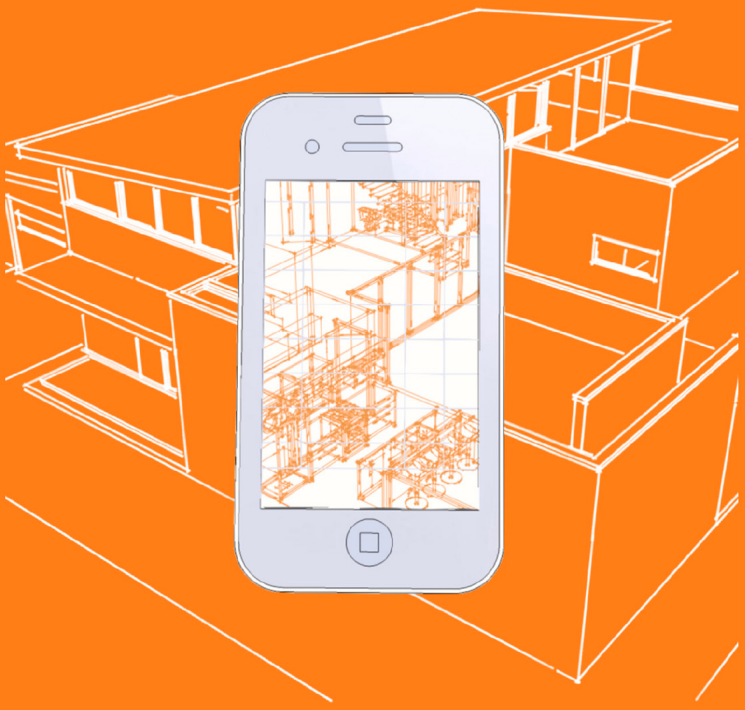
OTHMAR KYAS

HOW TO SMART HOME



KEY CONCEPT PRESS

How To Smart Home

A Step by Step Guide Using Internet, Z-Wave, KNX & OpenRemote

A Key Concept Book by

Othmar Kyas

Published by Key Concept Press e.K., Wyk, Germany
www.keyconceptpress.com
Cover design: Joerg Nestle
ISBN 978-3-944980-00-3

First Edition 10 2013

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.

About the Author

Othmar Kyas is an internationally renowned expert in communication technology and strategic marketing. He is the author of several books, among them Network Troubleshooting, Internet Security and Corporate Intranets. Kyas has held positions as technology advisor, director of strategic marketing and CTO at international technology companies such as Hewlett Packard, Tektronix and Danaher.

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, Webserver 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 a plug and play type of home automation solution, which various vendors and utilities are offering based on closed and proprietary solutions with limited functionality.

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.)

– sans — serif — typeface — font
Sequence of GUI commands (application or operating system)

For the purpose of the exercises in this book I have created the user account smart home (on both OS X and Windows XP/7. 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. 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.

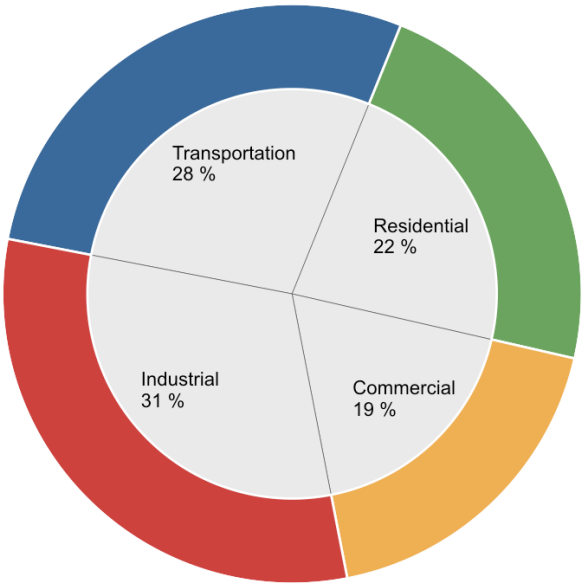
Of similar significance to the changes in what is possible in home automation have been the advances in user interface. The smartphone and tablet revolution has finally brought the personal, universal remote control device to the home. Proprietary, stationary panels and control devices are phasing out, 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 2.2).



