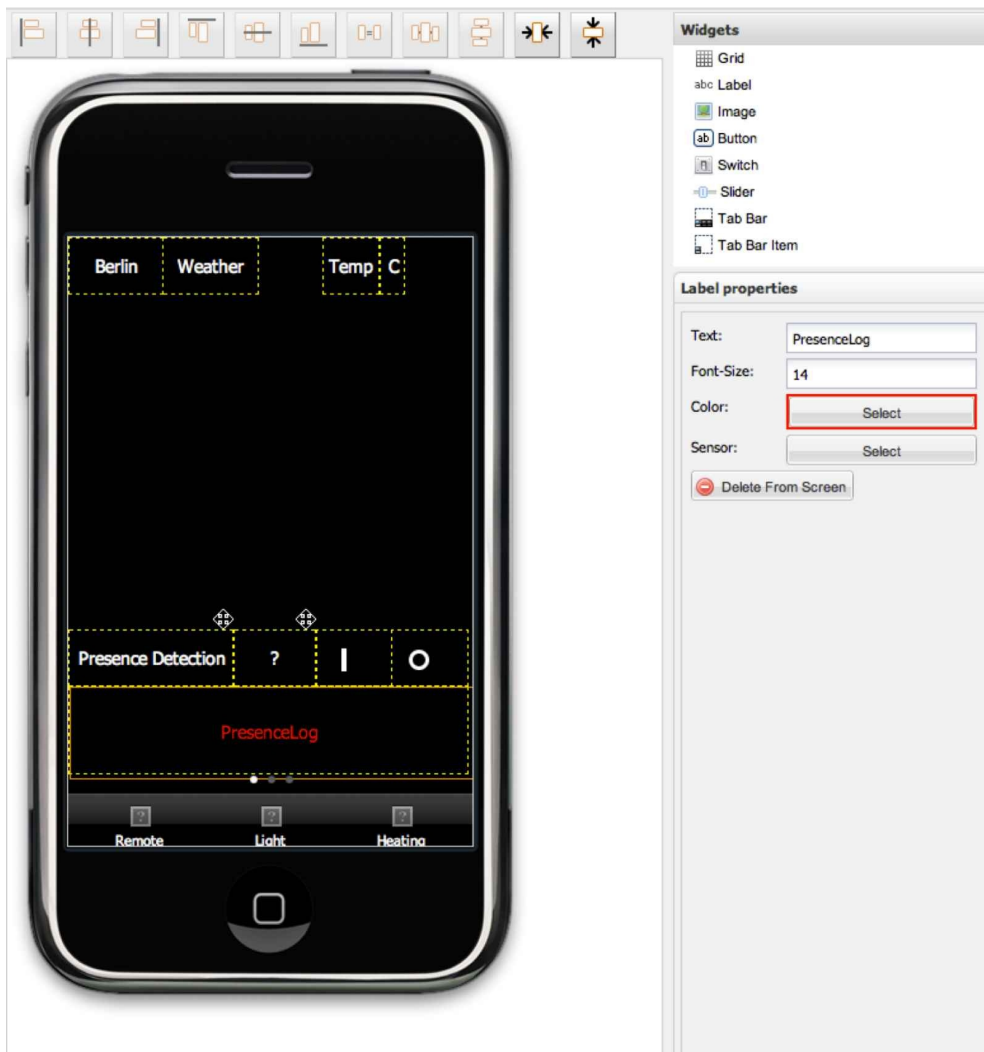
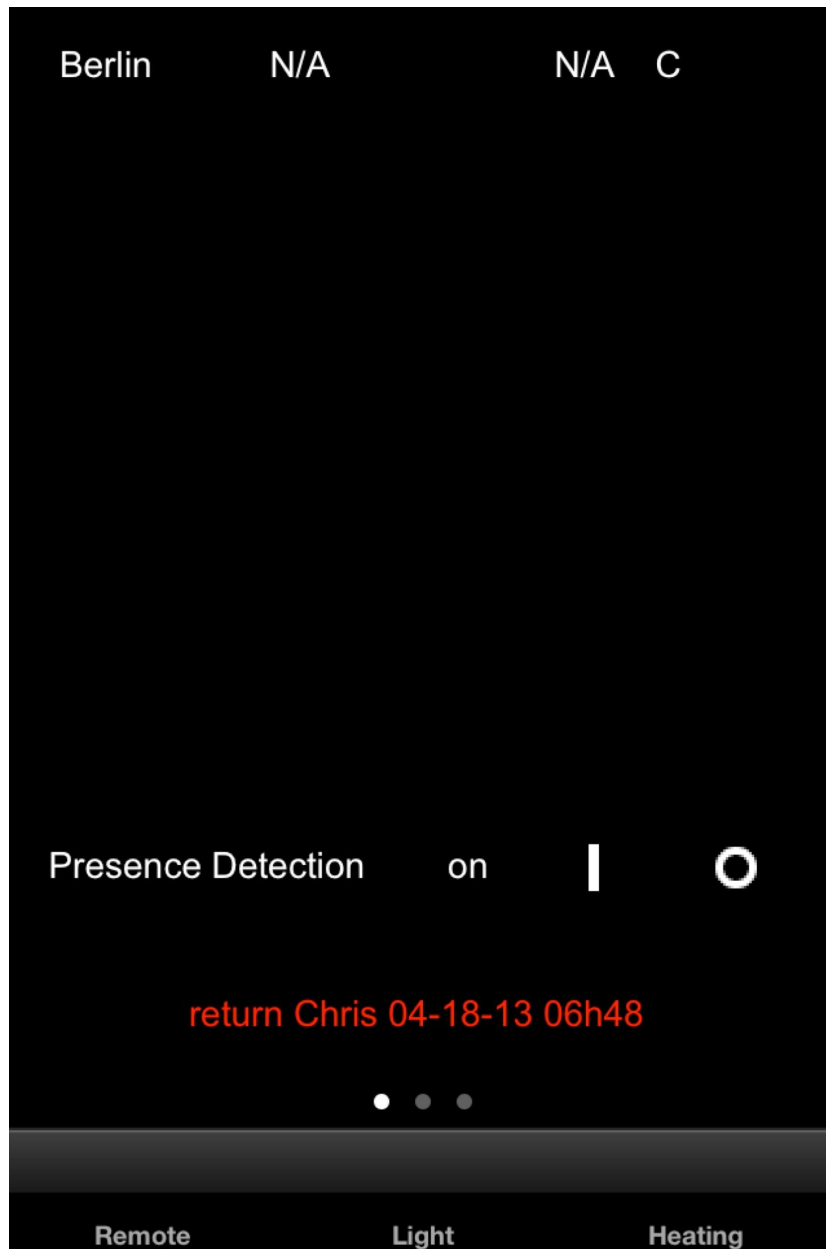


Figure 7.9 Designing the OpenRemote controls for presence detection

To display the log entry of the last return event we create a command called *PresenceLog*, which retrieves the last entry from the file *presencelog.html* using the HTTP protocol, the local URL `http://localhost:8080/controller/presencelog.html` the *HTTP method* GET and the regular expression `((?!.*return)).*$`. The core of the regular expression is the negative look ahead portion `(?!return)`, which does not match if followed by `return`. (We want to match the last occurrence of `return` in our string, since *presencelog.html* contains a history of all return events, and we are just interested in the last one.) To prevent the regex from matching at the line feeds at the end of each log entry, we expand the definition to only match, if the match occurs at the end of the string using the dollar sign. Since the first match for the string that is not followed by a `return` is `eturn`, we add a dot at the start of the expression, which adds `r` back, and we are done with the command. We now create a sensor for the above command, which we also call *PresenceLog*. In the OpenRemote UI designer we add a grid element below the presence function controls containing a label, which references the *PresenceLog* sensor and are done (Figure 7.10).



*Figure 7.10 Presence detection and weather display on the smart home app*

Writing the log entries and status displays into html files, which reside in the root directory of the OpenRemote webservice, has the advantage that we can access these files via a web browser. With that we can also easily design a smart home status summary webpage that displays the content of these files. We can now test our presence app by leaving the range of our Wi-Fi network and coming back, while we observe if our return is logged correctly. Make sure that your smartphone or tablet is configured in a way that the Wi-Fi function is active even while in sleep mode. On an iPhone under iOS 6, for example you need to configure *Settings — Auto — Lock — Never*. Then also in sleep mode, with the screen black, the Wi-Fi function is in hunt mode, and will connect to a Wi-Fi network once one detected. Compared to 3G, GPS, or the brightness of a large screen, today the battery drain of Wi-Fi is small, and in state of the art smartphones and tablets it will not be a major factor for battery life. With an increasing number of applications requiring push notification via Wi-Fi, the Wi-Fi always on configuration has become the preferred operating mode for an increasing number of users.



## 8 Integration of Multimedia: iTunes Remote

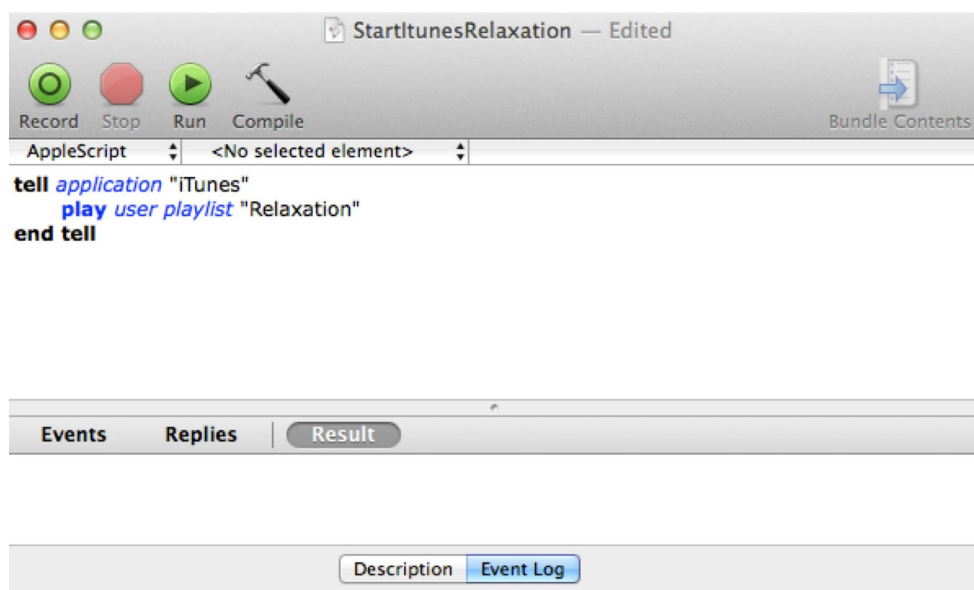
The ability to control scripts via OpenRemote also allows us to integrate the capabilities that state of the art multimedia PCs have, such as playing audio and video files, TV programs, and radio stations. In this chapter we will set up our OpenRemote smartphone app to function as a remote control for Apple's multimedia suite iTunes for Macs as well as for PCs. In chapter nine we will then demonstrate how to create automated rules, which allow us to put together the control components, we have created so far into a powerful, rule based smart home control application.

## 8.1 Script Based iTunes Control in OS X

Under OS X writing a script, which involves interaction with applications, is easiest using the powerful scripting language AppleScript. The AppleScript specification is based on the Open Scripting Architecture (OSA). Therefore any AppleScript can be executed from a shell using the `osascript` command. To get started with AppleScript it is best to start using the AppleScript Editor application. Go to *Applications — Utilities — AppleScript Editor* to start the AppleScript development application. Enter the following code, replacing Relaxation with a playlist that actually exists on your PC:

```
tell application "iTunes"
    play user playlist "Relaxation"
end tell
```

Then select run and iTunes should launch and start to play the selected playlist. (Fig. 8.1)



*Figure 8.1 Working with the AppleScript Editor*

If we want to run the above in a shell script, we just need to insert the `osascript` command in the first line and tell the script interpreter to handle all text following `exec osascript AS` an `osascript` until EOF is reached:

```
exec osascript << EOF
tell app "iTunes"
    play user playlist "$1"
end tell
EOF
```

Adding the possibility to set the playlist through an argument as well as an argument error procedure gives us the final version of our script `startItunes.sh`:

```
#!/bin/sh
#Presence detection using the smartphone DHCP request when booking ton a Wi-Fi (WLAN) network
if [[ $# -lt 1 || $# -gt 1 ]];then
```



```
    echo "$0: Argument error: itunesStart.sh [playlist]"
    exit 2
fi
exec osascript << EOF
tell app "iTunes"
    play user playlist "$1"
end tell
EOF
```

## The command

startItunes.sh Relaxations

now opens iTunes and starts playing the playlist Relaxation. As a last step we want the script to redirect the iTunes output via AirPlay to an output of choice. Since there is no direct AppleScript command for this, we need to simulate the user interaction, which is to click on the drop-down menu *Choose which speakers to use* and select the *AirPlay* device of choice. To do this we need to make *iTunes* the active window and then simulate the drop-down click

click (first UI element whose help is "Choose which speakers to use")

and the selection of the desired AirPlay device, in our case the Denon AVR-3313. The command

```
keystroke "DENON"
```

will literally type the letters "DENON" and by doing this select the AirPlay device called "DENON" in the drop down window opened by the `click` command. The command

```
key code 76
```

operates the return-key and the one-second delay makes sure the GUI can follow the speed of AppleScript. Before executing the `keystroke` command we insert the command line `tell application "iTunes" to activate` to avoid it being written into another window, which might have opened in the meantime:

```
tell application "iTunes" to activate
```

```
delay 1
```

```
tell application "System Events"
```

```
    tell window "iTunes" of process "iTunes"
```

```
        click (first UI element whose help is "Choose which speakers to use.")
```

```
    delay 1
```

```
    tell application "iTunes" to activate
```

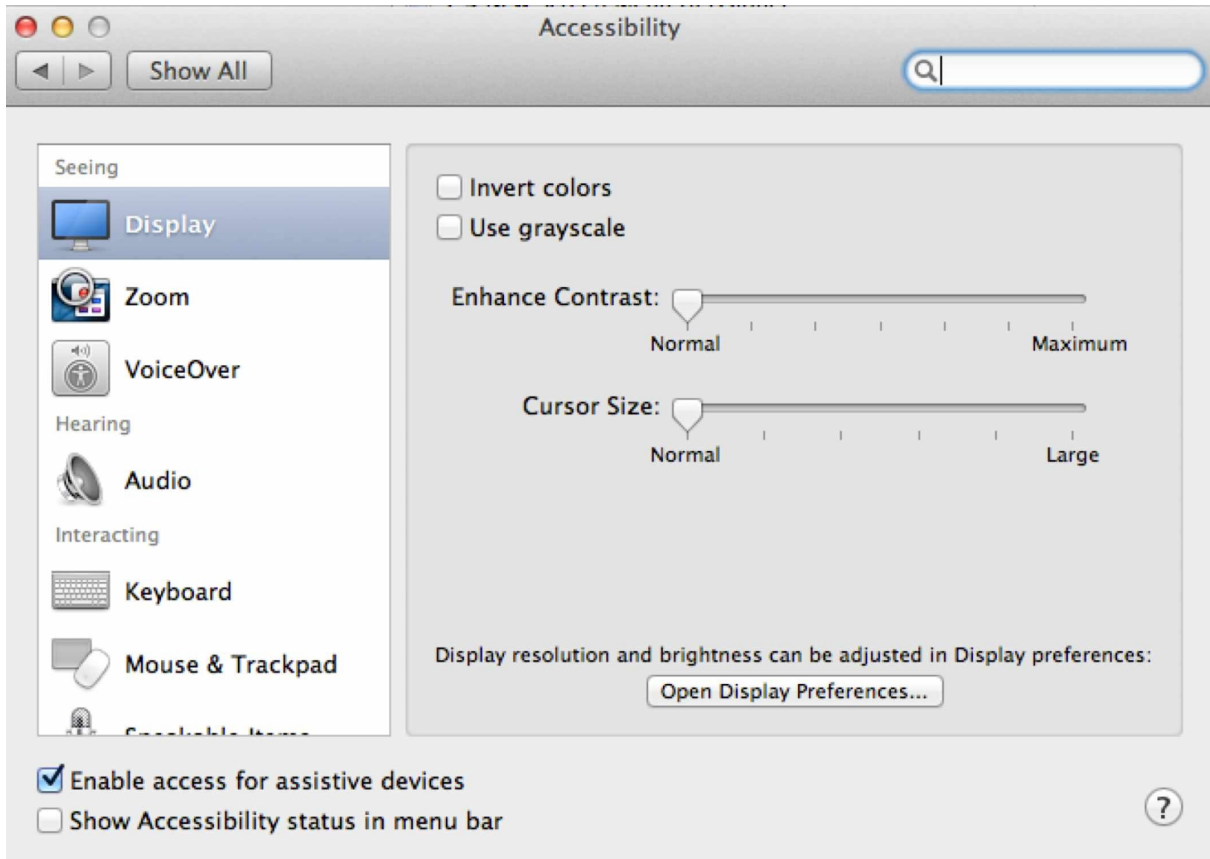
```
        keystroke "DENON"
```

```
    delay 1
```

```
    key code 76
```

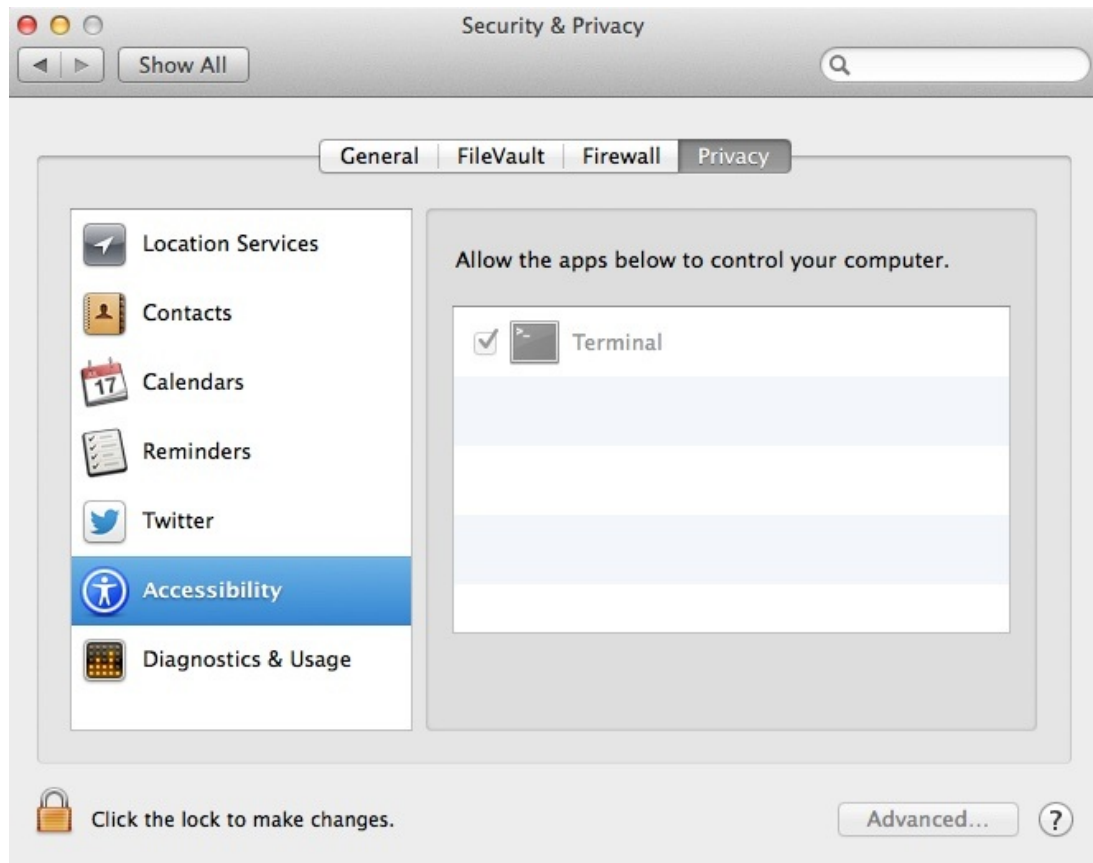
```
end tell
delay 1
end tell
```

In order to enable the above script to work, you need to ensure that under *System Preferences — Accessibility* on your Mac the box *Enable access for assistive devices* is checked (Figure 8.2).



*Figure 8.2 Enabling access for assistive devices on the Mac (OS X 10.8 and below)*

Under OS X 10.9 Mavericks you find this option under *System Preferences — Security&Privacy — Accessibility*. However, other than with an universal checkbox you need to run the application, which is intended to control the computer once first. In our case this means running an AppleScript command in the Mac Terminal application. A window will appear asking you, whether you want Terminal to control your computer. After confirming this message Terminal is added to the Accessibility list and from now on can execute control commands such as our AppleScripts.



*Figure 8.2 Enabling access for assistive devices on the Mac (OS X 10.9 and above)*

While this is not the most elegant solution, it demonstrates, how practically any application can be integrated in our smart home infrastructure. Our complete script `startItunes.sh` reads now:

```
startItunes.sh
#!/bin/sh
#Start iTunes and play playlist provided as an argument
if [[ $# -lt 1 || $# -gt 1 ]];then
    echo "$0: Argument error: itunesStart.sh [playlist]"
    exit 2
fi
exec osascript << EOF
tell application "iTunes" to activate
delay 1
tell application "System Events"
    tell window "iTunes" of process "iTunes"
        click (first UI element whose help is "Choose which speakers to use.")
    delay 1
    tell application "iTunes" to activate
        keystroke "DENON"
```

```

        delay 1
        key code 76
    end tell
    delay 1
end tell
tell application "iTunes"
    play user playlist "$1"
end tell
EOF

```

To complete our shell based iTunes control capabilities we add the scripts *stopItunes.sh*, *iTunesVolUp.sh*, *iTunesVolDown.sh* and *iTunesTo75.sh* to control the volume and close iTunes. (*iTunesTo75.sh* sets the *iTunes* volume to 75% of the maximum):

```

stopItunes.sh
#!/bin/sh
#Stop iTunes
exec osascript << EOF
tell application "iTunes"
    quit
end tell
EOF

iTunesVolUp.sh
#!/bin/sh
#Turn up iTunes volume
exec osascript << EOF
    tell application "iTunes"
        if it is running then
            if sound volume is less than 100 then
                set sound volume to (sound volume + 10)
            end if
        end if
    end tell
EOF

iTunesVolDown.sh
#!/bin/sh
#Turn down iTunes volume
exec osascript << EOF
    tell application "iTunes"
        if it is running then

```

```

        if sound volume is greater than 0 then
            set sound volume to (sound volume - 10)
        end if
    end if
end tell
EOF
iTunesVolTo75.sh
#!/bin/sh
#Set iTunes volume to 75
exec osascript << EOF
    tell application "iTunes"
        if it is running then
            set sound volume to 75
        end if
    end tell
EOF

```

Later, we will add the above scripts as commands to OpenRemote Designer, using the shell execution protocol. Since we will also display the track iTunes is currently playing, we need one more AppleScript based shell script, which determines if iTunes is playing, and, if this is the case, writes the result to the variable `state`:

```
state=`osascript -e 'tell application "iTunes" to player state as string`;
```

If the variable `state` is set to `playing` we write current artist and track to the variables `artist` and `track`:

```
artist=`osascript -e 'tell application "iTunes" to artist of current track as string`;
```

```
track=`osascript -e 'tell application "iTunes" to name of current track as string`;
```

We design the script with the log file for the result to be provided as an argument. So if iTunes is playing, track and artist are written to the file specified as shell argument. Since we plan to use an html file as log, which shall be displayed by the OpenRemote web server, the files location shall again be in the web server root directory:

```
/ORC/webapps/controller
```

With that our script *iTunesPlaying.sh* reads to:

```
#!/bin/sh
#Retrieves currently played title from iTunes and writes to logfile - last entry will be overwritten
if [[ $# -lt 1 || $# -gt 1 ]];then
    echo "$0: Argument error: playing.sh [log file]"
    exit 2
fi

```

```
state=`osascript -e 'tell application "iTunes" to player state as string`;
echo "iTunes is currently $state.";
if [ $state = "playing" ]; then
    artist=`osascript -e 'tell application "iTunes" to artist of current track as string`;
    track=`osascript -e 'tell application "iTunes" to name of current track as string`;
    echo "Current track $artist: $track" > /Users/smarthome/shProject/ORC/webapps/controller/$1;
    else
    echo "iTunes is currently $state." > /Users/smarthome/shProject/ORC/webapps/controller/$1;
fi
```

To test we start an iTunes playlist and type

```
iTunesPlaying.sh playing.html
```

in the terminal window. If we now open the URL

<http://localhost:8080/controller/playing.html>

in a web browser, we should see a display of the playlist. We now can configure all scripts as commands in OpenRemote Designer. (Section 8.3)

## 8.2 Script Based iTunes Control on Windows XP/7/8

*iTunes* for Windows has a *COM API*, which can be used to control the *iTunes* functions. *COM* (Component Object Model) is an *API* under Windows that allows software components to communicate. As the script environment we will again use Windows Powershell. (A detailed description on how to start using PowerShell you can find in section 7.9). We start by creating the *\$iTunes* object using the Cmdlet *New-Object*, which can be used to create either COM or .NET objects:

```
$iTunes = New-Object -ComObject iTunes.Application
```

Next we store the *iTunes* library name in *\$libraryName* and the reference to the selected playlist in *\$selectedPlaylist*:

```
$libraryName = $iTunes.LibraryPlaylist.Name
```

```
$selectedPlaylist = $iTunes.Sources.ItemByName($libraryName).Playlists.ItemByName('RadioClassic')
```

Now we can play the playlist:

```
$selectedPlaylist.PlayFirstTrack()
```

To be more flexible in using our script, we use the parameter variable *\$args[]* to set the playlist as parameter when calling the script. Further we add a check at the start of the script which verifies whether the playlist parameter was provided with the script command. Our script *startItunes.ps1* now reads as:

```
if (($args.count -lt 1) -or ($args.count -gt 1)){
```

```
    echo "Argument error: startItunes.ps1 [Playlist]"
```

```
    exit
```

```
}
```

```
$iTunes = New-Object -ComObject iTunes.Application
```

```
$libraryName = $iTunes.LibraryPlaylist.Name
```

```
$selectedPlaylist = $iTunes.Sources.ItemByName($libraryName).Playlists.ItemByName($args[0])
```

```
$selectedPlaylist.PlayFirstTrack()
```

We can now start the playlist *Relaxation*, for example by typing

```
.\startItunes.ps1 Relaxation
```

The commands to change the volume, to start and to stop *iTunes* are:

```
$iTunes.Mute = $true
```

```
$iTunes.Mute = $false
```

```
$iTunes.Stop()
```

```
$iTunes.Play()
```

Further commands for even more functions would be:

```
$iTunes.Pause()
```

```
$iTunes.PlayPause() (toggle between play and pause)
```

```
$iTunes.CurrentTrack.Name
```

```
$iTunes.NextTrack()
```

```
$iTunes.PreviousTrack()
```

The documentation for the *iTunes COM* interface can be downloaded from <http://developer.apple.com/sdk/itunescomsdk.html>. A good collection of iTunes Powershell Cmdlets can be found on <http://www.thomasmaurer.ch/projects/powershell-itunes/>.

We can now write the Powershell scripts *startItunes.ps1*, *stopItunes.ps1*, *iTunesVolUp.ps1* and *iTunesVolDown.ps1*:

```
startItunes.ps1
```

```
if (($args.count -lt 1) -or ($args.count -gt 1)){  
    echo "Argument error: startItunes.ps1 [Playlist]"  
    exit  
}
```

```
$iTunes = New-Object -ComObject iTunes.Application
```

```
$libraryName = $iTunes.LibraryPlaylist.Name
```

```
$selectedPlaylist = $iTunes.Sources.ItemByName($libraryName).Playlists.ItemByName($args[0])
```

```
$selectedPlaylist.PlayFirstTrack()
```

```
stopItunes.ps1
```

```
$iTunes = New-Object -ComObject iTunes.Application
```

```
$iTunes.Stop()
```

```
iTunesVolUp.ps1
```

```
$iTunes = New-Object -ComObject iTunes.Application
```

```
$iTunes.SoundVolume = $iTunes.SoundVolume + 2
```

```
iTunesVolDown.ps1
```

```
$iTunes = New-Object -ComObject iTunes.Application
```

```
$iTunes.SoundVolume = $iTunes.SoundVolume - 2
```

```
iTunesSetVol.ps1
```

```
$iTunes = New-Object -ComObject iTunes.Application
```

```
$iTunes.SoundVolume = $Volume
```

Since we also want to display the track that iTunes is playing after each control command, we want to read out the active iTunes track and store it to the html log file *playing.html*. This file we will regularly poll through a sensor and display its content. For this purpose we will use the Powershell cmdlet *set-content*:

```
Set-Content -Value $iTunes.CurrentTrack.Name -Path .\ORC\webapps\controller\playing.html
```

Specifying the playlist logfile as a script argument variable *\$args[0]* we can now write our script *iTunesPlaying.ps1*:

```
if (($args.count -lt 1) -or ($args.count -gt 1)){  
    echo "Argument error: iTunesPlaying.ps1 [Playlist logfile]"
```



```

    exit
}
$iTunes = New-Object -ComObject iTunes.Application
$currentTrack = $iTunes.CurrentTrack.Name
$playlistlog = $args[0]
Set-Content -Value $currentTrack -Path .\ORC\webapps\controller\$playlistlog

```

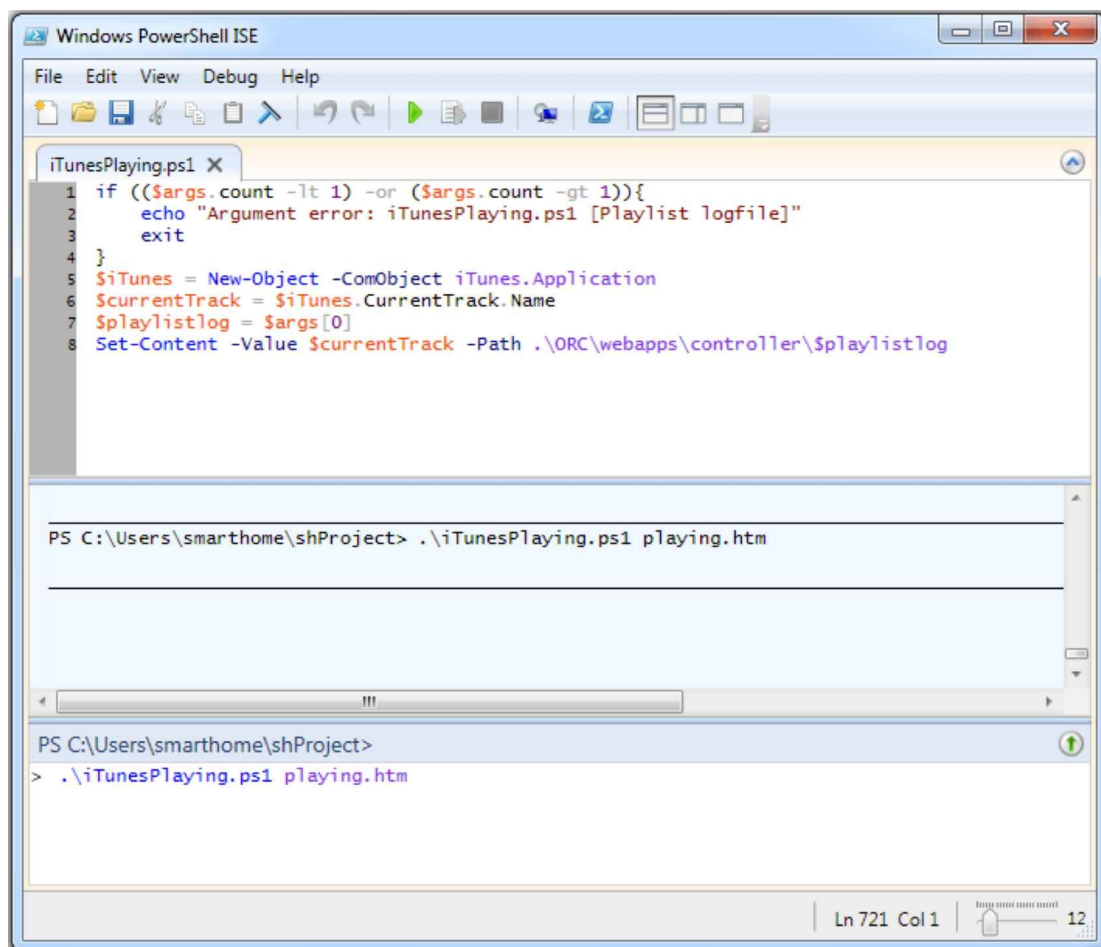
To test we start an *iTunes* playlist and type

```
.\iTunesPlaying.ps1 playing.html
```

in the Powershell window. If we now open the URL

```
http://localhost:8080/controller/playing.html
```

in a web browser, we should see a display of the playlist (Figure 8.3).



*Figure 8.3 Storing the active iTunes playlist via PowerShell to playing.html*

Don't forget that you have to set the Powershell execution policy so that your scripts are executed by typing:

```
Set-ExecutionPolicy -Scope CurrentUser RemoteSigned
```

In addition keep in mind you always need to specify the full path in order to execute your script, such as

```
C:\Users\smarthome\shProject\startItunes.ps1
```

or alternatively, if you reside in the directory where the script is,

```
.\startItunes.ps1
```

In order to start our Powershell scripts from OpenRemote, we will start them as command line without opening Powershell, which is done by typing `powershell.exe` followed by the full path to the script and the script parameters:

```
powershell.exe C:\Users\smarthome\shProject\startItunes.ps1 Relaxation
```

We now can move on to configure our iTunes remote control in OpenRemote Designer.

## 8.3 Creating the iTunes Smartphone Remote

As the last step, we add the control of *iTunes* to our *OpenRemote* based universal smartphone remote. We go to OpenRemote Designer and create the new device iTunes by selecting *New — New Device*. We select the iTunes device and open the command editor window: *New — New Command*. As the name for our command we enter `startItunes RadioClassic`, and for the protocol we select *Shell execution protocol*. On a Mac we enter the shell script name including its complete path and the parameter for our shell script, which is the playlist we want to start, `RadioClassic` (Figure 8.4).

The *OpenRemote* shell execution function also supports multiple parameters, which have to be separated by spaces. This is what we use under Windows, since we need to enter `powershell.exe` in the *Path* field followed by the path to our script and the name of the playlist in the field *Command parameter* separated with a space (Figure 8.5).

Even the parsing of script output using regular expressions for more complex script functions is supported by *OpenRemote* (as of controller version 2.1). In our simple example, however, we can leave the last three lines empty. We save the command and add, in a similar manner `startItunes` with the playlist `Recently Played` and `stopItunes`. We add the remaining commands `volumeupiTunes`, `volumedowniTunes` and a few more playlist initializations such as `startItunes RadioClassic`, `startItunes RadioJazz`, `startItunes RadioPop`, `startItunes RadioRock`, and `startItunes RadioNews`. Of course, you need to make sure the *iTunes* playlists exist and that they contain playlist items.

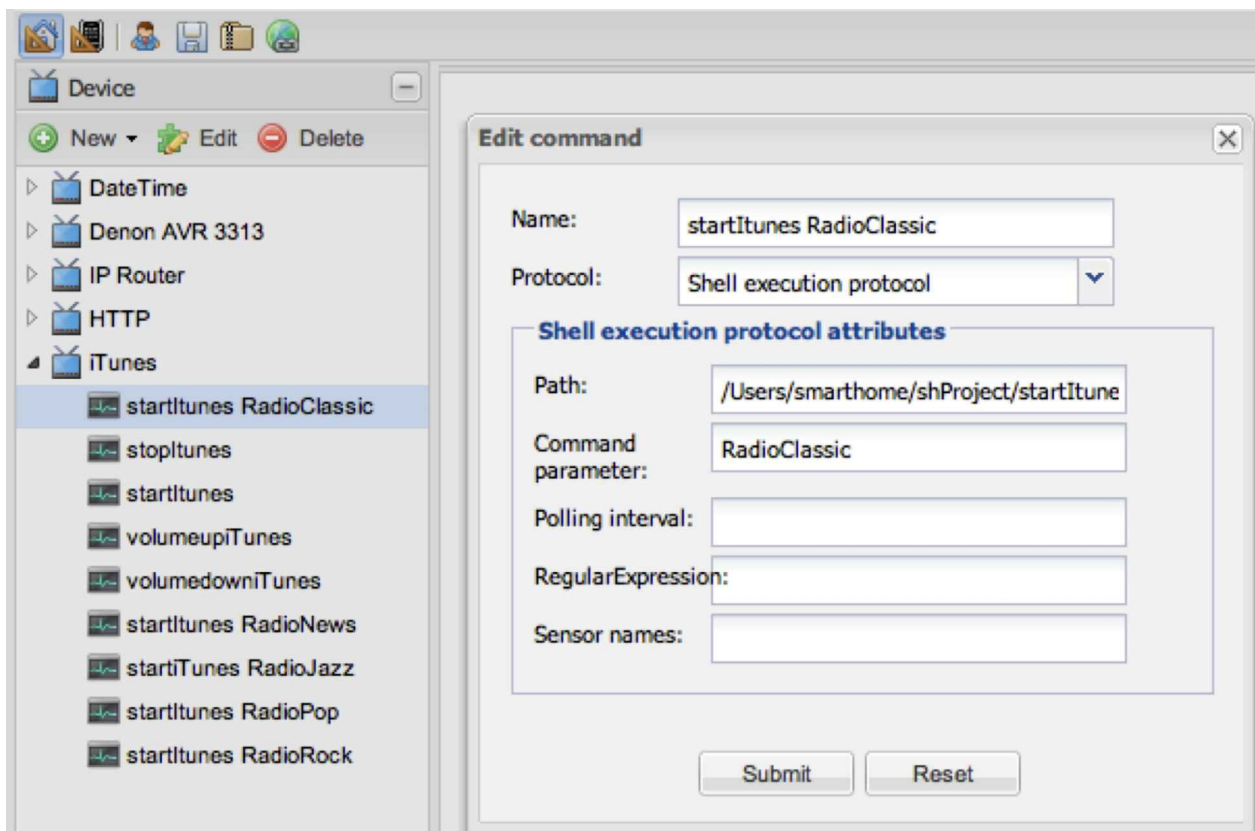


Figure 8.4 Definition of the `startItunes` command in OpenRemote Designer (OS X)

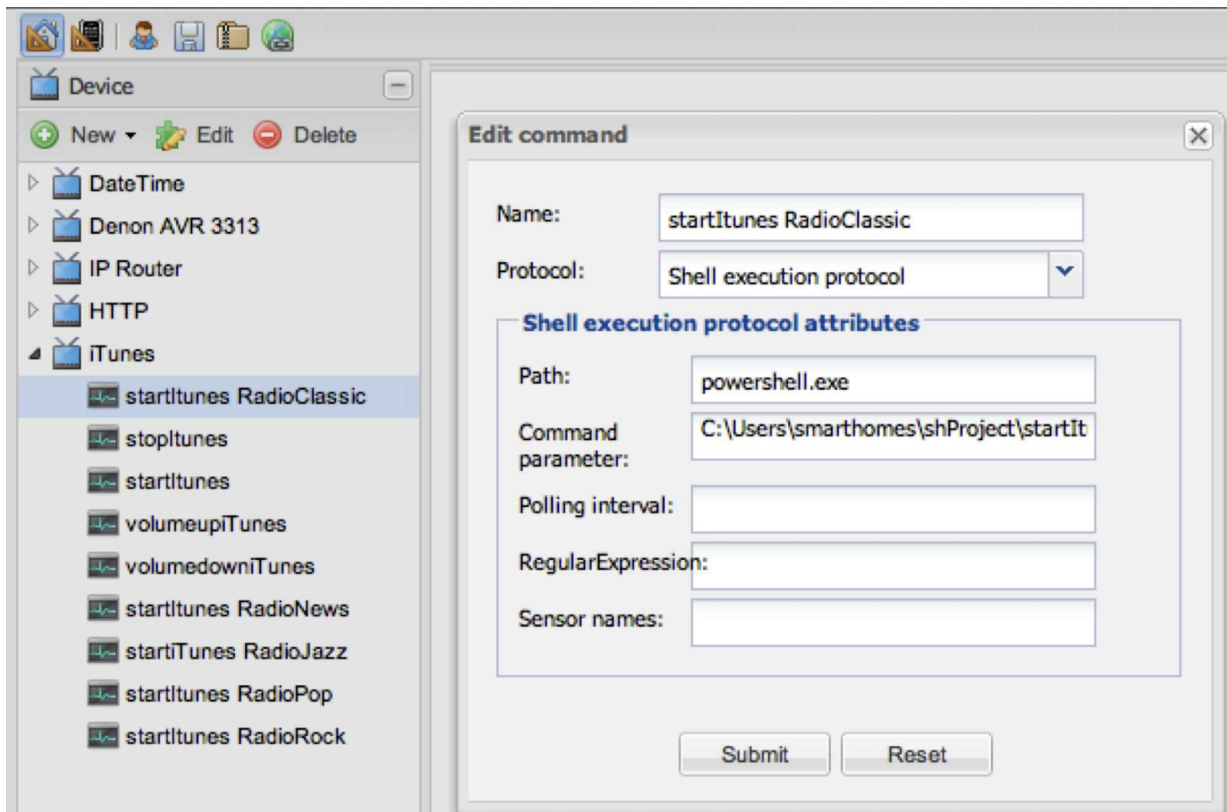


Figure 8.5 Definition of the startiTunes command in OpenRemote Designer (Windows)

Since we not only want to control *iTunes* but also display the active playlist, we need to execute the command that writes the current playlist to our log file *playing.html*. For this purpose we create the shell execution protocol command *iTunesPlaying* in *OpenRemote*. On a Mac we enter `/Users/smarthome/shProject/iTunesPlaying.sh` in the *Path* field, and *playing.html* in the *Command parameter* field.

In Windows we enter `powershell.exe` in the *Path* field, and `C:\Users\smarthome\shProject\iTunesPlaying.ps1` in the *Command parameter* field, followed by *playing.html*, separated by a space.

For the subsequent execution of commands, *OpenRemote* provides an easy to use macro function. To use it, we expand the *Macro* menu and select *New* on the left side of the *OpenRemote* GUI. We name our first macro *Classic* and can now drag the commands, which we want to be executed into the macro window, in our case the commands `startiTunes RadioClassic`, which starts *iTunes* with the playlist *RadioClassic* and `iTunesPlaying`, which stores the current track to our playlist log file. To make sure there are no timing problems between two commands, we insert a delay function in between.

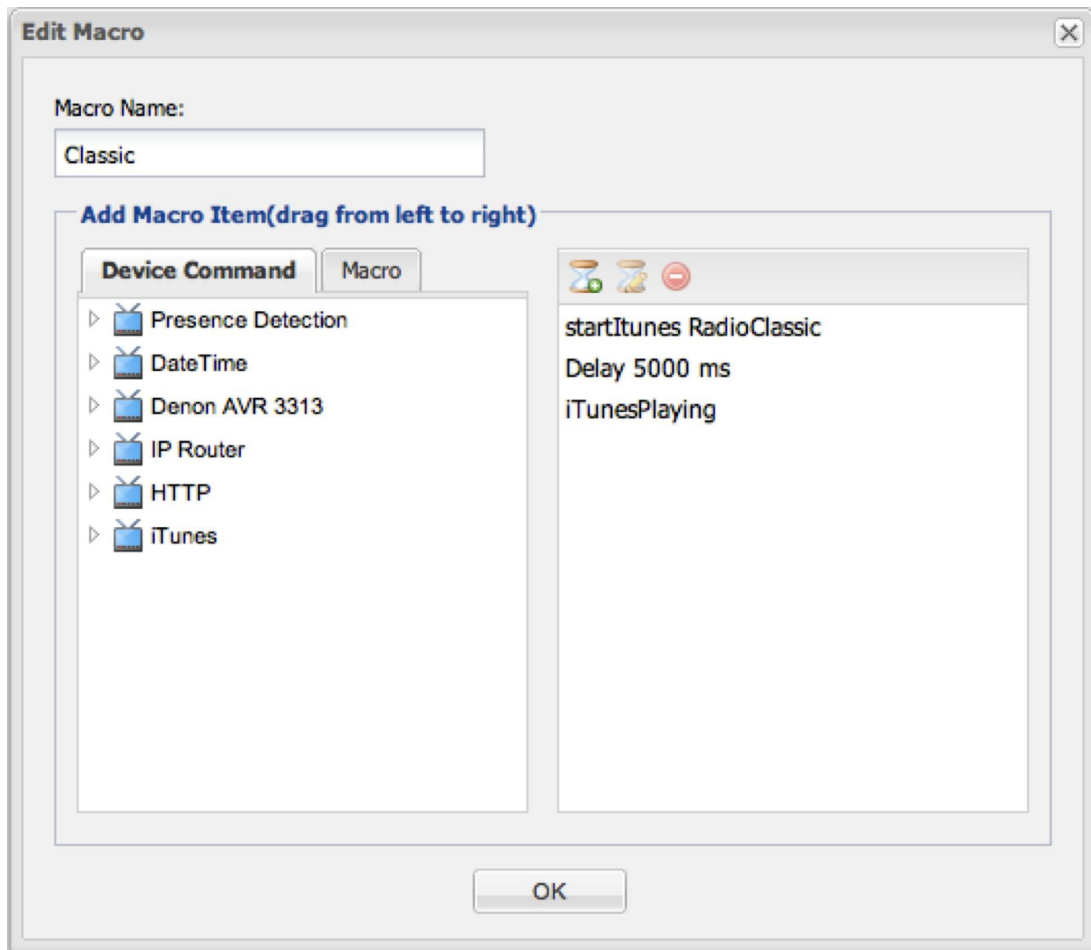


Figure 8.6 Macro definition: Creating sequences of OpenRemote commands

For each of our *iTunes* controls, we now define the macros *Classic*, *News*, *Pop*, *Rock*, *Jazz*, *Start* and *Stop*. Then we define the sensor, which displays the content of our playlist log file *playing.html*.