US Energy Consumption by Sector 2010

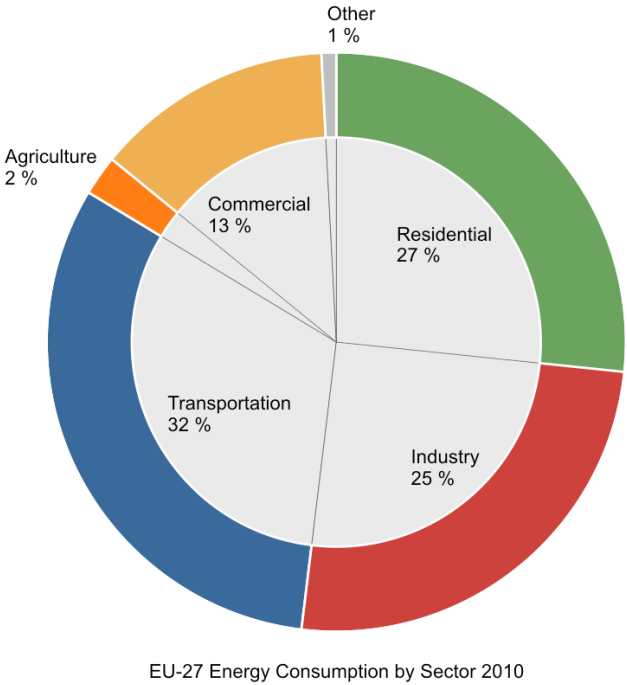
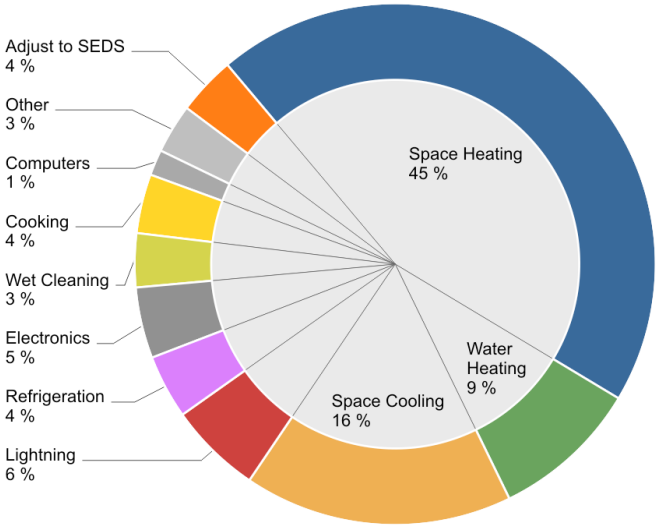
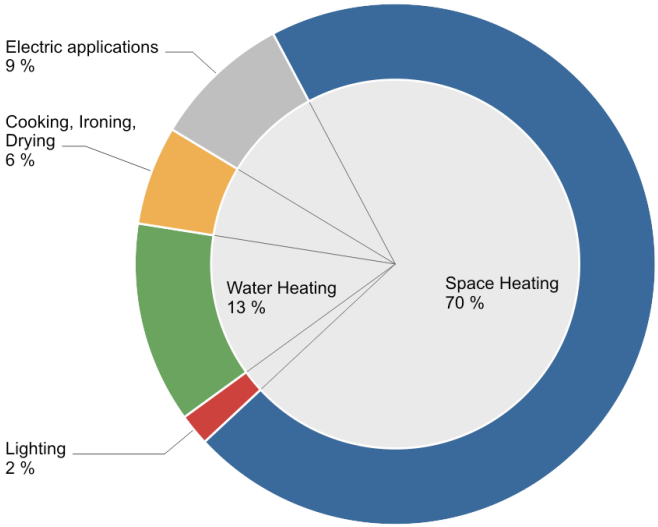


Figure 2.1 US and EU-27 Energy Consumption per Sector 2010 Source: EuroStat 2012, U.S. Department of Energy, September 2012



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.