Finally we need the OpenRemote command for the sensor, which displays the current playlist. We call it *Current Playlist* and configure it with HTTP in the *Protocol* field, <http://localhost:8080/controller/playing.html> in the *URL* field, select GET as *HTTP Method*, and 5 seconds as the *Polling interval* (Figure 8.7).

**Edit command**

Name:

Protocol:

**HTTP attributes**

URL:

HTTP Method:

Workload:

Username:

Password:

XPath Expression:

RegularExpression:

Polling interval:

JSONPath Expression:

*Figure 8.7 Command definition for the playlist Sensor*

We can now create the OpenRemote sensor called *Current Playlist* with *Readout Current Playlist* as its command. We now have all the commands and sensors we need and can move on to the *UI Designer* section. We add a grid with 2 rows, 5 columns and the dimensions 315 x 90 (for an iPhone), and add a *label* element for the name of the control sequence (in our case *iTunes*) and nine *buttons*, which we configure with the macro commands *Start*, *Stop*, *Classic*, *Jazz*, *Rock*, *Pop* and *News*, and with the standard commands *volumeupiTunes* and *volumedowniTunes*.

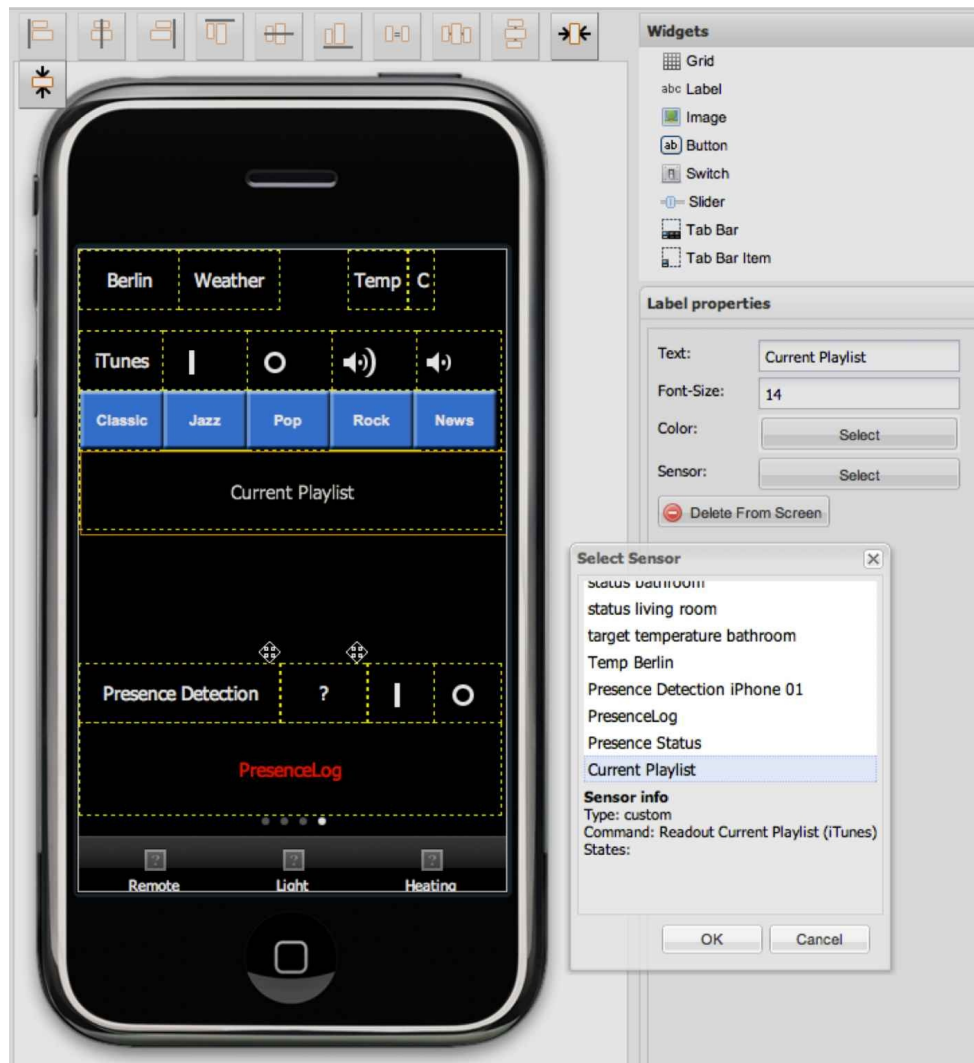
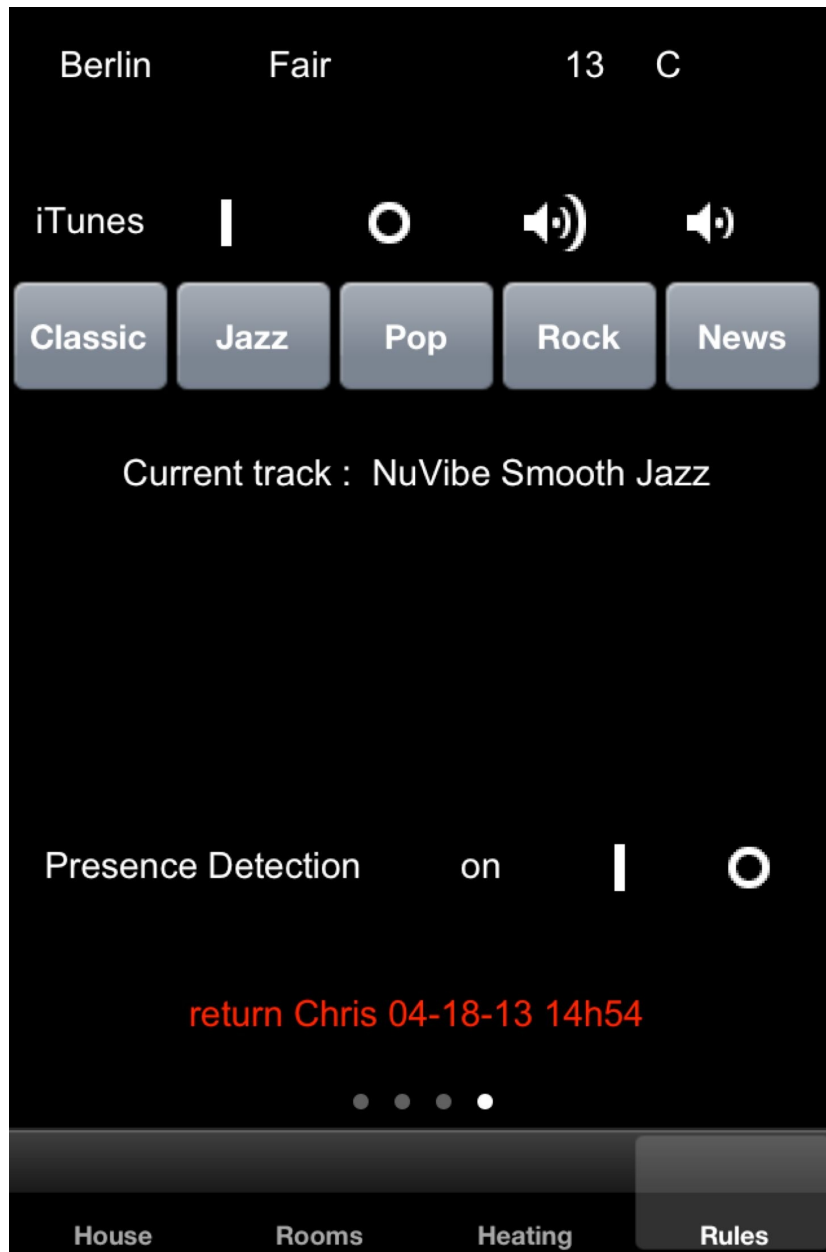


Figure 8.8 Designing iTunes controls in OpenRemote UI Designer

Below the controls section we add another grid, which serves as the container for our playlist sensor. We drag the *Label* symbol into the grid and configure it with our sensor *Current Playlist*.

We save our design, open our Internet browser with the *OpenRemote* controller window and synchronize our smartphone app with our local controller. On our smartphone we restart the smart home app and should now be able to control *iTunes* (Figure 8.9). To test if everything works as desired we operate *iTunes* from our smartphone app while watching *iTunes* react on the screen of our control PC.



*Figure 8.9 The universal smartphone remote app with weather display, presence and iTunes control*



## 8.4 Talk to Me

An important element of a modern smart home is that the communication between home and user is not restricted to computer GUIs, but also takes place at a human level such as speech. Thus, we will briefly cover how to implement speech output for our project.

## 8.4.1 Speech Output Under OS X

Under OS X we can initiate speech output using the command line function `say`. Typing

```
man say
```

gets us a display of the command options:

```
say [-v voice] [-r rate] [-o outfile [audio format options]] | [-n name:port] | [-a device]
```

Per default, OS X comes with a number of English voices such as *Alex* or *Victoria*. However, you can easily expand the available voices to additional flavors and languages. Select *System Preferences — Dictation&Speech — Customize* and you can select between a large variety of language and speaker options.

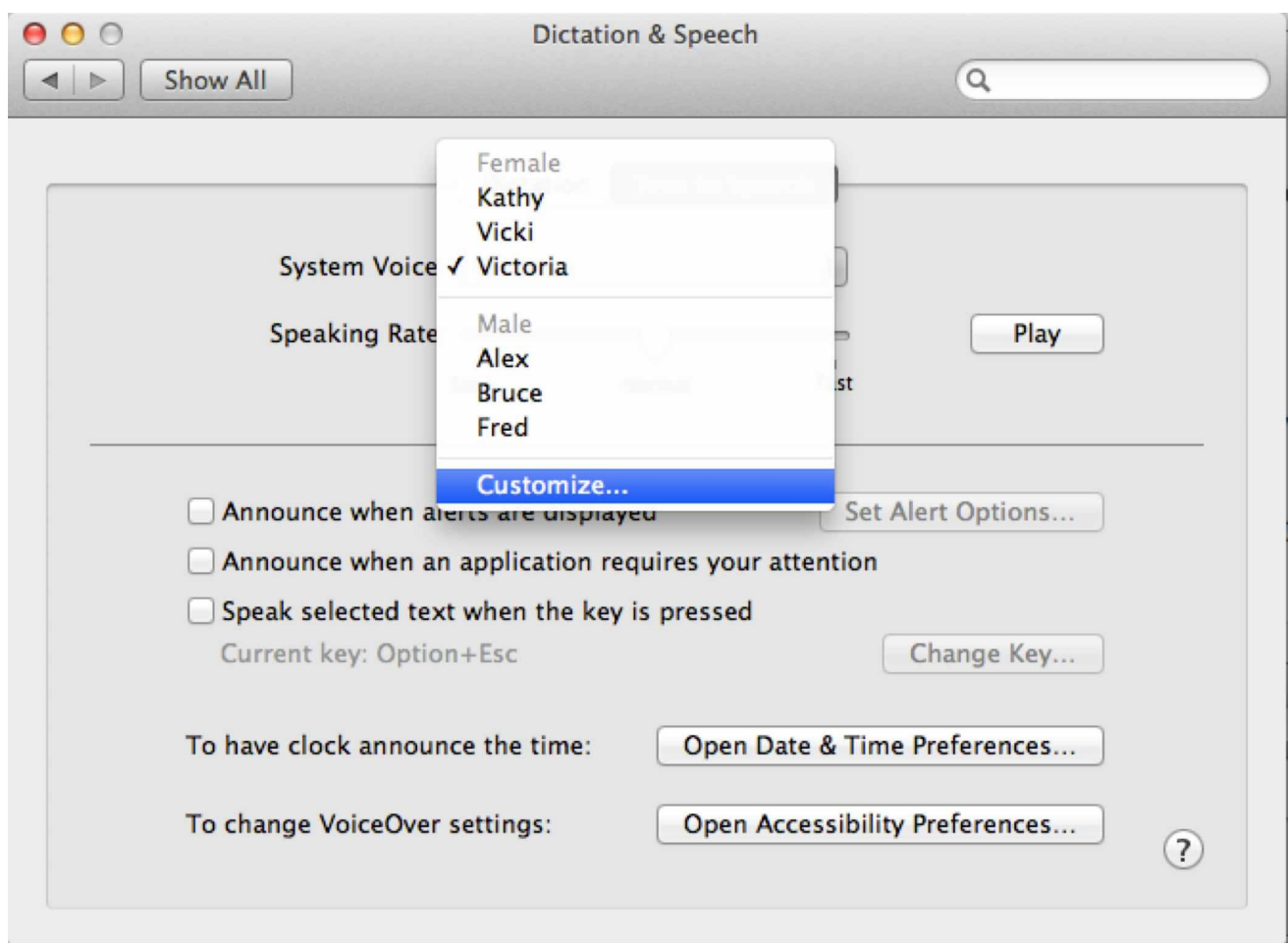


Figure 8.10 Mac OS X Speaker and language options menu

In order to change the audio output for the speech to an external speaker, we could use the AppleScript commands, which change the audio settings in *System Preferences*. This time, however, we will use a “hard coded” command line function, which avoids the delay the AppleScript solution would introduce when executing. There are several functions available as free downloads in the Internet. We choose one called *SwitchAudioSource*. (Internet search `switchaudio-osx`). After download we unzip the file and copy the function *SwitchAudioSource* to our directory *shProject*. Typing the command `SwitchAudioSource` gets us a listing of its functions. The `-a` option displays the available output options:

SwitchAudioSource

Please specify audio device.

Usage: SwitchAudioSource [-a] [-c] [-t type] [-n] -s device\_name

-a : shows all devices

-c : shows current device

-t type : device type (input/output/system). Defaults to output.

-n : cycles the audio device to the next one

-s device\_name : sets the audio device to the given device by name

SwitchAudioSource -a

Built-in Input (input)

AirPlay (output)

Built-in Output (output)

For our project we want to switch from built-in speakers to AirPlay and back. With the help of *SwitchAudioSource*, we can now easily create the two simple shell scripts *outputBuiltIn.sh* and *outputAirPlay.sh*:

*outputBuiltIn.sh*

```
#!/bin/sh
```

```
#Switch audio output to Built-in Output
```

```
SwitchAudioSource -s "Built-in Output"
```

*outputAirPlay.sh*

```
#!/bin/sh
```

```
#Switch audio output to AirPlay
```

```
SwitchAudioSource -s AirPlay
```

As a note to the above: The *SwitchAudioSource* parameter "Built-in Output" in the first script contains a space, which is why you need to put it in quotes. With these two shell scripts, we now create the OpenRemote commands Switch to AirPlay and Switch to Built-in Output. The *say* command also provides an option (-s) to choose the output, but this way we can create a generic speech shell script, with the output options available as separate commands. We now create *macSpeak.sh*, which uses the voice Victoria and the parameter \$1 as the text for speech output:

*macSpeak.sh*

```
#!/bin/sh
```

```
#Speaks the text given as parameter
```

```
if [[ $# -lt 1 || $# -gt 1 ]];then
```

```
    echo "$0: Argument error: macSpeak.sh [text]"
```

```
    exit 2
```

```
fi
```

```
say -v Victoria $1
```

In order to output the current time we create *sayTime.sh*. Here we simply use the `date` command, format its output to hours (%H) and minutes (%M), separated by a colon (this is the format the `say` command recognizes as time), and store it to a shell variable we call `ctime` (for current time). This is the format the `say` command recognizes as a time string.

```
sayTime.sh
#!/bin/sh
#Speaks the current time
ctime=$(date +"%H:%M")
say -v Victoria "It is $ctime"
```

As before we now create the corresponding OpenRemote shell protocol commands *Say the Time* and *Good Morning*. For *Good Morning* we call *macSpeak.sh* with the parameter `Good morning`. Time to get up. To test our commands we create a small *OpenRemote* macro called *demo*. The macro

- switches on our Denon AVR Zone 2
- Sets the volume to 25 dB
- starts iTunes with the Playlist RadioJazz
- sets the Audio Output to AirPlay
- has Victoria wish us a good Morning
- has Victoria tell us the time

Do not forget to insert delays between the commands, which allow the various commands to execute. In particular you need to allow *Victoria* to finish her first sentence, before asking her to tell us the time (Figure 8.11). For the final version of your code, you should verify whether the first `say` command has finished its execution before starting the next one.

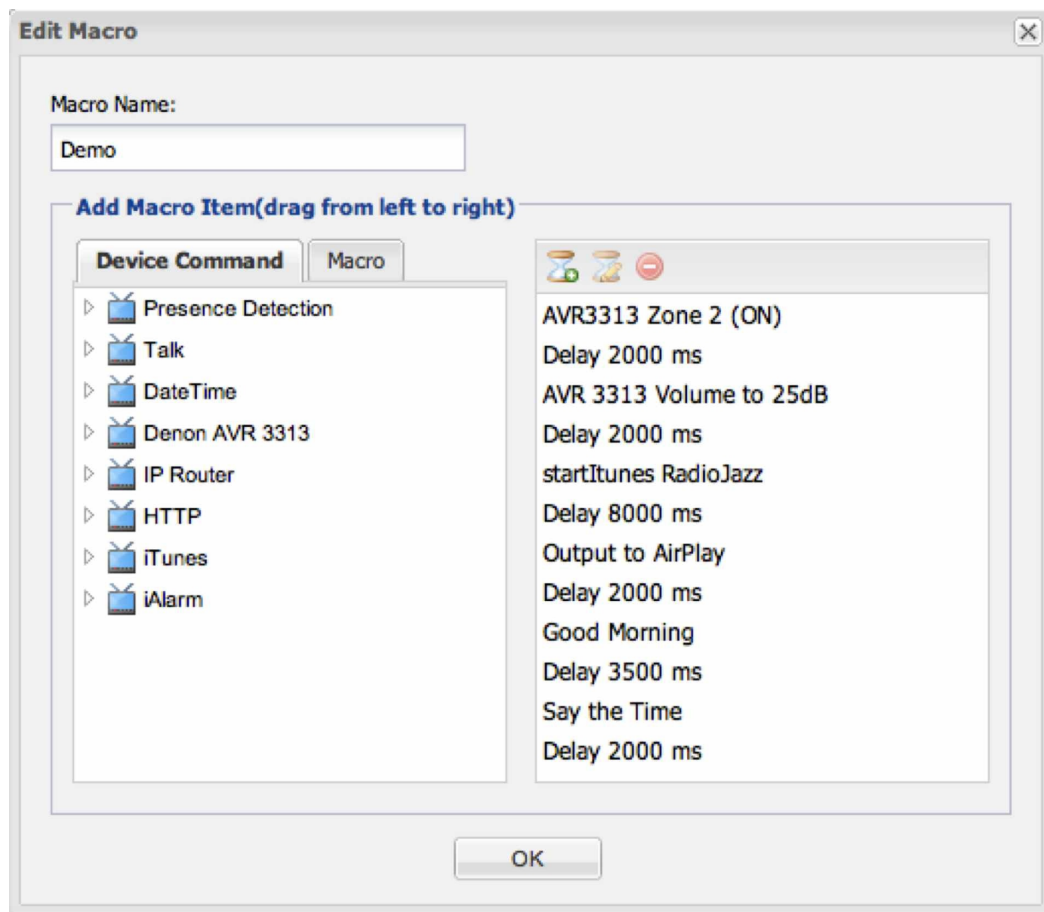


Figure 8.11 OpenRemote demo macro using speech commands

In a similar manner you can create a shell script which outputs a text file to the *say* function. With that you can create voice output for the current weather forecast or the calendar entries of the day. As an example under OS X we can get a print out of the day's *ical* events using the free utility *icalBuddy* from Ali Rantakari. (<http://hasseg.org/icalBuddy/>). With its number of options you can access events and tasks as shown below:

USAGE: icalbuddy [options] <command>

<command> specifies the general action icalBuddy should take:

'eventsToday' Print events occurring today

'eventsToday+NUM' Print events occurring between today and NUM days into the future

'eventsNow' Print events occurring at present time

'eventsFrom:START to:END' Print events occurring between the two specified dates

'uncompletedTasks' Print uncompleted tasks

'undatedUncompletedTasks' Print uncompleted tasks that have no due date

'tasksDueBefore:DATE' Print uncompleted tasks that are due before the given date

'calendars' Print all calendars

'strEncodings' Print all the possible string encodings

'editConfig' Open the configuration file for editing in a GUI editor

'editConfigCLI' Open the configuration file for editing in a CLI editor

For voice output of today's events we just need to route the output of *icalbuddy* to the *say* command:

icalbuddy eventsToday | say

We create a brief shell script *sayCalendar.sh* and an associated *OpenRemote* shell protocol command and can now use it as part of a rule or macro.

sayCalendar.sh

```
#!/bin/sh
```

```
#Voice output of todays events
```

```
icalbuddy eventsToday | say -v Victoria
```

## 8.4.2 Speech Output Under Windows

Under Windows Powershell we can use the object `SAPI.SPVoice` to turn text into speech output. As a first test we simply type the following commands in the Powershell window:

```
$Voice = new-object -com SAPI.SpVoice
$Voice.Speak( "Hello World!", 1 )
```

If you change the parameter 1 to 5, you can have Powershell read out a text file. Create a text file and have Powershell read it to you:

```
$Voice = new-object -com SAPI.SpVoice
$Voice.Speak( "C:\Text\smartHome.txt", 5 )
```

First we want to create a generic script, that reads out the text provided with the script as parameter. As we have done before, we start with the validation that exactly one parameter was provided when calling the script. Then we define the object `SAPI.SPVoice` and call `$Voice.Speak` with the content of `$args[0]`:

```
pcSpeak.ps1
if (($args.count -lt 1) -or ($args.count -gt 1)){
    echo "Argument error: psSpeak.ps1 [text for speech output]"
    exit
}
$Voice = new-object -com SAPI.SpVoice
$Voice.Speak( $args[0], 1 )
```

We call the script `pcSpeak.ps1` and can call it now with any text:

```
./pcSpeak.ps1 "Hello my name is John"
```

As a second script we want to output the current time. Since we just want hours and minutes, we format the `Get-Date` Cmdlet using the `%H` and `%M` options:

```
$a = Get-Date -format %H %M
```

With that we can finish our script `sayTime.ps1` as follows:

```
sayTime.ps1
$cTime = Get-Date -format "%H o %M"
$Voice = new-object -com SAPI.SpVoice
$Voice.Speak( "It is", 1 )
$Voice.Speak( "$cTime", 1 )
```

In order to start our Powershell scripts from OpenRemote we will again start them as command line without opening Powershell, which is done by typing `powershell.exe` followed by the full path to the script and the script parameters:

```
powershell.exe C:\Users\smarthome\shProject\sayTime.ps1
```

In OpenRemote Designer we can now create the command `Say Time`, selecting *Shell Execution Protocol* as the communication protocol. In the *Path* field, we enter `powershell.exe` and in the *Command parameter* field, we enter `C:\Users\smarthome\shProject\sayTime.ps1`.

As a first text-to-voice command we create the command *Good Morning*. In the *Path* field we enter powershell.exe. In the *Command parameter* field we enter

C:\Users\smarthome\shProject\pcSpeak.ps1 followed by “Good Morning, time to get up”, separated by a space. To test our commands we create a small OpenRemote macro which starts iTunes followed by the speech output commands *Good Morning* and *sayTime*.

Per default Windows 7 comes with only a single voice (*Microsoft Anna*, US English).

However, you can download additional voices for a variety of languages for free from several sites. (e.g. <http://www.zero2000.com/free-text-to-speech-natural-voices.html>). To configure the text-to-voice settings under Windows 7 you need to go to *Control Panel — Ease of Access — Speech Recognition — Advanced Speech Options*. In Windows 8 Microsoft Anna has been replaced by David (US male), Hazel (UK female) and Zira (US female).





## 9 A Little AI: Drools Rules

In this chapter we will explain how to set up rules for our *OpenRemote* controller, which is the always-on home automation component of *OpenRemote*. So far we control the sensors and applications, we have developed in the previous chapters, via our smartphone application. We now will control them with a rule engine, and with that will be able to implement powerful home automation scenarios such “iAlarm” and “Coming Home”. A few years ago, rule based expert systems were hyped up and referred to as artificial intelligence. While this is far from reality, it is still surprising, how well a small set of well through rules can perform in defined environments. For our smart home project the rules database is the brain, where the building automation intelligence resides, and where defined actions based on detected events are taken. The rules infrastructure, which OpenRemote uses, is called *Drools*, an open source object oriented rule engine written in Java. Besides in Java, rules however can also be specified in MVEL, Python or Groovy.

## Caution: Are you running the correct Java version?

Since the Drools rule engine version 5.1.1, which is part of OpenRemote, is not compatible with Java versions higher than 6.x, be sure you have Java 1.6 (under OS X) or the latest JDK 6.x version (JDK 6.45) (under Windows) installed. If not your rules will NOT work! A detailed description on installing the correct Java environment for your OpenRemote installation can be found in Chapter 5.

A rule engine is basically nothing but an if/then statement interpreter. Each if/then statement is called a rule. The commercial, productized version of Drools is called JBoss Rules by the company RedHat. Details can be found under <http://www.jboss.org/drools/>.

The Drools based rules can be very complex, the basic syntax is relatively simple. The rules are stored in files with the extension .drl. In OpenRemote per default rules from the OpenRemote Designer rule editor are stored in the subdirectory `webapps/controller/rules/modeler_rules.drl` of the OpenRemote installation.

The basic structure of a rule definition is:

```
rule "name"  
  attributes  
  when  
    LHS  
  then  
    RHS  
end
```

LHS stands for left hand side (the if part of an if/then rule), RHS stands for right hand side (the then part of an if/then rule). There are a number of keywords, which are reserved as Drools commands, and which must not be used as names or identifiers. Some of them are:

true, false, null, eval, when, then, end, not, or, end

A key role in Drools play rule attributes. They are optional and can be used to control the behavior of the rule. Examples for attributes are `timer` or `no-loop`. LHS (Left Hand Side) is the conditional part of the rule, RHS (Right Hand Side) contains the commands to be executed in case LHS becomes true. When LHS is left empty, the rule is assumed to be always true. All other rule elements, including a rule name, are mandatory for the rule to work. Single-line comments are marked using double backslashes `//`, multi-line comments are marked using `/*` and `*/`.

While the LHS part (the conditional part) of a rule always has to be written in the Drools syntax, the RHS (the consequences of the rule) is specified using Java. (In addition to Java the RHS can also be specified using the Java scripting language MVEL if the option dialect "mvel" is set. However, since MVEL also supports Java syntax and we will specify our rule consequences using Java we do not have to worry about the difference).

Lets take a look at a first simple yet important example, which every thirty-seconds reads out the value of the OpenRemote sensor `CurrentTemperatureBathroom` and sends it to the terminal

along with the current date and time:

```
rule "TemperatureReport"  
timer (cron:0/30 * * * * ?)  
when  
$temp : Event( source == "CurrentTemperatureBathroom", $tBath : value );  
then  
Date dateRp = new Date();  
System.out.println($tBath + " „+dateRp);  
end
```

In the first line we specify the name "TemperatureReport" of the rule. The timer expression in the next line tells Drools to evaluate this rule every 30 seconds. Next the `when` statement marks the beginning of the rule's LHS, which comprises of the line

```
$temp : Event( source == "CurrentTemperatureBathroom", $tBath : value );
```

What is happening here is that we declare a new local variable of type `Event` called `$temp`. Inside the brackets we have the rule condition, which searches for an entity with the name „CurrentTemperatureBathroom" and which assigns its value to the variable `$tBath`. If the entity `CurrentTemperatureBathroom` is found, the RHS (the `then`) part of the rule is being executed. We declare the local variable `dateRp` of type `Date` and assign it the current date. Then we print the content of `dateRp` and of `dateRp` to the terminal window. The full specification for the Drools language can be obtained from

<http://docs.jboss.org/drools/release/5.2.0.Final/drools-expert-docs/html/ch05.html>

## 9.1 Wake me up Early if it Rains: iAlarm

As our first rule we will be creating our intelligent alarm *Wake me up early if it rains*, for which we have designed the weather sensor in chapter 6. Our rule shall execute the following actions:

- at 6 a.m. retrieve the current weather from our OpenRemote sensor “Weather Condition Berlin”
- at 6 a.m. retrieve the value (on/off) of *iAlarmfunction.html*
- determine if the weather sensor value contains the text strings `rain` OR `snow`
- in case the strings `rain` or `snow` are found, and *iAlarmfunction.html* contains `on`, start iTunes with playlist `RadioPop`
- set *iAlarmfunction.html* to `off`
- update the playlist logfile *playing.html*

A second simple rule will let the alarm go off at 6h45 a.m., in case *iAlarmfunction.html* is set to `on`.

## 9.2 Controlling iAlarm via Smartphone

Before we move on to design the rule for our intelligent alarm, we want to set up the capability to switch our iAlarm function on and off from our smartphone with the help of the html file *iAlarmfunction.html* in a similar fashion as we have done it for our presence detection application. OS X and Linux users create the two shell scripts *turniAlarmOn.sh* and *turniAlarmOff.sh*, Windows users create the two Powershell scripts *turniAlarmOn.ps1* and *turniAlarmOff.ps1*:

```
turniAlarmOn.sh
```

```
#!/bin/sh
```

```
echo "on" > /Users/smarthome/shProject/ORC/webapps/controller/iAlarmfunction.html
```

```
turniAlarmOff.sh
```

```
#!/bin/sh
```

```
echo "off" > /Users/smarthome/shProject/ORC/webapps/controller/iAlarmfunction.html
```

```
turniAlarmOn.ps1
```

```
echo "on" > C:\Users\smarthome\shProject\ORC\webapps\controller\iAlarmfunction.html
```

```
turniAlarmOff.ps1
```

```
echo "off" > C:\Users\smarthome\shProject\ORC\webapps\controller\iAlarmfunction.html
```

We test the scripts and validate if the file *iAlarmfunction.html* is created and updated with on respectively off. Under OS X do not forget, that you have to enable the execution rights of the new files:

```
chmod +x ./shProject/ORC/bin/turniAlarmOn.sh
```

```
chmod +x ./shProject/ORC/bin/turniAlarmOff.sh
```

```
./shProject/ORC/bin/turniAlarmOn.sh
```

In OpenRemote Designer we now create a new device called *iAlarm* and define the three commands *Turn iAlarm On*, *Turn iAlarm Off* and *iAlarm Status*. For the commands *Turn iAlarm On* and *Turn iAlarm Off* we select the *Shell Execution Protocol*.

The OpenRemote Shell Execution Protocol supports multiple parameters, which simply have to be separated by spaces. This is what we use under Windows now, since the command lines we need to configure read:

```
powershell.exe C:\Users\smarthome\shProject\turniAlarmOff.ps1
```

```
powershell.exe C:\Users\smarthome\shProject\turniAlarmOn.ps1
```

We enter *powershell.exe* in the *Path* field and the actual path to our script as the first parameter in the field *Command parameter* (Figure 9.1).

Under OS-X we simply enter

```
/Users/smarthome/shProject/turniAlarmOn.sh
```

and

```
/Users/smarthome/shProject/turniAlarmOff.sh
```

in the *Path* field of the command definition window, and can leave the *Command*

*parameter* field empty.

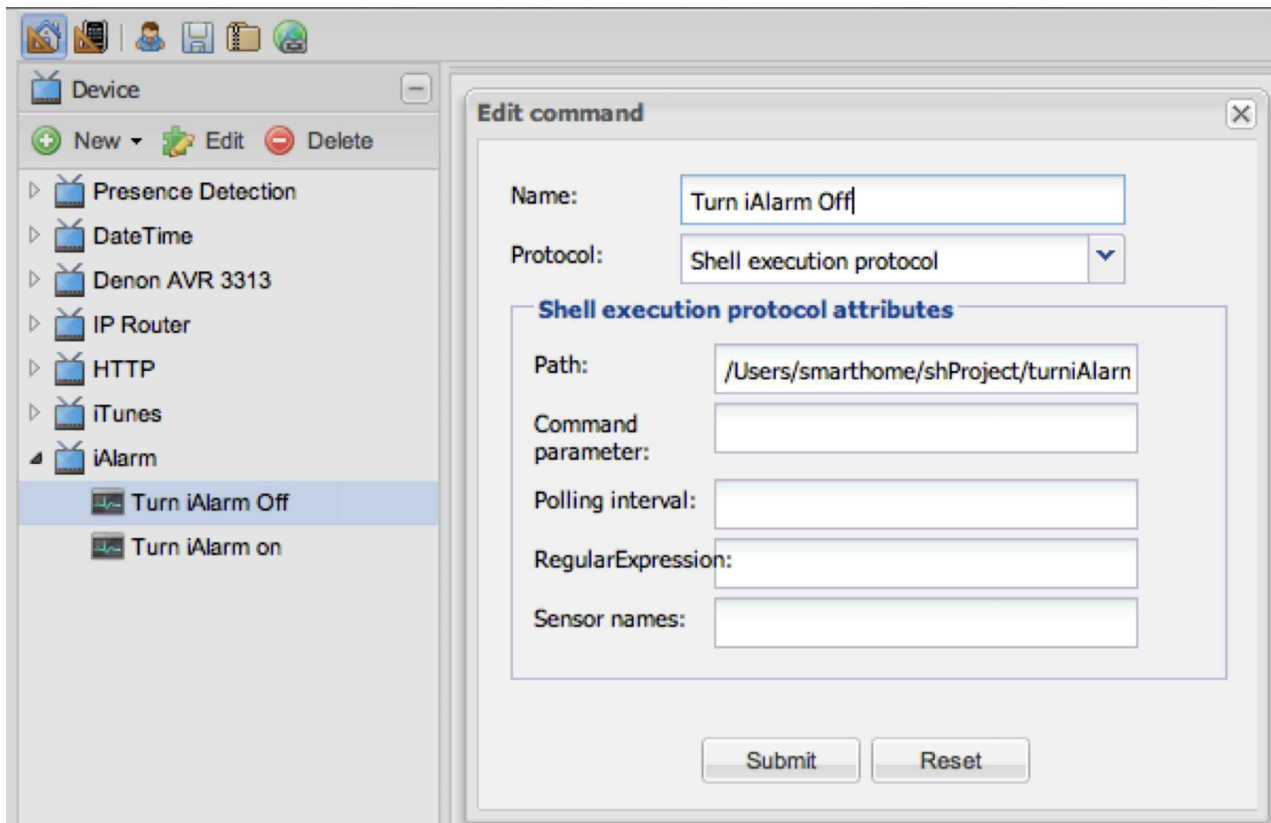


Figure 9.1 OpenRemote Command Definition Turn iAlarm Off for OS X

For the command *iAlarm Status* we select the HTTP protocol, provide the URL to the local OpenRemote web server for the file *iAlarmfunction.html* in the *URL* field <http://localhost:8080/controller/iAlarmfunction.html> and select GET for the field *HTTP Method* (Figure 9.2).

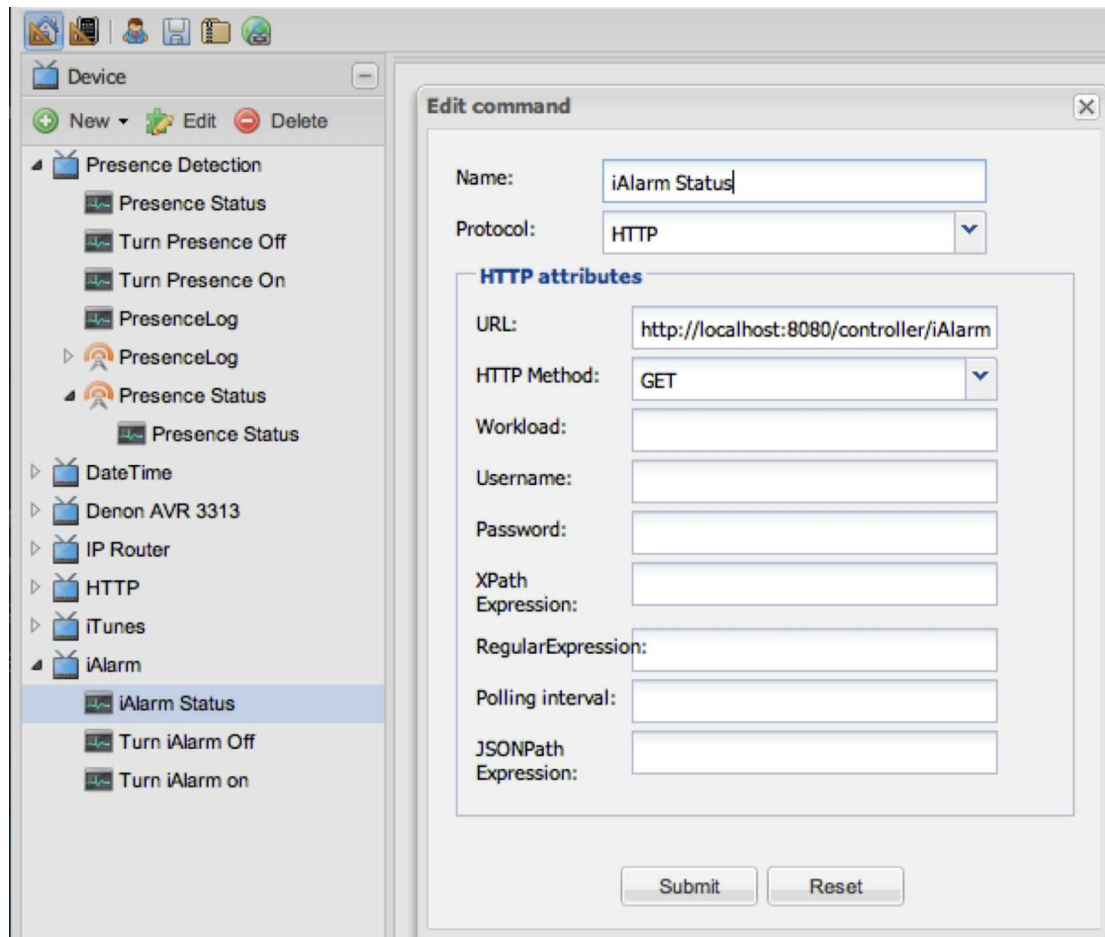


Figure 9.2 OpenRemote Command Definition for *iAlarm Status* using HTTP GET

Now we can create the sensor *iAlarm Status* using the command *iAlarm Status* we just defined (Figure 9.3).



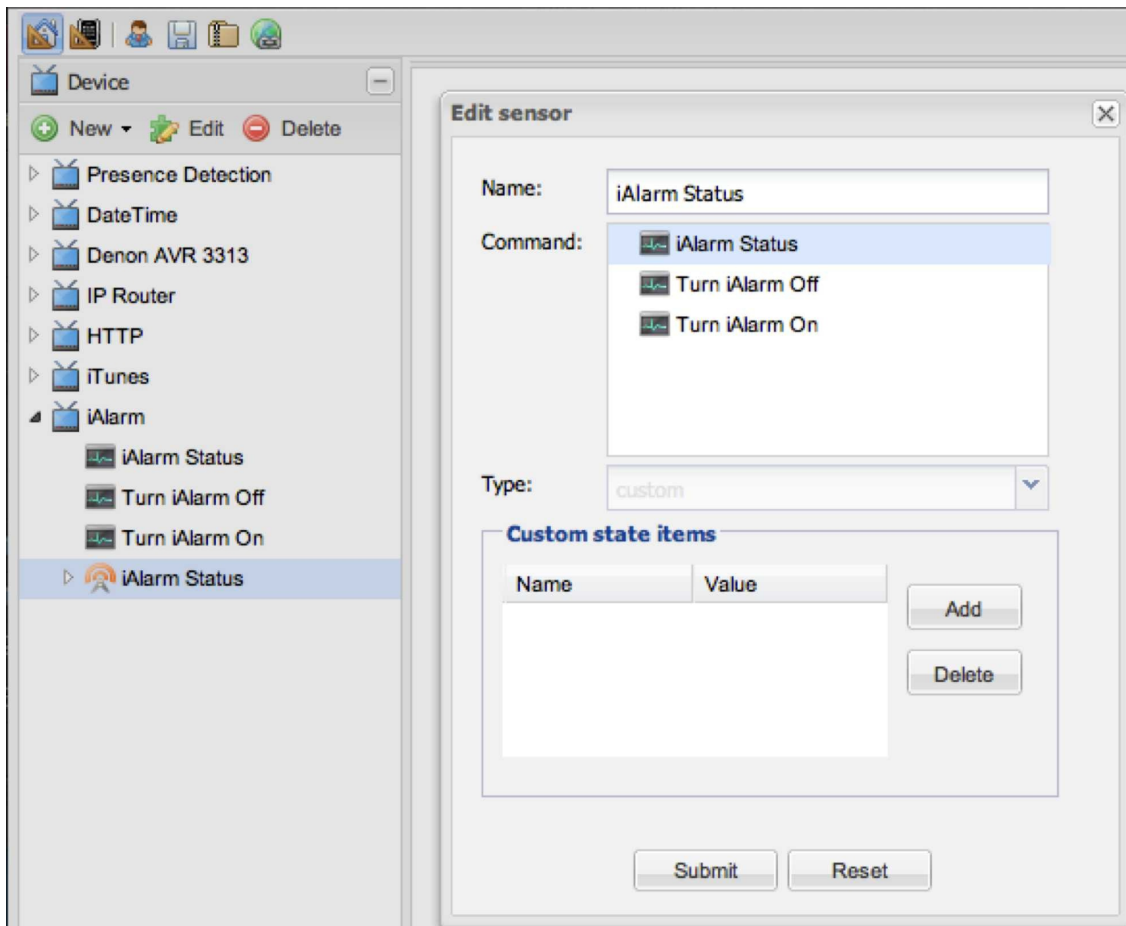


Figure 9.3 OpenRemote Sensor Definition for *iAlarm Status*

As the last step we add the GUI controls in the OpenRemote UI Designer window. We add the four grid elements, name field (*iAlarm*), status display sensor (*iAlarm Status*), *iAlarm On* and *iAlarm Off*. In the grid element for the name of our control function we drag an *abc label* element and name it *iAlarm*. For the *iAlarm* status display we do the same, but select as sensor *iAlarm Status*, which we have defined before. And finally for the switch controls we use the button element and configure it with the two commands *Turn iAlarmOn* and *Turn iAlarmOff* (Figure 9.4). The symbols for the switch commands you can design yourself or you can use some of the ones provided by OpenRemote. You can also download the designs I have developed for the purpose of this project (*poweron.png*, *poweronpress.png*, *poweroff.png*, *poweroffpress.png*) from the book website <http://www.howtosmarthome.com>. After synchronizing our local controller with the updated design, we can toggle our *iAlarm* rule on and off, and get an updated display of its status.

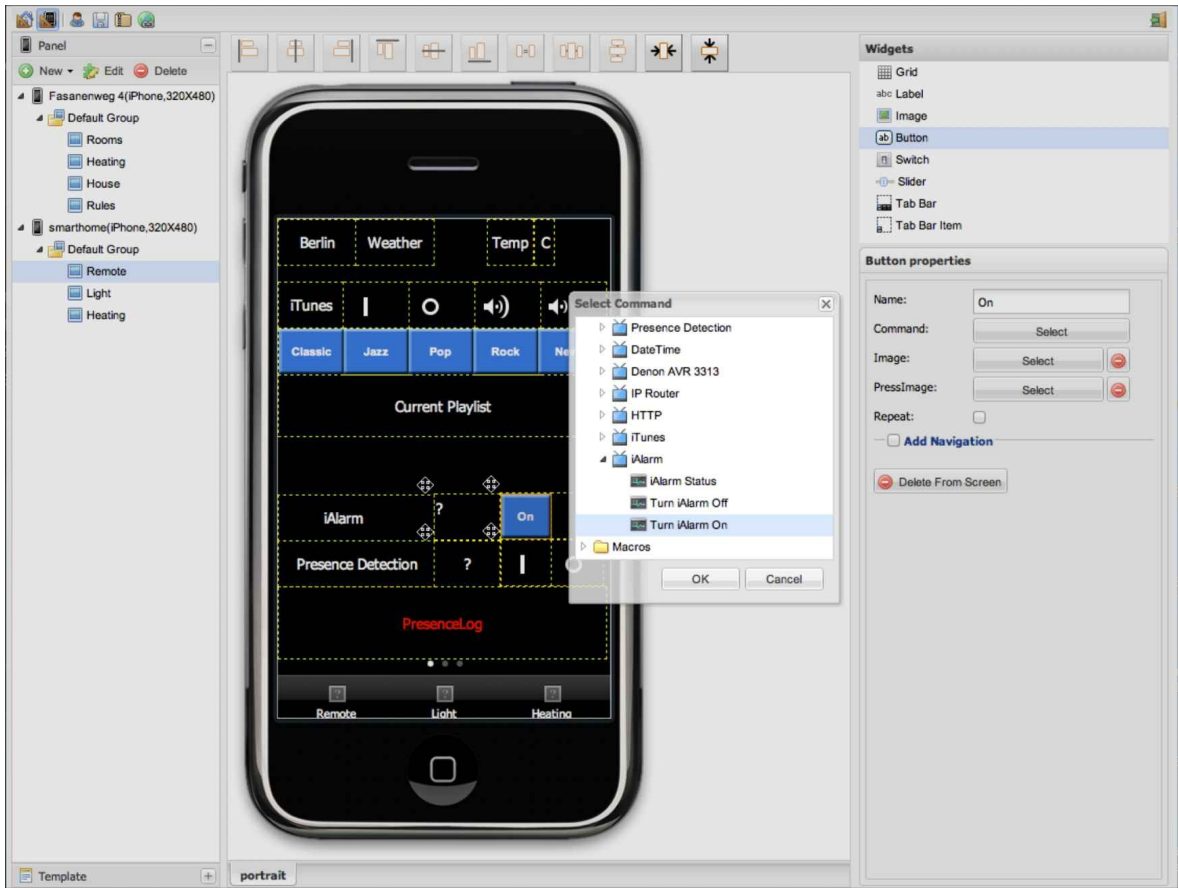


Figure 9.4 OpenRemote GUI design for iAlarm control

## 9.3 The iAlarm Rule Script

We can now begin the design of our rules script. We start with the name statement followed by a timer, which shall kick off the rule at six a.m. Attention! In Drools the timer uses the format of UNIX cron expressions, which consist of six mandatory fields and the optional year field, each separated by a white space:

```
timer (cron: <seconds> <minutes> <hours> <day-of-month> <month> <day-of-week> [year])
```

Valid values are

<i>seconds</i>	<i>0-59 or * for any value</i>
<i>minutes</i>	<i>0-59 or * for any value</i>
<i>hours</i>	<i>0-23 or *</i>
<i>day-of-month</i>	<i>1-31, L,?,*</i>
<i>month</i>	<i>1-12 or JAN-DEC</i>
<i>day-of-week</i>	<i>1-7 or SUN-SAT</i>
<i>Year</i>	<i>1970-2199</i>

The value L stands for “last day of month”, \* stands for “any value” and ? stands for “no value”. If any value other than ? is specified in day-of-month, a ? must be chosen for the ‘day-of-week’ field. Similarly, when a “day-of-week” value is specified, the “day-of-month” field must contain ?. Day and month ranges can be specified using the hyphen character (-), minute increments with the backslash character (/).

As an example the time expression for a rule being evaluated every 15 minutes would be

```
timer (cron:0 0/15 * * * ?)
```

A rule with the timer expression

```
timer (cron:0/5 * * * * ?)
```

would be evaluated every 5 seconds.

We want our rule to execute every Monday through Friday at 6 a.m. which gets us to the timer expression

```
timer (cron:0 0 6 ? * MON-FRI)
```

Alternative to cron based timers Drools also supports interval based timers with the following syntax:

```
timer ( <initial delay> <repeat interval> )
```

As an example the following timer fires 30 seconds after start of Drools and then every ten minutes:

```
timer ( int: 30s 600s )
```

With the CustomState command we can execute OpenRemote sensors. With the command name : value we write the value of the sensor to the variable name. In the then part of our rule we simply print out the content of our variable name followed by current date and time:

```
(1) //Rule reading out weather sensor at 6a.m. and starting iTunes in case of rain or snow//
```

```
(2) rule “Wake me up early if it rains”
```

```
(3) timer (cron:0 35 17 ? * MON-FRI)
```

```
(4) when
```

(5) CustomState(source == “Weather Condition Berlin”, name : value)

(6) then

(7) System.out.println(“name”);

(8) Date date = new Date();

(9) System.out.println(date.toString());

(10) end

(1) Single-line Comment

(2) The name of our rule

(3) Timer which stops the script here until the time condition is met

(4) Begin of the rule if condition: when

(5) Read out OpenRemote sensor “Weather Condition Berlin”, storage of its content into variable name

(6) Begin of the rule then condition: then

(7) Print out the content of variable name

(8) Write current date and time to variable date

(9) Print out the content of variable date

(10) End

We now want to test what we have so far. In OpenRemote Designer on the left hand side of the screen we expand *Config for Controller* and select the menu entry *rules*. We are now in the OpenRemote rules editor. In addition to the above script we need to insert the import commands for several OpenRemote scripts and Java packages. We just need to do this once at the beginning of our rules definition file. So insert the following definitions at the very beginning of the file, followed by our first script in the rules editor window (Figure 9.5):

```
//Package, globals and imports:
```

```
package org.openremote.controller.protocol
```

```
global org.openremote.controller.statuscache.CommandFacade execute;
```

```
global org.openremote.controller.statuscache.SwitchFacade switches;
```

```
global org.openremote.controller.statuscache.LevelFacade levels;
```

```
import org.openremote.controller.protocol.*;
```

```
import org.openremote.controller.model.event.*;
```

```
import java.lang.Float;
```

```
import java.sql.Timestamp;
```

```
import java.util.Date;
```

```
rule “Wake me up early if it rains”
```

```
timer (cron:0 35 17 ? * MON-FRI)
```

```
when
```

```
CustomState(source == “Weather Condition Berlin”, name : value)
```

```
then
```

```
System.out.println(“name”);
```

```
Date date = new Date();
```

```
System.out.println(date.toString());
```

```
end
```

After inserting the rule definition and the import packages click on the *submit* button twice and you will receive the confirmation Property saved successfully. Next you go to the *UI Designer* window and save your designer project by clicking on the disc symbol. You get the message UI Designer Layout saved at .... Then synchronize your local rules definition file with your Online Designer project by clicking *Sync with Online Designer* in the OpenRemote Controller window. Now your local rules definition file

```
.../ORC/webapps/controller/rules/modeler_rules.drl
```

is updated. Open it with a text editor to validate that your rule definition has loaded. (When looking for the source of a problem related to rules definitions it is always a good idea to check the content of your local *modeler\_rules.drl* file, since this is, what actually gets executed. Due to an incomplete synchronization process it can happen, that your local file is actually different that your latest rule definition in OpenRemote Designer. At the very bottom of the OpenRemote GUI you can also monitor the saving progress.

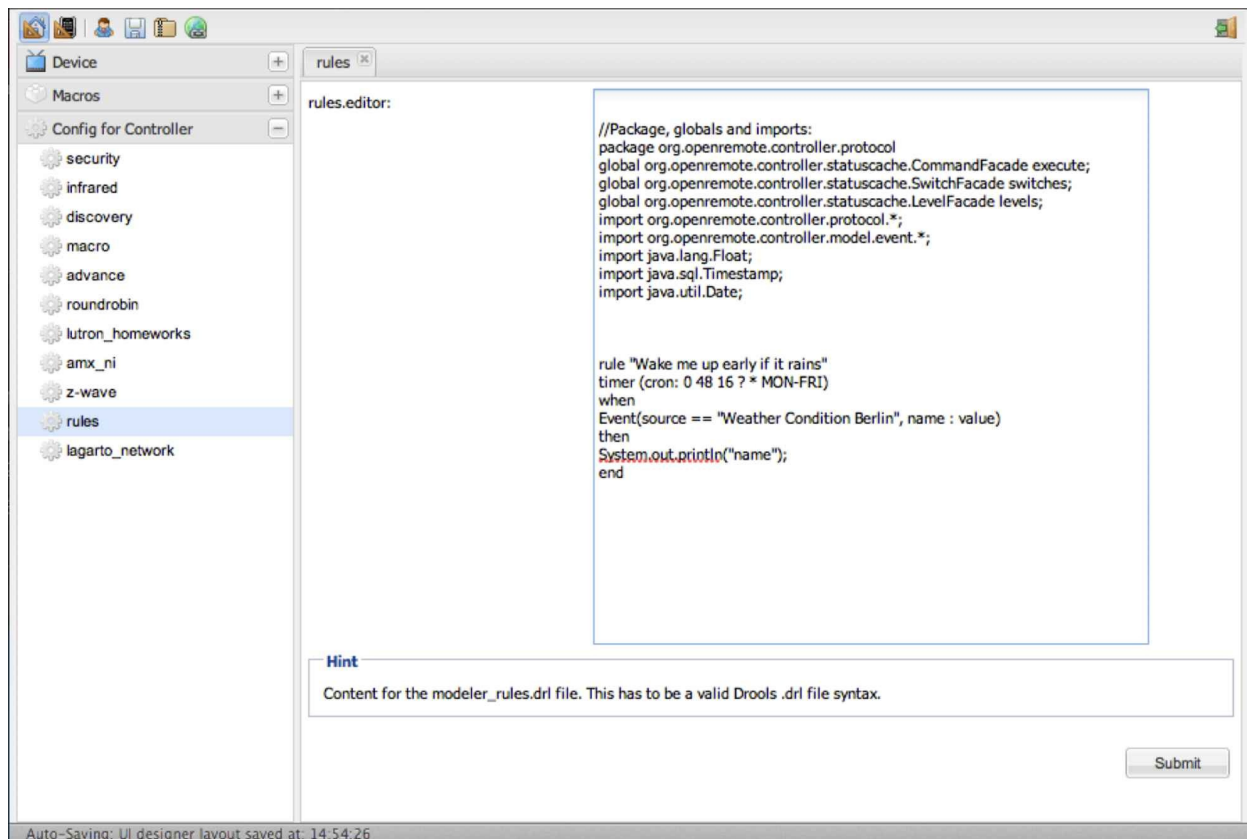


Figure 9.5 OpenRemote rules definition editor

When the OpenRemote controller now loads, observe closely the control messages in the terminal window. At the very beginning you will now see a line similar to

```
INFO 2013-05-09 14:54:13,328 : Initialized event processor : Drools Rule Engine
```

This tells us, that the Drools definition file has been parsed correctly without any error. In case the parser finds an error, this is where you will find the according error messages. In order to test our rule now, we need to change the time in the timer definition of our rule to a time two or three minutes ahead of our current time. Then we synchronize our rules

definition again and observe the terminal window of our controller. After startup our rule should print the current weather condition from our weather sensor followed by time and date in the controller window, once the time specified in our timer definition has passed. We should see something like:

```
INFO 2013-05-09 17:35:54,915 : Startup complete.
```

```
Partly Cloudy
```

```
Thu May 09 17:37:00 CEST 2013
```

We now know our rule is working and can proceed. First, on the conditional part of our rule, we need to validate if *iAlarm* is switched on. This is done by determining if the value of our custom sensor *iAlarm Status* is on:

```
CustomState(source == "iAlarm Status", value == "on")
```

In general, depending on what type of sensor you have defined (custom, range, level, switch), you can reference your sensors with the four commands

```
custom state
```

```
range
```

```
level
```

```
switch
```

Valid values for switch sensors are `on` and `off`, for level and range sensors any integer, and for custom sensors any arbitrary string. The event value is the value the sensor must report for the rule to be triggered. Examples are

```
Range ( source == "Livingroom", value == "25" )
```

```
Level ( source == "brightness", value == "15" )
```

```
Switch ( source == "OutdoorLight", value == "on" )
```

```
value != minValue, value != maxValue
```

When the value of a sensor changes, an event is triggered and sent to the rule engine. You can also store the value of the sensor to a variable, and process it later in the rule. In the below example the value of the level sensor `brightness` is stored to the variable `lamp01`:

```
Level ( source == "brightness", lamp01 : value )
```

Since for combining several rule elements the most common condition is a logical `and`, it is implicit on the LHS side of a Drools rules definition. This means we simply need to insert the above line below our first `Event` command, and both events are logically connected through an `and` condition. For our *iAlarm* rule we further need to conduct a search substring operation for the occurrence of `rain` or `snow` in the output of our weather sensor. For this purpose we use the Java command `matches` for regular expression based string searches. We define the string variable `testStr`, which we assign the content of our variable `name` (it holds the value of our weather sensor) and the string variables `lookUp1` and `lookUp2`, which we assign our search terms `rain` and `snow`. The regular expression search for our two substrings shall be case insensitive, which is why our regex definitions have to start with `(?i)`:

```
(?i).*rain.*
```

```
(?i).*snow.*
```

The asterisk (\*) stands for any number, the dot (.) for any character except new line. With that our Java commands for the substring search of our test string `testStr`, using the two substrings in variable `lookUp1` and `lookUp2`, read:

```
testStr.matches("(?i).*" + lookUp1 + ".*")
```

```
testStr.matches("(?i).*" + lookUp2 + ".*")
```

We combine both substring searches in an `if` command, connecting them with a logical `or`, which in Java is the double pipe `||`:

```
rule "Wake me up early if it rains"
```

```
timer (cron:0 41 18 ? * MON-FRI)
```

```
when
```

```
CustomState(source == "Weather Condition Berlin", name : value)
```

```
CustomState(source == "iAlarm Status", value == "on")
```

```
then
```

```
String testStr = (String) name;
```

```
String lookUp1 = "rain";
```

```
String lookUp2 = "snow";
```

```
if ((testStr.matches("(?i).*" + lookUp1 + ".*")) || (testStr.matches("(?i).*" + lookUp2 + ".*")))
```

```
{
```

```
System.out.println("iAlarm going off "+name);
```

```
Date date = new Date();
```

```
System.out.println(date.toString());
```

```
}
```

```
end
```

We can now test our rule definition again. In order to get an alarm, we again need to adjust the time in our timer definition to a few minutes ahead of us, and we need to replace one substring search (e.g. `snow`) by a keyword, which currently is being displayed by our weather sensor, e.g. `cloudy`. (And in case you are working on a Saturday or Sunday, adjust the day-of-week statement to `MON-SUN`). If the above is working we are almost done. In addition to the print statement, which we can leave in as logging information, we just need to add the commands to start iTunes (`startItunes RadioPop`), to update the playlist log file `playing.html` (`iTunesPlaying`) and to switch our alarm function off (`Turn iAlarm Off`). The command execution uses the format

```
execute.command("OpenRemote command name");
```

Since `OpenRemote` macros are not supported to be called from the rules environment, we cannot just call our macro `Pop`, which we defined to contain the two commands `startItunes RadioPop` and `iTunesPlaying` with a five-second delay timer in between. (The timer was needed to give iTunes the chance to load the playlist, before updating the playlist log file). Instead we need to call the two commands within our rule and insert a Java program delay by using the `Thread.sleep` command, which causes our Java thread to suspend execution for a

specified period. When using `Thread.sleep` we also need to deal with the so called `InterruptedException`, which is why the Java code we need to insert for the delay is exactly as follows:

```
try {
    Thread.sleep(5000);
} catch(InterruptedException ex) {
    Thread.currentThread().interrupt();
}
```

We can now finish our first rule script, which reads in its final version to:

```
rule "Wake me up early if it rains"
timer (cron:0 0 6 ? * MON-FRI)
when
    CustomState(source == "Weather Condition Berlin", name : value)
    CustomState(source == "iAlarm Status", value == "on")
then
    String testStr = (String) name;
    String lookUp1 = "rain";
    String lookUp2 = "snow";
    if ((testStr.matches("(?i).*" + lookUp1 + ".*")) || (testStr.matches("(?i).*" + lookUp2 + ".*")))
    {
        System.out.println("iAlarm going off "+name);
        Date date = new Date();
        System.out.println(date.toString());
        execute.command("startItunes RadioPop");
        execute.command("Turn iAlarm Off");
        try {
            Thread.sleep(5000);
        } catch(InterruptedException ex) {
            Thread.currentThread().interrupt();
        }
        execute.command("iTunesPlaying");
    }
end
```

What is left is the second simple rule, which should start iTunes at 6h45a.m., in case the alarm function is still on (which means, *iAlarm* did not go off at 6.a.m.):

```
rule "Wake me at 6h45"
timer (cron:0 45 6 ? * MON-FRI)
when
```



```
CustomState(source == "iAlarm Status", value == "on")
```

```
then
```

```
System.out.println("iAlarm going off");
```

```
Date date = new Date();
```

```
System.out.println(date.toString());
```

```
execute.command("startItunes RadioPop");
```

```
execute.command("Turn iAlarm Off");
```

```
try {
```

```
    Thread.sleep(5000);
```

```
} catch(InterruptedException ex) {
```

```
    Thread.currentThread().interrupt();
```

```
}
```

```
execute.command("iTunesPlaying");
```

```
end
```

Once you have understood the above rules and got them working, you will be able to easily expand them, or build others using a similar structure. Many rules will also be simpler than the ones above, and just execute one or two commands based on a simple on/off condition of a sensor.

## 9.4 Coming Home

The base version for our second scenario “Coming Home” combines the smartphone based presence detection and the iTunes control functionality to welcome a person returning home with their individual playlist playing. Also this scenario we will later expand by adding functions such as turning on outdoor and corridor lights, room lights or heating. The exact functionality will of course depend on which components are made controllable via the smart home infrastructure, and which functions are desired to act automated. Our “Coming Home” rule in its basic version shall execute the following actions:

- determine if the sensor *PresenceEvent* contains the value `return_Chris`
- determine if the sensor *Presence Status* is set to `no`
- in case both of the above conditions are met, start iTunes with playlist *RadioClassic*
- update the playlist logfile *playing.html*

With what we have learned so far, we can now easily write our *Coming Home* rule:

```
rule “Coming Home Chris”
when
  CustomState(source == “PresenceEvent”, value == “return_Chris”)
  CustomState(source == “Presence Status”, value == “on”)
then
  Date date = new Date();
  System.out.println(date.toString());
  System.out.println(“Chris coming home....”);
  execute.command(“startItunes RadioClassic”);
  try {
    Thread.sleep(5000);
  } catch(InterruptedException ex) {
    Thread.currentThread().interrupt();
  }
  execute.command(“iTunesPlaying”);
end
```

On the LHS side we just have the two sensor condition commands:

```
CustomState(source == “PresenceEvent”, value == “return_Chris”)
CustomState(source == “Presence Status”, value == “on”)
```

On the RHS side we write the current date to the variable `date` and print it to the terminal window, followed by the string “Chris coming home....” . This just serves as a log entry. Then we execute the `OpenRemote` command `startItunes` with the parameter `RadioClassic`, after which we insert the Java `Thread.sleep(5000)` command sequence. As we remember, these lines halt

the Java program execution for five-seconds, which gives iTunes the time to load and start the desired playlist. After the wait we execute the command `iTunesPlaying`, which updates the html file *playing.html*. This gets us an updated display of the currently active iTunes playlist on our smartphone app.



# 10 More iDevices

In this chapter we will now enhance our existing automation scenarios by adding additional components to our smart home control infrastructure. Many new generation home appliances contain integrated web or Telnet server, which allow them to be controlled via HTTP or Telnet commands. Others, which are either simpler or older (or both), such as coffee machines or lamps, need to be connected to smart power-outlets. For the later and for controlling the building infrastructure, we will demonstrate how to use and incorporate Z-Wave components in our project.

## 10.1 Denon / Marantz Audio System Control

We will start by showing how to integrate a Denon AV receiver into our smart home control system using Telnet commands. The two leading Japanese audio equipment manufacturers Denon and Marantz belong to the same holding company, which is why they use the identical control protocol. So the basic set of commands should work on all recent models from both manufacturers. From the Denon website (see bibliography) one can download the full specification of the DENON AVR control protocol, which allows for full control of all functions. Before we start setting up the configuration of our new device in OpenRemote, we open a terminal window and validate if we can reach our target device using a Telnet connection. We type `telnet` and at the Telnet prompt `open` followed by the IP address of our device:

```
telnet> open 192.168.178.16 23
Trying 192.168.178.16...
Connected to denonavr3313.fritz.box.
Escape character is '^['.
```

For security reasons the built in Telnet servers of many consumer electronic devices close the connection after every single command they receive. This does not hurt for the one-way remote control scheme, which we are implementing, but is bothering when trying to test more complex things using an online terminal session. On the DENON AVR we are using we also have to deal with this behavior, which is why after each command, which we send to the device via Telnet, we need to close the terminal session and open a new one for the next command.

Also make sure to configure your target device (in our case the DENON AVR 3313) with a fixed IP address rather than leaving it with the default setting of DHCP. If the latter is the case, the moment your router assigns a new IP address to your target device, the settings we will configure in OpenRemote, which contain a static IP address, will not work anymore. So we assign a fixed IP address to our device and test if control via Telnet actually works by sending a command such as Z2ON:

```
telnet> open 192.168.178.16 23
Trying 192.168.178.16...
Connected to denonavr3313.fritz.box.
Escape character is '^['.
Z2ON
```

The DENON AVR should activate zone two and on the front LED you should be able to see the according status display.

Now since Telnet control is working, we can get started with the OpenRemote configuration. We go to OpenRemote designer and create a new device, which we call *Denon AVR*. We want to create four commands: *switch on*, *switch off*, *volume up* and *volume down*. In our OpenRemote Designer Building Modeler we select *Denon AVR — New — New Command*, and enter the command names, Telnet as the protocol, the default Telnet port

number 23, the IP address of our Denon receiver, and, as a first example, the command to switch Zone 2 on, which we look up in the Denon protocol manual to be Z2on. (The Denon 3313, which we use in our example, is capable of sending its audio output to multiple zones. A zone is typically a separate room, with loudspeakers connected to the according zone output. By switching from zone to zone one can control to which room the audio-output is sent). If the target control device provides a Telnet service without a prompt, as is the case with the Denon Telnet daemon, the OpenRemote implementation requires to prepend each command with `null|` (null pipeline) without leaving a space to the actual command. To find out if your target device runs Telnet with or without a prompt, just open a Telnet session to the device using a terminal window and look at the output. With that the command for switching on Zone 2 of our Denon AVR, which we enter in the OpenRemote command window reads:

```
null|Z2ON
```

Figure 10.2 shows the finished command.

The image shows a screenshot of the 'Edit command' dialog box in the OpenRemote application. The dialog has a title bar with 'Edit command' and a close button. It contains several input fields and a dropdown menu. The 'Name' field is filled with 'AVR3313 Zone 2 (ON)'. The 'Protocol' dropdown is set to 'Telnet'. Below these is a section titled 'Telnet attributes' which contains several more input fields: 'IP Address' (192.168.178.16), 'Port' (23), 'Command' (null|Z2ON), 'Read Timeout (s)', 'Read Regex Filter', 'Read Regex Group', 'Default Read Response', and 'Polling interval'. At the bottom of the dialog are two buttons: 'Submit' and 'Reset'.

*Figure 10.1 OpenRemote Telnet command for switching Denon AV3313 Zone2 On*

In the same way we configure the Denon commands

Z2OFF Switch Zone2 off

Z2UP Zone 2 Volume Up

Z2DOWN Zone 2 Volume Down

Now we just need to set up the button controls for our four commands in the OpenRemote UI Designer menu and we are done. As a final step we want to add push buttons for four presets, which we have programmed with our favorite Internet radio stations. The according commands for turning on presets 1 through 4 are:

```
null|NSB01  
null|NSB02  
null|NSB03  
null|NSB04
```

This gives us another four commands to associate with four button controls. With that we are done with our Denon remote control (Figure 10.2).

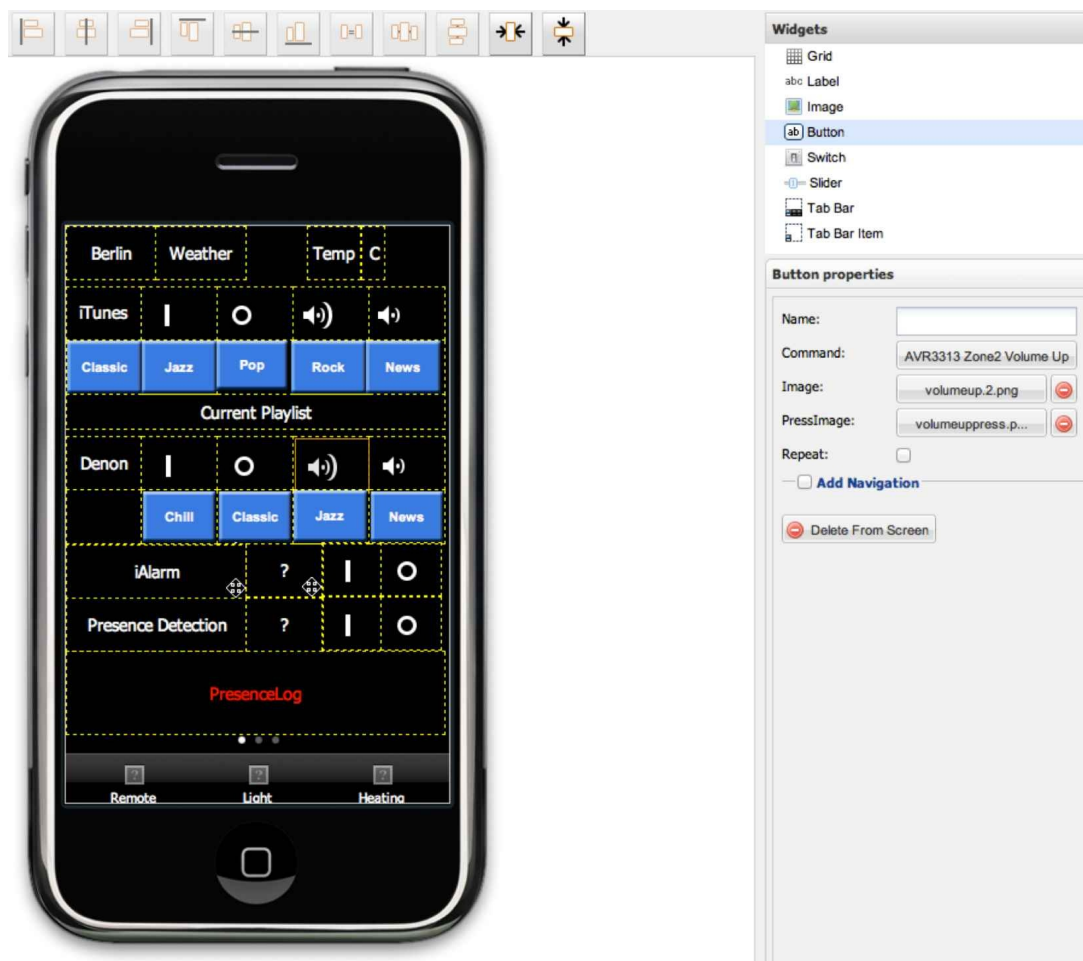


Figure 10.2 The OpenRemote screen layout for Denon AV3313 control

Of course we now can easily correlate our Denon commands inside macros with other commands to create single push button scenarios such as

- dinner (switch some lights off, dim other lights, switch on the Denon audio system and select a radio station) or
- leave home (switch music and lights off, change heating operating state to standby, switch on outside lights for three minutes)

Equally we can combine these commands with our iTunes control commands, which now lets us choose in which zone we want to listen to our iTunes playlist. As an example we create a Denon command AVR 3313 Volume to 25 dB, which sets the volume for zone 2 to 25



dB. The according Denon protocol command would be

null|Z225

In our macro definitions for the iTunes controls (*Classic, News, Pop, Rock, Jazz*) we now add the OpenRemote commands we have defined AVR3313 Zone 2 (ON) and AVR 3313 Volume to 25 dB, separated by a 2 second delay, to give the first Telnet command the chance to execute before the second is sent out.

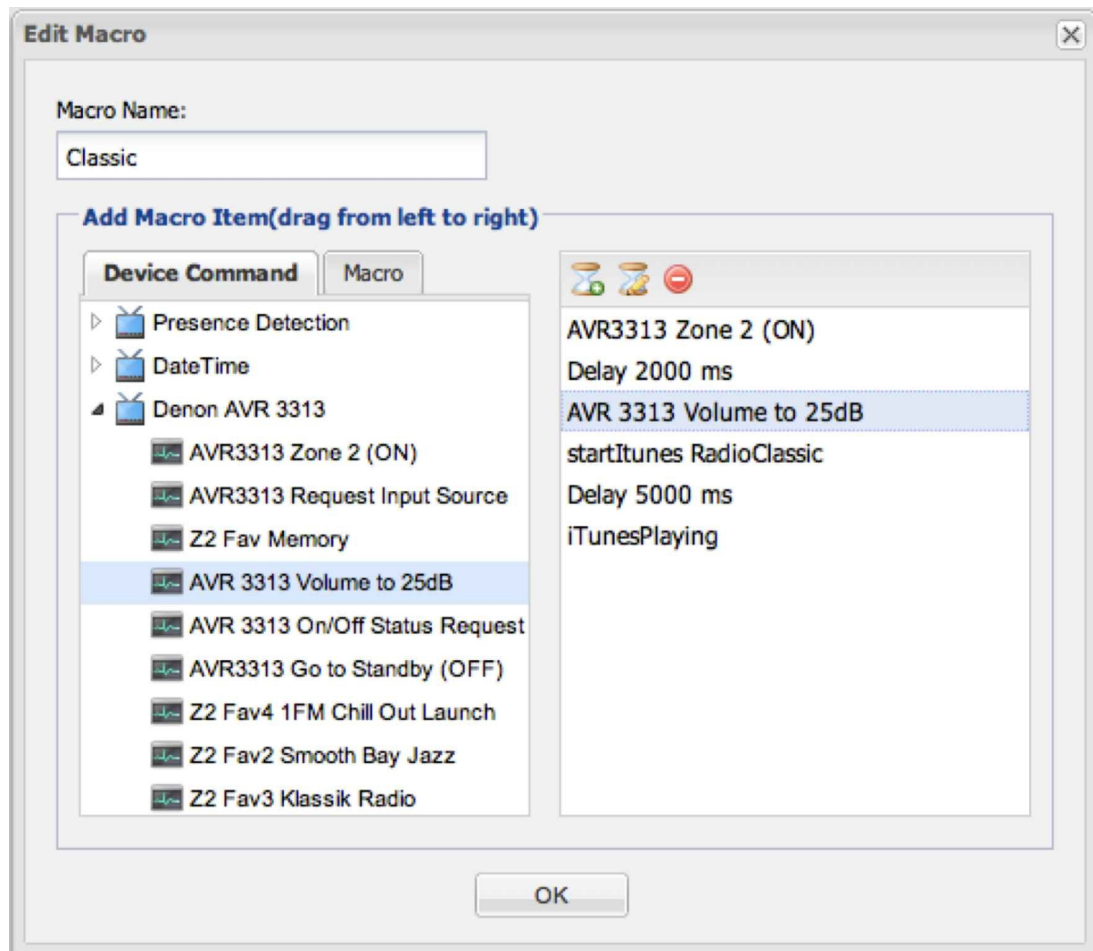


Figure 10.3 Enhancing the iTunes macros with Denon control commands

In a similar manner we can update our rule definitions with Denon control commands as needed. As an example we can add the two commands

```
execute.command("AVR3313 Zone 2 (ON)");
```

```
execute.command("AVR 3313 Volume to 25dB");
```

to our rule "Wake me up early if it rains" as shown below, which would ensure, that the iTunes playlist, which is triggered by the alarm condition, is played on the Denon zone 2 at a volume of 25 dB. By inserting one command before and one after the Java sleep sequence, we ensure that there is enough time between the two commands to execute:

```
rule "Wake me up early if it rains"
```

```
timer (cron:0 0 6 ? * MON-FRI)
```

```
when
```

```
CustomState(source == "Weather Condition Berlin", name : value)
```

```

CustomState(source == "iAlarm Status", value == "on")
then
String testStr = (String) name;
String lookUp1 = "rain";
String lookUp2 = "snow";
if ((testStr.matches("(?i).*"+lookUp1+".*") || (testStr.matches("(?i).*"+lookUp2+".*")))
{
System.out.println("iAlarm going off "+name);
Date date = new Date();
System.out.println(date.toString());
execute.command("startItunes RadioPop");
execute.command("Turn iAlarm Off");
execute.command("AVR3313 Zone 2 (ON)");
try {
    Thread.sleep(5000);
} catch(InterruptedException ex) {
    Thread.currentThread().interrupt();
}
execute.command("iTunesPlaying");
execute.command("AVR 3313 Volume to 25dB");
}
end

```

The same we can do with the rule "Coming Home Chris":

```

rule "Coming Home Chris"
when
CustomState(source == "PresenceEvent", value == "return_Chris")
CustomState(source == "Presence Status", value == "on")
then
Date date = new Date();
System.out.println(date.toString());
System.out.println("Chris coming home...");
execute.command("startItunes RadioClassic");
execute.command("AVR3313 Zone 2 (ON)");
try {
    Thread.sleep(5000);
} catch(InterruptedException ex) {
    Thread.currentThread().interrupt();
}
}

```

```
execute.command("iTunesPlaying");  
execute.command("AVR 3313 Volume to 25dB");  
end
```

We see, that once the basic structure is in place and functioning, it is very easy to expand the functionality of a rule.

## 10.2 Device Control Using Z-Wave

As outlined in the introductory chapters of this book, for the control of the building infrastructure there are a number of technology options. One of the most popular and wide spread open standard based technology is Z-Wave, which we will cover in this section. The Z-Wave communication protocol is also supported by OpenRemote, which is why we can seamlessly integrate any Z-Wave controlled device into our smart home control center. To build a Z-Wave network all you need is a Z-Wave controller, typically in form of a USB-stick, and Z-Wave controllable devices. There are a large number of Z-Wave devices available from various vendors, starting from power-switches, sensors (water, door, energy, light), door-bells or general purpose controllers. Other examples, which demonstrate what is possible with Z-Wave, are LED light-bulbs with built in Z-Wave switches or Z-Wave controllable film, that converts glass from transparent to opaque for energy and privacy needs. (<http://aeotec.com/homeautomation> ).

## 10.2.1 Z-Wave Network Setup

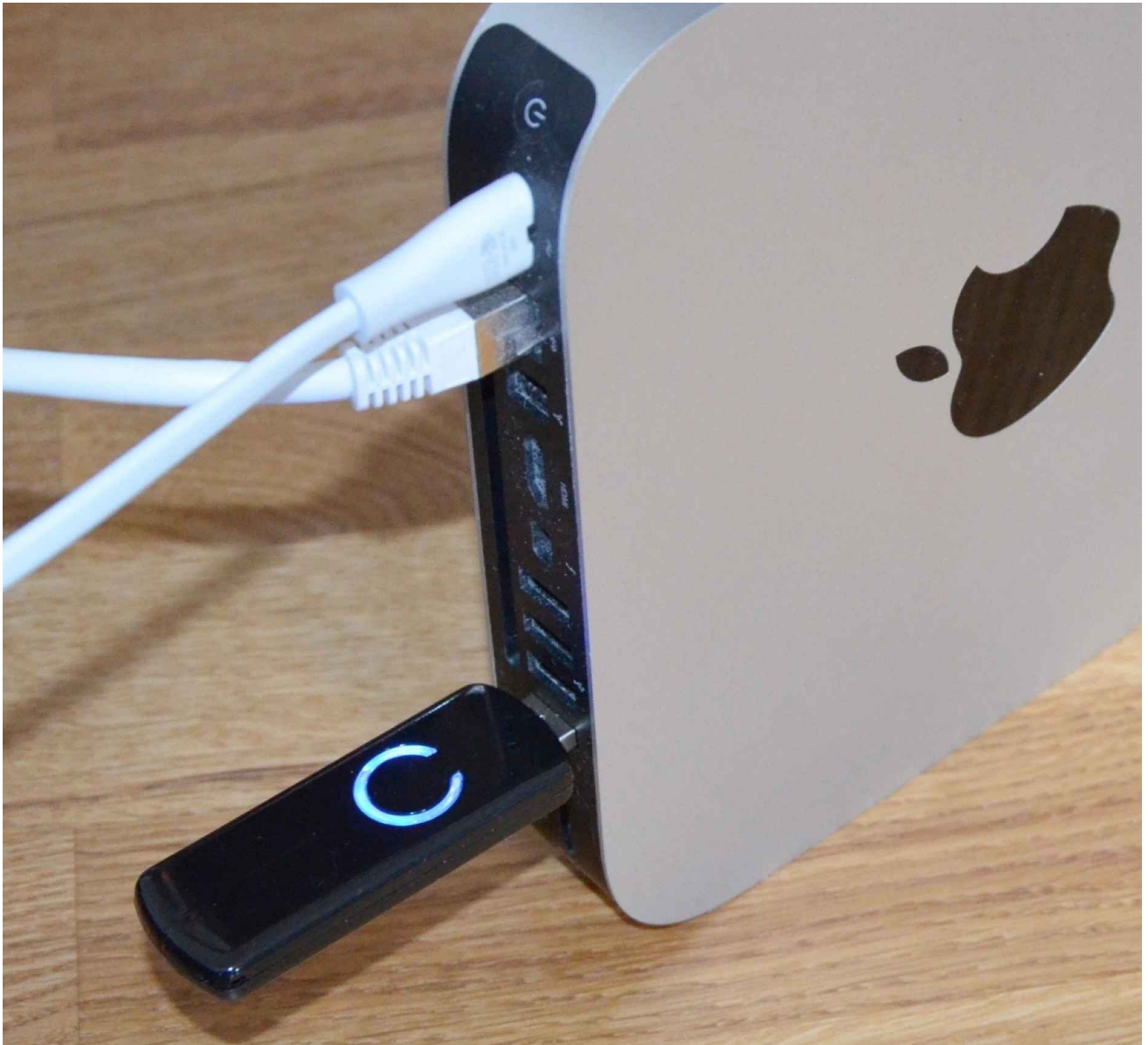
For our project we will use the Aeotec USB stick, currently probably one of the most popular Z-Wave controllers on the market. To install it we download the necessary software, a USB to UART driver (available for OS X, Windows and Linux) from

<http://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx>

After driver installation under OS-X we can validate the correct installation by validating the presence of the file `tty.SLAB_USBtoUART` in the device directory `/dev` as shown below:

```
cd /dev
ls
...
....
tty
tty.Bluetooth-Incoming-Port
tty.Bluetooth-Modem
tty.SLAB_USBtoUART
ttyp0
...
...
```

Under Windows the USB stick is assigned a COM port (e.g., COM3) after insertion. Go to *Device Manager — Ports* to identify the port number, which you will need later for the configuration in OpenRemote. Also be aware, that when you move the stick from one USB-port to another, the COM port might change, and your configuration might not work anymore (Figure 10.4).



*Figure 10.4 The Aeotec Z-Wave Controller USB Stick*

## 10.2.2 Connecting Z-Wave Devices

As a first Z-Wave device we connect an Everspring AN158 to our Aeotec USB stick based Z-Wave network. The AN158 is a combined power-switch and power meter. In order to include it in our Z-Wave network, we need to disconnect our Aeotec USB stick and press its control key, which starts flashing slowly. Now we put it next to the plugged in Everspring AN158, which we now also switch to inclusion mode by pressing its control button three times within 1.5 seconds. The Aeotec LED now starts blinking fast, and then stays solid for three-seconds. This is the sign that we have successfully associated our Everspring AN158 with our Z-Wave network. We now plug the Aeotec USB stick back to our controller PC.

Before we start integration with OpenRemote, we want to verify the functionality of our Z-Wave network. To do this we can use the free Z-Wave controller from the smart home company Qees, which can be downloaded (for Windows and OS X) from

<http://www.qees.eu/products/z-wave/z-wavepctool>

When opening the Qees controller for the first time, under *All Devices* the software automatically displays all devices, which are associated with the controller, in our case the Z-Wave power switch AN158 from Everspring (Figure 10.5).

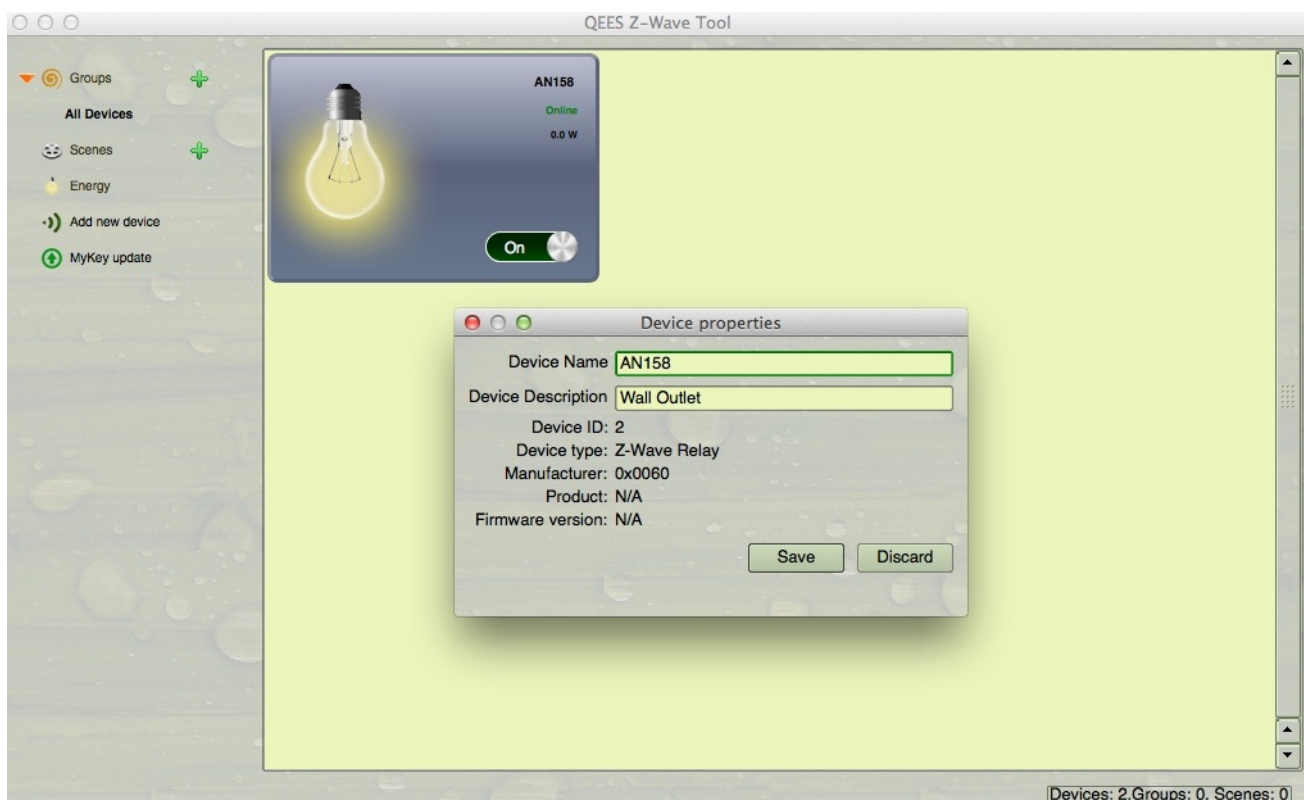


Figure 10.5 The QEES Z-Wave Controller Software

To turn devices *on* or *off*, simply use the slider in GUI window of the device. A right click on the device GUI displays other actions that can be performed. Selecting the last option *Properties* opens a dialog box which shows the name of the device, its description, its ID and its type. Take a note of the ID, which you will need when configuring the device under OpenRemote.

## 10.2.3 Configuring OpenRemote for Z-Wave Operation

We now know that our Z-Wave components work, and can integrate them into our OpenRemote control infrastructure. Before starting make sure you have the latest OpenRemote controller software, version 2.1 January 2015 or later, installed. If in doubt double check the creation date of *openremote.sh* or *openremote.bat*. Also download the file *zwave.jar* from

<http://download.openremote.org/free/zwave>

and copy it into the directory *ORC/webapps/controller/WEB-INF/lib/*, replacing the existing version of *zwave.jar*. Due to license restrictions from Z-Wave chip designer Sigma the Z-Wave code in *zwave.jar* cannot be distributed as part of an open source product. It would then become open source as well. This is why — while being free — it has to be downloaded and installed separately.

Now we go to OpenRemote Designer and open the Z-Wave configuration screen by selecting *Config for Controller — Z-Wave*. For the field *zwave.Comm.Layer* we select *RXTX* from the drop down menu. In the field *zwave.com.Port* under MS Windows we enter the COM port we use for our USB-stick, under OS X we enter the path to the device driver */dev/tty.SLAB\_USBtoUART* we have installed before (Figure 10.6).

zwave.comPort:	/dev/tty.SLAB_USBtoUART
protocol.zwave.classname:	org.openremote.controller.protocol.zwave.ZWaveCommandBuilder
zwave.commLayer:	RXTX
zwave.pad.host:	localhost
zwave.pad.port:	7876

**Hint**  
Configuration for the Z-Wave protocol

Reset to defaults

*Figure 10.6 Configuring Z-Wave in OpenRemote*

Now we can create Z-Wave OpenRemote commands. For our Everspring AN158 we need the three commands, *on*, *off* and *status*. We go to OpenRemote Designer, define a new device called *Z-Wave* and create three commands called *Power Off*, *Power On* and *Power On/Off Status*. In each command definition we select Z-Wave as the communication protocol and enter the Z-Wave node id, which we looked up in the QEES Z-Wave controller software (Figure 10.7). (Further down in this chapter we will learn how to associate Z-Wave devices and how to identify their node ID just using OpenRemote as well. However, for Z-Wave beginners the QEES Z-Wave Controller Software is convenient and easy to use.)