	Thermal Energy				Electrical Energy			
BACS Class	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
Wholesale&Retail	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)

Class A	**High energy performance BACS** Networked room automation with automatic demand control Scheduled maintenance Energy monitoring Sustainable energy optimization
Class B	**Advanced BACS and some specific TBM functions** Networked room automation without automatic demand control Energy monitoring
Class C	**Corresponds to standard BACS** Networked building automation of primary plants No electronic room automation, thermostatic valves for radiators No energy monitoring
Class D	**Non energy efficient BACS** Without networked building automation functions No electronic room automation No energy monitoring

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". U.S. 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/UmweltoekonomischeGesamtrechnungen/EnergieRohstoffeEmissionen/Tabellen/EnergieverbrauchHaushalte.html

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

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:

– Powerline 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 20kHz to 100MHz. 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 HomePlug standard is able to achieve transmission speeds of up to 500MBit/s. The main advantage of power line communication is the low price for its components and that fact, that no additional wiring is required. The downside is that power line distribution units can significantly 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.

3.3.2 Wireless Transmission

Today, there are a large number of wireless transmission technologies available for building and home automation. Transmission speeds and distance depend on transmission frequency and modulation of the very technology and range from 20kBit/s to 250kBit/s and 60ft (20m) to 3000ft (1000m) respectively. Other important considerations are power consumption and location accuracy. Technology advances have significantly improved all performance aspects of wireless transmission technologies over the past 10 years. The main drivers leading to wireless technology finally to take off in home automation were:

– proprietary home automation systems have migrated towards Internet technologies

– all main building automation systems have become open, international standards

– new standard releases have increased throughput and have further reduced power consumption

– cost and size of components have come down

– integration with wire line based building automation standards such as KNX or LON through gateways.

	First Released	Range (indoor / outdoor)	Maximum Speed	Frequency	Modulation	Standard	Location Accuracy
Z-Wave	1999	30m	250 kBit/s	908.42 MHz	GFSK	IEEE 802.15.4 (*)	10m (**)
ZigBee	2003 / 2006	30m - 500m	250 kBit/s	2.4GHz (Global), 915MHz (North America), 868 MHz (Europe)	QPSK	ITU-G.9959	10m (**)
Wireless Hart	2004	50m / 250m	250kBit/s	2.4GHz	DSSS, O-QPSK	IEEE 802.15.4 (*), IEC 62591	10m (**)
MiWi	2003	20m / 50m	20kBit/s 40kBit/ 250kBit/s	868MHz, 915MHz, 2.4GHz	O-QPSK	IEEE 802.15.4 (*)	10m (**)
EnOcean	2001	30m - 300m	125kBit/s	902MHz (North America) 868MHz (Europe) 315MHz (International)	ASK	ISO/IEC 14543-3-10	N/A
DASH7 (active RFID)	2004	- 1000m	200kBit/s	433MHz	GFSK	ISO/IEC 18000-7	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

While wireless building control for years has been the plan B for lower end, post-construction projects, the adoption of new, reliable low power technologies has changed the industry. Today, Z-Wave, ZigBee, BLE (Bluetooth low energy), and RFID interfaces are available fully integrated in controllable power-outlets, light switches, and household appliances. Many audio and video consumer electronic devices come with WLAN (Wi-Fi), ready to stream content from the Internet, and

ready to be fully controlled via smartphones. A new generation of energy harvesting technology based devices (e.g. EnOcean) are even capable of operating wireless control links exclusively with energy retrieved from the environment through temperature changes, light changes or the mechanical energy when pressing a switch. Table 3.1 lists the main open standards used for wireless building automation today.

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. 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 wire line 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 as a home remote makes the capability of remote building control via Internet or the mobile telephone network a feature which is available as a 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
- integration of web/IP enabled home appliances
- integration of web/IP enabled consumer electronics
- integration of Internet based information and services (supply / demand based rates from energy providers, weather / traffic information, location information).

With that trend, the slow moving market of building automation has been changing quickly and radically. New players and start-ups are taking on the opportunity which the intersection of new technologies and new demands are offering in the market for building and home automation. While the big promise of the “Internet of Things” is still more vision than reality, this vision has become a lot closer in recent years.

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.

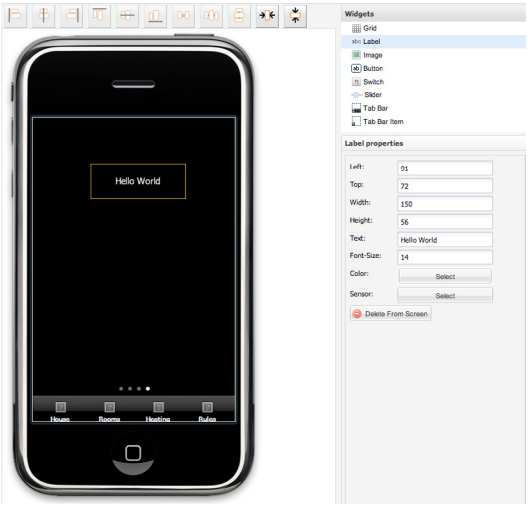


Figure 4.1 Project phase 1: "Hello World" on the custom smart home app

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. (Fig. 4.2)

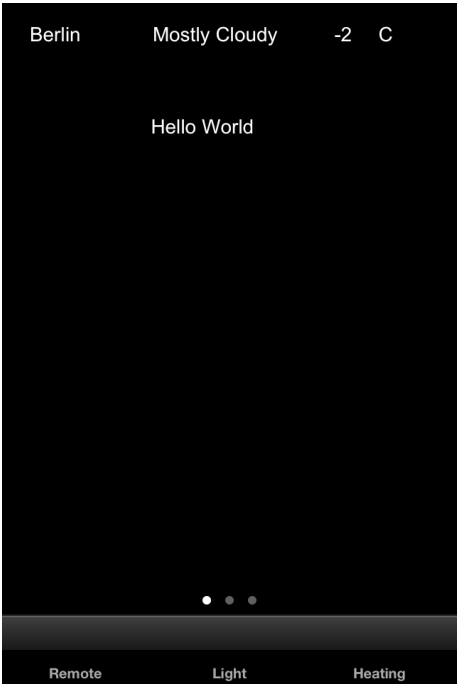


Figure 4.2 Project phase 2: Retrieving and displaying weather information

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.