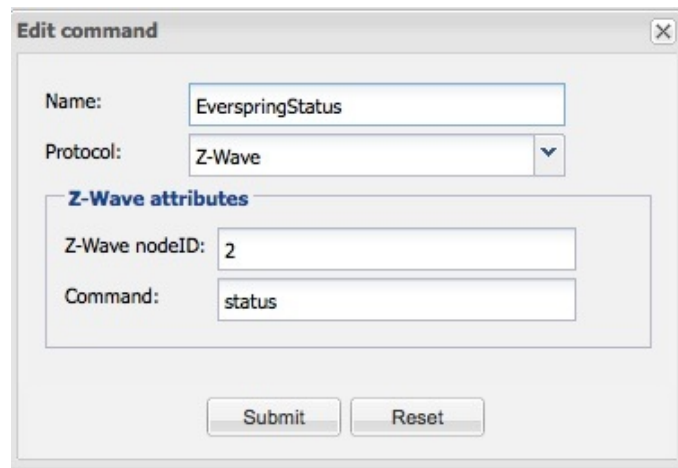


Figure 10.7 Configuring a Z-Wave status command in OpenRemote

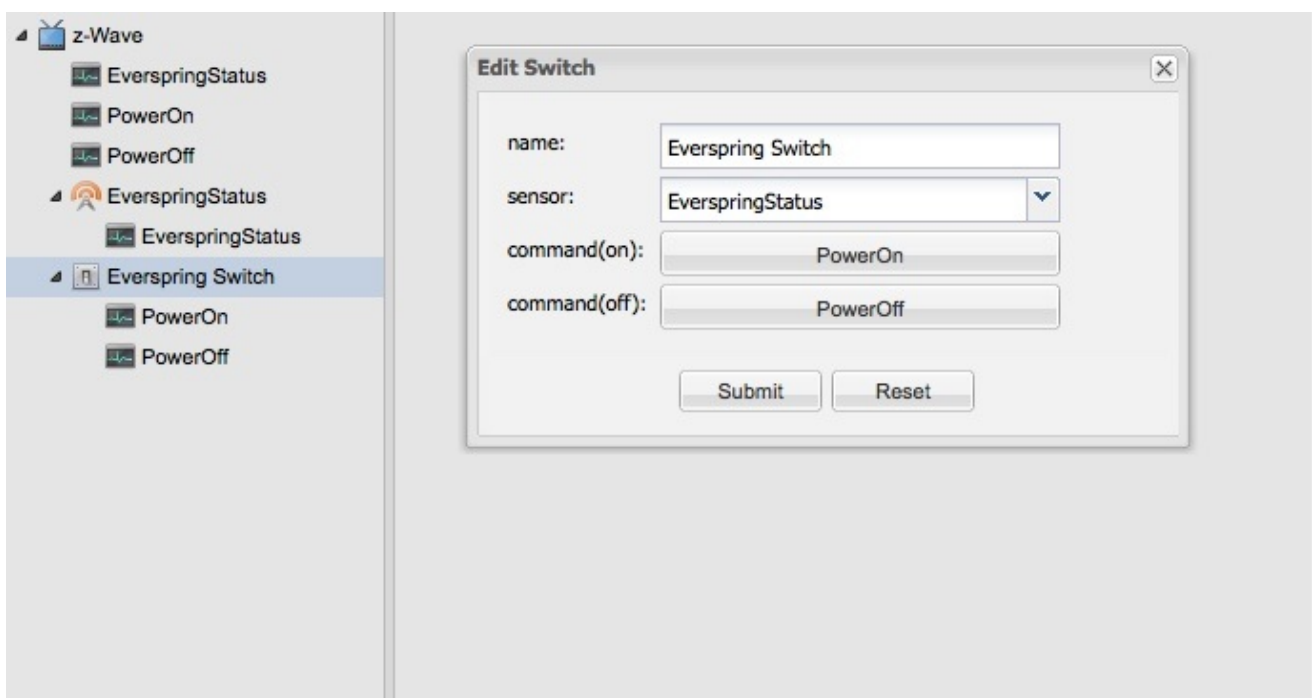


Figure 10.8 Configuring a Z-Wave Sensor in OpenRemote

Next we create an OpenRemote sensor, using the status command we just created (Figure 10.8). With the sensor and the two *on off* commands we then build an OpenRemote switch, which can control our Everspring AN158. We synchronize the new Z-Wave design with our local OpenRemote controller and perform a controller restart by closing the terminal window. (You have to actually close the terminal window and conduct a manual restart. Synchronizing the controller is not sufficient). At restart in the controller window we see that our new sensor has been registered:

```
INFO 2015-03-25 12:16:41,640 : Registered sensor : Switch Sensor (Name = 'AN158Status', ID = ,106905468')
```

```
INFO 2015-03-25 17:13:40,937 : Startup complete.
```

In addition OpenRemote has placed a Z-Wave log file in xml format in the Z-Wave directory

```
/ORC/webapps/controller/zwave
```