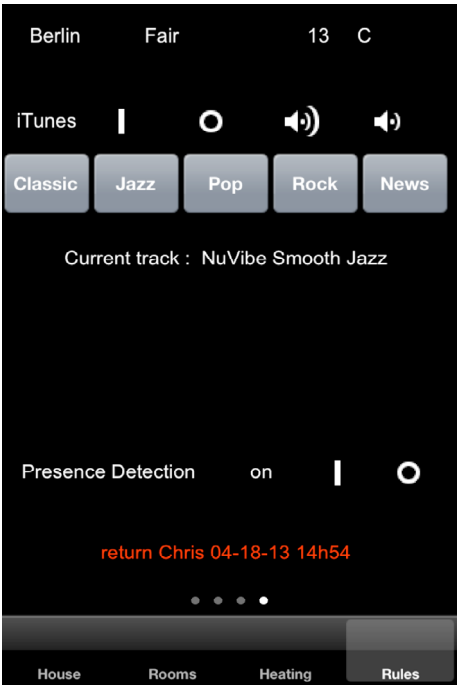


Figure 4.3 Project phase 3: Adding controls for iTunes

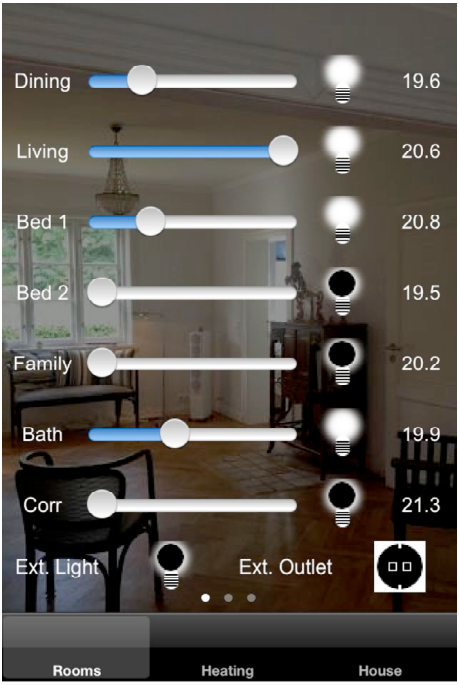


Figure 4.4 Project phase 4: Controlling lights and power outlets

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.

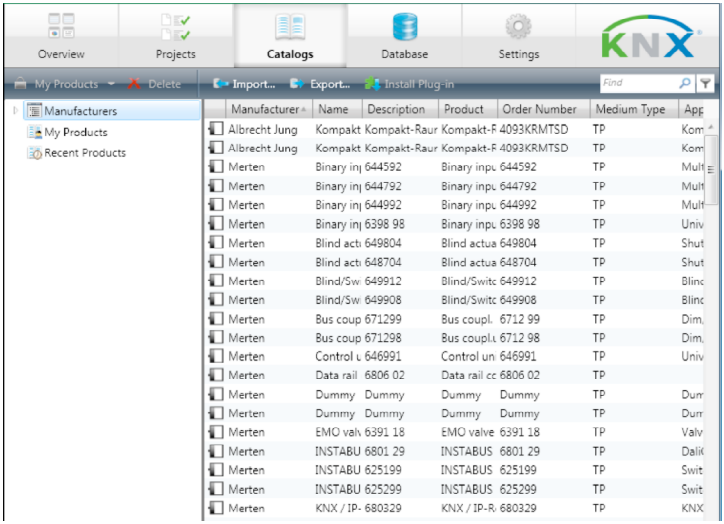


Figure 4.5 Project phase 6: Adding KNX control

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

Sensoring 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 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: OpenRemote

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. It requires little effort and no programming skills. It will, however, allow the reader to build a custom, professional smartphone app, which will serve as our mobile control centre. In addition, the OpenRemote controller, which is supported on OS X, Linux, Windows and other platforms, will run the "always on" automation rules for our project.

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 http://www.knx.org
TCP/IP, UDP, Telnet, HTTP	Internet protocols
Insteon	Home automation system based on power line and radio frequency (RF). http://www.smartlabsinc.com
Shell execution protocol	Execution of shell scripts.
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 ISO/IEC 14543-3-10 http://www.enocean.com
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. http://www.z-wavealliance.org
AMX Controller	AMX Inc. proprietary device control protocol.
1-Wire Protocol	Low data rate communication bus for Maxim Integrated Products. http://www.maximintegrated.com
ISY-99	Control protocol for Universal Devices home automation solution.
panStamp lagarto	Open source protocol for PanStamp wireless modules. http://www.panstamp.com

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. http://www.globalcache.com
XBMC	Open source media player platform. http://xbmc.org
xPL,IRTrans, VLC, FreeBox, MythTV	Commercial and OpenSource home automation solutions.

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 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-2.0.1,* 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 commands you need to know in order to survive at the beginning:

`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

First, we validate if we have Java on the system by typing

```
java -version
```

On a new Mac, most likely you will not find Java, but in response to the above command, the system will propose to download and install the Java Runtime Environment (JRE). After its installation, you should see something like what is below in response to the command `java -version`:

```
java -version

java version "1.6.0_37"

Java(TM) SE Runtime Environment (build 1.6.0_37-b06-434-11M3909)

Java HotSpot(TM) 64-Bit Server VM (build 20.12-b01-434, mixed mode)
```

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 `$PATH` variable to the definition of our local user account. We save .bash_profile in TextEdit, close the terminal window and open it up again (which forces the system to process the new `$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
```

The start script for the OpenRemote controller is *openremote.sh* in */*

shProject/ORC/bin. When we do a long listing (`ls -l`) of the files in the ORC/bin directory we can see that we do not have the execution right for the 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 5.2) 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.

5.4 Installation under Windows 7 and Windows XP

Under Windows (XP/7) we first 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, and we should see an entry that says something like