In our case for the Everspring Z-Wave device with its node ID 2 the Z-Wave log file is called *node2.xml* and looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<node id="2">
  <manufacturer id="96">Everspring</manufacturer>
  <basic-device-class id="0x04">BASIC_TYPE_ROUTING_SLAVE</basic-device-class>
  <generic-device-class id="0x10">GENERIC_TYPE_SWITCH_BINARY</generic-device-class>
  <specific-device-class id="0x01">SPECIFIC_TYPE_POWER_SWITCH_BINARY</specific-device-class>
  <product-type id="0x0004" />
  <product id="0x0002" />
  <listening>true</listening>
  <routing>true</routing>
  <command-classes>
    <command-class id="0x20" version="1" name="COMMAND_CLASS_BASIC" type="supported" />
    <command-class id="0x70" version="1" name="COMMAND_CLASS_CONFIGURATION" type="supported" />
    <command-class id="0x32" version="2" name="COMMAND_CLASS_METER_V2" type="supported">
      <meter-type id="0x01">ELECTRIC_METER</meter-type>
      <scales>
        <scale id="0x02">ELECTRIC_METER_SCALE_W</scale>
        <scale id="0x00">ELECTRIC_METER_SCALE_KWH</scale>
      </scales>
      <meter-reset>true</meter-reset>
    </command-class>
    <command-class id="0x72" version="1" name="COMMAND_CLASS_MANUFACTURER_SPECIFIC"
type="supported" />
    <command-class id="0x25" version="1" name="COMMAND_CLASS_SWITCH_BINARY" type="supported" />
    <command-class id="0x85" version="2" name="COMMAND_CLASS_ASSOCIATION_V2" type="supported" />
    <command-class id="0x86" version="1" name="COMMAND_CLASS_VERSION" type="supported" />
  </command-classes>
  <configuration hash="1FD99131DD59F877B177895EF925C84D">
    <associations>
      <association-group id="1" capacity="1">
        <association>
          <node>1</node>
        </association>
      </association-group>
      <association-group id="2" capacity="4">
        <association>
          <node>1</node>
        </association>
      </association-group>
    </associations>
  </configuration>
</node>
```

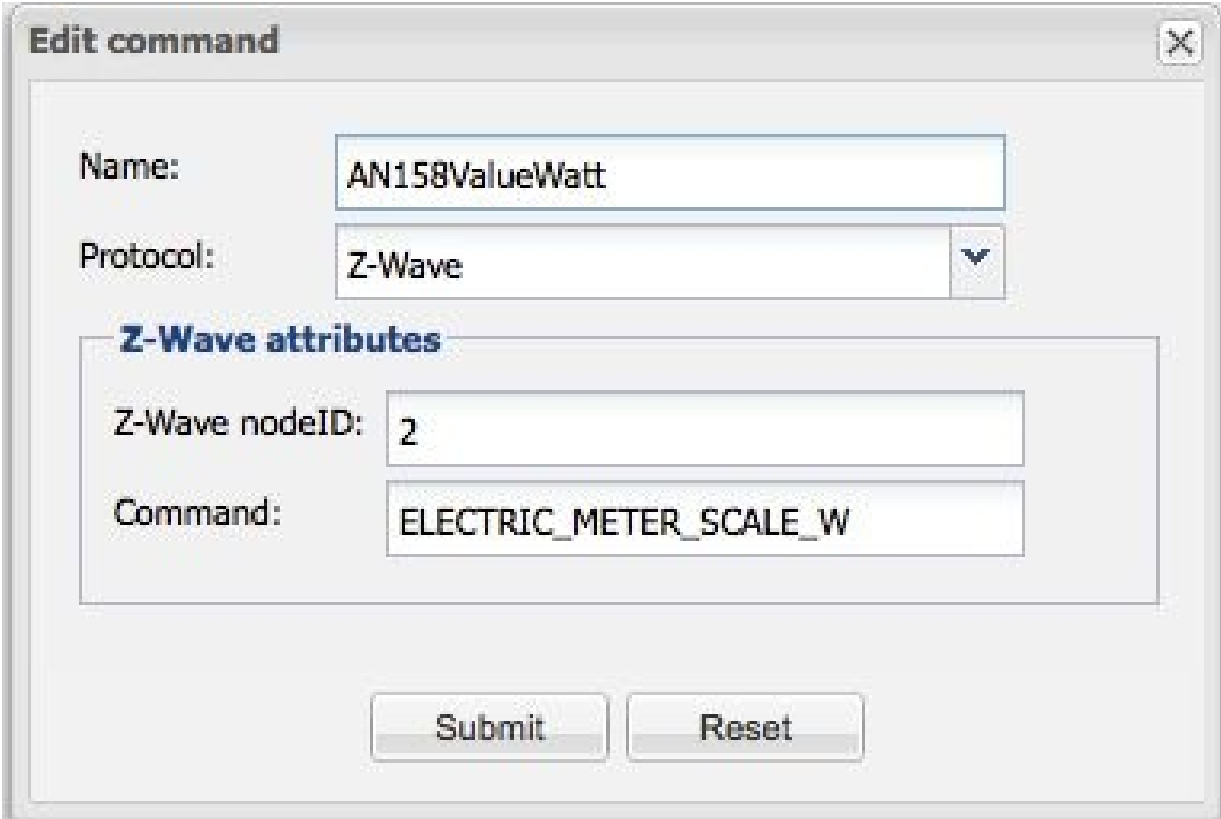
```
<parameters />
</configuration>
</node>
```

The file contains a configuration section (<configuration> ... </configuration>) and a command class section (<command-classes> ... </command-classes>). In the command class section we find the two commands

ELECTRIC\_METER\_SCALE\_W and

ELECTRIC\_METER\_SCALE\_KWH

Both commands can also directly be used to create OpenRemote commands. We want to retrieve the power reading of the AN158 in Watts and create the command *AN158Watt* using the command ELECTRIC\_METER\_SCALE\_W. (Figure 10.9). Next we create the sensor *AN158Watt* using the command *AN158Watt*, which we just created. As sensor *type* we select *level*.

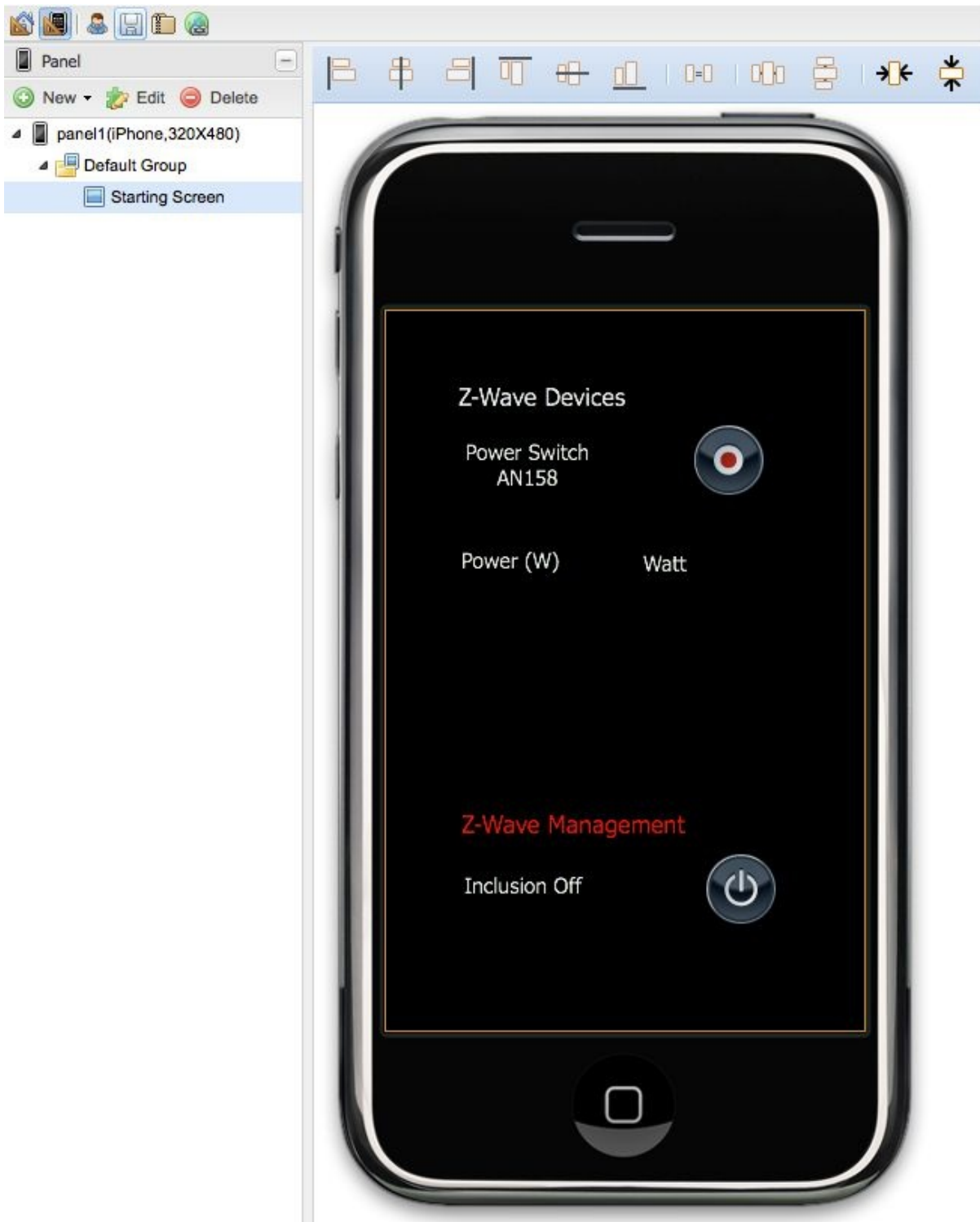


The screenshot shows a dialog box titled "Edit command" with a close button in the top right corner. The dialog contains the following fields and controls:

- Name:** A text input field containing "AN158ValueWatt".
- Protocol:** A dropdown menu showing "Z-Wave" with a downward arrow.
- Z-Wave attributes:** A section containing:
  - Z-Wave nodeID:** A text input field containing "2".
  - Command:** A text input field containing "ELECTRIC\_METER\_SCALE\_W".
- Buttons:** "Submit" and "Reset" buttons at the bottom.

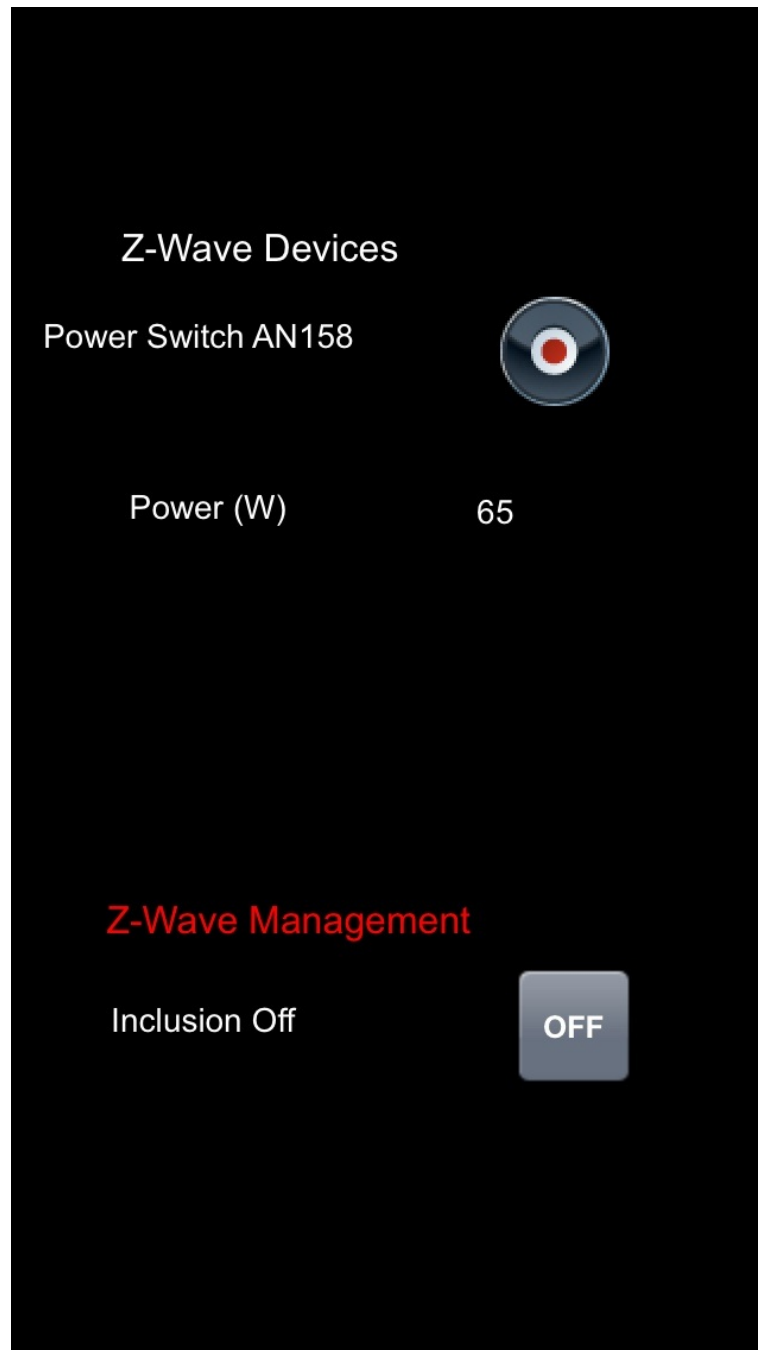
*Figure 10.9 Creating the OpenRemote Power Meter Readout Command for the AN158*

Now all what is left to do is to create a GUI in OpenRemote Designer, which contains the necessary GUI elements: the switch element for switching our AN158 *on* and *off* and the sensor, to display the power meter reading of our device (Figure 10.10).



*Figure 10.10 Creating the Z-WAVE GUI for the AN158 in OR Designer*

We can now control the AN158 power switch from our smartphone app and monitor its power level (Figure 10.11). Using the above procedure we can integrate any Z-Wave item of our building infrastructure. Equally we can integrate any OpenRemote Z-Wave command, sensor or switch into any of our OpenRemote rules.



*Figure 10.11 The AN158 Control GUI on a Smartphone*

Alternative to configuring the Z-Wave network with an external Z-Wave controller software, you can also use the following dedicated OpenRemote Z-Wave commands for device inclusion and exclusion:

INCLUSION\_MODE\_ON

INCLUSION\_MODE\_OFF

INCLUSION\_MODE\_STATUS

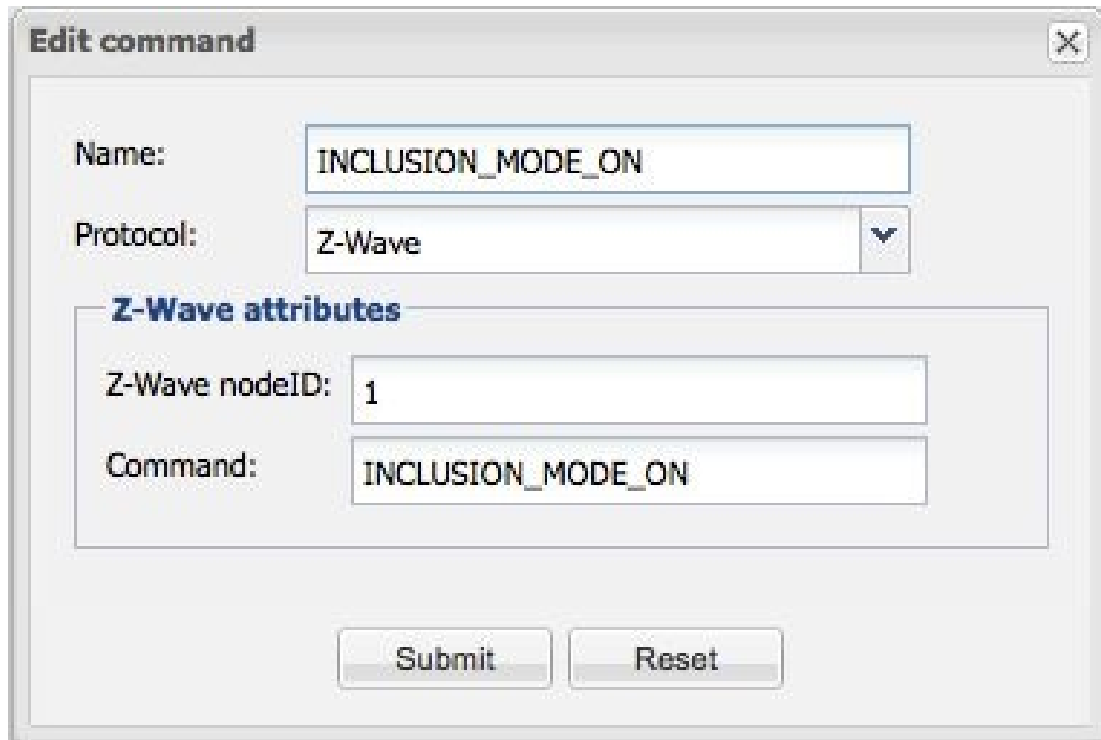
EXCLUSION\_MODE\_ON

EXCLUSION\_MODE\_OFF

EXCLUSION\_MODE\_STATUS

(Inclusion is the process which adds a device to the Z-Wave network, exclusion the process which removes a device.) The inclusion/exclusion mode automatically turns off after 60 seconds or after a Z-Wave node has been added or removed. To use this feature of

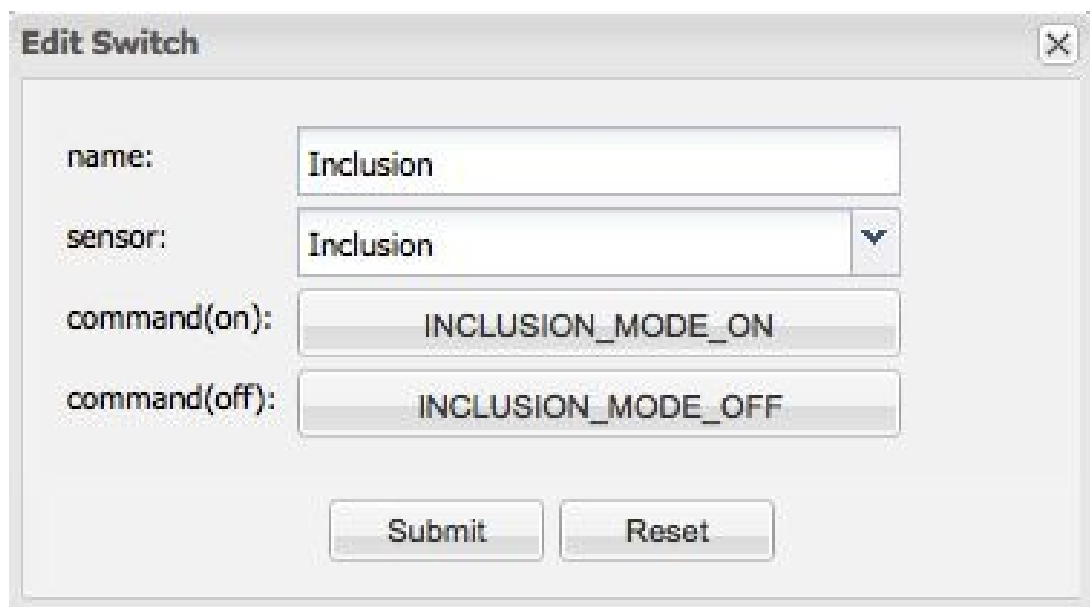
the OpenRemote Z-Wave implementation simply create two OpenRemote sensors, one using the command INCLUSION\_MODE\_STATUS and one the command EXCLUSION\_MODE\_STATUS. (Make sure to use the exact command for the name of the command as well as the command itself (Figure10.12).



The screenshot shows a dialog box titled "Edit command". It has a close button in the top right corner. The "Name:" field contains "INCLUSION\_MODE\_ON". The "Protocol:" field is a dropdown menu currently showing "Z-Wave". Below this is a section titled "Z-Wave attributes" which contains two fields: "Z-Wave nodeID:" with the value "1" and "Command:" with the value "INCLUSION\_MODE\_ON". At the bottom of the dialog are two buttons: "Submit" and "Reset".

*Figure 10.12 Creating a Z-Wave Inclusion Command in OR Designer*

Then create two OpenRemote switches, one for inclusion and one for exclusion. The inclusion switch consists of the INCLUSION\_MODE\_STATUS sensor and the two commands INCLUSION\_MODE\_OFF and INCLUSION\_MODE\_ON. The exclusion switch accordingly uses the EXCLUSION\_MODE\_STATUS sensor and the two commands EXCLUSION\_MODE\_OFF and EXCLUSION\_MODE\_ON (Figure 10.13). After adding the according GUI elements to our control panel we can now use OpenRemote to manage our Z-Wave network.



The screenshot shows a dialog box titled "Edit Switch". It has a close button in the top right corner. The "name:" field contains "Inclusion". The "sensor:" field is a dropdown menu currently showing "Inclusion". Below this are two fields: "command(on):" with the value "INCLUSION\_MODE\_ON" and "command(off):" with the value "INCLUSION\_MODE\_OFF". At the bottom of the dialog are two buttons: "Submit" and "Reset".

*Figure 10.13 Creating a Z-Wave Inclusion Switch in OR Designer*



*Figure 10.14 The Z-Wave combined power switch / power meter Everspring AN158*





## Bibliography

DD&M Holdings Inc., “DENON AVR control protocol 5.2a”.2013

[http://usa.denon.com/US/Downloads/Pages/InstructionManual.aspx?FileName=DocumentMaster/US/AVR-3808CISerialProtocol\\_Ver5.2.0a.pdf](http://usa.denon.com/US/Downloads/Pages/InstructionManual.aspx?FileName=DocumentMaster/US/AVR-3808CISerialProtocol_Ver5.2.0a.pdf)

Christian Paetz, Serguei Polterak. “ZWay Manual”, Z-Wave.Me, 2011

[http://en.z-wave.me/docs/zway\\_manual\\_en.pdf](http://en.z-wave.me/docs/zway_manual_en.pdf)



# 11 Industry Grade Home Infrastructure

## Control: KNX

## 11.1 What is KNX?

KNX (short for Konnex) is a European (EN50090, 2003) and international (ISO/IEC 14543-3, 2006) standard for industry grade home and building automation. It specifies how technical building infrastructure such as light, heating, ventilation, shutters, alarm-systems or power-outlets be controlled by switches, sensors, tablets or smart-phones based on a dedicated wireline control infrastructure. By merging three previous standards (EIB, EHS, BatiBus) for building automation KNX achieves compatibility across technologies (light, heating, power-outlets etc.) and vendors. In other words: In an all KNX compliant building any switch can be configured with a few mouse-clicks to control any technology. For example a room-controller from vendor A can be configured to control a heating system from vendor B, dim lights from vendor C and report room temperature to a smart-phone app from vendor D. Since KNX is built upon a dedicated, standardized cable infrastructure it is more expensive (and more reliable) than automation technologies using wireless technologies or the existing electric power lines.

## 11.2 How does KNX Work?

KNX is build up of the following components:

- A centralized cable infrastructure consisting of :
  - all power cabling (light, power-outlets, shutters, etc.) leading to a central control location
  - a KNX control cabling infrastructure, connecting all switches and sensors to the same central control location via dedicated, standardized KNX cabling
- KNX actuators, located at the central control location. The actuators conduct the actual control (switching) of the building infrastructure through their connection to the various power cables (dim lights, control heating systems, activate or deactivate wall-outlets, etc.)
- KNX switches and sensors in the building, which are connected through the KNX control cabling infrastructure to the KNX actuators, telling them when to switch which system.
- KNX communication protocol, which is the means of communication between the devices in the KNX network
- KNX-USB module to connect a PC for configuring the KNX infrastructure
- KNX/IP router to integrate the KNX infrastructure with IP other IP devices (smartphones, tablets)

Through this concept, the control layer and the building engineering layer are physically and logically separated. With that any switch can control any device independent of its location. As a result, the operation of the building engineering infrastructure can be controlled in a much more sophisticated and flexible way compared to traditional infrastructure. KNX switches, KNX sensors, KNX devices and KNX actuators communicate using the KNX protocol, which uses IP as transport protocol. Via a KNX-USB module the PC with ETS software is connected to the KNX network. If, as in our project, additional control and monitoring functions via networked devices (smartphones, tablets) are desired, a KNX/IP router module is needed. The devices - in our example the OpenRemote controller - accesses the IP address of the KNX/IP router and through it is capable of sending KNX commands or reading KNX status messages, as we will see further down.

## 11.3 The KNX Software Infrastructure: ETS

ETS stands for Engineering Tool Software and is a manufacturer independent software application developed by the KNX-Organization (<http://www.knx.org>). It has been designed to configure KNX based building control installations.. There are also vendor specific implementations for KNX configuration and installation, however the KNX ETS tool is widely used in the industry and the quasi-industry standard for KNX installations. Since the KNX API is standardized, any KNX software should work with any KNX certified component. The individual software functions of the KNX components are provided via product libraries, which the vendors typically offer as free downloads from their websites. ETS in its current version 5 runs on the Windows operating systems and is available in three licenses:

ETS5 Demo:	a free test and trial version for very small test projects with up to three KNX devices only
ETS5 Lite:	a low cost license (about 200€) for small to mid-range projects with up to 20 KNX devices
ETS5:	the full professional license for all projects sizes and functions (about 1000€)

Previous versions (you might run into) were (ETS1: 1993-1996, ETS2: 1996-2004, ETS3: 2004-2010, ETS4: 2010).

ETS5 was released in 2014 with higher performance (in particular for larger projects) and several usability improvements compared to ETS4. Previous ETS versions can be upgraded to ETS5, which however requires the usage of a hardware dongle.

ETS can be purchased and downloaded directly from the [knx.org](http://knx.org) web site. While ETS is used for KNX configuration, the KNX infrastructure can also be monitored and controlled via smartphone, tablet or PC. Among other tools our project platform OpenRemote also supports the KNX protocol. After configuring our infrastructure with ETS thus we will integrate the KNX components into our OpenRemote based project.

## 11.4 Which Operating Systems does ETS Support?

The following Windows environments are supported:

Windows XP Professional; SP3 (32 Bit), Windows XP Professional; SP2 (64 Bit), Windows Vista; SP2 (32/64 Bit), Windows 7; SP1 (32/64 Bit), Windows Server 2003; SP2 (32/64 Bit)

Windows Server 2003 R2 SP2 (32/64 Bit), Windows Server 2008; SP2 (32/64 Bit), Windows Server 2008 R2; SP1 (32/64 Bit)

The required computer hardware requirements are:

CPU:  $\geq 2$  GHz, RAM:  $\geq 2$  GB, HDD:  $\geq 20$  GB, RES:  $\geq 1024 \times 768$

## 11.5 ETS on a Mac

While the KNX-Forum does not officially support ETS running in virtual environments without problems. I have been running ETS4 and ETS5 on Mac OS-X 10.7 through OS-X 10.10 using Parallels Desktop and Windows 7/8 with no issues.



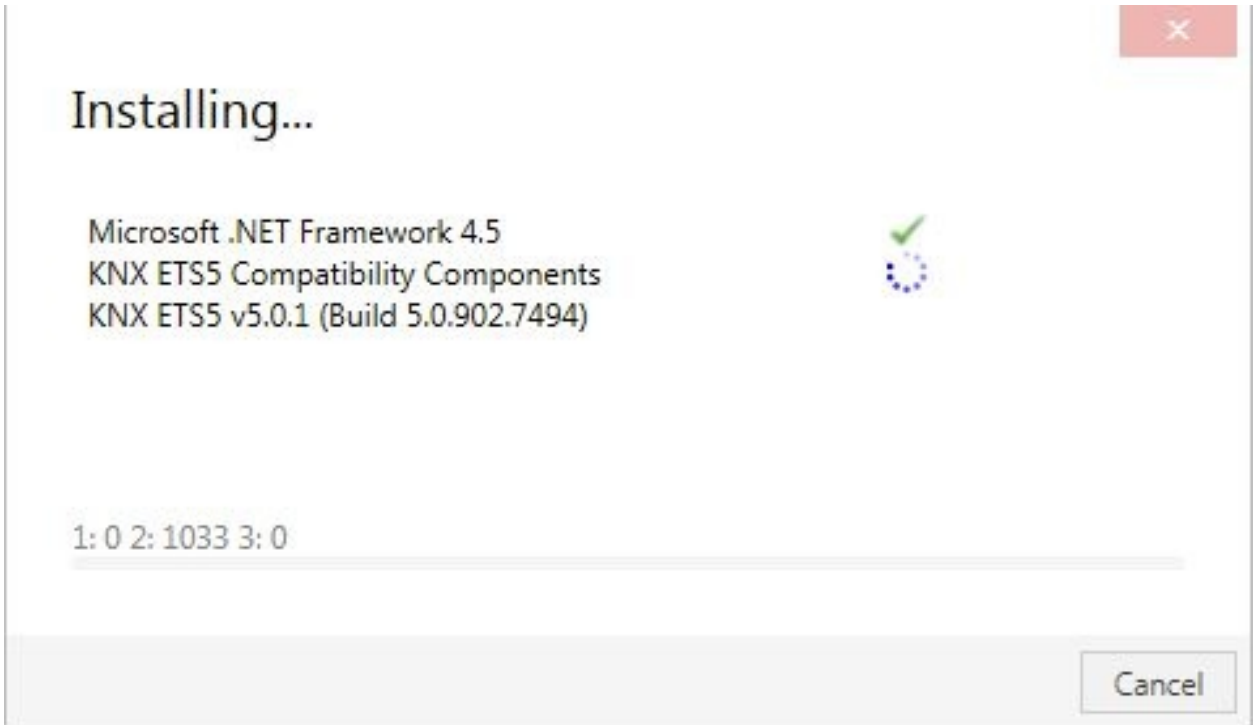
## 11.6 Other KNX.org Software Tools

There are also some other software tools, which are provided by the KNX organization:

- iETS Server is a gateway application creating an interface between KNX and the IP world using the EIBlib/IP protocol (other than KNXnet/IP). It allows iETS clients to remotely connect to the KNX network for maintenance and troubleshooting purposes. The software dates back from 2003. In the meantime many new tools from a number of vendors (e.g. aycontrol.com) are available, so you probably do not want to go with this.
- Falcon is a network connection library for ETS, and is used by software developers to write KNX device drivers.
- EITT is the KNX Interworking Test Tool used to test the interoperability of KNX devices by manufacturers
- KNX Manufacturer Tool is used by manufacturers of KNX hardware for the creation of the ETS product database entries of their devices.

## 11.7 ETS5 Installation

The ETS5 installation file can be downloaded (for free) directly from [knx.org](http://knx.org). Its size is about 500 Mbytes. After decompressing (unzip) the downloaded file you run the resulting .exe file. The ETS install procedure automatically sets up all required components (Figure 11.1).

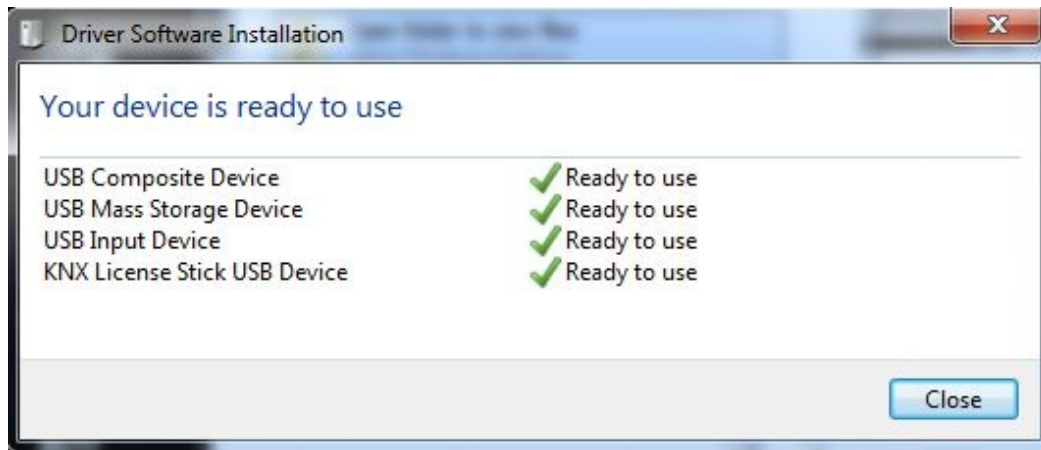


*Figure 11.1 Installing ETS5*

After successful installation you can immediately run ETS without any license key as a demo version, which restricts the maximum number of KNX devices to three. If you have purchased a Lite or Professional license insert the KNX USB dongle into your computer and wait until Windows finishes installing the required device drivers (Figure 11.2, 11.3).



*Figure 11.2 The KNX ETS5 Dongle*



*Figure 11.3 Installing the ETS5 dongle*

When you start ETS5 its license status is displayed at the bottom of the ETS5 user interface. Now click on the *Licenses* button at the bottom of the GUI and the licensing window opens. Now copy the dongle ID which is displayed on the left column of the licensing window to the Windows Clipboard. With the ID you can now download the required license file from your account at the [KNX.org](http://KNX.org) website. Go to *My Account — My Products*, enter the dongle ID and select *Add Key*. The product key field now turns into a link, which you can use to download the license file (format \*.license) to your computer. Now you can go back to the ETS5 license window and select the green + button. Browse to the license file, select it and click on *Open*. Now your license is activated and displayed at the bottom of the GUI (Figure 11.4).

ETS ?

Overview Bus Catalogs Settings

Your Projects + ✎ ↓ ↑

Name	Last Modified	Status
Pheasant	04.03.2015 15:53	Unknown

### KNX News

**KNX Userclub Greece 1st meeting in 2015!**  
18.02.2015

More than 55 participants from Greece and Cyprus joined the first meeting in 2015 of the KNX Userclub Greece on February 11th, 2015.

This event, which was held at the premises of KNX Scientific Partner 'National Technical University of Athens (NTUA)', offered the unique opportunity to learn about the currently ongoing research project by NTUA about KNX. Furthermore, the event was accompanied by presentations, given by representatives of KNX Training Centres and KNX Association International.

The event concluded with the confirmation of KNX Userclub Greece's President, Mr. Ioannis Stathopoulos. As first action, Mr. Stathopoulos drafted the agenda for the upcoming events of 2015 and the future outlook for the KNX Userclub Greece.

The first event in 2015 underlined the growing importance amongst the KNX integrators in Greece and more events are soon to be followed in

### New KNX Products

**Synco OZW772** ◀ ▶  
Siemens Schweiz AG (Switzerland)

For the first time, version V5 of the Synco OZW772 web server from Siemens allows joint web access to Synco HVAC controllers and KNX electrical devices via one web server, which makes operation, data collection and data analysis for these disciplines significantly easier. This is an essential requirement for integrated applications from Siemens – comprehensively tested applications that make it possible to achieve significant energy savings by automatically exchanging data between the HVAC primary plant and room automation devices. Version V5 of the OZW772 web server supports up to 250 Synco controllers and provides for the integration of up to 230 KNX communication objects.

**Certified KNX Products**  
[See a list of all certified KNX products here.](#)

ETS Version ETS 5.0.1 (Build 902) i Licenses ETSS Lite Active Apps 1

Figure 11.4 The ETS5 main screen

## 11.8 Importing Vendor Catalogs

For each KNX device you use in your project, you need to download and install its product specific KNX library file (catalog file). Some vendors offer a single file with all product libraries they provide, for others you need to individually locate and download the catalog files for the products you need. Older ETS catalog files have the extensions .vd3 or .vd4, new files (in XML format) carry the extension .knxprod. Once you have downloaded the catalog files you open the *catalog* menu in ETS5, select the vendor files and ETS will either convert them to XML (in case of older files such as vd3 or vd4) or it will directly import them into the ETS catalog (Figure 11.5).

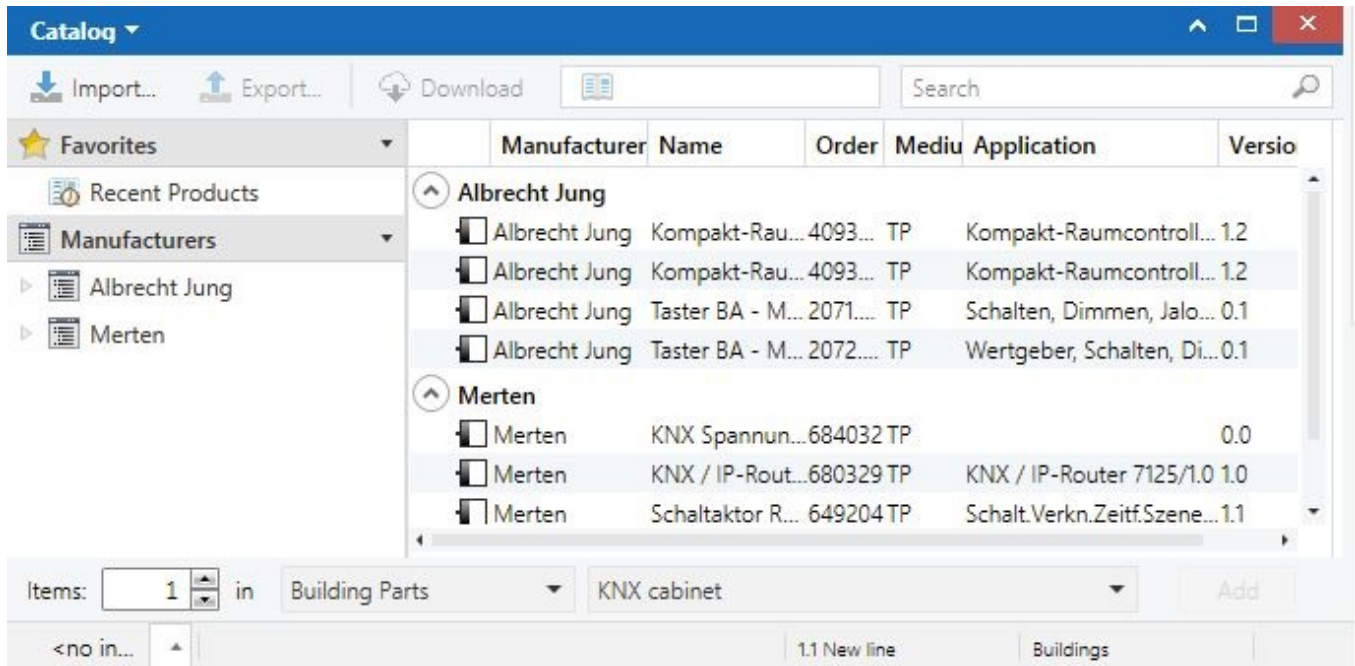


Figure 11.5 Vendor entries in the ETS5 catalog menu

## 11.9 ETS5 Infrastructure Configuration

In order to set up a KNX infrastructure you need to go through the following five steps:

- create new project and the building topology in ETS
- add the KNX elements to the topology
- create logical functions (with KNX addresses as identifiers)
- connect KNX devices through the logic functions (e.g. a KNX switch with a KNX dim actuator)
- download the configuration to the KNX hardware

The first step is to create a new project (Figure 11.6).

New project (1) Import Date: 10.03.2015 Last Modified: 10.03.2015 15:57

Details Project Log Project Files

Name: New project (1) Password: [ ] Set Password

Project Number: [ ] BCU Key: [ ] Set Key

Contract Number: [ ] Codepage: US-ASCII

Start Date: Select a date [15] Group Address Style:  Free  Two Level  Three Level

End Date: Select a date [15] Compatibility:  Hide extended group address range for plugins

Status: Unknown

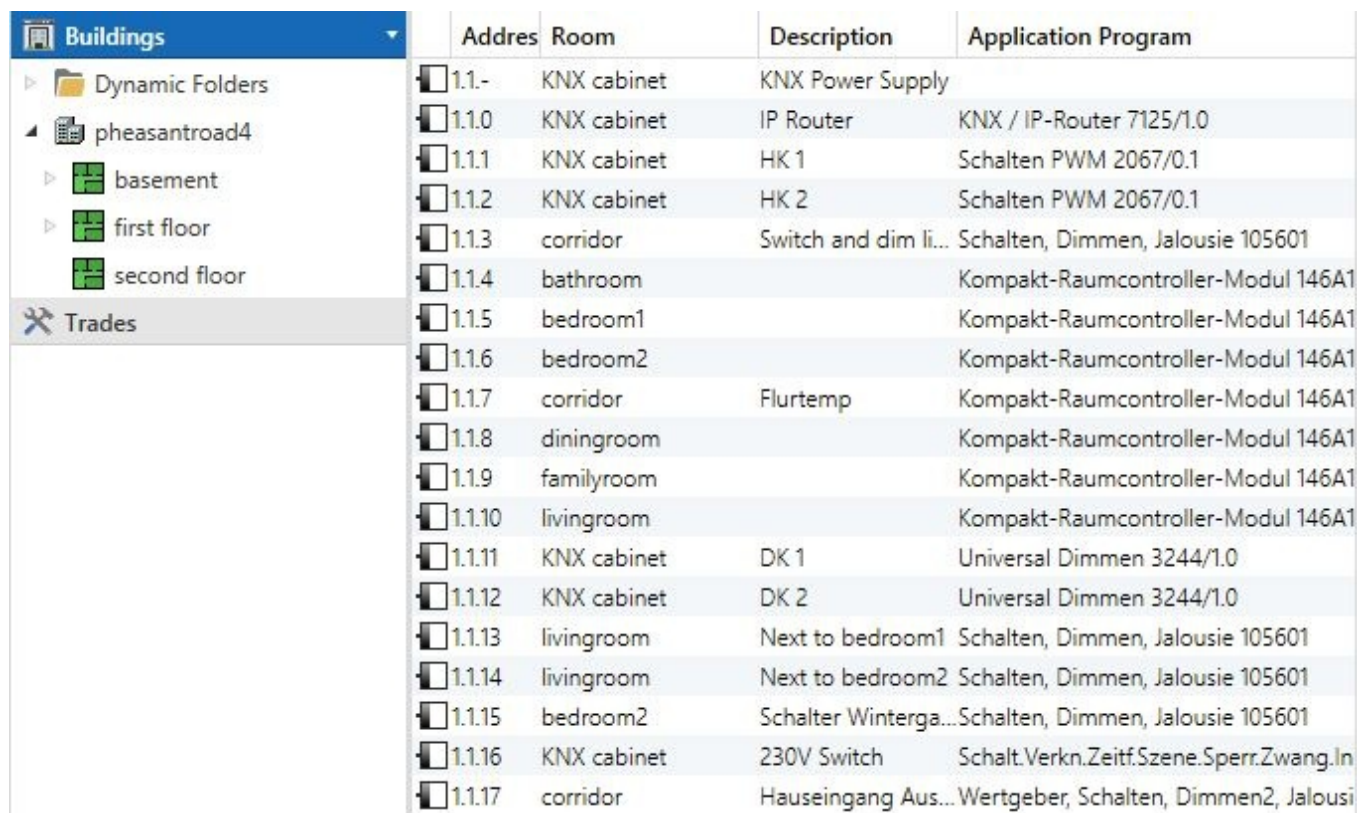
Comment: [ ]

Figure 11.6 Starting a new project in ETS5



## 11.10 ETS5: Adding the Building Infrastructure

We start by adding a building, and assigning it a name. Then we add the floors by right clicking on the building entry. We create three levels and call them basement, first floor and second floor. Then we configure (again through menus, which pop up through a right-mouse-click) rooms, the corridor and in the basement the KNX cabinet (Figure 11.7).

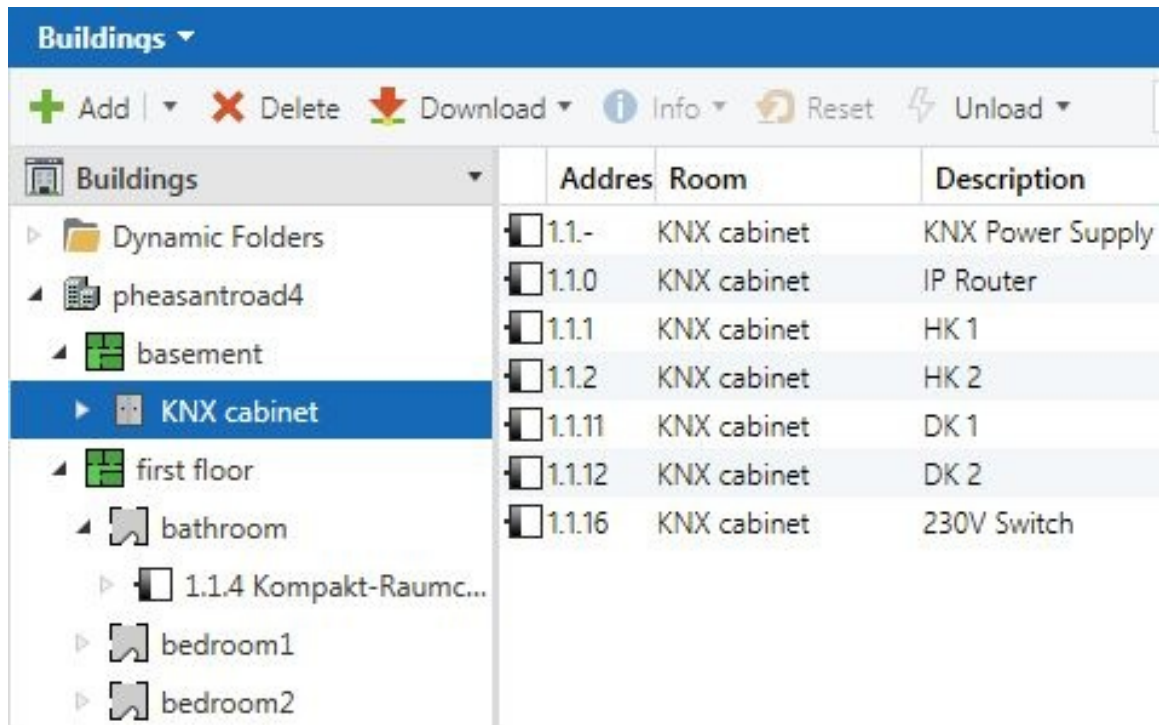


Buildings	Address	Room	Description	Application Program
Dynamic Folders	1.1.-	KNX cabinet	KNX Power Supply	
pheasantroad4	1.1.0	KNX cabinet	IP Router	KNX / IP-Router 7125/1.0
basement	1.1.1	KNX cabinet	HK 1	Schalten PWM 2067/0.1
first floor	1.1.2	KNX cabinet	HK 2	Schalten PWM 2067/0.1
second floor	1.1.3	corridor	Switch and dim li...	Schalten, Dimmen, Jalousie 105601
Trades	1.1.4	bathroom		Kompakt-Raumcontroller-Modul 146A1
	1.1.5	bedroom1		Kompakt-Raumcontroller-Modul 146A1
	1.1.6	bedroom2		Kompakt-Raumcontroller-Modul 146A1
	1.1.7	corridor	Flurtemp	Kompakt-Raumcontroller-Modul 146A1
	1.1.8	diningroom		Kompakt-Raumcontroller-Modul 146A1
	1.1.9	familyroom		Kompakt-Raumcontroller-Modul 146A1
	1.1.10	livingroom		Kompakt-Raumcontroller-Modul 146A1
	1.1.11	KNX cabinet	DK 1	Universal Dimmen 3244/1.0
	1.1.12	KNX cabinet	DK 2	Universal Dimmen 3244/1.0
	1.1.13	livingroom	Next to bedroom1	Schalten, Dimmen, Jalousie 105601
	1.1.14	livingroom	Next to bedroom2	Schalten, Dimmen, Jalousie 105601
	1.1.15	bedroom2	Schalter Winterga...	Schalten, Dimmen, Jalousie 105601
	1.1.16	KNX cabinet	230V Switch	Schalt.Verkn.Zeitf.Szene.Sperr.Zwang.In
	1.1.17	corridor	Hauseingang Aus...	Wertgeber, Schalten, Dimmen2, Jalousi

Figure 11.7 Adding building, floors and rooms in ETS5

## 11.11 ETS5: Configuring the KNX Elements

Once we are done with the building layout, we can start adding the KNX modules (actuators, switches, sensors) we want to use. For this we need to select each room and add its KNX components by selecting it from the vendor catalog listing in ETS. Once we are done with the rooms we add the KNX devices (actuators, power-supply, IP-Router, etc.), which we have installed in the central KNX cabinet (Figure 11.8).



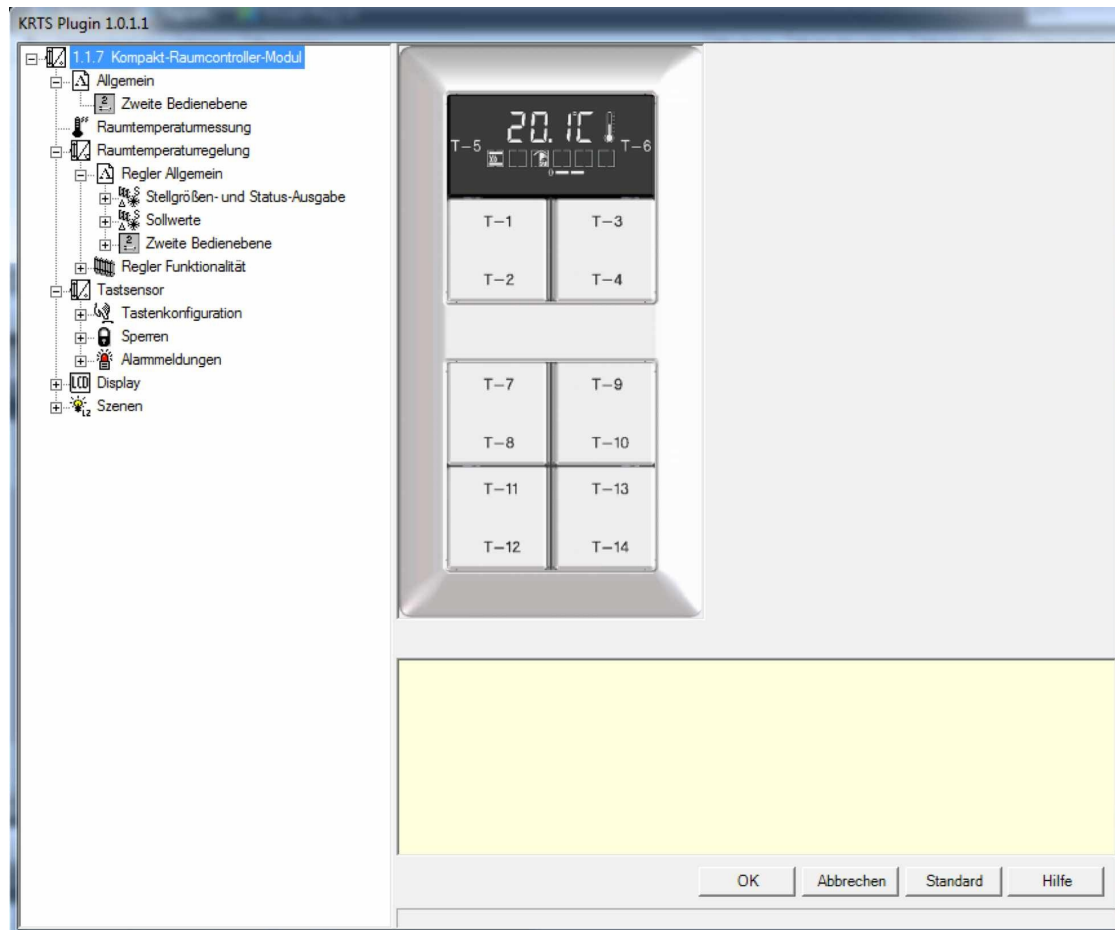
The screenshot shows the 'Buildings' configuration window in ETS5. The left sidebar shows a tree view of the building structure, with 'KNX cabinet' selected. The main table lists the installed devices with their addresses, room names, and descriptions.

Buildings	Address	Room	Description
Dynamic Folders	1.1.-	KNX cabinet	KNX Power Supply
pheasantroad4	1.1.0	KNX cabinet	IP Router
basement	1.1.1	KNX cabinet	HK 1
KNX cabinet	1.1.2	KNX cabinet	HK 2
first floor	1.1.11	KNX cabinet	DK 1
bathroom	1.1.12	KNX cabinet	DK 2
1.1.4 Kompakt-Raumc...	1.1.16	KNX cabinet	230V Switch
bedroom1			
bedroom2			

*Figure 11.8 Configuring the devices in the KNX cabinet*

Depending on the type of element and the individual functionality, for every KNX component we need to set a number of parameters. For example in case of a switch we can configure it to function as a simple light switch, or a more sophisticated dimmer. Figure 11.9 shows the configuration screen of the Jung Compact Room Controller. The switches left and right to the LED display can be used for room temperature control, switches T1 through T14 for light control, shutter control, wall-outlet control, presence detection or scenarios combining multiple controls.





*Figure 11.9 Setting parameters for a room controller*

The control parameters for the heating system are typically the most complex to understand and define. While you can get started with default parameters in many other cases, for the heating system you need to take the time to understand what you are doing. Most importantly you have to make the right selection between underfloor heating and central heating, each of which operates at different water temperatures. And you need to set the correct control algorithm - continuous PI or switched PI (PIW). (Some systems also offer a two point PI mode). Switched PI, which is used for underfloor heating systems, just sends one single Bit to the according heating actuator and its connected valves, just sending an open or close command. Continuous PI sends its control message in form of an entire byte, telling the actuator and valve how far to open or close. These systems are often used in larger central heating configurations. To a certain extent ETS prevents basic misconfigurations. For example is it not possible, to assign a one byte heating control command to a heating actuator, which can only process binary (1 bit) valve commands - open or close. ETS will respond with an error message and refuse the connection to be made.

## 11.12. ETS5: Connecting Infrastructure to Controls

At this point there are three steps left to do, until we can actually start testing our configuration:

- the assignment of KNX addresses for each function
- the assignment of sensors, switches and actuators to the functions represented by KNX addresses
- the actual programming of the KNX hardware by downloading the configuration from the ETS software to the KNX devices

The group addresses are logical addresses, which are assigned to each function in the building, e.g. switch on light bathroom or dim light bedroom). For better organization we start by defining three address groups (Switch to the *Group Adresses* menu and select the green + sign), which shall contain the controls for related functions (Figure 11.10):

- heating
- light
- power sockets

For each group we add the individual functions we want to implement in the group address menu. The actual KNX addresses for the functions are selected automatically by ETS (Figure 11.11). Now for each KNX address (which is equivalent to a function) we need to assign the devices (switches and actuators), which actually realize the functions. This is simply done by dragging the devices from the *Devices* pane into the *Building* window (Figure 11.12).

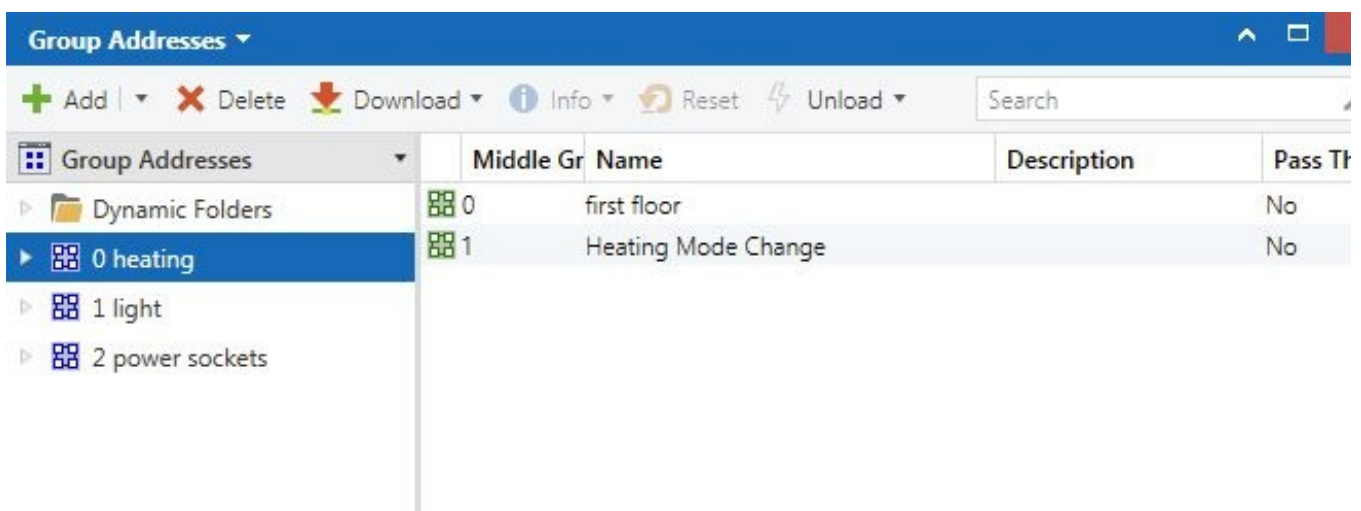
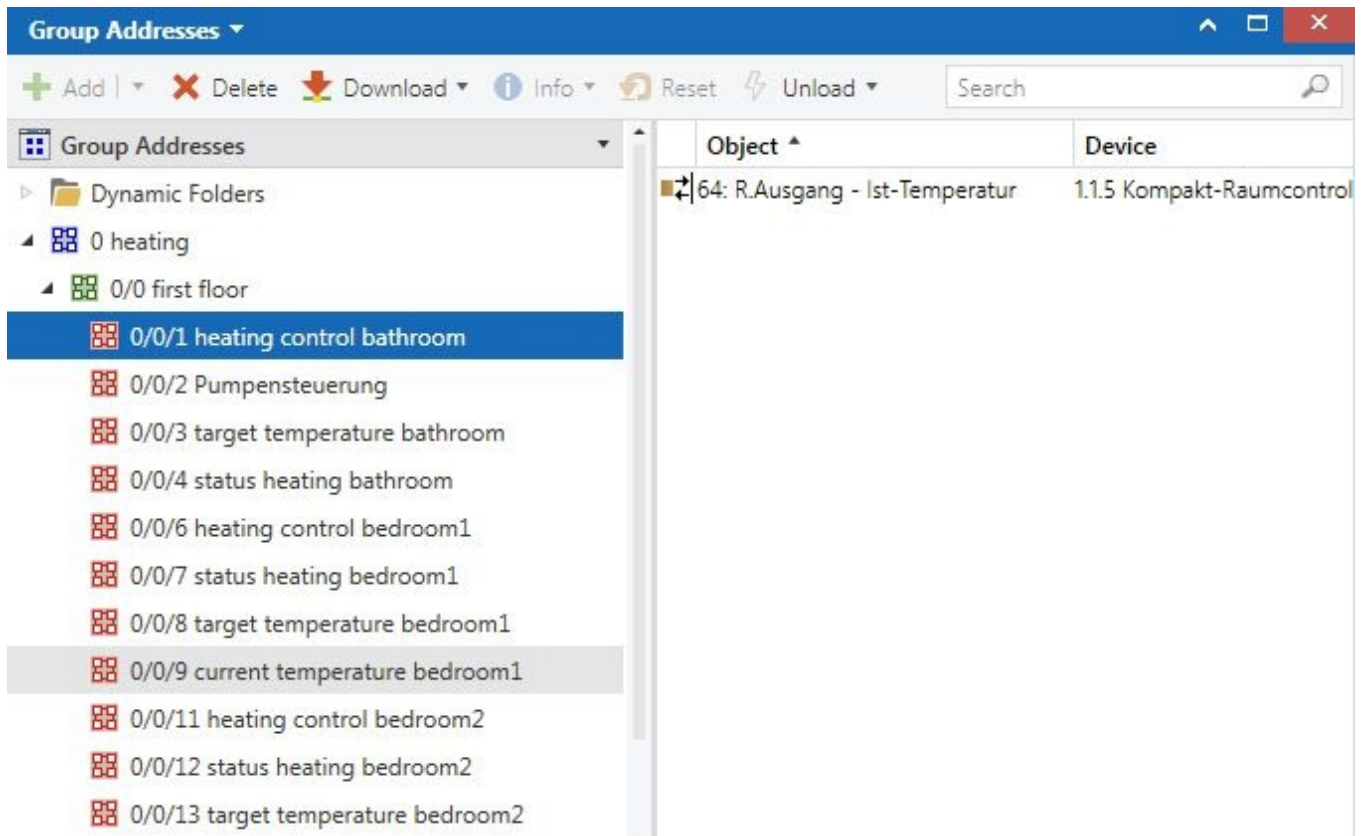


Figure 11.10 Creating Address Groups in ETS



*Figure 11.11 Configuring KNX Addresses in ETS*

Once we have configured all desired functions, we need to download the configuration from ETS to the individual switches, sensors and actuators. Via an IP Router or a KNX USB module we connect our PC to the KNX network. Then in the **Building** menu we select the first device we want to program, right click it and in the pop-up-menu select **Download All**. Now we need to physically go to the device we want to configure and press the program button. The ETS software will now start download to the target device. Once we are finished with all the devices of a given KNX address we can do the test and switch on our device.

Buildings

+ Add | - Delete | Download | Info | Reset

Search

Address	Room	Description	Application
1.1.-	KNX cabinet	KNX Power Supply	
1.1.0	KNX cabinet	IP Router	KNX / IP-Rc
1.1.1	KNX cabinet	HK 1	Schalten PV
1.1.2	KNX cabinet	HK 2	Schalten PV
1.1.3	corridor	Switch and dim livin...	Schalten, D
1.1.4	bathroom		Kompakt-R
1.1.5	bedroom1		Kompakt-R
1.1.6	bedroom2		Kompakt-R
1.1.7	corridor	Flurtemp	Kompakt-R
1.1.8	diningroom		Kompakt-R
1.1.9	familyroom		Kompakt-R
1.1.10	livingroom		Kompakt-R
1.1.11	KNX cabinet	DK 1	Universal D
1.1.12	KNX cabinet	DK 2	Universal D
1.1.13	livingroom	Next to bedroom1	Schalten, D
1.1.14	livingroom	Next to bedroom2	Schalten, D
1.1.15	bedroom2	Schalter Wintergarte...	Schalten, D
1.1.16	KNX cabinet	230V Switch	Schalt.Verke
1.1.17	corridor	Hauseingang Aussen...	Wertgeber,

first floor

- bathroom
  - 1.1.4 Kompakt-Raumcontroller-Modul
- bedroom1
  - 1.1.5 Kompakt-Raumcontroller-Modul
- bedroom2
- corridor
- diningroom
- familyroom
- kitchen
- livingroom

second floor

Trades

Devices | Parameter | Building Parts

Figure 11.12 Assigning devices to KNX addresses

## 11.12.1 Notes on Configuring KNX Devices

There are a couple of things which are important to know when configuring KNX devices:

- The parameters which are available for the configuration of KNX objects in the menu (room controllers, heating actuators etc.) often depend on the general setting for the object. For example the important parameter *dimming value status* for reading out the current setting of a dimmer is only available on a dimming actuator, if in *General Settings* the *status value object* is activated.
- Similarly for a heating actuator the control parameter expected by the heating actuator needs to be set in the general settings for the object (*1 Bit switching value* OR *1 Byte continuous control value*).
- Another example how tricky it sometimes is to configure complex devices is the Jung compact controller display. Depending on a button being configured as *push-button* or *rocker*, different configuration values are possible. For example important functions such as change of operating mode (*comfort, night, stand-by*) or change of *set point temperature* can only be assigned to a button, which is configured as *push-button*!
- An important configuration setting is the read flag. If you want to read out the value of a device, the reading flag has to be set. Often the reading flag is not set by default (Figure 11.13).

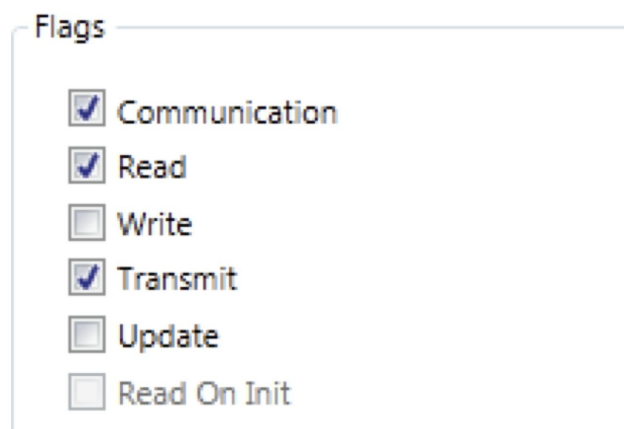


Figure 11.13: Setting the KNX Read Flag in order to be able to retrieve values

To summarize, especially with more complex devices containing multiple objects, one should carefully look at general settings and flags, watching out on possible impacts of their settings on secondary menu options or functionality. Also once done, taking notes and writing a brief documentation is not a bad idea for later reference.



# 12 KNX Control via OpenRemote Designer

When entering your KNX configuration into OpenRemote Designer the first step is to import the KNX Group Addresses. For this purpose we first create a CSV-export file from our ETS installation. We select the top folder of the Group Addresses in the ETS software, right-click and select *Export Group Addresses* in the pop-up-menu (Figure 12.1)

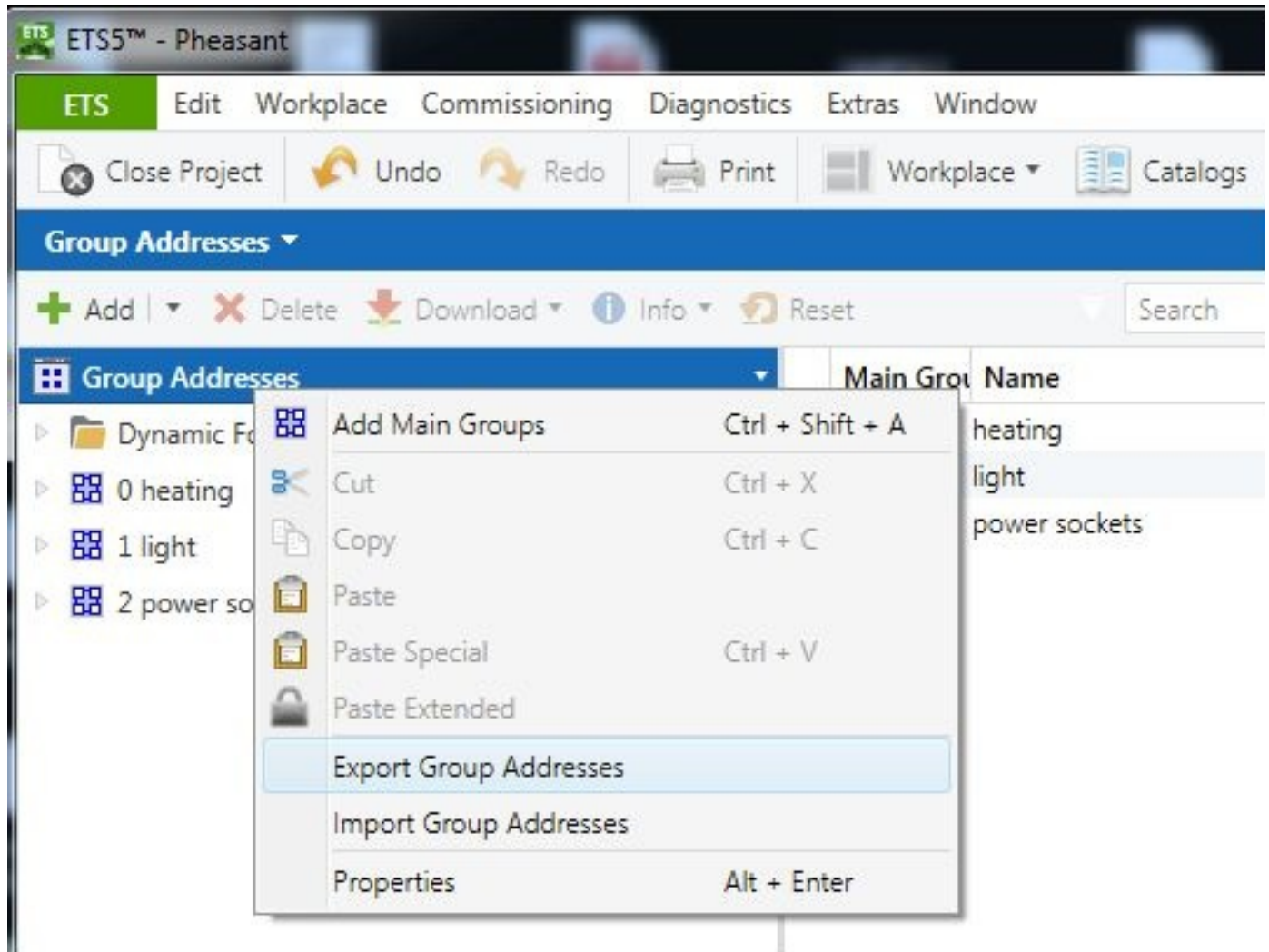
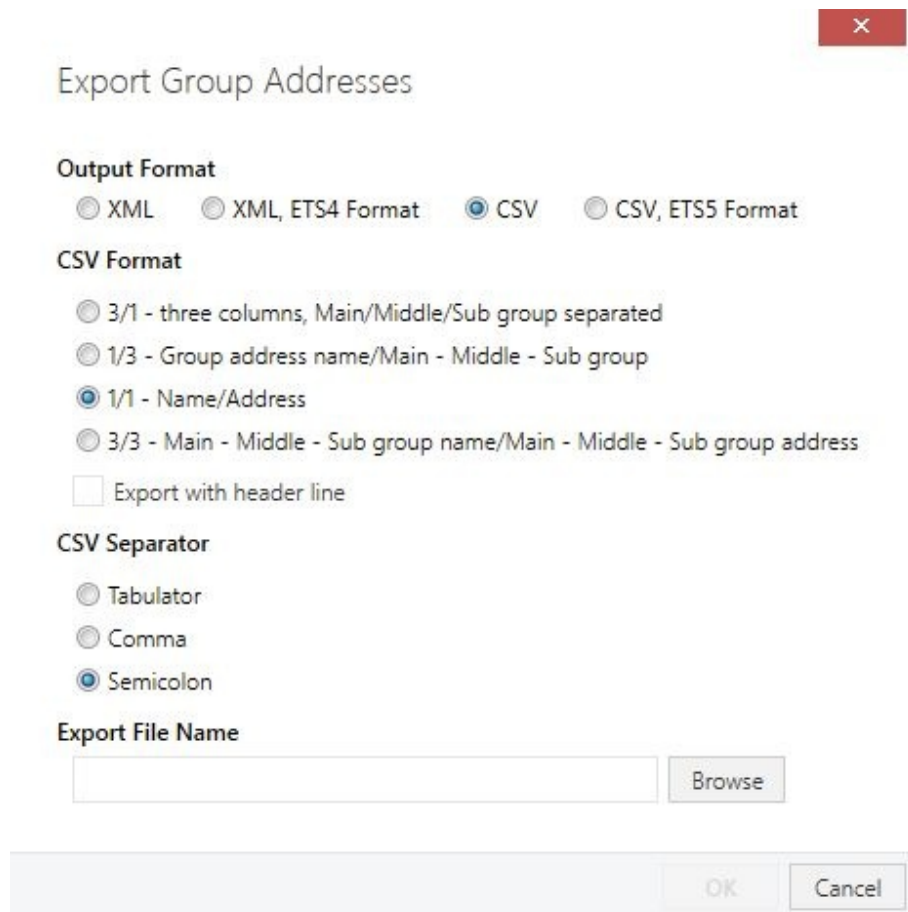


Figure 12.1 Export Group Addresses in ETS5





*Figure 12.2 Exporting Group Addresses in ETS5*

In the window which appears we select **CSV**, **1/1**, **Semicolon** and click **OK** (Figure 12.2). This will get us a simple CSV-file. Before importing it, we open the CSV-file with a text editor and make sure no other characters appear. Several KNX versions frame the information elements with quotes. If this is the case, we need to delete those until the entries look like the following:

- Switch light diningroom;1/0/16
- Value light diningroom;1/0/17
- Dim light diningroom;1/0/18
- Dim value light diningroom;1/0/19
- Switch light familyroom;1/0/20

In OpenRemote Designer as the first step we then create a new device selecting **New — New Device** and giving it a name, for example IP KNX Router. (The naming conventions for the devices are just of administrative nature, and do not have an impact on functionality). We mark our device and can now import our ETS Group Addresses by selecting **New — Import ETS data** and importing our CSV file as shown in Figures 12.3 and 12.4.



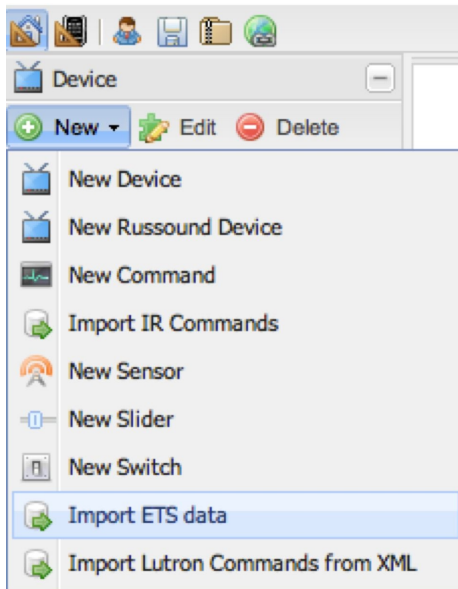


Figure 12.3 Importing ETS data to OpenRemote

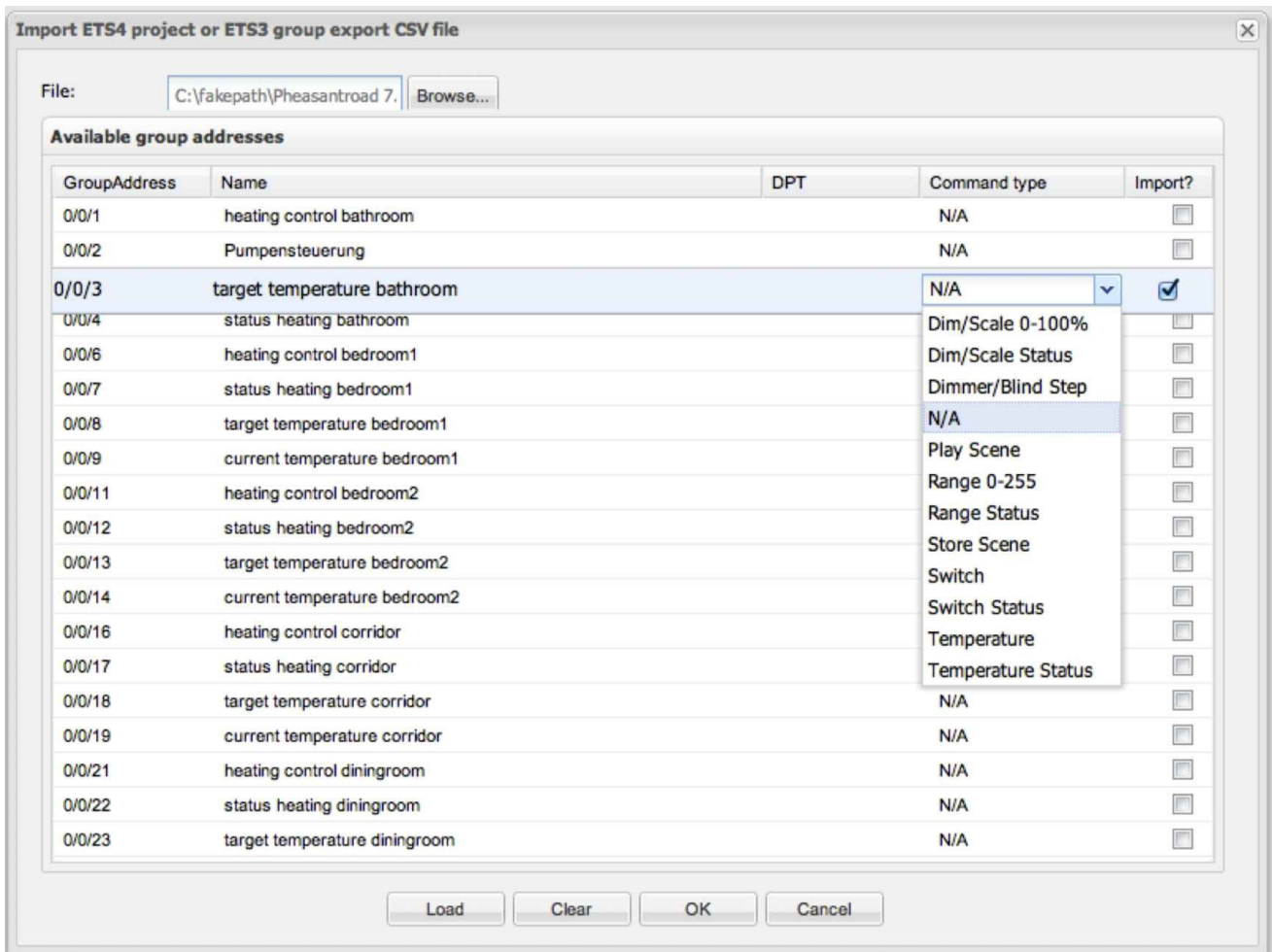


Figure 12.4 Open Designer Import Dialogue for ETS data

After selecting the *Load* command, the ETS data appears in the Open Remote Designer import window. We now need to select the addresses we want to import and assign a command type for each group address. Selecting the correct command type is essential for the functioning of our application. For Group Addresses, which just report data (temperature, range of values, state of a dimmer or a switch) we need to select the appropriate *Status* command type (Temperature Status, Switch Status, etc.). For Group

Addresses, which actually initiate an action (e.g. activate a switch, change the dim value or temperature) we need to select the according command type (e.g. Temperature, Switch or Dim/Scale 0-100%). Once we are done we hit OK and your selected KNX commands appears on the left side of our Open Remote configuration screen with the correct KNX command and the correct KNX Data Point Type (DPT) selected. (Figure 12.5)

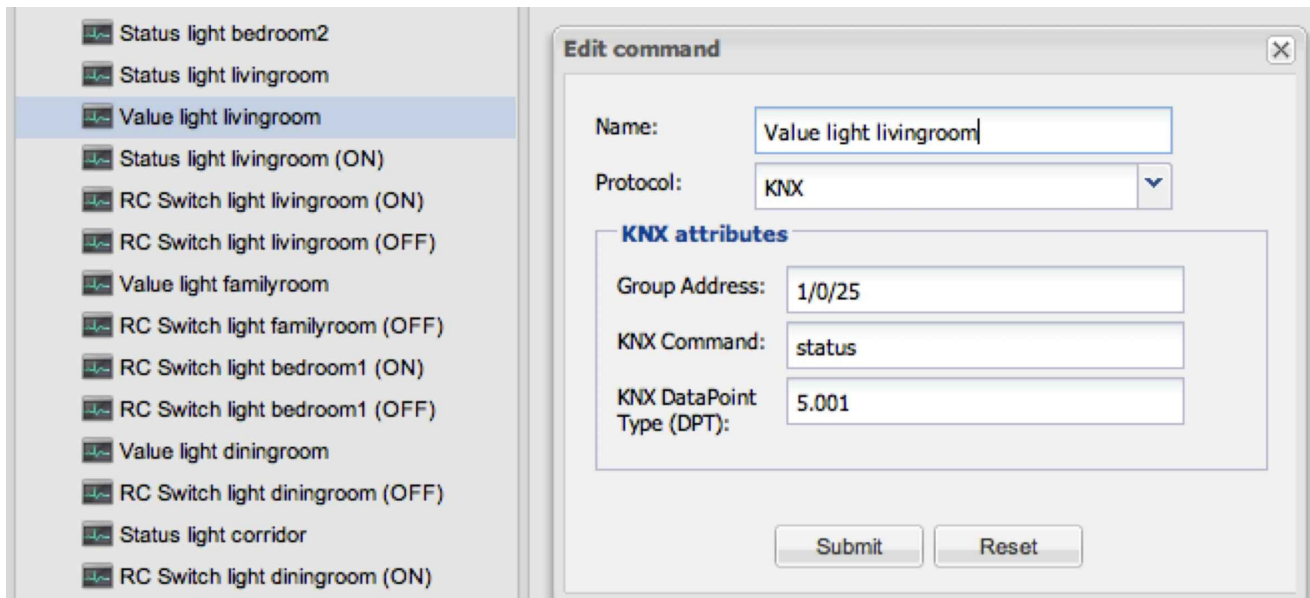


Figure 12.5 Configuring a dim status command: Report the light value from the living room

Alternatively we can also manually add KNX commands by selecting *New — New Command* or edit imported data records. For a functioning configuration it is essential to configure the correct DPTs and commands for each entry. Table 12.1 shows a listing with some important DPTs, Figure 12.6 the command editing dialogue in OpenRemote Designer.

DPT ID	Format	DPT Name
1.001	B <sub>1</sub>	DPT_Switch
2.001	B <sub>2</sub>	DPT_Switch_Control
5.001	U <sub>8</sub>	DPT_Scaling
6.001	V <sub>8</sub>	DPT_Percent
7.001	U <sub>16</sub>	DPT_Value_2_U_Count
8.001	U <sub>16</sub>	DPT_Value_2_Count
9.001	F <sub>16</sub>	DPT_Value_Temp
10.001		DPT_TimeOfDay
11.001		DPT_Date

Table 12.1 Select KNX Data Point Types

B<sub>1</sub> one bit Boolean  
 B<sub>2</sub> two bit Boolean  
 U<sub>8</sub> unsigned integer 8 bit  
 U<sub>16</sub> unsigned integer 16 bit  
 F<sub>16</sub> floating point 16 bit

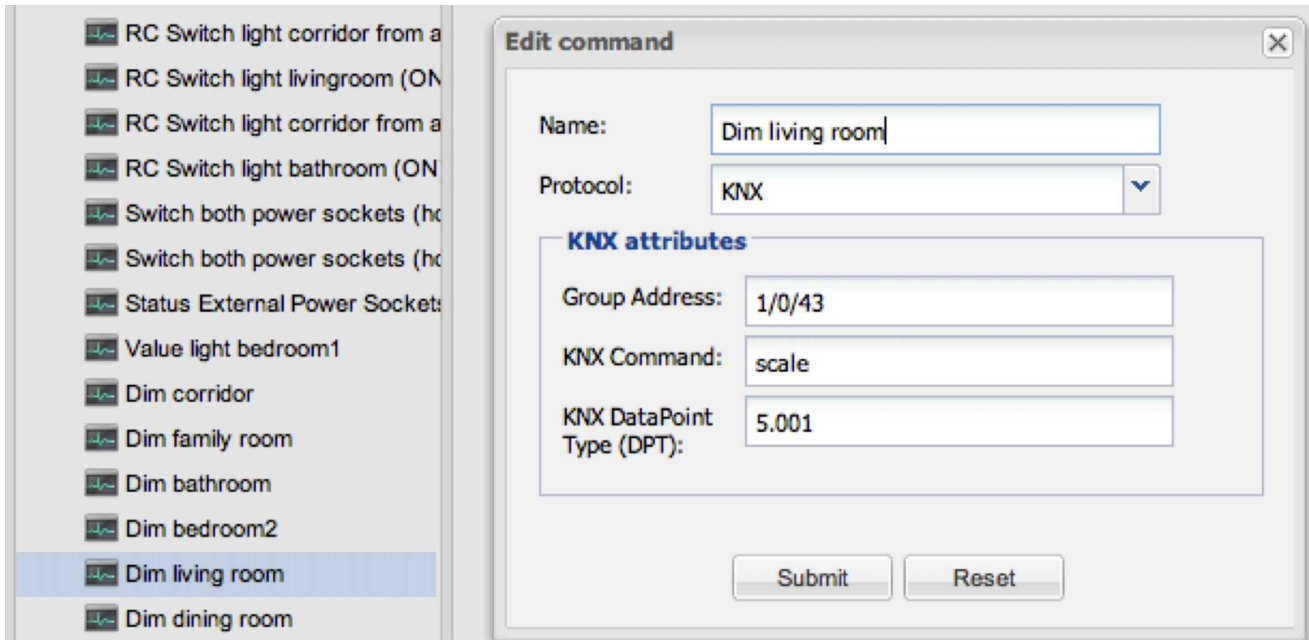


Figure 12.6 Configuring the dim command using “scale”: Dim living room light

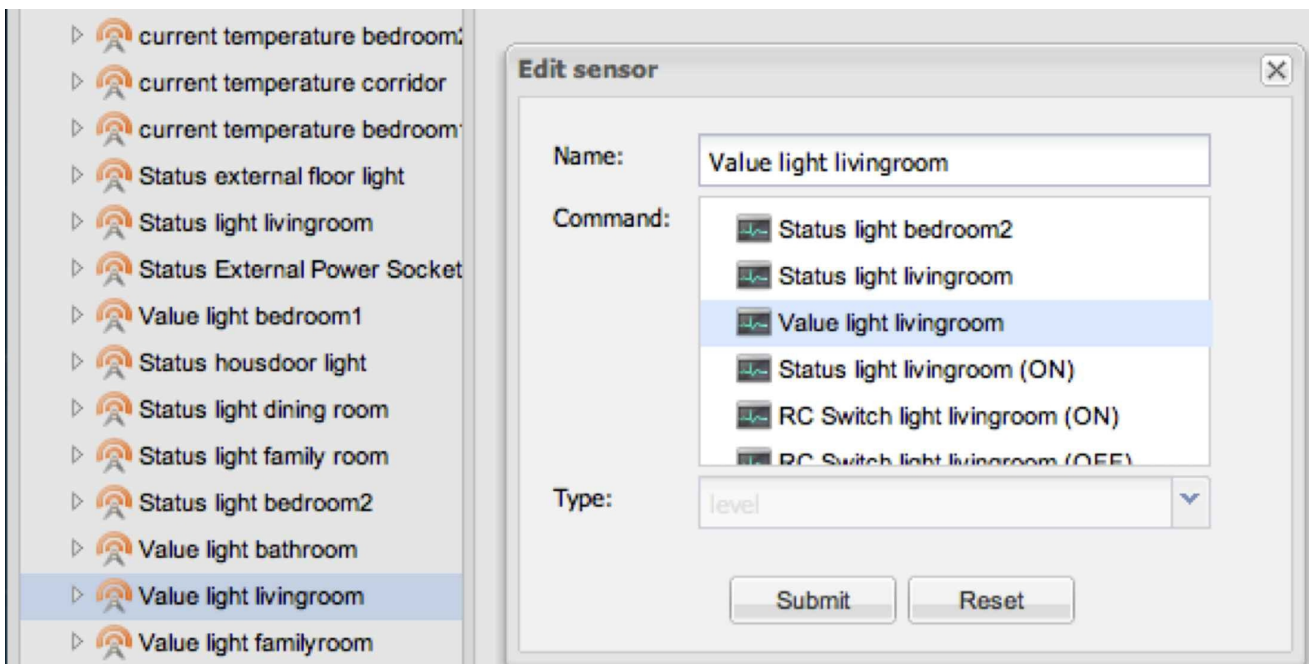


Figure 12.7 Configuring a sensor: Dim sensor living room

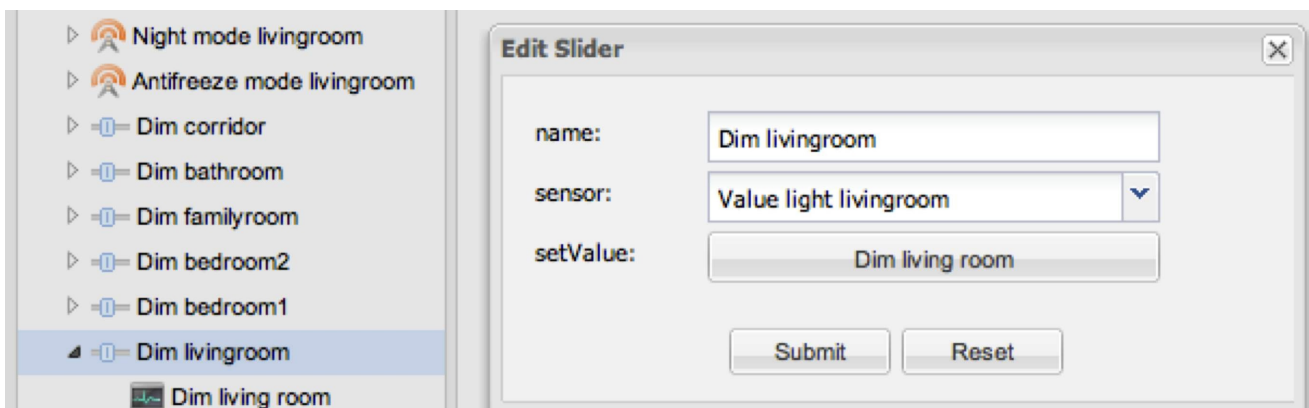
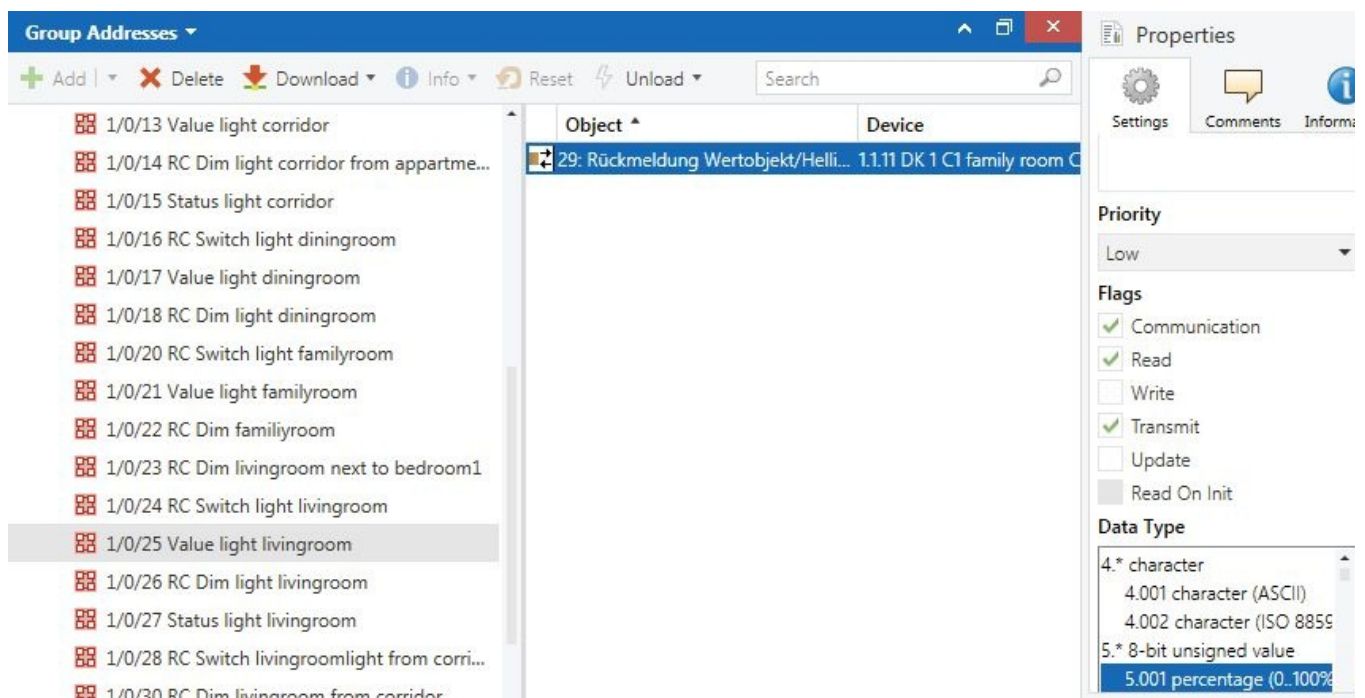


Figure 12.8 Configuring a slider: Dim slider living room

Once we are done with the commands, we need to configure sensors for each of our functions (switches, sliders, value displays). We select *New — New Sensor* and select the according status command for it. The final step is then to configure switches and sliders. For a slider to dim light we for example select *New — New Slider* and then add both - the according action command as well as the according sensor containing the light brightness value.

In our example we want to configure a slider to dim the light in the living room. We first need the status command reporting the current value of the light in the living room (Figure 12.5). Second we need the command which actually dims the light. The according scale command is shown in figure 12.6. Third we configure the sensor for our dim slider (Figure 12.7), which uses the status command from figure 12.5. And finally we configure the slider itself (Figure 12.8), consisting of the sensor and the scale command “Dim living room”.

To clarify the KNX ETS side of things at this point as well: The dim status KNX Group Address (in our example 1/0/25, Figure 12.9) in the ETS software links to the *Status feedback value object/brightness* of the KNX dim actor unit, and has also, as you can see in figure 12.5, a DPT of 5.001. Similarly the dim command through KNX Group Address 1/0/43 (Figure 12.6) links on the KNX side to the *General value object* of the dim actor, which allows the setting of the dim level.



*Figure 12.9 KNX Group Address and linkage to the dim actor value brightness object in ETS5*

In the same fashion as explained above we now configure switches or simple sensors for displaying status information. We can now move on and design the OpenRemote GUI for our controls as shown in previous chapters.



## 12.1 Background Pictures for the Smartphone and Tablet App

Besides defining your own GIFs for buttons, sliders and other control element, you can also upload your custom background picture for your OpenRemote based smartphone or tablet app. The background picture in most cases probably needs to be darkened using photo editing software. Figure 12.10 shows an example background screen on an iPhone. If you desire you can even position the grid for the light switch above the position of the lamp. You then take two pictures - one with the lamp switched on and one with the lamp switched off. You assign the two pictures to the switch configuration and can now tab on the lamp in the picture. The lamp will go on, while the picture in the screen changes accordingly.

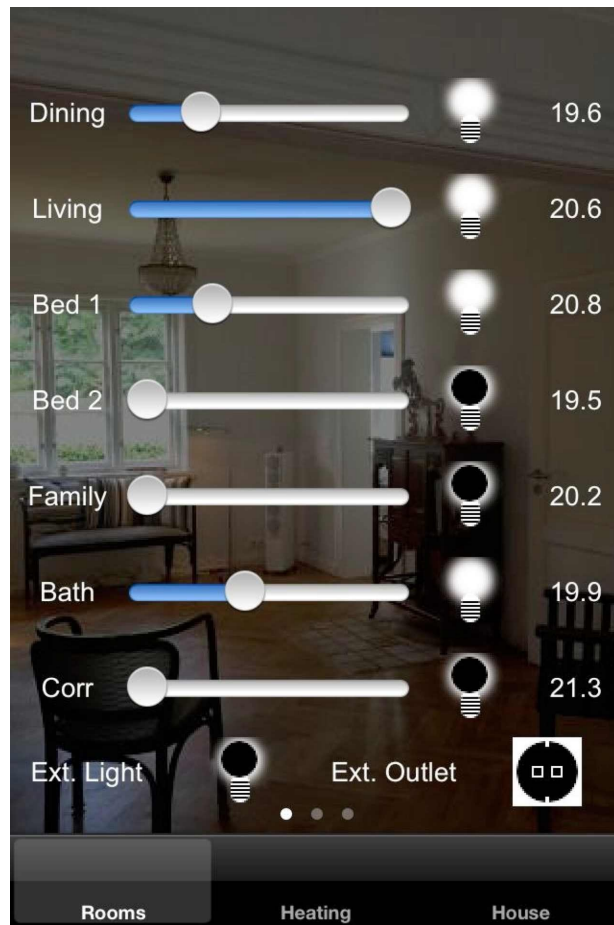


Figure 12.10 OpenRemote Based iPhone app with custom background screen

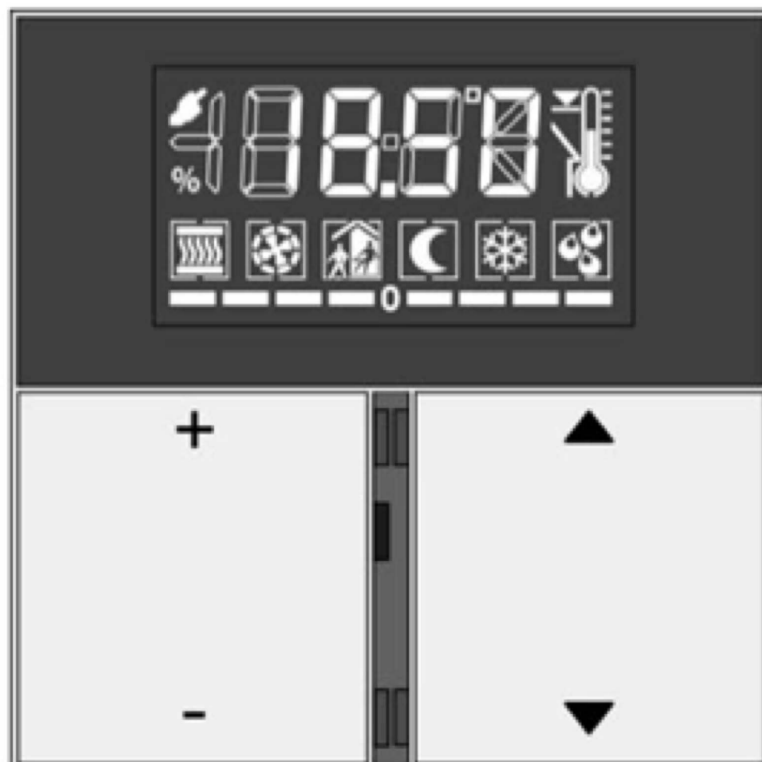
## 12.2 Configure KNX Based Heating Mode Control

As the final example for OpenRemote to KNX interaction we will implement the classic time triggered operating mode change for a heating system. Typically heating systems are configured in operating states such as

comfort	comfort temperature for rooms during the day
night	lower temperature during the night
antifreeze	to keep the house well above the freezing point during longer periods of absence
standby	below night temperature level for shorter periods of absence

Traditional heating control systems are often complex to handle. They execute a fixed schedule, set often years ago, when the system was installed. As a result the systems operate very inefficient, more or less dependent on how often the owner is ready to make an adjustment. At the same time heating (or cooling) in most regions of the world represents the by far largest portion of household energy consumption. Thus integrating heating management into the smart building control system will in most cases provide a significantly potential for energy saving.

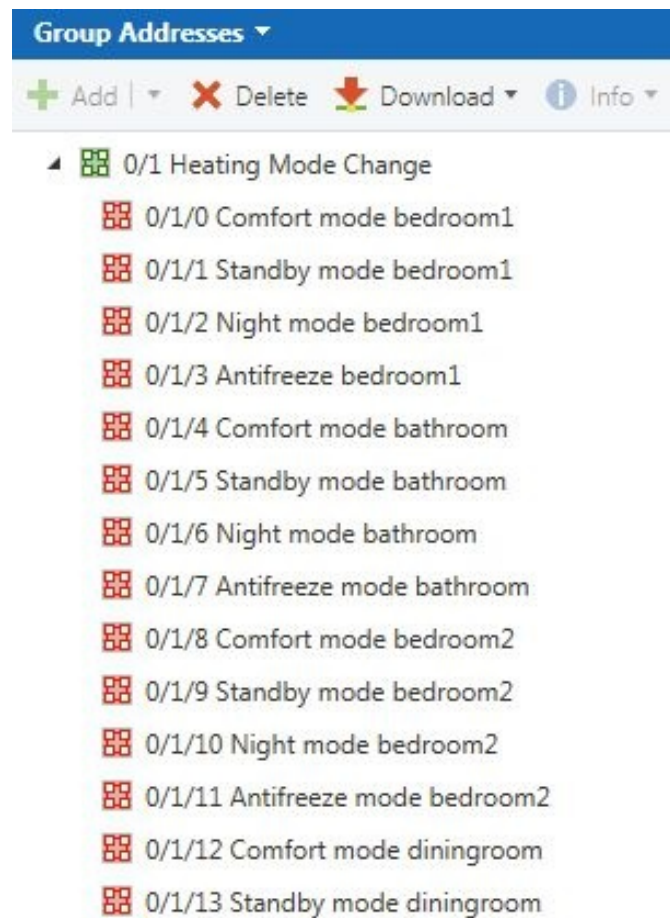
In our example we will control Jung room controller units, which contain a thermostat, reporting the room temperature, and a management unit, which controls the heating actor, dim lights and control blends. It is one of the more complex units with a wealth of functionality (Figure 12.11).



*Figure 12.11 The Jung compact room control unit*

Besides configuring the manual switches on the room control unit itself, the heating operating state can be configured by either by sending a one byte control value or by setting a combination of four one bit switches, which we will use for our heating control

function. In the ETS configuration menu for the room controller we select the four bit control mode, which activates four 1 Bit KNX objects of the type DPT 1.001. We now can link each of the four objects to a KNX Group Address which allows us to access them through OpenRemote (Figure 12.12).



*Figure 12.12 KNX Group Address linkage to the four switch objects heating operating mode change*

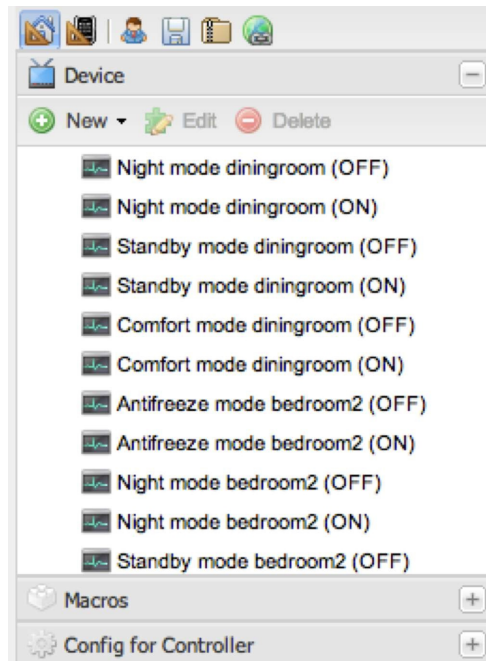
In order to switch to a particular operating state, the four switch objects have to be set as outlined in the matrix in Table 12.2:

<i>Antifreeze</i>	<i>Comfort</i>	<i>Standby</i>	<i>Night</i>	
<i>1</i>	<i>X</i>	<i>X</i>	<i>X</i>	<i>Antifreeze</i>
<i>0</i>	<i>1</i>	<i>X</i>	<i>X</i>	<i>Comfort</i>
<i>0</i>	<i>0</i>	<i>1</i>	<i>X</i>	<i>Standby</i>
<i>0</i>	<i>0</i>	<i>0</i>	<i>1</i>	<i>Night</i>

*Table 12.2 Control matrix for Jung compact room control unit*

Thus for example, in order to switch the heating status to night, we need to set four switches (Antifreeze to 0, Comfort to 0, Standby to 0 and Night to 1). For the other states we just need to set three, two or one switches respectively. (X stands for don't care). We import the newly defined KNX Group Addresses into OpenRemote, assigning the command types "Switch" to each group address. OpenRemote imports each group address

automatically into two commands, one extended by (ON) and one by (OFF) (Figure 12.13).



*Figure 12.13 Heating state commands after import in OpenRemote*

We will now first configure our OpenRemote control panel so we can change the operating mode for each room by pressing a single button. Once the manual control via OpenRemote is working, it will be easy to implement Drools rules, which manage the heating system according to our needs.



## 12.3 Smartphone Based Heating Control

Since we need to set up two four single commands in order to change the heating state for a room we will use the OpenRemote macro function for implementation. For each room state we will create an according macro. In the Building Modeler we expand the *Macro* menu and select *New*. We give the macro a name (e.g. *bedroom2*) and simply drag the commands from the left side of the macro window to the right side (Figure 12.14).

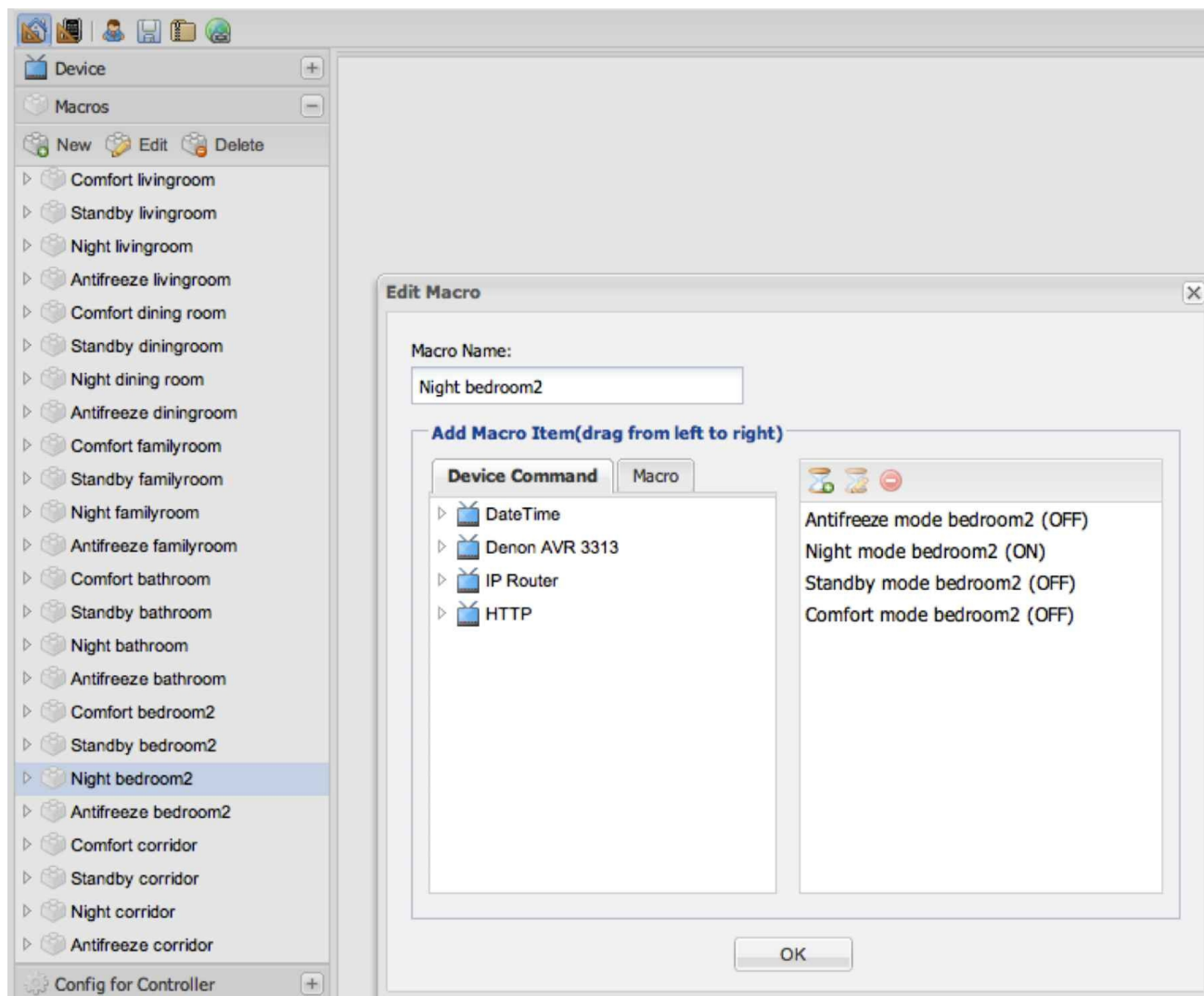


Figure 12.14 Macro for switching bedroom two room controller to night mode

We do this for all rooms and all operating states and can now assign the macros to push buttons on our control panel screen. In addition to the push buttons we also want to display the current and the target temperature on our smartphone app. Both - current and target temperature - are also KNX read objects, which our Jung room controller provides with a DPT of 9.001. We link KNX Group Addresses to each current and target temperature object in ETS and import them assigning the command type “temperature status” in the OpenRemote import dialogue. OpenRemote automatically creates the associated commands as well as sensors, so all which is left to do is configure the screen on our panel (Figure 12.15).

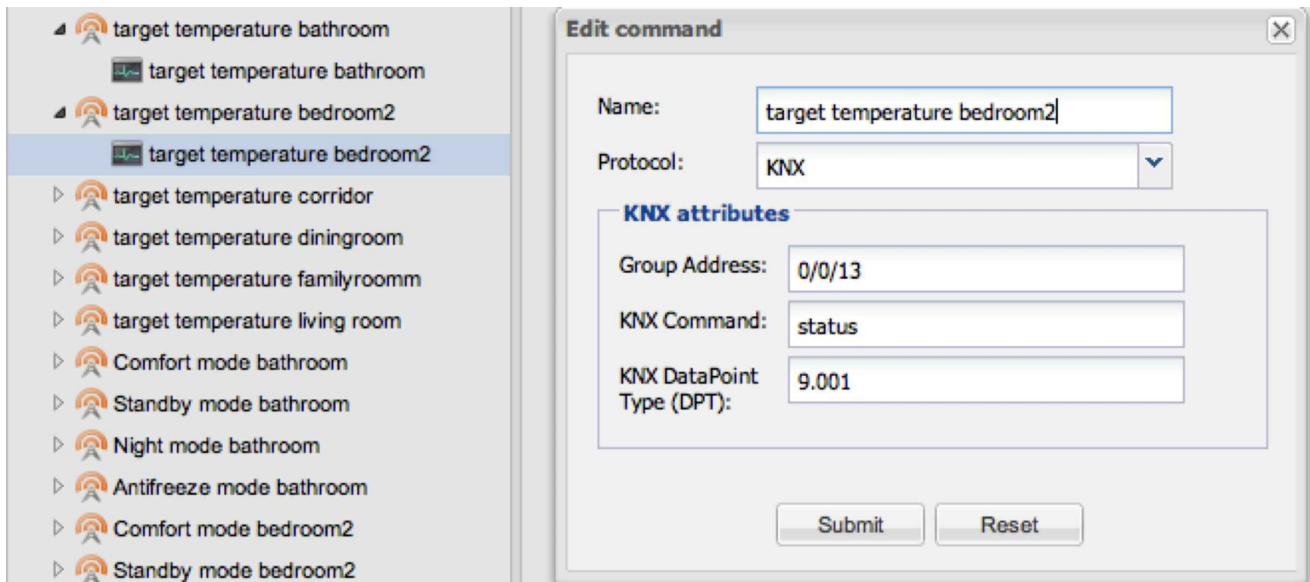


Figure 12.15 Sensors for displaying target and current room temperature

After setting up the layout using several grids we drag four *Button* widgets for each room into the grid and assign the according macros. For the display of the current and the target temperature we use the *Label* widget. We use two for each room and link them with the current and target temperature sensors respectively. We save the design, synchronize our controller and test the functionality on our smartphone or tablet (Figure 12.16). To round off our design we create two symbol images (one *button image* and one *pressed button image*) for each heating state, choose a background picture and are done (Figure 12.17).

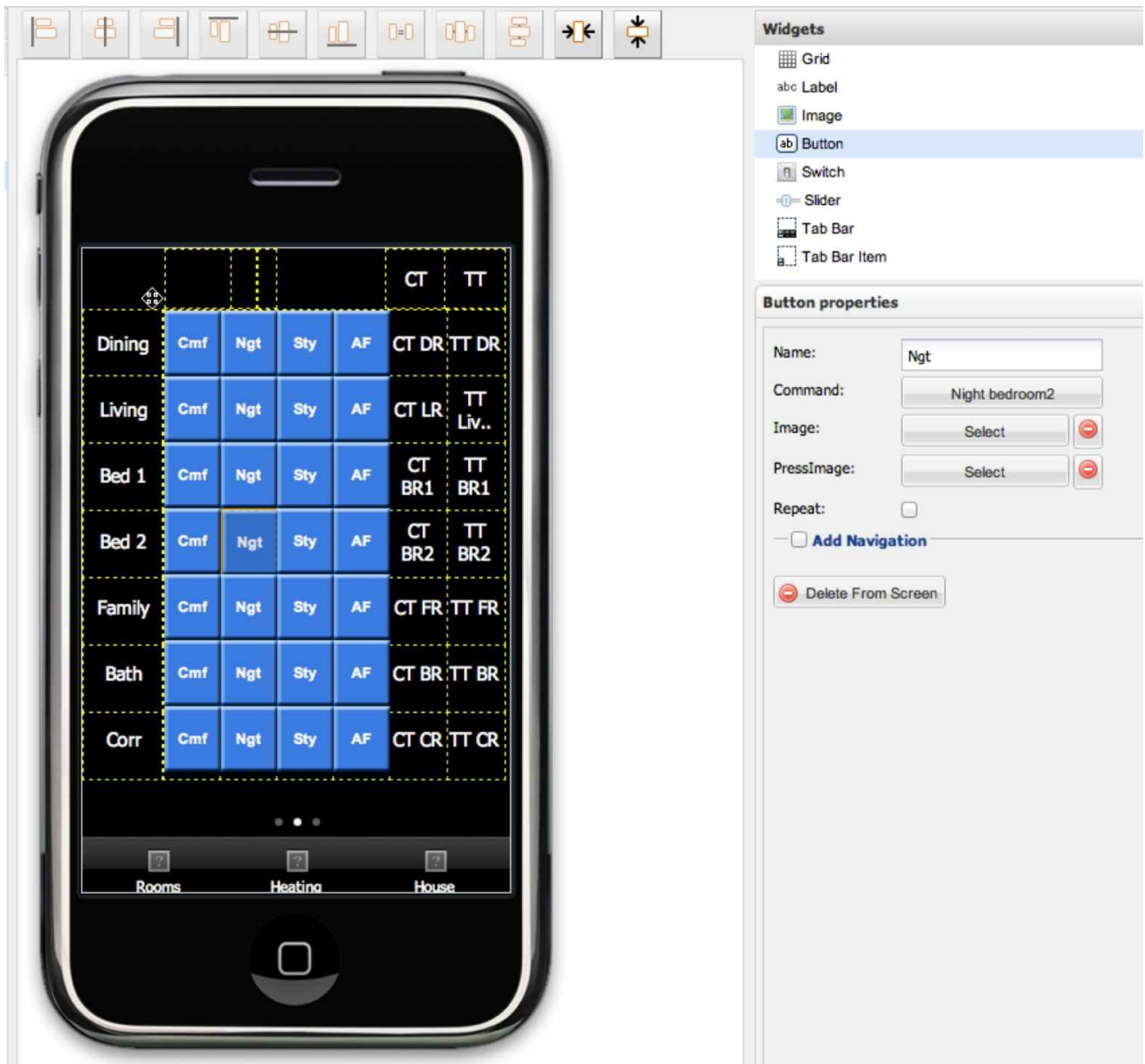


Figure 12.16 Configuration of the smartphone screen for our heating management

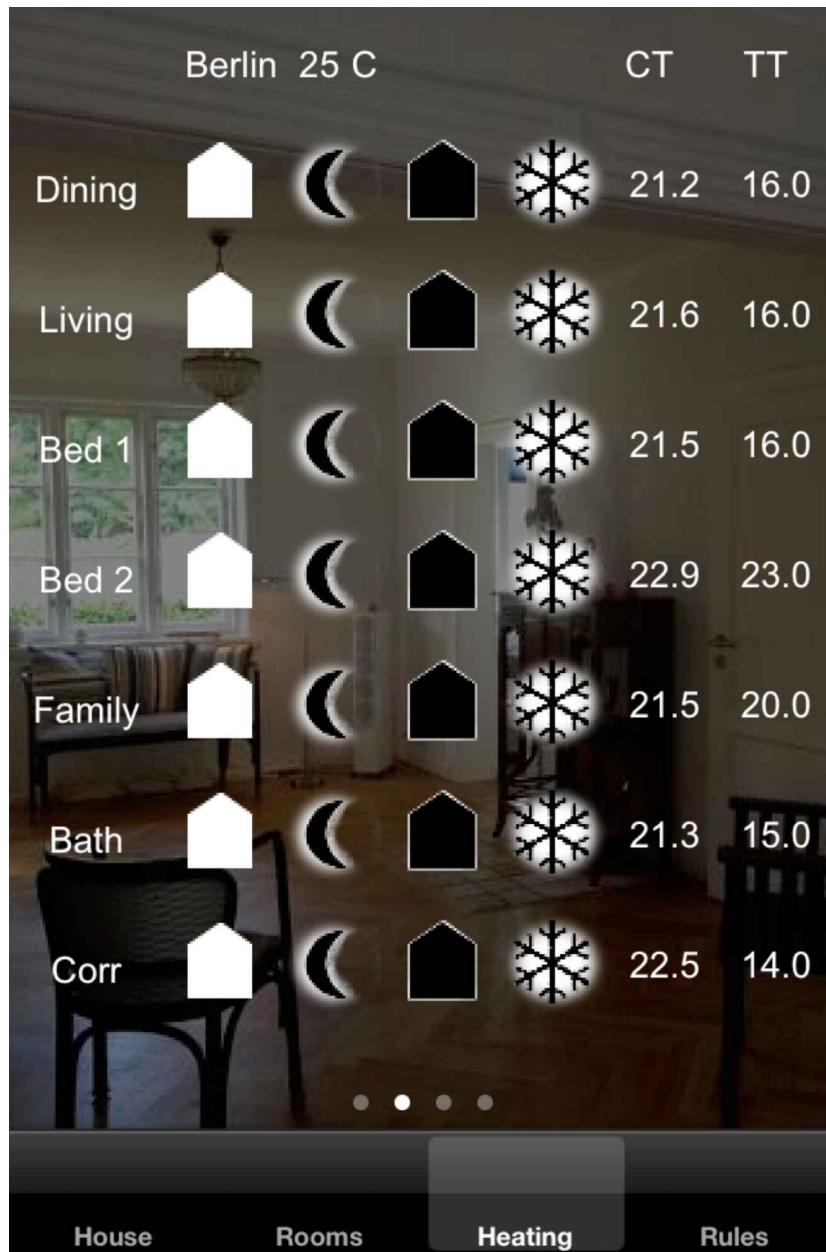


Figure 12.17 OpenRemote smartphone app for heating management

## 12.4 Drools Based Heating Automation

Finally we want to automate the change of heating operating modes based on rules. Initially we will simply switch to comfort mode at 5:30 a.m. As in our previous example with the time controlled start of iTunes we use the rules attribute `timer` and specify a schedule of Monday through Friday together in the macro. Here we need to list them individually in the rules statement, since it is not possible to use macro statements in rules. In addition we add a logging function to our rule. So after every execution of our command we want a brief status report along with date and time information: at 5:30 a.m.:

```
timer (cron:0 30 5 ? * MON-FRI)
```

For switching to comfort mode we need to execute two switch commands (antifreeze (OFF) and comfort (ON)). For the panel based control we put these commands

```
System.out.println("!!!bedroom1 switched to comfort!!!");
```

```
Date date1 = new Date();
```

```
System.out.println(date1.toString());
```

With that the rules code for switching bedroom1 and bedroom2 to comfort Monday through Friday at 5:30 a.m. (including package definitions) would look like follows:

```
package org.openremote.controller.protocol
```

```
global org.openremote.controller.statuscache.CommandFacade execute;
```

```
global org.openremote.controller.statuscache.SwitchFacade switches;
```

```
global org.openremote.controller.statuscache.LevelFacade levels;
```

```
import org.openremote.controller.protocol.*;
```

```
import org.openremote.controller.model.event.*;
```

```
import java.sql.Timestamp;
```

```
import java.util.Date;
```

```
rule "heating management switch to comfort at 5:30 a.m."
```

```
timer (cron:0 30 5 ? * MON-FRI)
```

```
when
```

```
eval(true)
```

```
then
```

```
execute.command("Antifreeze bedroom1 (OFF)");
```

```
execute.command("Comfort mode bedroom1 (ON)");
```

```
System.out.println("!!!bedroom1 switched to comfort!!!");
```

```
Date date1 = new Date();
```

```
System.out.println(date1.toString());
```

```
execute.command("Antifreeze bedroom2 (OFF)");
```

```
execute.command("Comfort mode bedroom2 (ON)");
```

```
System.out.println("!!!bedroom2 switched to comfort!!!");
```

```
Date date2 = new Date();
```

```
System.out.println(date2.toString());
```

end

In the controller terminal window you will be able to monitor the activity as reported through our logging function.



# 13 Remote Smarthome Control

A key element for a smart home is the capability to access all control functions from remote via the Internet. This allows the realization of various important use cases such as

- control and monitoring of the vacant home (temperature, energy, gas, water, smoke, wind)
- feeding and watching pets
- watering plants indoors and outdoors
- checking on elderly and handicapped people to ensure they are safe and well.

From a technical perspective, the above functions require the ability to access the home Wi-Fi network from a smartphone, tablet or notebook via the Internet. In order to do this, we will need to do two things:

- configure our Internet/DSL router to run a Dynamic DNS service (DDNS)
- set up a Virtual Private Network (VPN) connection between the mobile device we plan to use for accessing our smart home Wi-Fi network and our Internet/DSL router at home.



## 13.1 Configuring a Dynamic DNS Service

The Dynamic DNS service resolves the problem that the IP address for your Internet/DSL router is dynamically assigned by your Internet service provider (ISP) and typically changes every 24 hours or every time you reboot your router. (This assumes that you do not have Internet access via a permanently assigned, fixed IP address, since this is the case for most residential homes.) Working with a Dynamic DNS service your router periodically announces its current IP address to the DynDNS server, where it is associated with a domain name you choose at registration. This allows you to access your router any time using that domain name. There are a number of free and pay DDNS service companies. A good listing can be found under

<http://dnslookup.me/dynamic-dns/>

So all you need to do is to select a DDNS service, and register. At registration you select a domain name, a user name and a password. You then log into your Internet/DSL router and look for a menu item called *remote access / Dynamic DNS* or something similar. You activate this function and enter your DDNS provider name as well as domain name, user name, and the password of your account.

## 13.2 Configuring a VPN

The second step is to configure a VPN connection between the mobile device that you plan to use for remote smart home access and your home network. A VPN is basically a secure (encrypted) point-to-point connection between two networks or computers. On each end point of the connection, a VPN software agent needs to be installed. Both VPN agents need to be configured, among other things, with a so called “shared secret”, which is the key for the encrypted network connection. It allows the two VPN agents to establish a connection (sometimes referred to as a VPN tunnel) with each other. In our case (as with most residential homes), one end of the VPN connection will be the DSL/Internet router of the smart home, the other end a smartphone, tablet or notebook.

It is best to start with your DSL/Internet router and check the manual or the support web site for how to set up a VPN connection. Most vendors provide a small utility that automatically creates the necessary configuration files for the router as well as for the second VPN end point. This is necessary, since you cannot manually generate the encryption keys for the shared secret. For the configuration utility, you will need to have the following information at hand:

Dynamic DNS (DDNS) domain name

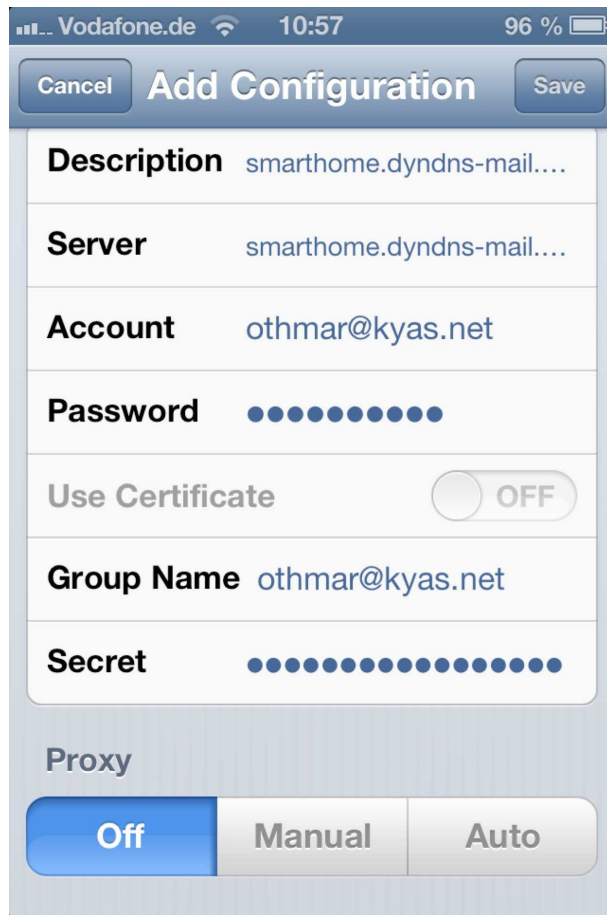
DDNS user name

DDNS password

IP address of your DSL/Internet router (the one inside your Wi-Fi network)

Subnet address mask of your DSL/Internet router

The utility will then typically generate two configuration files. One is for the DSL/Internet router, which you will use when activating its VPN capability, and a second one is for your smartphone, PC or Mac. As an example, on an iPhone you go to *Settings — General — VPN — Add VPN Configuration*. You then select the communication protocol (typically IPSec) and enter the DDNS credentials along with the VPN shared secret you have received from the VPN utility (Figure 13.1).



*Figure 13.1 VPN Configuration on an iPhone*

When you now activate the VPN connection from your smartphone (while it is connected to the Internet, e.g. via 2G or 3G), a secure VPN connection will be established to your Internet/DSL Router. Now from the perspective of the apps on your smartphone or tablet you are connected to your home Wi-Fi network, as if you were at home. You can now start your OpenRemote app and have full access to your smart home control functions.



*Figure 13.2 Activating VPN on an iPhone*



# 14 Cold Start: Launch Automation

In the case of the smart home controller rebooting (due to a power outage or due to maintenance downtime), we need to ensure to have a running system in a defined operating state in place once the system is back up. For this purpose we will prepare our computer system for an automatic restart of the OpenRemote controller after a system shutdown. Since such functionality is highly operating system specific, its implementation under OS X /Linux and Windows are very different. For OS X we will use `launchd`, for Windows Task Scheduler.

## 14.1 Windows Task Scheduler

Under Windows scheduling a task to be executed at startup is quite straight forward. We open Task Scheduler by selecting *Start — Control Panel — System and Security — Administrative Tools — Task Scheduler*. Next we click on the *Action* menu and select *Create Basic Task*. We give the task a name, select *Next*, and then *When the computer starts*. Now we select *Start a program — Next*, browse to `openremote.bat`, enter `run` in the *Add arguments* field, and select *Next* again, to finish the creation of the task (Figure 14.1).

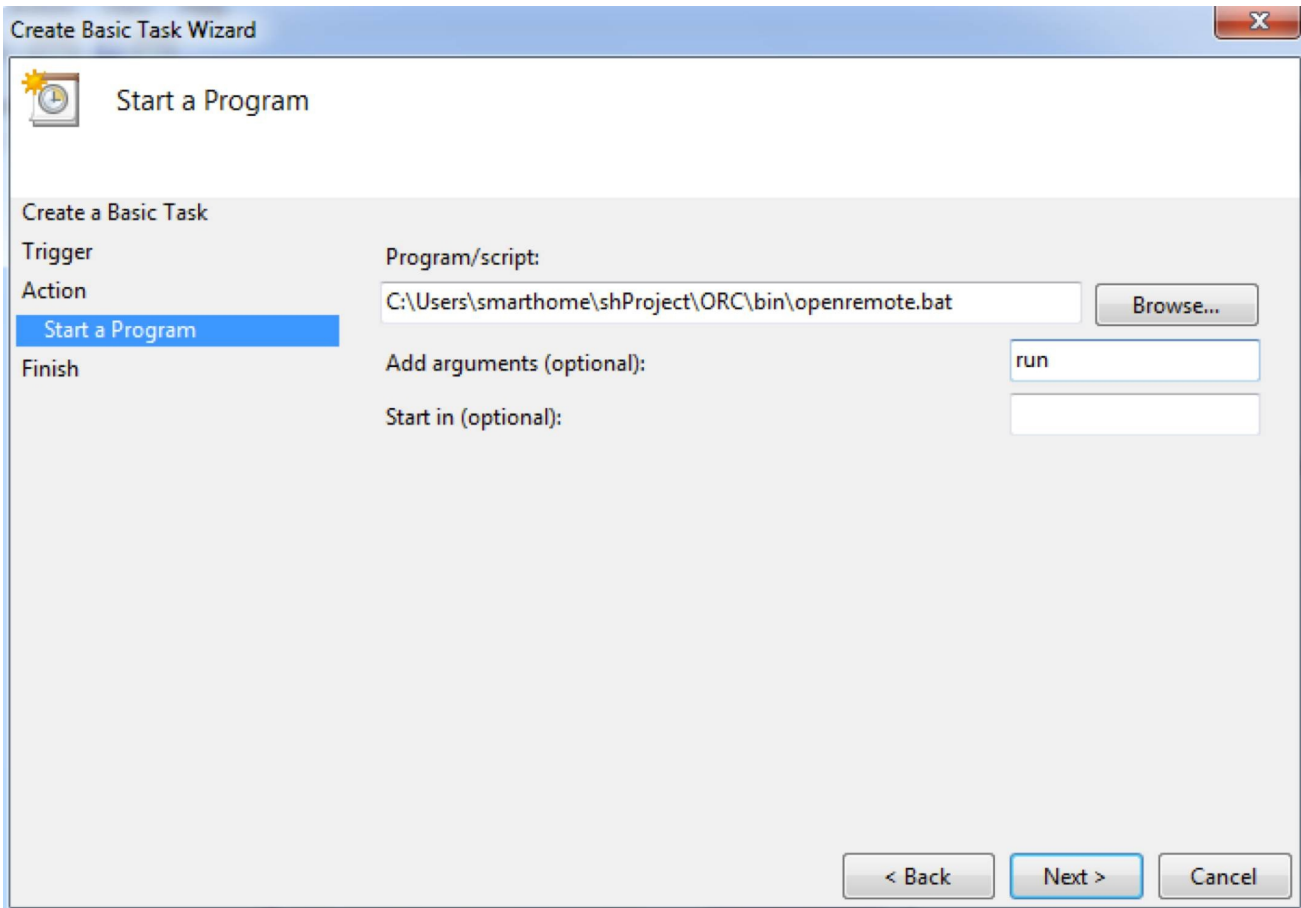


Figure 14.1 Windows Task Scheduler configuration

## 14.1.1 Sending a Reboot Notification Email

In addition to automatically starting OpenRemote we want our smart home controller to send out a reboot notification email. While Windows Task Manager can be configured to send an email at system startup, we want to be more flexible. For example we might want our OpenRemote application to trigger sending an email under certain circumstances as well. Therefore we will use the command line email application *mailsend*, which can be downloaded for free from:

<https://github.com/muquit/mailsend/releases/>

The *mailsend* documentation can be found under

<https://code.google.com/p/mailsend/wiki/mailsendFAQ>

In order to send an email with *mailsend* we open Windows Terminal, change to the *mailsend* directory and type `mailsend.exe` followed by the specific options for our mail server. Most mailservers today require authentication via user name and password and encryption using STARTTLS or SSL. If you are not sure about the capabilities of your SMTP mailserver type the below command (the example for Gmail):

```
mailsend.exe -info -port 587 -smtp smtp.gmail.com
```

Now you should be able to construct the complete command for sending an email. Below the command for sending an email using the Gmail SMTP server with authentication and STARTTLS encryption:

```
mailsend.exe -to ok@keyconceptpress.com -from info@smarthomeserver.com -starttls -port 587 -auth -smtp smtp.gmail.de -sub "Open Remote Server Notification" +cc +bc -v -user info@smarthomeserver.com -pass "yourpassword" -M "Open Remote Server restarted"
```

The *mailsend* options we use in the above example are listed below. A complete listing of all options can be found under

<https://github.com/muquit/mailsend/blob/master/README.mediawiki>.

Option	Description
-smtp	Hostname/IP address of the SMTP server
-to	email address/es of the recipient/s
-from	email address of the sender
-ssl	SMTP over SSL
-starttls	Check for STARTTLS and if server supports, do it
-auth	authenticating trying: CRAM-MD5,LOGIN,PLAIN in that order
-user	username for authentication
-pass	password for ESMTP authentication
-M	Content line

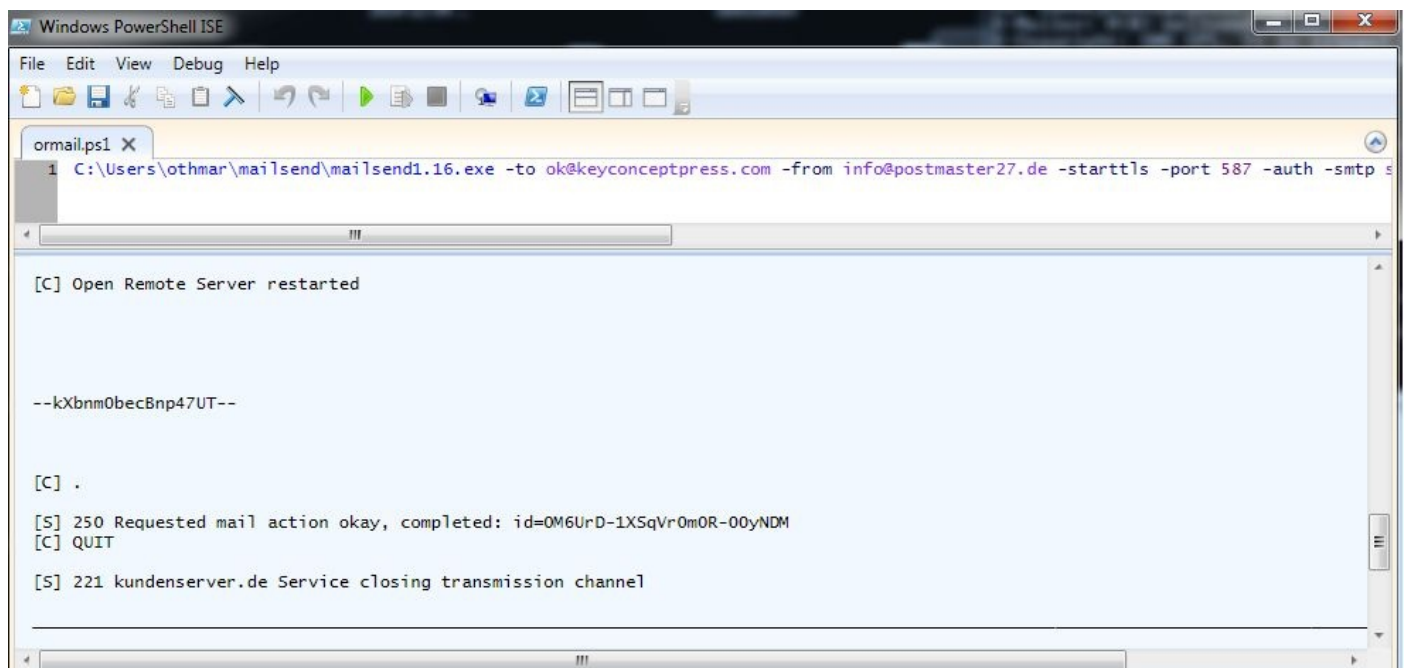
*Table 14.1 Essential options for the command line email tool mailsend*

Once we have validated our email command in Windows Terminal, we open the Windows



PowerShell editor, paste in the command (along with the complete path to our mailsend directory) and store it under something like *ormail.ps1* (Figure 14.2). Again we test our work by executing the script from within the PowerShell editor.

As the last step we now need to configure Task Scheduler to run *ormail.ps1* at startup. We select *Start — Control Panel — System and Security — Administrative Tools — Task Scheduler*. Then we click on the *Action* menu and select *Create Basic Task*. We give the task a name like *OR Reboot Notification*, select *Next*, and then *When the computer starts*. Now we select *Start a program* and enter *powershell.exe* in the *program/script* field and *C:\Users\smarthome\mailsend\ormail.ps1* in the *Add arguments* field (Figure 14.3). Of course we can now also add the *openremote.bat* run command to our Powershellscript *ormail.ps1* and only schedule a single task for startup. Finally, with a few reboots of our system we validate the functionality of our auto reboot and notification email capabilities.



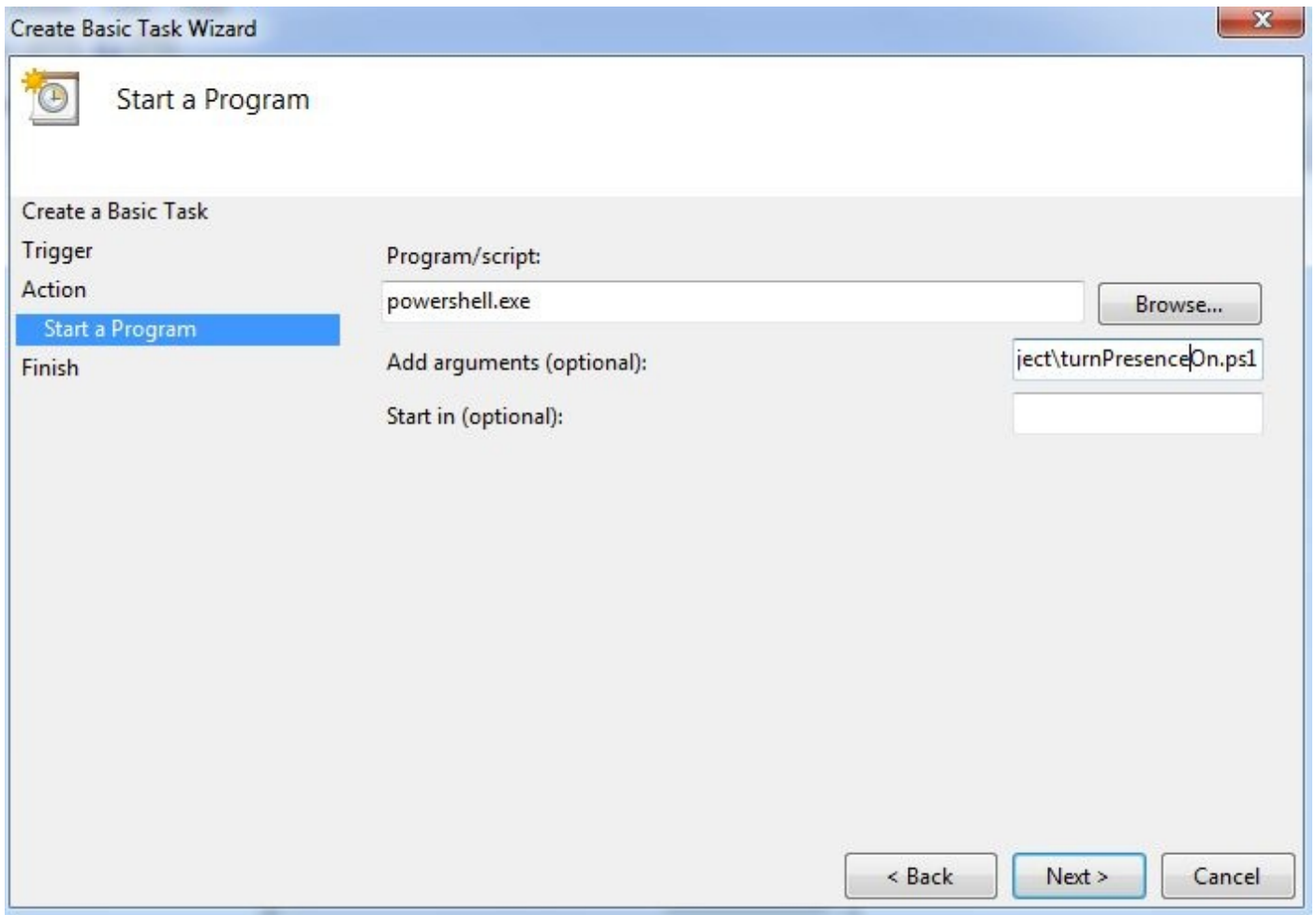
```
Windows PowerShell ISE
File Edit View Debug Help
ormail.ps1 X
1 C:\Users\othmar\mailsend\mailsend1.16.exe -to ok@keyconceptpress.com -from info@postmaster27.de -starttls -port 587 -auth -smtp s