```
Java SE Development Kit 7 Update 9
```

If you find you do not have Java installed, you need to go to *http://www.oracle.com/technetwork/java/javase/downloads/index.html* and download and install the Java SE Development Kit for your Windows version. (While you will not do any Java Development work, OpenRemote will not work with the Java Execution Environment (JRE) under Windows 7, under Windows XP it will). Before the next step, determine the exact directory path of your Java installation. It will be something like

```
C:\Program Files\Java\jdk1.7.0_09\
```

We now need to go to the Windows menu for environment variables. We open System in Control Panel. On the Advanced tab, we click 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
```

```
JAVA_HOME=C:\Program Files\Java\jdk1.7.0_09
```

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-2.0.1* 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.

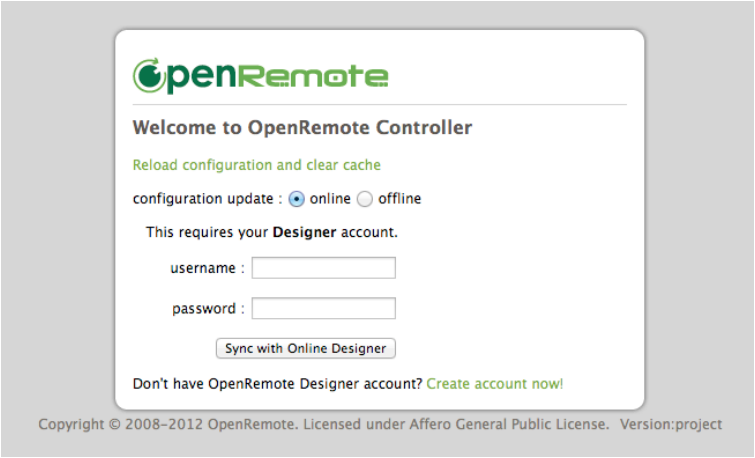


Figure 5.1 Open Remote Controller login screen

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 consist 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).

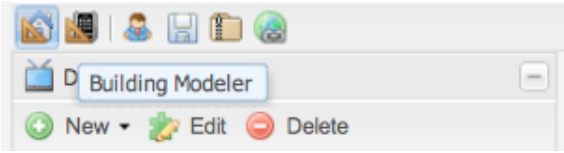


Figure 5.2 Selecting the Building Modeler Screen in Open Remote Designer

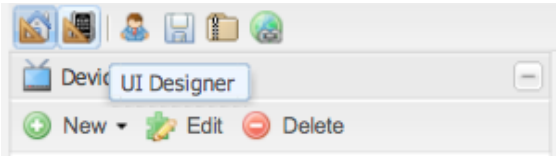


Fig. 5.3 Selecting the UI Designer Screen in Open Remote Designer

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.