[C] Open Remote Server restarted

--kXbnm0bec8np47UT--

[C] .
[S] 250 Requested mail action okay, completed: id=0M6UrD-1XSqVr0m0R-00yNDM
[C] QUIT
[S] 221 kundenserver.de Service closing transmission channel
```

*Figure 14.2 PowerShell script for sending the reboot notification email*



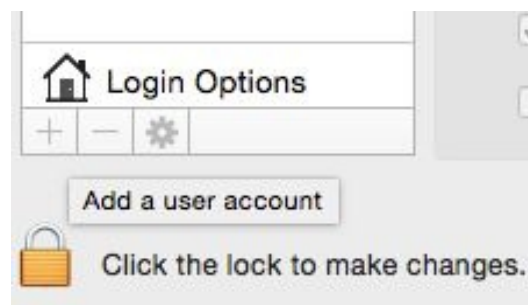
*Figure 14.3 Notification email in Task Scheduler*

## 14.2 OS X launchd

With OS X Tiger (10.4) Apple introduced `launchd` as the new service-management framework for its operating system. It replaces older services such as `init`, `crond` or `watchdogd`. Its purpose is to start, stop and manage daemons, applications, processes and scripts. While `launchd` is open source and also available for Linux, most Linux distributions still use `systemd` or `Upstart` for service management. This section focuses on the description of preparing an OS X system for auto starting OpenRemote when rebooting using `launchd`.

## 14.2.1 Setting up a Standard User for OpenRemote

If you have not done yet, I strongly recommend to set up a dedicated user for your smarthome controller. Running OpenRemote in operation using an administrator account is a significant security risk, and should never be done. If anything goes wrong, may it be an erroneous script, someone else making unauthorized changes, a virus or an attack from outside, under administrator rights the consequences will be much more severe, than under a standard user account. Administrators can create, manage, and delete other users, install and remove software, and change your Mac's settings. For these reasons, server software, which can be accessed from other devices (as in our case using smartphones or tablets) or even from the Internet (in case you set up a VPN connection for OpenRemote) should never be installed using a user account with administrator privileges. Go to [System Preferences — Users and Groups](#) and click on the plus sign on the left hand side to add a new standard user account (Figures 14.4, 14.5).



*Figure 14.4 Adding a standard user under OS X (step 1)*

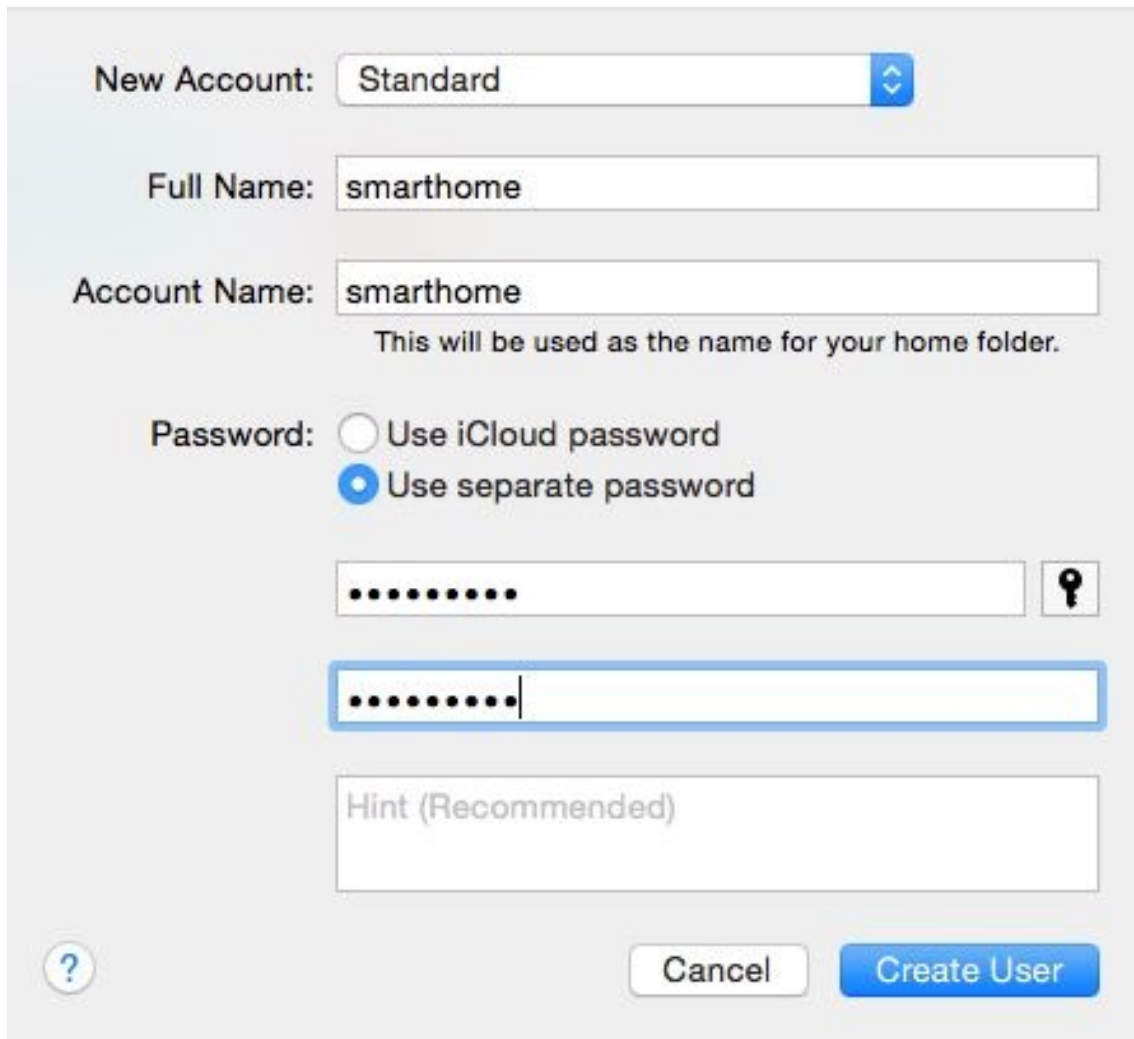
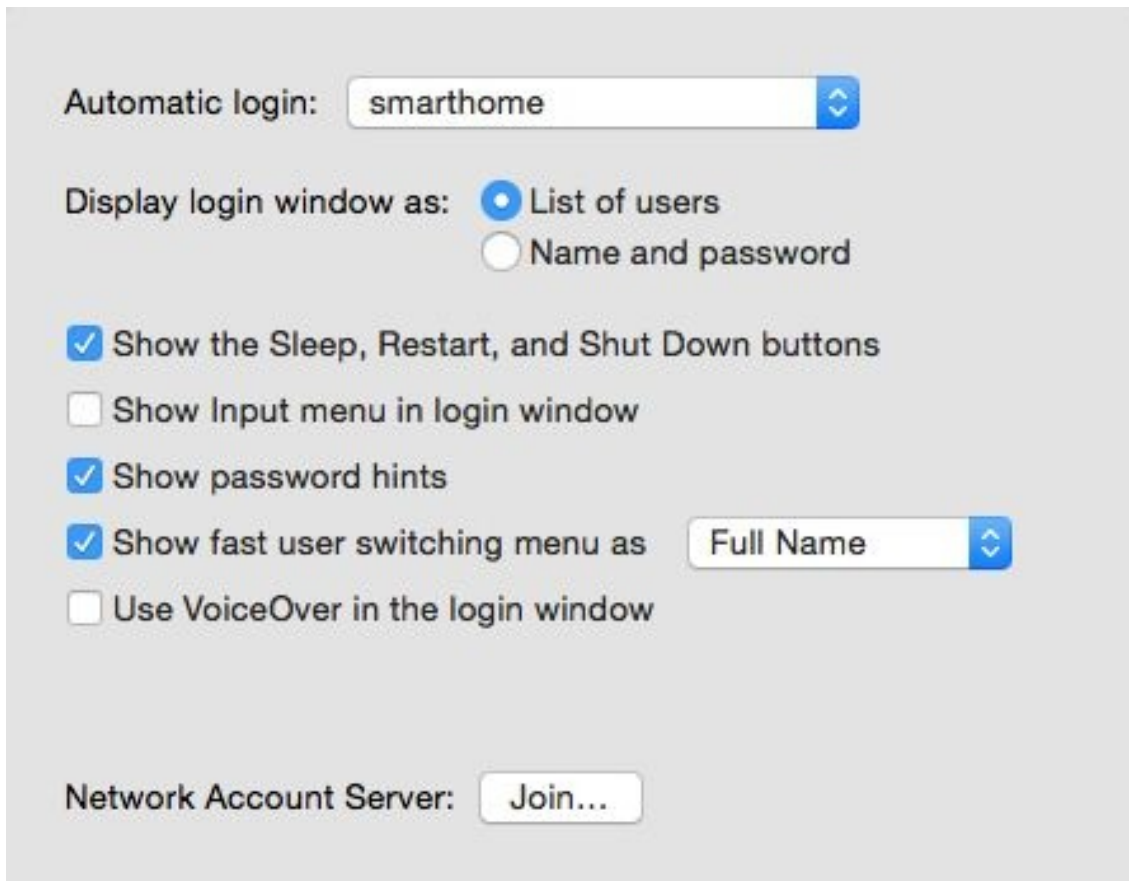


Figure 14.5 Adding a standard user under OS X (step 2)

Then select *Login Options* on the left side of the menu and configure *Automatic login* for your new user, in our case the user account *smarthome*. When you log in to your new account for the first time, the system will ask you to configure your AppleID, which you must decline. Then set up a dedicated email account for the *smarthome* controller, if you want to send notification emails as described below. Now, every time the system reboots, the new standard user *smarthome* will automatically login (Figure 14.5). This is what we need, since among other things our start up sequence will use AppleScript to send a notification email. The usage of AppleScript and the email application requires our *smarthome* user to be logged in. From a security perspective, this is acceptable, since this user cannot make changes to the system and only contains our OpenRemote system. If you want even more security, change the OpenRemote startup script `openremote.sh` to read only, with write access restricted to the administrator. This prevents the classic attack, which modifies the start up script and then restarts the system in the hope, that during the start up sequence the script would be executed using administrator rights. However, since we will use our standard user to restart OpenRemote, as you will see below, this attack would not be possible in our system setup even without changing access rights.



*Figure 14.6 Adding a standard user under OS X (step 3)*

## 14.2.2 Daemons and Agents

Once you are done with setting up OpenRemote under a dedicated standard account we can start building the autostart functionality. There are two different types of processes the `launchd` service is dealing with:

- agent processes and
- daemon processes

An agent is run on behalf of the logged in user. This means it only runs, when the user is logged in. Daemons run on behalf of the root user or any user specified with the `User` option, independent of the user being logged in or not. The system differentiates three agent and two daemon types. However, you should never have to edit System Agents or System Daemons:

Type	Run on behalf of
User Agents	Currently logged in user
Global Agents	Currently logged in user
Global Daemons	Root or the specified user
System Agent	Currently logged in user
System Daemon	Root or the specified user

*Table 14.2 Agent and daemon types*

As our autostart sequence requires a logged in user as explained above, we will use a User Agent for our startup script.

### 14.2.3 The Startup Shell Script

Before we go about configuring our User Agent, we need to create the shell script, which the agent shall start. If you just want to start OpenRemote the according shell script is simple:

```
#!/bin/sh
cd ORC/bin
./openremote.sh run
```

We save it under a name such as `orstart.sh` in our `shProject` directory, set the execution permission with

```
chmod +x orstart.sh
```

and move on to configuring our user agent. Even though the user agent will not open a Terminal window and thus will start OpenRemote without a GUI, I recommend to use the the startup option `openremote run` rather than the background option `openremote start`, since the first option allows us to track the server output from the default output variables `standard output` and `standard error`, as we will see below.



#### 14.2.4 Hint: Clean directory management

I recommend to place all custom scripts and files in the root directory of *shProject*, rather than in the OpenRemote directory (which in our project we called *ORC*) or one of its subdirectories. Otherwise you will have to manually move your custom scripts and files from your old to your new OpenRemote installation in case of a OpenRemote software update. Since we save *orstart.sh* to the *shProject* root directory, we need to insert the change directory command `cd ORC/bin` before the OpenRemote launch command `./openremote.sh` run in the above startup script.

## 14.2.5 The Utility LaunchControl

Agents and Daemons consist of XML files with well defined syntax, which are not easy to write for beginners. In order to get things done quickly I strongly recommend to use the `launchd` utility `LaunchControl` from `soma-zone`. For testing you can download the application with full functionality for free from

<http://www.soma-zone.com/LaunchControl/>

The purchase price of less than 10€/US\$ for the application is very fair considering the functionality you get. After downloading and opening `LaunchControl` you select *User Agent* in the GUI and *File — New*. In the field *Label* you enter the name of your task. Apple recommends to use reverse URL notation, since label names must be unique. So you could use something like `com.coldstart.openremote`. In the left pane you rename the default file name `local.job` to the name you want it to give. I recommend to use the label name as file name as well. The label is the name of our agent, under which it is recognized by `launchd`, the file name is simply the name of the agent definition file. There is really no reason why you should call it differently.

Next you drag the symbol *WorkingDirectory* from the utility pane on the right hand side of the `LaunchControl` GUI into the main window. Now you should have a total of three configuration blocks in the main window of your agent configuration:

- *Program to run*
- *Working Directory*
- *Run at load*

(*Program to run* and *Run at load* are automatically loaded when opening a new configuration file). In *Program to run* we now enter the name of our shell script (`orstart.sh`), in *Working Directory* the path to it's directory:

```
/Users/shProject/
```

If you enter a file which does not exist or which does not have execution rights, the entry will remain red. The same is true for the working directory path. Both, file name and directory path, need to appear in green for the agent to be able to execute.

Finally we drag the *StandardErrorPath* symbol into our program window. Two command boxes appear: *Standard Output* and *Standard Error*. They allow us to monitor the standard and error outputs which our `OpenRemote` startup script generates. Edit the path definitions and the file names according to your needs. We save our daemon definition with *File — Save*.

When we now select *Load* from the upper right corner of the GUI we run our agent for the first time (Figure 14.7).

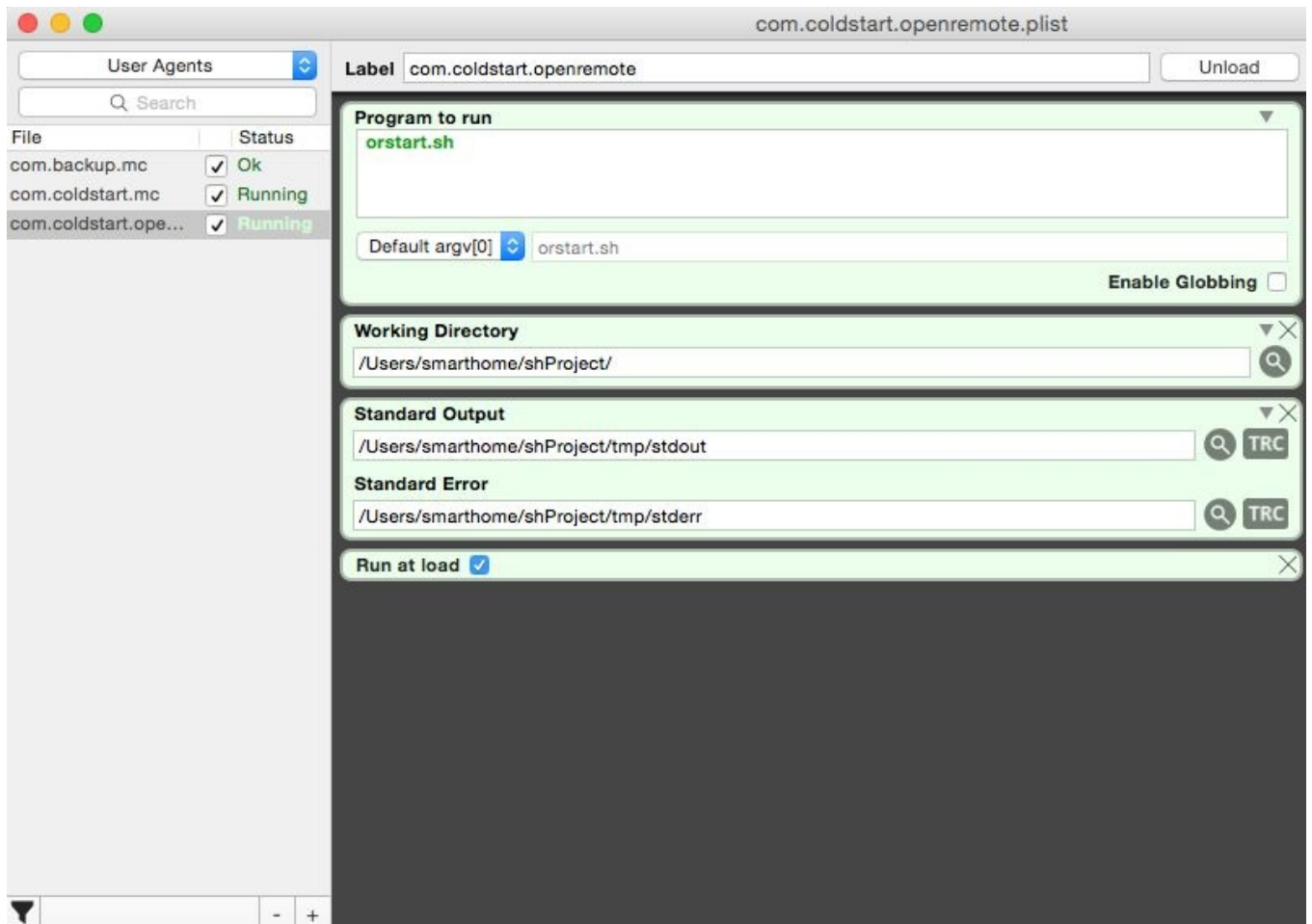


Figure 14.7 OpenRemote startup User Agent using LaunchControl

In the activity pane on the left hand side we can see if our agent has successfully loaded. To validate if our startup script has actually started OpenRemote we open a Terminal window and use the `ps` (process status) command, which displays every active process along with its job number. In order to filter out the process of interest, in our case the OpenRemote process, we pipe the output to a filter using `|grep openremote`:

```
ps aux|grep openremote
1290  0,0  0,0  2432772  640 s000  S+  4:31pm  0:00.00  grep openremote
```

In the above case however, OpenRemote is NOT running. The only process that matches “openremote” is the `grep` process itself (the process doing the searching). Below an example of the process state command with OpenRemote running:

```
ps aux|grep openremote
smarthome  4168  7.6  4.1  8373520  681644  ??  S   Tue12PM  318:44.94
/Library/Java/JavaVirtualMachines/jdk1.8.0.jdk/Contents/Home/bin/java -Dcatalina... ...
smarthome  4187  0.0  9.8  8547572  1636312  ??  S   Tue12PM  218:59.03 /...
smarthome  8320  0.0  0.0  2424580   384 s001  R+  8:26AM   0:00.00  grep openremote
```

In case the startup agent is not working as desired, click on the **TRC** (Trace) symbol in the **Standard Output** and the **Standard Error** window of LaunchControl. This will get you the standard and error output trace of your startup script with the necessary information for debugging (Figure 14.8).

To stop a process you can use the `kill` command. With `kill` you can either specify the process

you want to stop by PID (Process ID) which is displayed when using the process listing command `ps`, or by name. The command

`kill 571`

will stop the process with the PID 571. Alternatively you can monitor and stop processes using the OS X Activity Monitor *Applications — Utilities — Activity Monitor App*.

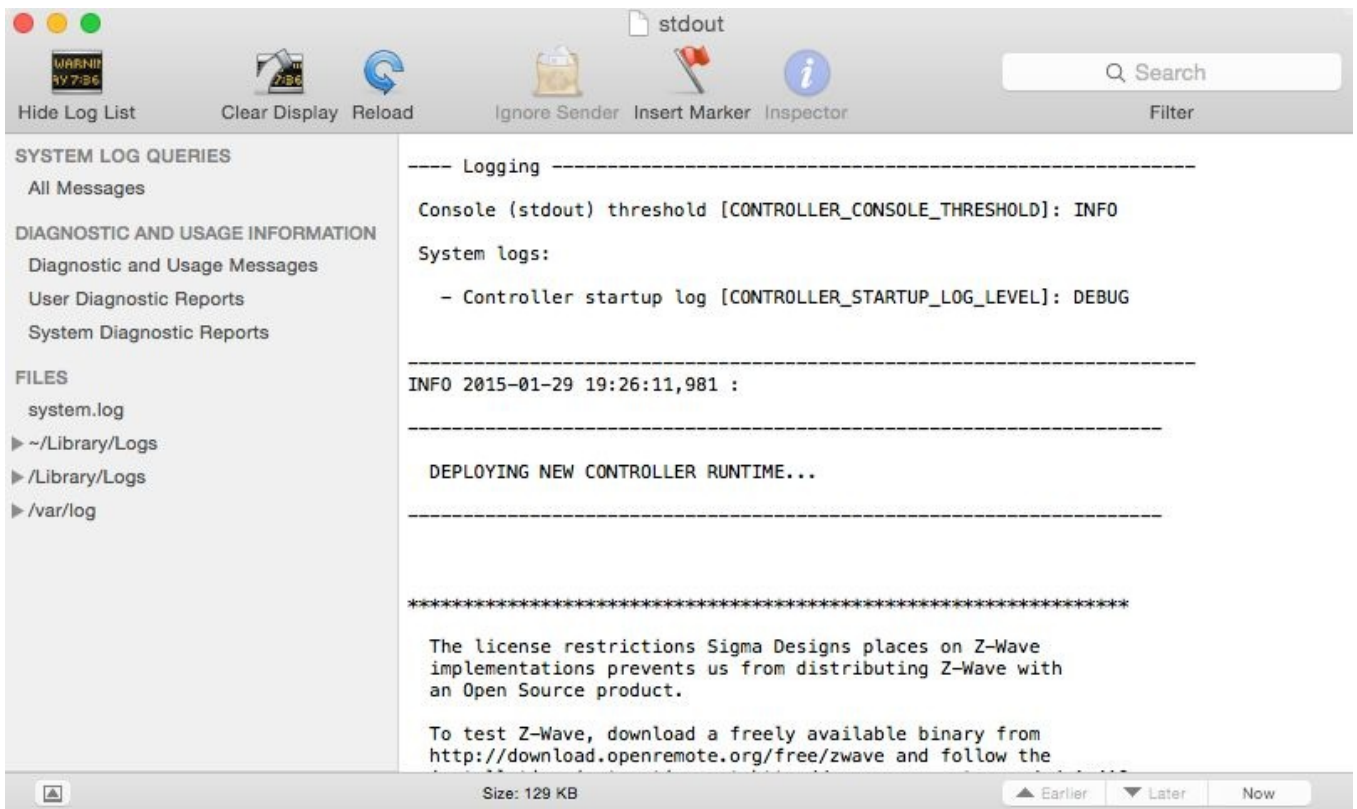


Figure 14.8 Analyzing the startup sequence using LaunchControl

## 14.2.6 Reboot Notification via E-Mail

Finally we want to send a notification email every time the server is rebooting. Since we want to avoid setting up a SMTP server on our system, which is not a trivial task, we will use the Apple Mail email account of our standard user. Alternatively we could also use the command line mailer application *mailsend* as described in the Windows section above.

*Mailsend* is also available for OS X. For our project however we will simply control our smart home email account using AppleScript commands, which we will then - in a second step - call from a shell script. As explained above, in order for this approach to work you will need to set the user account of your OpenRemote server to *Automatic Login* (open *System Preferences* and select *Users&Groups — Login Options*) and you will need to setup an Apple Mail account for this user.

Below the simple AppleScript, which sends out an email with the current date in its message body. At first the AppleScript variables `recipientName`, `recipientAddress`, `theSubject`, `theTime`, `theText` and `theContent` are set. The command `current date` generates the current date, which we want to be contained in our notification message. The command `tell application "Mail"` activates the Apple mail application, and `Create the message`, `Set a recipient`, `send create` and `send the email message`. Below the complete AppleScript code:

```
set recipientName to "Smarthome"
set recipientAddress to "info@youremail.com"
set theSubject to „Server restart"
set theTime to current date
set theText to "Server restart at "
set theContent to theText & theTime

tell application "Mail"

    ##Create the message
    set theMessage to make new outgoing message with properties {subject:theSubject, content:theContent,
    visible:true}

    ##Set a recipient
    tell theMessage
        make new to recipient with properties {name:recipientName, address:recipientAddress}

        ##Send the Message
        send

    end tell

end tell
```

Now open the AppleScript editor, paste in the above script (inserting your email address for `recipientAddress`) and hit *Run* to test if the script is working. Almost immediately an email

should be sent to the recipient address in the script (Figure 14.9).

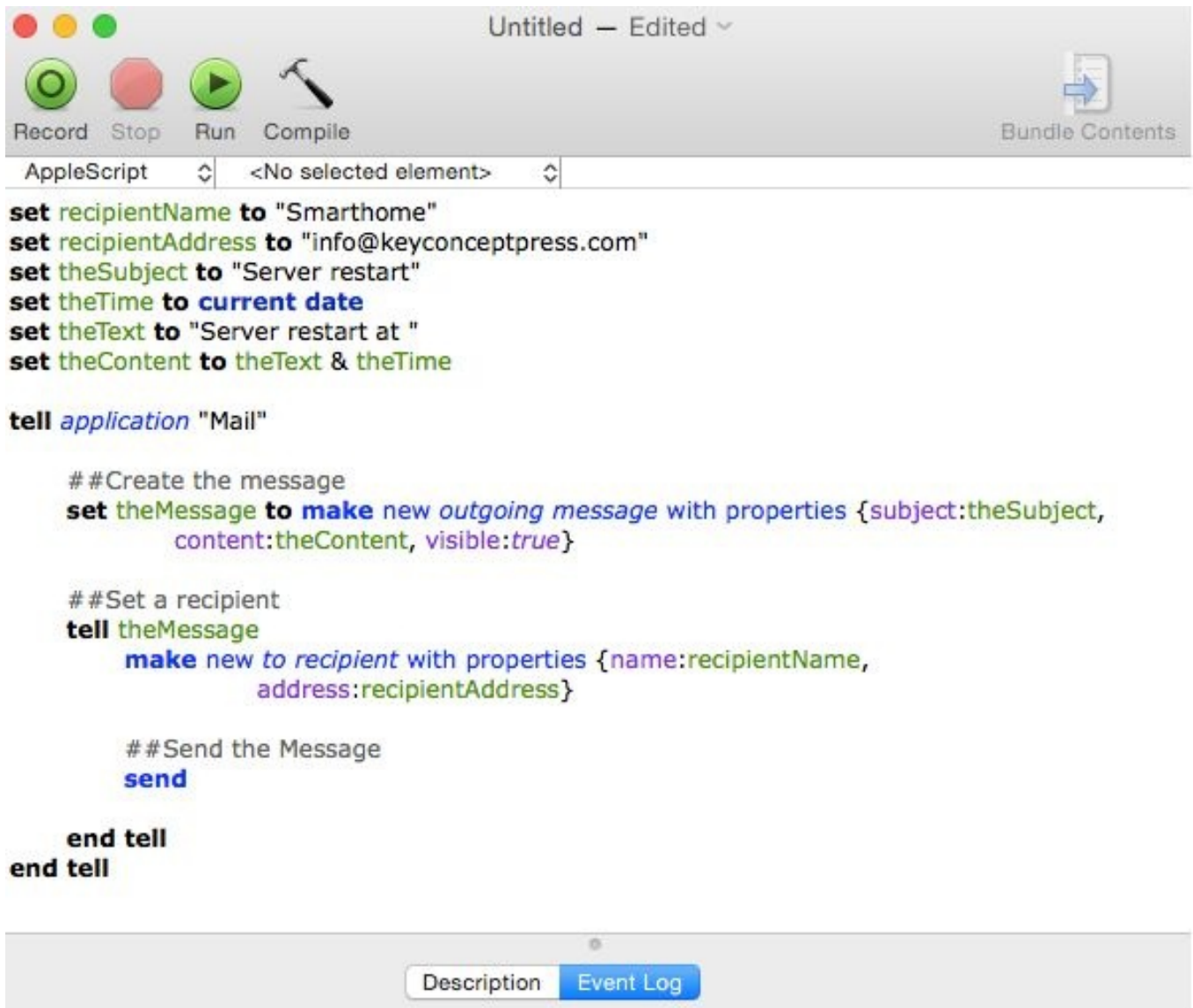


Figure 14.9 AppleScript for sending an email

Now we want to run the above AppleScript within a shell script. For this we just need to insert the `osascript` command in the first line and tell the script interpreter to handle all text following this command as AppleScript until EOF is reached:

```
#!/bin/sh
exec osascript << EOF
set recipientName to "Smarthome"
set recipientAddress to "info@keyconceptpress.com"
set theSubject to "OpenRemote Server restarted"
set theTime to current date
set theText to "OpenRemote Server Beachroad 4, Uprville restarted at "
set theContent to theText & theTime
tell application "Mail"
    set theMessage to make new outgoing message with properties {subject:theSubject, content:theContent,
        visible:true}
    tell theMessage
```

```
        make new to recipient with properties {name:recipientName, address:recipientAddress}
    send
end tell
end tell
EOF
```

We save the above code as `tmailor.sh` (terminalmail openremote). All which is left to do now is to add `tmailor.sh` to our autostart script `orstart.sh`, which now reads as follows.

```
#!/bin/sh
sleep 60
./tmailor.sh
cd ORC/bin
./openremote.sh run
```

The command `sleep 60` introduces a 60 second delay to the start of the script, since we want to make sure the autologin procedure for our *smarthome* user is finished, by the time we start OpenRemote and send the notification email.

### 14.2.7 Curly or straight, this is the question!

Finally an important hint for editing AppleScript and shell scripts in general, where quotation marks have an important function. Double quotes (in shell scripts, AppleScripts and UNIX in general) are used for quoting phrases as you have seen in some of the above scripts. However, these quotes need to be the straight quotation marks, rather than the curly quotation marks, which are used by word processor software. Now sometimes, when you edit a shell script or an AppleScript using TextEdit or another word processor, it can happen, that the straight quotes are converted to curly quotes resulting in runtime errors, which at first sight are hard to analyze. So always take a close look at the quotation marks: „Curly or straight, this is the question!“ (Figure 14.10).

```
„curly double quotes“  
"straight double quotes"  
`backtick`
```

*Figure 14.10 Curly versus straight double quotes and the single backtick*





# 15 Troubleshooting and Testing

In many projects with high dependency on reliable software, the aspect of software test is underestimated. In particular in control engineering, which is what smart home control really is, a systematic and planned test phase is important in order to put a reliable, redundant system in place. In a smart home environment scripts and system configurations interact with people and assets, which can get hurt or damaged if something goes wrong. Thus a wait and see approach is not an option. While a detailed tutorial on test strategies and tactics is beyond the scope of this book, I cannot emphasize enough the importance of this topic. First considerations for test planning already need to be made before the start of the project. Test concepts and code testability have to be part of high level software design. Then for each software module, in parallel to writing the actual code, relevant test cases need to be defined. Systematic, planned test execution in the end ensures a software quality level which meets the requirements of a 24/7 smart home operation.

Once the software is tested and ready for deployment, the so called dress rehearsal tests start. During this phase, the system is tested by end users, who are given a heads up to watch out for funny or faulty behavior. Once the software passes this phase, the first stable release is rolled out. As part of the now beginning release management, the latest tested and stable version is well documented and saved as a backup. A roll back to this version must always be possible at any time.

For troubleshooting problems or unexpected behavior of the smart home control system, besides monitoring processes and log files on the controller, it is always a good idea to be able to monitor the smart home network itself as well. A powerful troubleshooting tool for this purpose is the open source protocol analysis tool Wireshark (former Ethereal - <http://www.wireshark.org> ) (Figure 15.1).

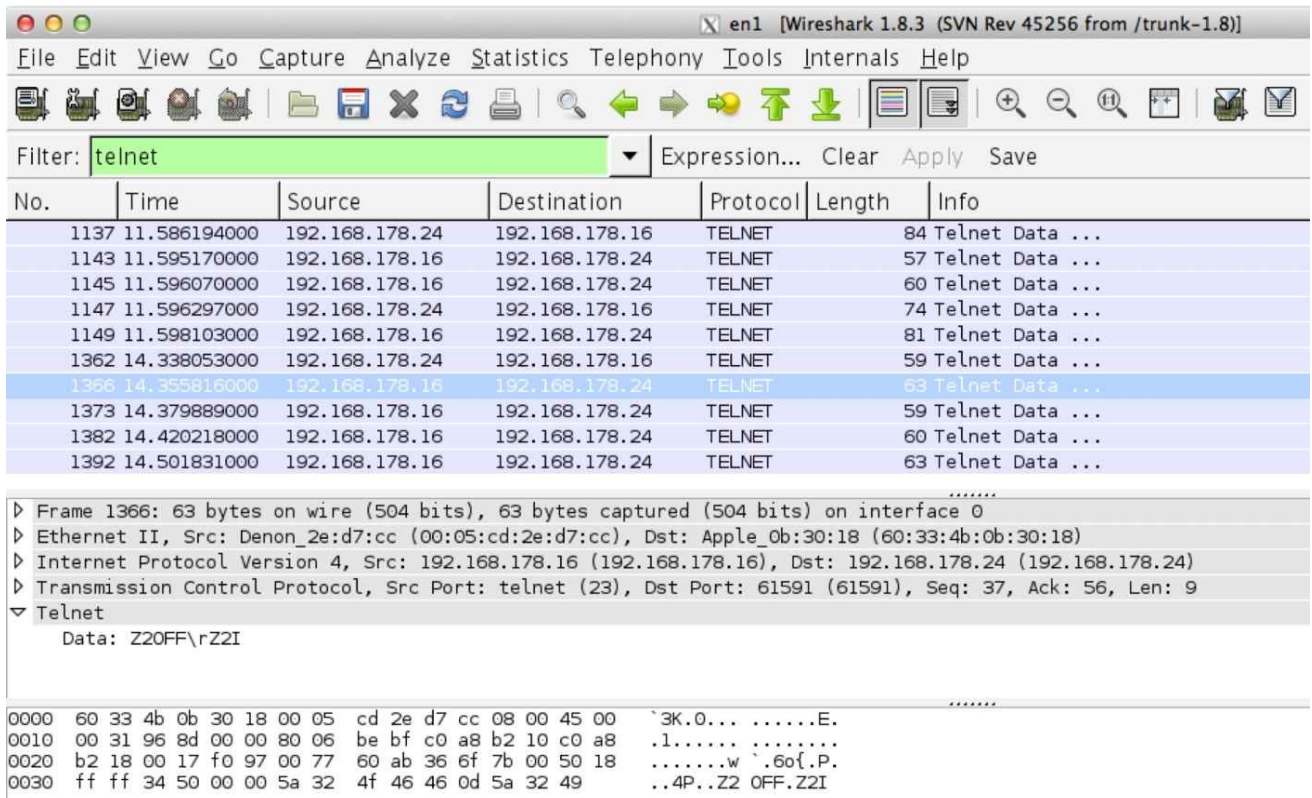


Figure 15.1 Wireshark protocol trace of the Denon 3313 Telnet response to a “Z2?” command

## 15.1 Preventive Maintenance

Contrary to traditional troubleshooting, the approach of preventive maintenance is to try to identify and repair a problem before users are impacted. Typically such systems consist of a problem detection component and (ideally) a self-healing component.

A common method for problem detection in real time high availability control systems are watchdog scripts. These are scripts that continuously monitor the key software processes. If a process stops working, they set an alarm. Some watchdog implementations browse the listing of active processes in order to verify whether a process is still alive. There are, however, cases in which a process is still listed as active, while in reality it already has stopped working. Commonly these are referred to as Zombie processes. Thus the proper way to reliably monitor the health of a process is to insert a small routine - the heartbeat - which periodically (e.g. every minute) writes a value (e.g. the number 100) to a file or variable. A separate watchdog script then decrements this value (e.g. every minute) by 10 and compares the resulting value to zero. If the process stalls for several minutes, the variable is not set back to its initial value and eventually reaches zero, by when the watchdog sets an alarm. An alternate approach is to have the heartbeat script store the current timestamp to a file or variable. The watchdog script then periodically compares its current timestamp with the heartbeat timestamp. When the delta exceeds a certain limit, the watchdog script sets an alarm.

In addition to monitoring software execution, watchdogs can also be used to verify whether an action actually reaches the real world, as intended. As an example, a watchdog for a reminder system that announces an important message in a smart home could validate the output of the message by recording the announcement via microphone, and comparing it with the intended output.

## 15.2 OpenRemote Heartbeat and Watchdog

As an example, we will monitor our key process, the OpenRemote controller, using a heartbeat and a watchdog script. As heartbeat we will use the file *heartbeat.txt*, which an OpenRemote rule will overwrite every minute, storing the current date and time to it.

The watchdog script *watchdog.sh* (*watchddog.ps1* for Windows) will read out the content of *heartbeat.txt* every ten minutes. The script will then compare the current time at execution with the time contained in *heartbeat.txt*. If the delta of the two time stamps is larger than 10 minutes, the watchdog will send an alarm email. The times can of course be modified and adapted to the availability requirements of your smart home project. In our case the maximum downtime of the smarthome system until the alarm email is being sent is  $10+1=11$  minutes. (The delay of one minute is being added by the resolution of the heartbeat, the 10 minutes delay by the threshold set for the watchdog).

We start with the OpenRemote heartbeat rule. Since we want to use the Java Input/Output class `java.io` we need to load `java.io.*` with the command

```
import java.io.*;
```

The heartbeat shall occur every minute, which we achieve with the timer expression:

```
timer (cron:30 * * * * ?)
```

At the 30th second of every minute the rule will be evaluated. We want to make sure, that the heartbeat I/O operation does not collide with other I/O operations, which might be scheduled to take place at exactly the full hour, which is why we avoid the expression `timer (cron:0 * * * * ?)`.

For the output process we use the heartbeat (Hb) variable `writerHb`, which we associate with the Java class `Writer` and its subclass `FileWriter`. With `new Date()` we assign the current date to the variable `dateHb`. With the string formatting option `%ts` we store the current date in form of the UNIX timestamp (number of seconds since Jan 1, 1970) in the variable `strHb`. The command `writerHb.write(dateHb.toString()); now` writes the UNIX timestamp to the text file *heartbeat.txt* as outlined below:

```
Date dateHb = new Date();
String strHb = String.format("%ts", dateHb );
Date dateHbTime = getTime();
FileWriter("/Users/smarthome/shProject/heartbeat.txt",false);
writerHb.write(strHb.toString());
writerHb.close();
```

The `FileWriter` option `false` instructs the function to generate a new file every time it is called, while the option `true` would append the content to the file in case it exists. In case no file exists, a new file is generated in any case. We want *heartbeat.txt* to only contain the current date and time, which is why we use the `FileWriter` option `false`. Since we want *heartbeat.txt* to be stored in the `shProject` directory we specify the path accordingly. The complete code for the heartbeat rule including the necessary package definitions and library imports now reads as follows:

```

//Package, globals and imports:
package org.openremote.controller.protocol
global org.openremote.controller.statuscache.CommandFacade execute;
global org.openremote.controller.statuscache.SwitchFacade switches;
global org.openremote.controller.statuscache.LevelFacade levels;
import org.openremote.controller.protocol.*;
import org.openremote.controller.model.event.*;
import java.sql.Timestamp;
import java.util.*;
import java.lang.Float;
import java.io.*;

//_____
//Rule to generate a OpenRemote heartbeat
//Every minute the string "OR Heartbeat!!!! current date" is sent to the Terminal
//Every minute current UNIX timestamp is written to the file heartbeat.txt
//_____
rule "OR Watchdog"
timer (cron:30 * * * * ?)
when
eval(true)
then
Date dateHb = new Date();
String strHb = String.format("%ts", dateHb );
System.out.println("!!!OR Heartbeat!!! "+dateHb+" UNIX Timestamp: "+strHb);
//true option: FileWriter appends to file
//false option: FileWriter overwrites file
FileWriter("/Users/smarthome/shProject/heartbeat.txt",false);
writerHb.write(strHb.toString());
writerHb.close();
end

```

Now we move on to the watchdog script. Under Windows PowerShell the command

```
[int][double]::Parse((Get-Date -UFormat %s))
```

returns the UNIX timestamp. We store it in the variable \$watchdog\_time:

```
$watchdog_time = [int][double]::Parse((Get-Date -UFormat %s))
```

With Get-Content we read the timestamp of *heartbeat.txt* and store it in the variable \$heartbeat\_time:

```
$heartbeat_time = Get-Content "C:\Users\smarthome\shProject\heartbeat.txt"
```

The delta between \$watchdog\_time and \$heartbeat\_time we store in \$deltaHb:

```
$deltaHb = $watchdog_time - $heartbeat_time
```

Now we add the below if-statement, which sends the server down notification using the command line mailer application `mailsend.exe`, as explained in section 14.1.1:

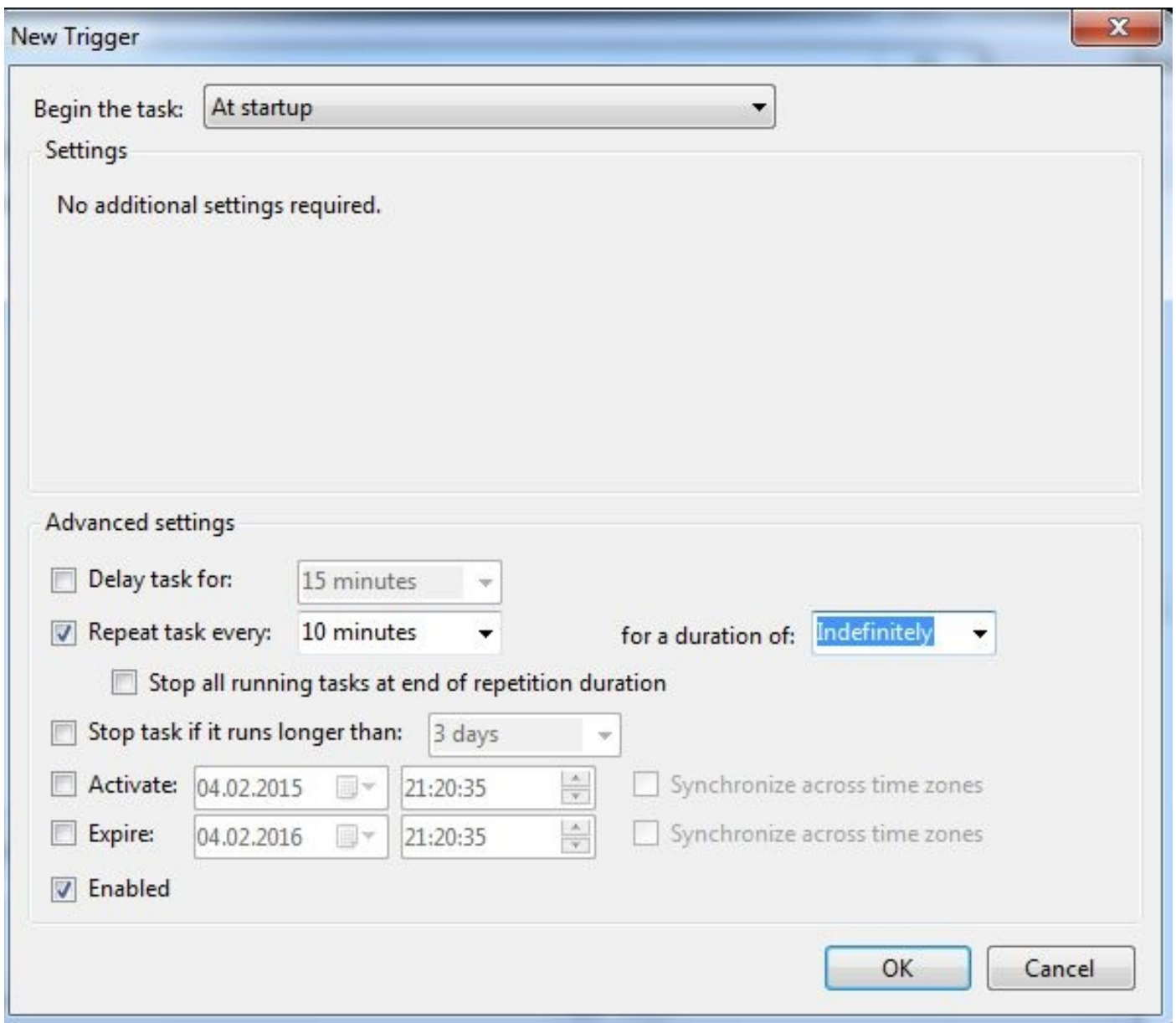
```
if ($deltaHb -gt 600) {  
    mailsend.exe -to info@keyconceptpress.com -from info@smarthomeserver.com -starttls -port 587 -auth -smtp  
smtp.gmail.de -sub "Open Remote Server Notification" +cc +bc -v -user info@smarthomeserver.com -pass  
"yourpassword" -M „!!Open Remote Server down!!“  
}
```

The complete watchdog script, which we store as `watchdog.ps1` now reads as below:

```
$watchdog_time = [int][double]::Parse((Get-Date -UFormat %s))  
$heartbeat_time = Get-Content "C:\Users\smarthome\shProject\heartbeat.txt"  
$deltaHb = $watchdog_time - $heartbeat_time  
if ($deltaHb -gt 600) {  
    C:\Users\smarthome\mailsend\mailsend1.16.exe -to info@keyconceptpress.com -from info@postmaster27.de -starttls -  
port 587 -auth -smtp smtp.1und1.de -sub "Open Remote Server Notificaton" +cc +bc -v -user info@postmaster27.de -  
pass "kyas.net" -M "Open Remote Server down!!!"  
}
```

You should be able to use the Drools heartbeat rule pretty much as listed above. When using the code of `watchdog.ps1` just make sure to adapt the file name and path definitions for `heartbeat.txt` and `mailsend1.16.exe` to your environment. Also make sure to download and install a copy of the command line mailer application `mailsend` as described in section 14.1.1.

As the last step under Windows we now need to configure Task Scheduler to run `watchdog.ps1` at startup and at 10 minute intervals. We select *Start — Control Panel — System and Security — Administrative Tools — Task Scheduler*. Then we click on the *Action* menu and select *Create Task*. We give the task a name like `ORWatchdog`, and configure under *Actions* to run `watchdog.ps1`. Under *Triggers* set Task Scheduler to start the task at system startup and to repeat it every 10 minutes (Figure 15.2).



*Figure 15.2 Watchdog configuration with Windows Task Scheduler: Run at startup and repeat every 10 minutes*

The watchdog shell script for OS X, which we also place in the *shProject* directory, looks as follows:

```
#!/bin/sh
watchdog_time=$(date +%s);
heartbeat_time=`cat heartbeat.txt`;
deltaHb=$(expr $watchdog_time - $heartbeat_time);
if [ "$deltaHb" -gt "600" ]
then
./tmailORDown.sh
fi
```

A few words of explanation to the above code lines: The `date` function provides current date and time. The option `+%s` formats the output of `date` to a Unix timestamp. Using the `cat` command we read the content of `heartbeat.txt` into the variable `heartbeat_time`. Using an `if`



statement we compare the delta between `watchdog_time` and `heartbeat_time` to the watchdog threshold of 600 seconds. In case `$deltaHb` is larger than 600 seconds the script `tmailORDown.sh` is being executed, which sends out a downtime alert email. (See chapter 14 for a detailed description of the email sending script.) When you are using the script as above make sure to adapt the path statement in the code to your environment in case you use different directory names.

As the last step we configure the OS X `launchd` agent for our watchdog script using LaunchControl is described in more detail in Chapter 14. In LaunchControl we select *User Agent* and *File — New*. In the field *Label* we enter as the name for the task something like `com.watchdog.openremote`. In the left pane we rename the default file name, which is `local.job` to the same name we just chose for the label. Next we drag the symbols *WorkingDirectory* and *StartInterval* from the utility pane on the right hand side of the LaunchControl GUI into the main window. Now we should have a total of four configuration boxes in the main window of our agent configuration:

- *Program to run*
- *Working Directory*
- *Run at load*
- *StartInterval*

(*Program to run* and *Run at load* are automatically loaded when opening a new configuration file).

In *Program to run* we now enter the name of our shell script (`watchdog.sh`), in *Working Directory* the path to its directory:

```
/Users/shProject/
```

If you enter a file name which does not exist or which does not have execution rights, the entry will remain red. The same is true for the working directory path. Both, file name and directory path, need to appear in green for the agent to be able to execute. In *StartInterval* we enter 600. The watchdog script will now be called at startup and from then every ten minutes. We save our agent definition with *File — Save*. When we now select *Load* from the upper right corner of the GUI we run our agent for the first time (Figure 15.3).

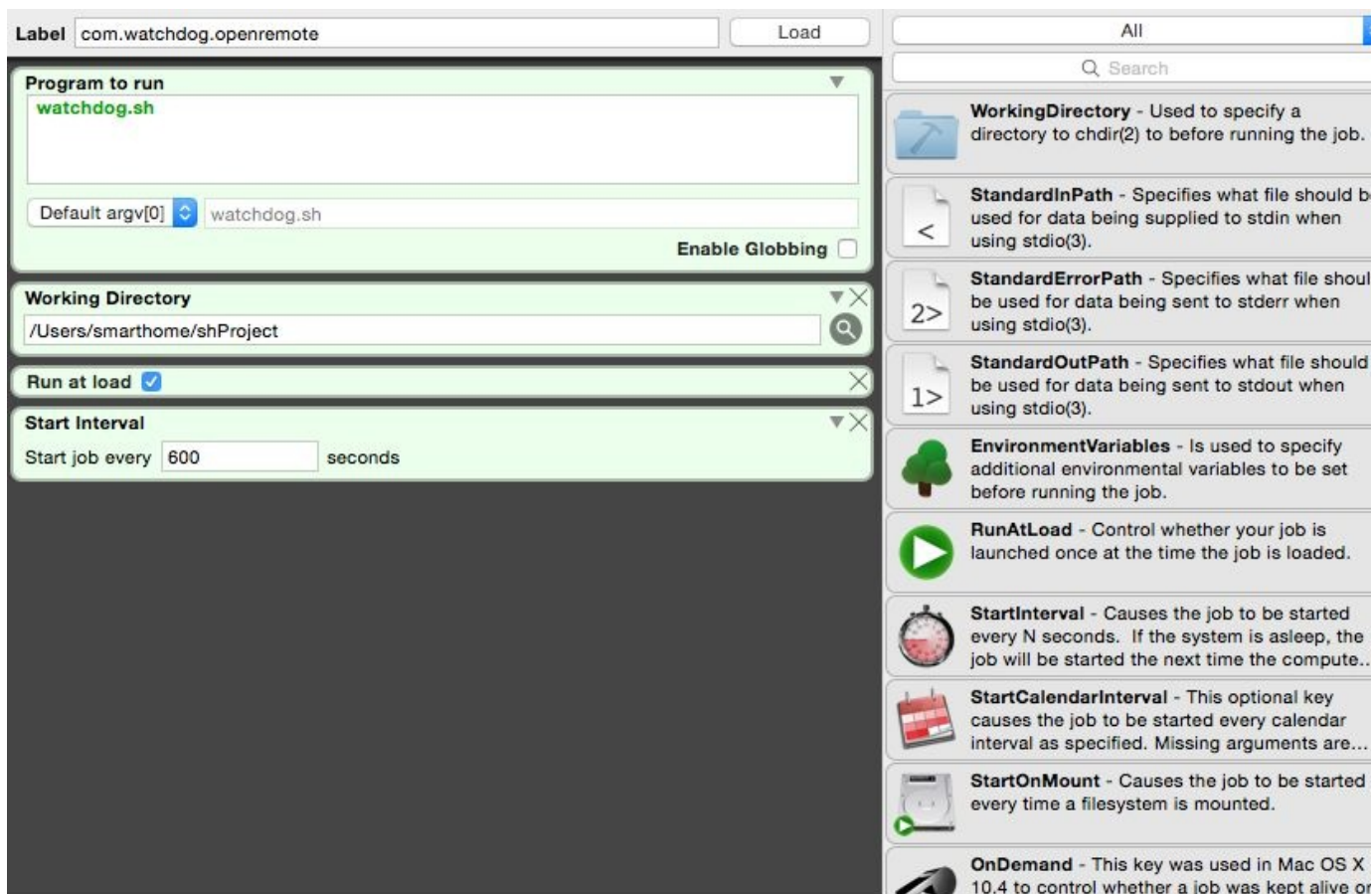


Figure 15.3 Watchdog configuration in OS X with LaunchControl



## 16 ... we proudly present: Reporting

Equally important as setting up a well tested and safe control infrastructure is the generation of reports on the key operating parameters. Reports are the basis for evaluating the operational parameters of a building and how they play together with its usage as well as the environmental conditions. Most buildings undergo constant changes due to additions, upgrades or changes in usage. Reports can aid in planning and evaluating these changes from the building control perspective. In addition reports play a vital role in monitoring and optimizing the energy consumption of a building. For our project we will create an automated report which hourly records the room temperature for each room, the outside temperature and the weather condition. The data will be stored in a CSV formatted file. Once per day a report for the past 24 hours will be sent out per email to an external email address. In addition all data will be consolidated on a monthly basis and stored accordingly.

## 16.1 A Drools Reporting Rule

The first step will be to write a Drools rule which stores the data of our room sensors, the outside temperature as well as the weather condition to a CSV file. The code for our corridor temperature sensor reporting rule including the necessary package definitions and library imports reads as follows:

```
//Package, globals and imports:
package org.openremote.controller.protocol
global org.openremote.controller.statuscache.CommandFacade execute;
global org.openremote.controller.statuscache.SwitchFacade switches;
global org.openremote.controller.statuscache.LevelFacade levels;
import org.openremote.controller.protocol.*;
import org.openremote.controller.model.event.*;
import java.sql.Timestamp;
import java.util.*;
import java.lang.Float;
import java.io.*;
//_____
//Rule to store corridor temperature in CSV format
//_____
rule "CorridorTempReport"
timer (cron:0 30 * * * ?)
when
$temp7 : Event( source == "CurrentTemperatureCorridor", $tCorr : value );
then
String newLine = System.getProperty("line.separator");
String CorrTmp ="",CorrTemp,""+$tCorr+newLine;
//data output to terminal
System.out.println(CorrTmp);
//data output to file tmpreport.txt
Writer writerRep = new FileWriter("/Users/smarthome/shProject/reports/tmpreport.txt",true);
writerRep.write(CorrTmp.toString());
writerRep.close();
end
```

A few words of explanation to the above code lines. Following the rule name we start with the timer command. Since we want the rule to execute at every full hour (second 10, minute 30 of every hour) we specify the following timer expression:

```
timer (cron:10 30 * * * ?)
```

The `when` statement in the next line marks the beginning of the rule's LHS. If the sensor exists, its value is stored into a variable. The condition reads:

```
$temp7 : Event( source == "CurrentTemperatureCorridor", $tCorr : value );
```

The code line declares a new local variable of type `Event` called `$temp7`. Inside the brackets we have the rule condition, which searches for an entity with the name „CurrentTemperatureCorridor” and which, in case it exists, assigns its value to the variable `$tCorr`. If this condition is true, we have the value of our corridor sensor stored in `$tCorr` and the RHS (the then part) of the rule is being executed. By the way, the `$` sign of some of the variables is not a syntax requirement. It is simply used to keep track of various groups and types of variables.

In the RHS side of our rule we first declare the local variable `dateRp` of type `Date`, assign it the current date and store it in form of a UNIX timestamp to `strRp`. Then we define the string `newline` and assign the system property `line.separator` to it:

```
String newLine = System.getProperty("line.separator");
```

At the end of the last sensor data (which in our case is the corridor data) we append this string as a line feed command. Rather than appending `/n`, which works in some cases (but not in other cases like in Windows environments), this approach achieves a system independent line feed command. Now we write the corridor temperature data followed by a colon and the `newLine` string to the variable `CorrTmp`. Using Java `FileWriter` we write the content of `CorrTmp` to the file `tmpreport.txt`.

As the first two entries for each CSV data record we want the current date and the UNIX time stamp, which we add to the first data record, in our case the `WeatherConditionBerlin` record. Below for the sake of completeness the eight Drools rules for weather condition, outside temperature, bathroom, familyroom, living room, bedroom1, bedroom1 and corridor:

```
//-----  
//Rule to store Weather Condition in CSV format  
//-----  
rule "WeatherConditionReport"  
timer (cron:10 30 * * * ?)  
when  
$weather : Event( source == "WeatherConditionBerlin", $WcBerlin : value );  
then  
Date dateRp = new Date();  
String strRp = String.format("%ts", dateRp);  
String WeatherData = dateRp+" "+strRp+" WeatherBerlin,"+ $Berlin;  
//data output to terminal  
System.out.println(WeatherData);  
//data output to file tmpreport.txt  
Writer writerRep = new FileWriter("/Users/smarthome/shProject/reports/tmpreport.txt",true);  
writerRep.write(WeatherData.toString());  
writerRep.close();  
end
```

```
//-----
//Rule to store outside temperature in CSV format
//-----
rule "OutsideTempReport"
timer (cron:15 30 * * * ?)
when
$Temp1 : Event( source == "Tempberlin", $TempBerlin : value );
then
String OutsideTmp =",TempBerlin,"+$TempBerlin;
//data output to terminal
System.out.println(OutsideTmp);
//data output to file tmpreport.txt
Writer writerRep = new FileWriter("/Users/smarthome/shProject/reports/tmpreport.txt",true);
writerRep.write(OutsideTmp.toString());
writerRep.close();
end
```

```
//-----
//Rule to store bathroom temperature in CSV format
//-----
rule "BathTempReport"
timer (cron:20 30 * * * ?)
when
$Temp2 : Event( source == "CurrentTemperatureBathroom", $tBath : value );
then
String BathTmp =",BathTemp,"+$tBath;
//data output to terminal
System.out.println(BathTmp);
//data output to file tmpreport.txt
Writer writerRep = new FileWriter("/Users/smarthome/shProject/reports/tmpreport.txt",true);
writerRep.write(BathTmp.toString());
writerRep.close();
end
```

```
//-----
//Rule to store livingroom temperature in CSV format
//-----
rule "LivingTempReport"
timer (cron:25 30 * * * ?)
when
$Temp3 : Event( source == "CurrentTemperatureLivingroom", $tLiv : value );
```

```
then
String LivingTmp =",LivingTemp,"+$tLiv;
//data output to terminal
System.out.println(LivingTmp);
//data output to file tmpreport.txt
Writer writerRep = new FileWriter("/Users/smarthome/shProject/reports/tmpreport.txt",true);
writerRep.write(LivingTmp.toString());
writerRep.close();
end
```

```
//-----
//Rule to store familyroom temperature in CSV format
//-----
rule "FamilyTempReport"
timer (cron:30 30 * * * ?)
when
$temp4 : Event( source == "CurrentTemperatureFamilyroom", $tFam : value );
then
String FamilyTmp =",FamilyTemp,"+$tFam;
//data output to terminal
System.out.println(FamilyTmp);
//data output to file tmpreport.txt
Writer writerRep = new FileWriter("/Users/smarthome/shProject/reports/tmpreport.txt",true);
writerRep.write(FamilyTmp.toString());
writerRep.close();
end
```

```
//-----
//Rule to store bedroom1 temperature in CSV format
//-----
rule "Bedroom1TempReport"
timer (cron:35 30 * * * ?)
when
$temp5 : Event( source == "CurrentTemperatureBedroom1", $tBed1 : value );
then
String Bed1Tmp =",Bed1Temp,"+$tBed1;
//data output to terminal
System.out.println(Bed1Tmp);
//data output to file tmpreport.txt
Writer writerRep = new FileWriter("/Users/smarthome/shProject/reports/tmpreport.txt",true);
writerRep.write(Bed1Tmp.toString());
```



```

writerRep.close();
end

//-----
//Rule to store bedroom2 temperature in CSV format
//-----
rule "Bedroom2TempReport"
timer (cron:40 30 * * * ?)
when
$temp6 : Event( source == "CurrentTemperatureBedroom2", $tBed2 : value );
then
String Bed2Tmp ="Bed2Temp,"+$tBed2;
//data output to terminal
System.out.println(Bed2Tmp);
//data output to file tmpreport.txt
Writer writerRep = new FileWriter("/Users/smarthome/shProject/reports/tmpreport.txt",true);
writerRep.write(Bed2Tmp.toString());
writerRep.close();
end

//-----
//Rule to store corridor temperature in CSV format
//-----
rule "CorridorTempReport"
timer (cron:45 30 * * * ?)
when
$temp7 : Event( source == "CurrentTemperatureCorridor", $tCorr : value );
then
String newLine = System.getProperty("line.separator");
String CorrTmp ="CorrTemp,"+$tCorr+newLine;
//data output to terminal
System.out.println(CorrTmp);
//data output to file tmpreport.txt
Writer writerRep = new FileWriter("/Users/smarthome/shProject/reports/tmpreport.txt",true);
writerRep.write(CorrTmp.toString());
writerRep.close();
end

```

## 16.1.1 I/O Exception Handling

In case you have not written code with I/O commands before, be aware, that you also should add basic error (exception) handling along with the `FileWriter` commands. Chances are, that events occur during I/O operations, which cause the software to produce errors and terminate. For example, when writing to a file, the operating system may not permit to do so, perhaps because the disk is locked, or there is no space available. You must prepare your code to catch this exception and therefore prevent the program from terminating. In Java to catch an exception the statements that might throw the exception are encapsulated in a `try` block. When we catch an exception we name the class of the exception (in our case `IOException`) and define a variable that will have that type. Within the body of the catch clause we can print out an exception message. For our example a basic exception handling code could look as follows:

```
try
{
    output = new PrintWriter(new FileWriter("data.txt"));
    ...
}
catch(IOException e)
{
    System.out.println(" File could not be created: " + e);
}
```

## 16.1.2 Report file management

For the handling of the report file generated by the above Drools rule under OS X we create two simple shell scripts. (Under Windows the analog functionality would be implemented using Powershell). One script sends out an email with the CSV file `tmpreport.txt` attached. The second one renames the file to a date based filename, creates a new, empty file version of `tmpreport.txt` and removes all report files older than 90 days. We start with the script `reportmail.sh`, which sends out an email with the file `tmpreport.txt` as an attachment. As explained in detail in chapter 14 we use an Applescript, which we insert inside a shell script. Compared to the example from chapter 14 we just add the command `make new attachment` as shown below:

```
make new attachment with properties {file name:"Macintosh HD:Users:smarthome:shProject:reports:tmpreport.txt" as alias}
```

With that the code for `reportmail.sh` looks as follows:

```
#!/bin/sh
exec osascript << EOF
set recipientName to "Smarthome"
set recipientAddress to "info@keyconceptpress.com"
set theSubject to "Smart Home Report"
set theTime to current date
set theText to "OpenRemote Server Beachroad 4, Uprville daily report"
set theContent to theText & theTime
tell application "Mail"
    set theMessage to make new outgoing message with properties {subject:theSubject, content:theContent,
visible:true}
    tell theMessage
        make new to recipient with properties {name:recipientName, address:recipientAddress}
make new attachment with properties {file name:"Macintosh HD:Users:smarthome:shProject:reports:tmpreport.txt" as
alias}
        send
    end tell
end tell
EOF
```

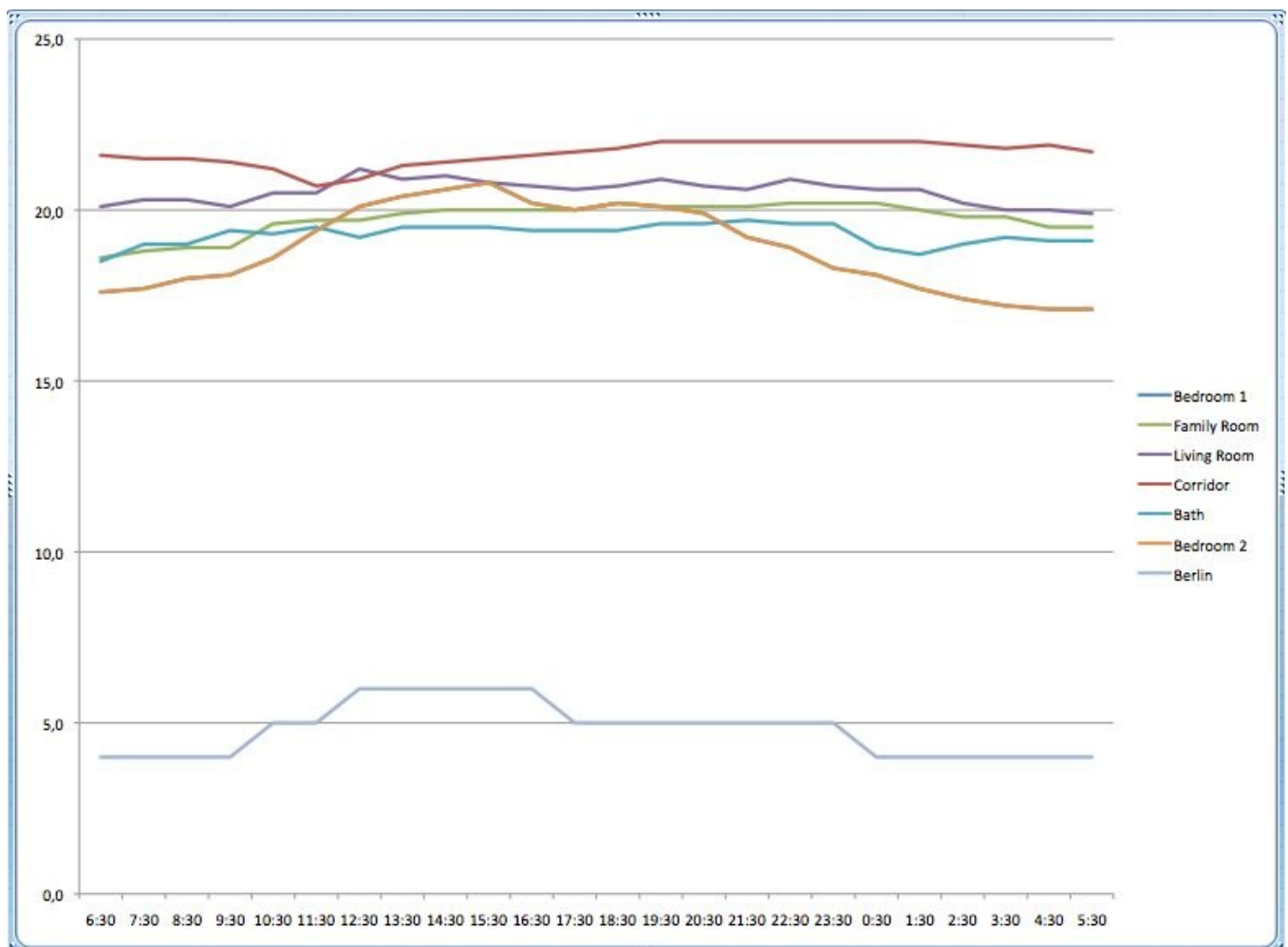
Now we start with the second script `reportmgm.sh`. At the beginning we call the script `reportmail.sh`, which sends out the email with the report file as an attachment. Then the command `sleep 10s` stops the script processing for ten-seconds to ensure the email send process finishes before continuing with the remaining script operations. We then define the variables `day` (containing the current date), `filename` (containing the date based filename) and `path` (containing the path to the report directory). The move command (`mv`) renames the current report file `tmpreport.txt` to a date based filename, the `touch` commands creates a new, empty report file. And finally the command

```
find $path/*.txt -mtime +90 -exec rm {} \
```

finds all files in our report directory which for the last time were modified more than 90 days ago (-mtime +90) and removes them (rm {}). Following the above description our shell script reportmgm.sh looks as follows:

```
#!/bin/sh
# Daily OpenRemote report handling script
# Variables for the script
./reportmail.sh
sleep 10s
day=$(date +%F)
filename="$day.txt"
path="/Users/smarthome/shProject/reports"
mv $path/tmpreport.txt $path/$filename
touch $path/tmpreport.txt
# Removal of report files older than 30 days
find $path/*.txt -mtime +90 -exec rm {} \
```

After thorough testing of our script we can create an OpenRemote command which calls our script reportmgm.sh and write a Drools rule, which calls this command once per day. For a detailed description see chapter 8, where we initiate an iTunes script from Drools. Alternatively we can schedule the start of reportmgm.sh using launchd (OS X) or Task Scheduler (for starting the Powershell variant of the script under Windows) as discussed in chapter 14.



*Figure 16.1 The daily smart home temperature report*



# 17 Appendix

## 17.1 OpenRemote Professional Designer

With Professional Designer OpenRemote offers an extended version of its open source software platform, targeting professional installers and consultants. In addition to the features of Free Designer, OpenRemote Professional Designer comes with

- an images library for applications such as entertainment, lighting, energy and security
- unlimited Z-wave support with automatic device recognition (Free Designer supports up to 10 devices without automatic device recognition)
- remote panel access via VPN
- Pronto CCF infrared protocol support
- AMX NetLinx controller protocol support
- three months of email / phone support
- the ability to backup and restore Professional Designer projects

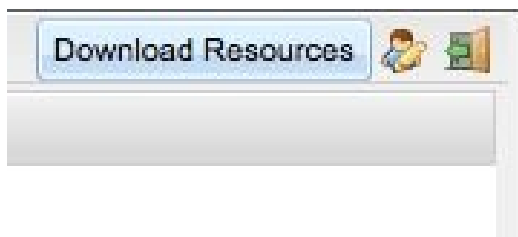
OpenRemote Professional Designer can also be ordered as a bundle with an embedded server called *eBox*. The *eBox* controller hardware comes pre-installed with the Professional Designer controller software, and is available in various configurations, covering a wide range of performance and redundancy requirements. As a third option OpenRemote Professional Designer can be ordered as a cloud service. Starting with one hundred hosted OpenRemote accounts and the optional corporate branding of the client app (iOS, Android and web console), the cloud service is a convenient solution for organizations, which need to manage large number of clients.



## 17.1.1 Installing OpenRemote Professional Designer

OpenRemote Professional Designer resides on <http://www.openremote.com> (in contrast to the OpenRemote open source version, which is located at under <http://www.openremote.org>). To start working with Professional Designer you open <http://designer.openremote.com/login.jsp>

in your Internet browser. After entering the account details the familiar OpenRemote designer windows opens. There are a few subtle differences however, as you discover when taking a closer look. One is the *Download Resources* button at the upper right corner of the user interface, which you need in order to download the professional version of the OpenRemote controller (Figure 17.1).



*Figure 17.1 The Download Resources button in OpenRemote Professional Designer*

Click on *Download Resources* and you are presented with the options to download the software packages *Controller Pro* or *eBox image* (Figure 17.2).



*Figure 17.2 Download Menu for the OpenRemote Professional Controller*

If you do not use the *eBox* select *Controller Pro*. After downloading the software move the file to your local project directory. On a Mac and under Linux do not forget to set the execution permission for the start up file `openremote.sh`, which is located in the `/bin` directory of the controller software:

```
chmod +x ./shProject/ORCPRO/bin/openremote.sh
```

You can now start the controller from the `/bin` directory by typing

`openremote.sh run` (on a Mac and under Linux) or

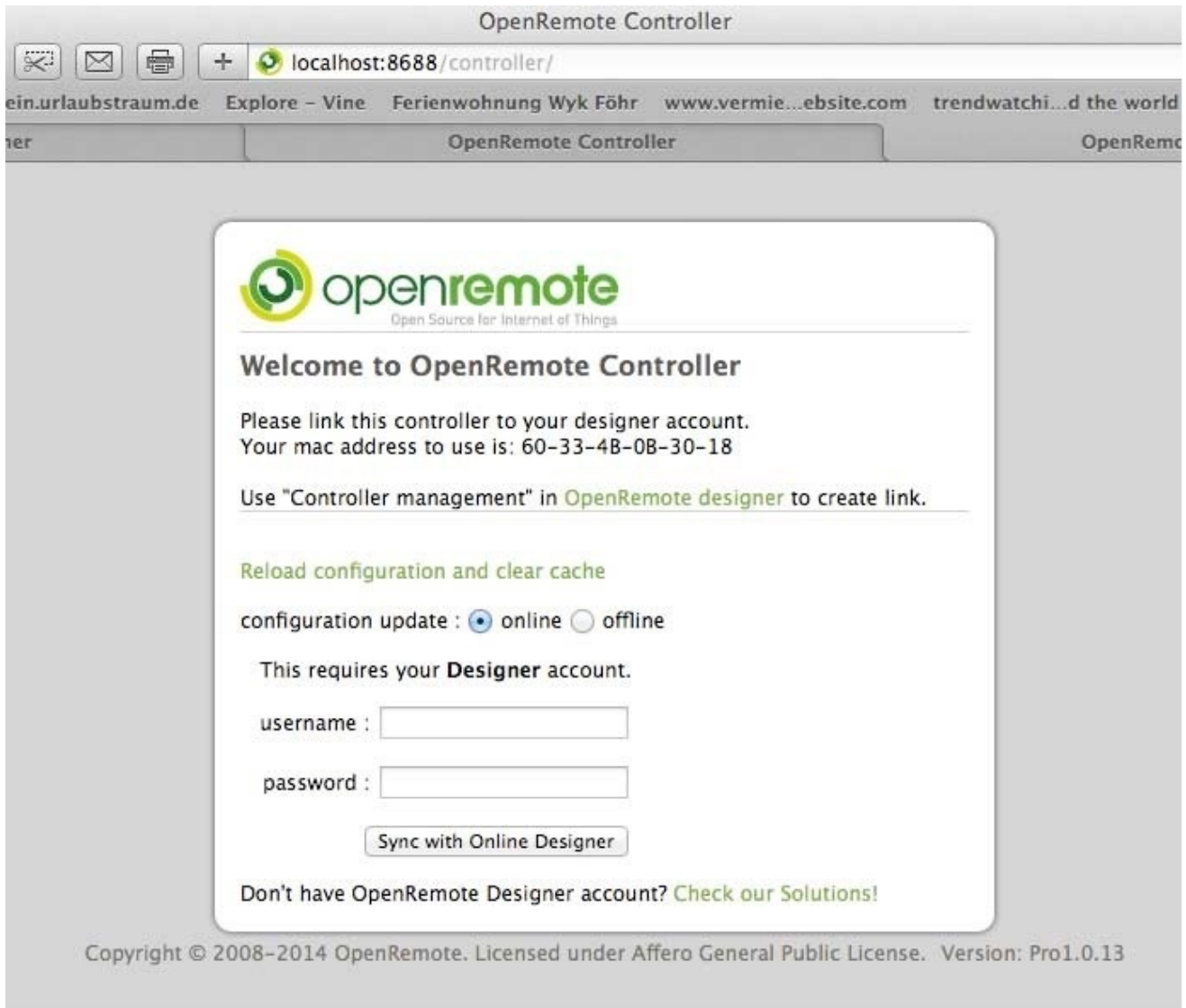
`openremote.bat run` (under MS Windows).

Next you connect with your Internet browser to the controller by opening the URL

<http://controller-machine-IP:8688/controller>

For controller-machine-IP you insert the IP address of the hardware, your controller is running on. If you run the controller on your local machine, the IP address you need to use is simply 127.0.0.1 or localhost, in which case you open the URL

<http://localhost:8688/controller>



*Figure 17.3 The OpenRemote Professional Designer synchronization window*

The *OpenRemote Professional Designer* controller synchronization window opens, showing the MAC address of the controller hardware in the upper half of the window (Figure 17.3). Other than with *Free Designer*, there is one more step required, before the first synchronization can take place. This is to configure the controller MAC address in the *Professional Designer* menu. In order to do that, you have to log on your *Professional Designer* account, open the *Controller Management* dialogue, and add your controller with its associated MAC address (Figures 17.4, 17.5).



Figure 17.4 Controller management in OpenRemote Professional Designer

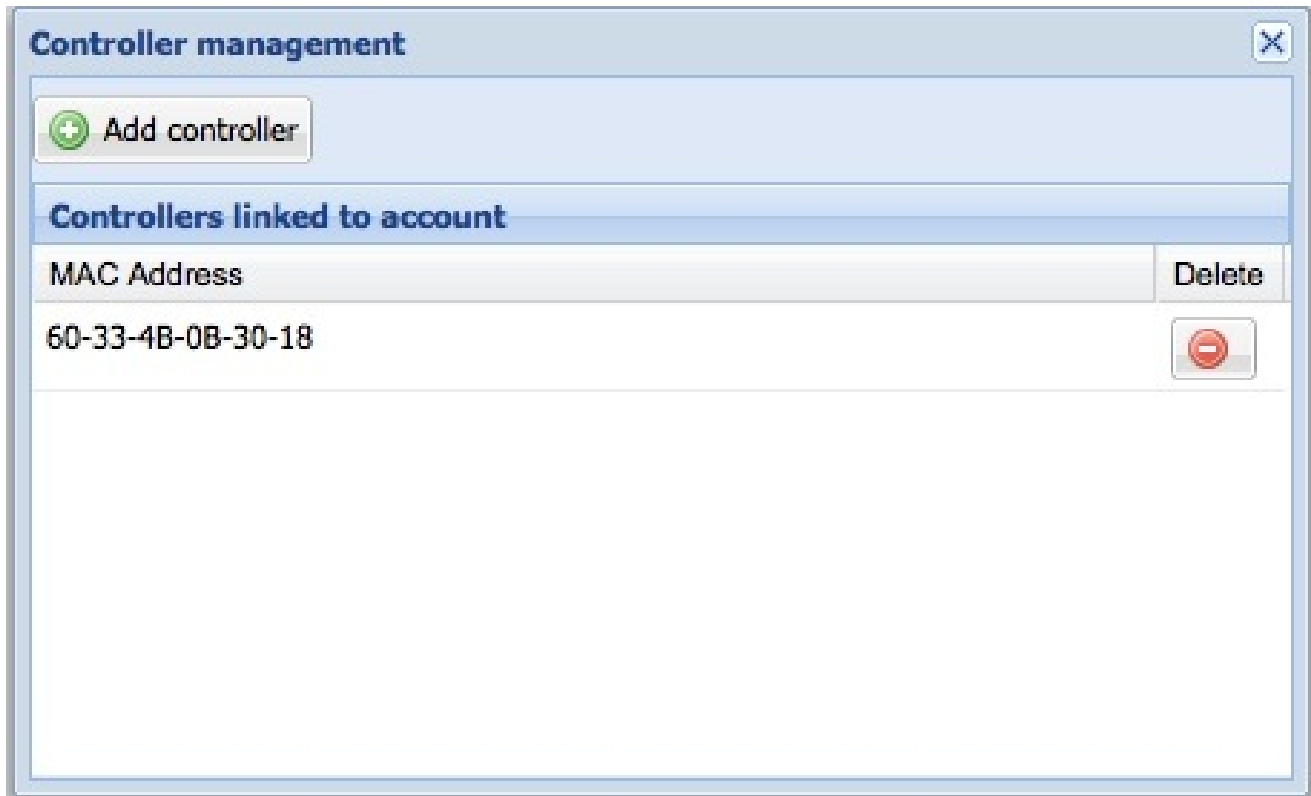
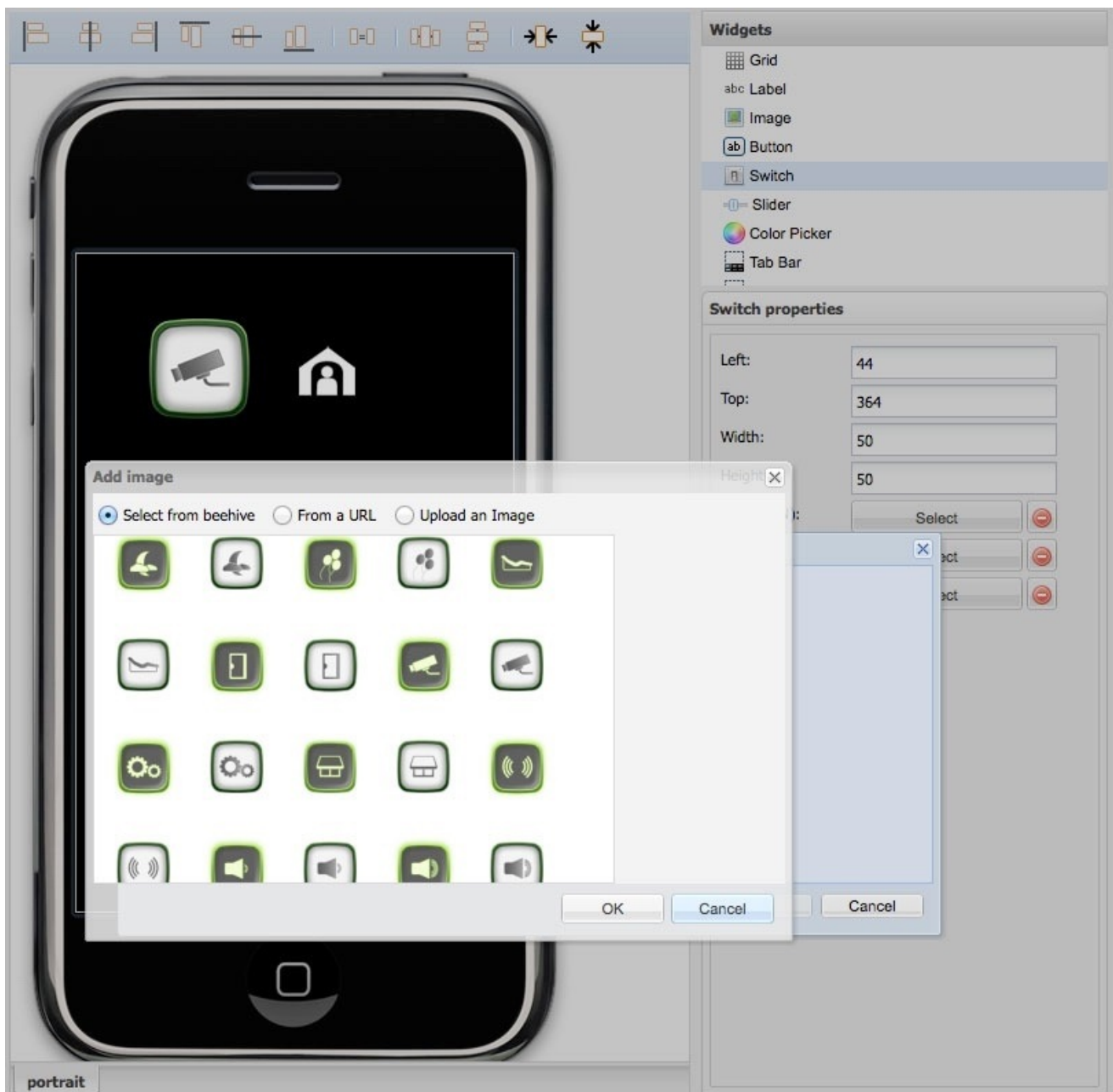


Figure 17.5 Adding a controller in OpenRemote Professional Designer

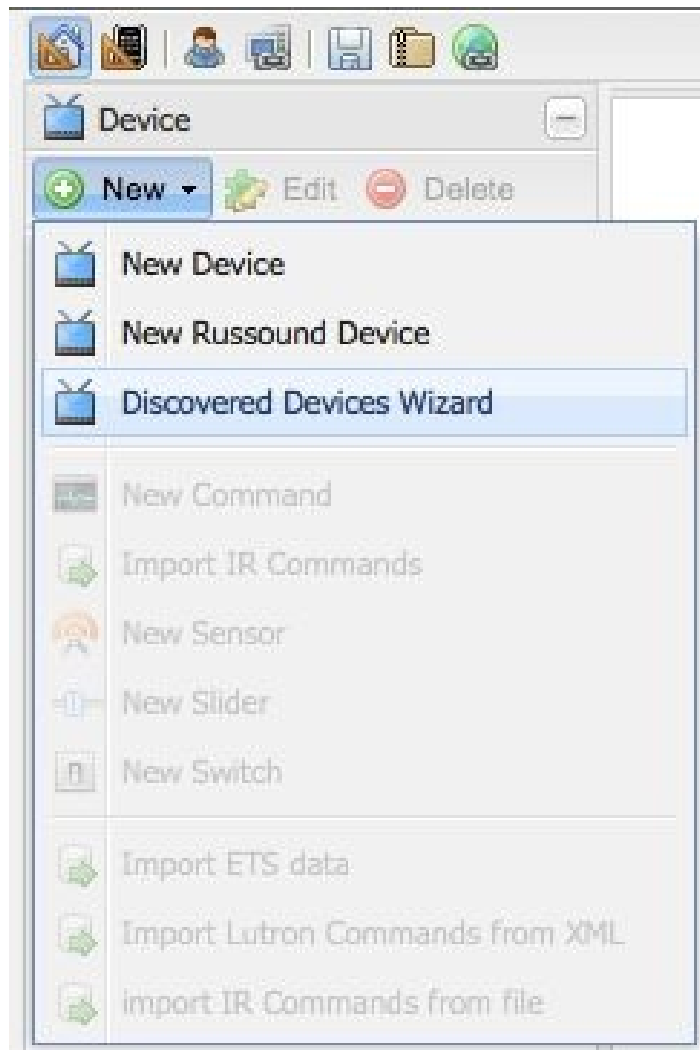
Now you go back to the synchronization window in your Internet browser, enter your *Professional Designer* login details, and select the button *Sync with Online Designer*. A few seconds later the synchronization window shows the message Sync complete.

You can now start designing your smart home system as discussed extensively in chapters 5 through 12. When adding control elements to the GUI of your panel design, you will find the icons and symbols from the images library, after selecting *Add image* (Figure 17.6).



*Figure 17.6 The images library in OpenRemote Professional Designer*

The automatic Z-Wave configuration function of Professional Designer can be found under *Device - New - Discovered Devices Wizard*. The wizard automatically detects all Z-Wave devices and creates the according commands and sensors (Figure 17.7).



*Figure 17.7 The Z-Wave Device Wizard in OpenRemote Professional Designer*



# Bibliography





## Chapter 2:

“Buildings Energy Data Book”. US Department of Energy, March 2012

<http://buildingsdatabook.eren.doe.gov/TableView.aspx?table=2.1.5>

“Annual Energy Review 2011”. US Department of Energy, September 2012

<http://www.eia.gov/aer>

“Energieverbrauch der privaten Haushalte für Wohnen”. Statistisches Bundesamt, Wiesbaden, November 2012

<https://www.destatis.de/DE/ZahlenFakten/GesamtwirtschaftUmwelt/Umwelt/Umweltoeko>

-

“Final energy consumption, by sector.” Eurostat European Commission, April 2012

[http://epp.eurostat.ec.europa.eu/portal/page/portal/energy/data/main\\_tables](http://epp.eurostat.ec.europa.eu/portal/page/portal/energy/data/main_tables)

Angelo Baggini, Lyn Meany. “Application Note Building Automation and Energy Efficiency: The EN 15232 Standard”, European Copper Institute, May 2012

<http://www.leonardo-energy.org/good-practice-guide/building-automation-and-energy-efficiency-en-15232-standard>



## Chapter 3

“IEEE Standard for Local and metropolitan area networks Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)”. IEEE Computer Society, June 2011

<http://standards.ieee.org/getieee802/download/802.15.4-2011.pdf>

“Recommendation ITU-T G.9959 Short range narrow-band digital radio communication transceivers – PHY and MAC layer specifications”. International Telecommunication Union, February 2012

<http://www.itu.int/rec/T-REC-G.9959-201202-I/en>

“EnOcean Wireless Standard ISO/IEC 14543-3-10”. EnOcean Alliance, May 2013

<http://www.enocean-alliance.org/en/home/>



## Chapter 10

DD&M Holdings Inc., “DENON AVR control protocol 5.2a”.2013

[http://usa.denon.com/US/Downloads/Pages/InstructionManual.aspx?FileName=DocumentMaster/US/AVR-3808CISerialProtocol\\_Ver5.2.0a.pdf](http://usa.denon.com/US/Downloads/Pages/InstructionManual.aspx?FileName=DocumentMaster/US/AVR-3808CISerialProtocol_Ver5.2.0a.pdf)

Christian Paetz, Serguei Polterak. “ZWay Manual”, Z-Wave.Me, 2011

[http://en.z-wave.me/docs/zway\\_manual\\_en.pdf](http://en.z-wave.me/docs/zway_manual_en.pdf)