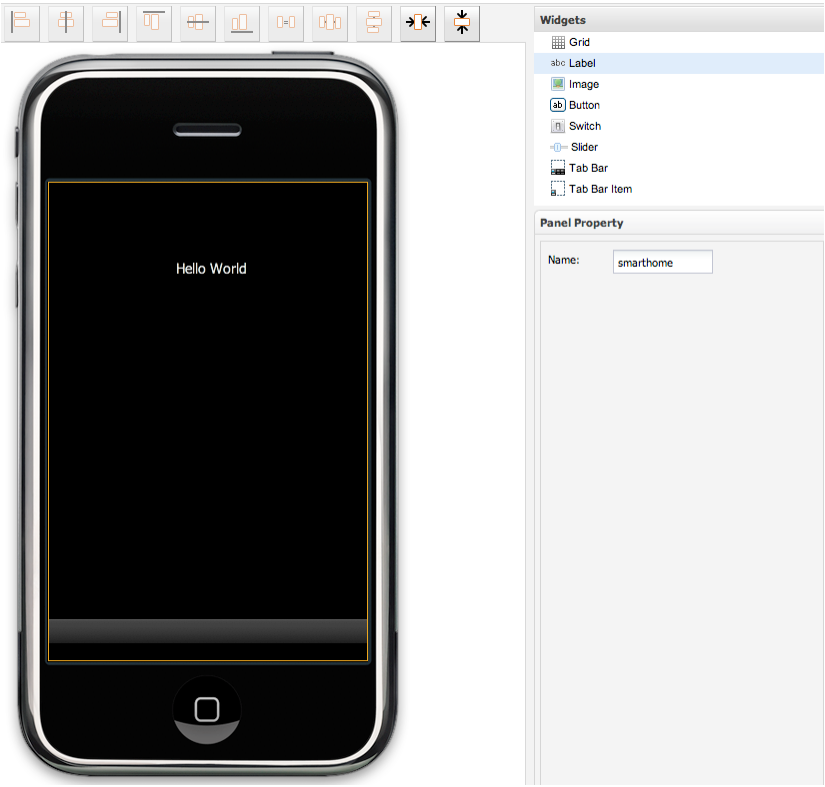


Figure 5.4 First steps with OpenRemote Designer

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”. 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, we start the Network Utility by selecting Applications — Utility — Network Utility. Under Windows (XP / 7) we open the terminal window and type

```
ipconfig /all
```

We enter the controller IP address, followed by `:8080/controller` in the configuration screen of our OpenRemote panel app, something like

```
http://192.168.178.41:8080/controller
```

When you start the OpenRemote app for the first time, it starts right with the configuration screen. Once the app has loaded, access to the configuration screen is gained by shaking the phone:

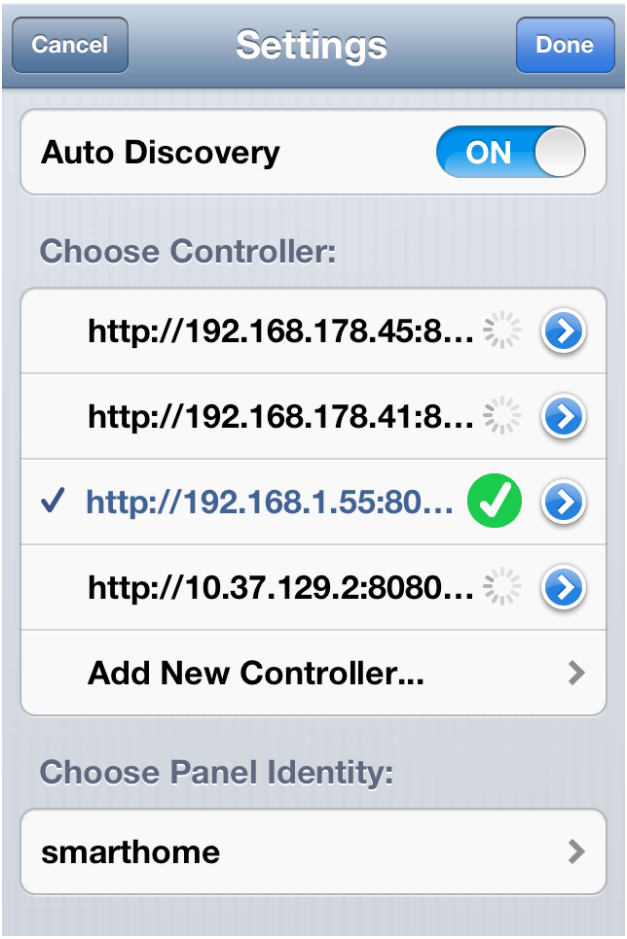


Figure 5.5 The OpenRemote App's configuration screen (shake phone for access)

As the final step, we select the panel identity of our online-design (Figure 5.5), which in our case shall be `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 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 updated in *panel.xml*. After a restart of our smartphone app, we see “Hello World” on the screen of our smart home app. Now 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....

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 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.

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 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 behaviour 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 `*/`. 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 `turniAlarmOn.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 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.

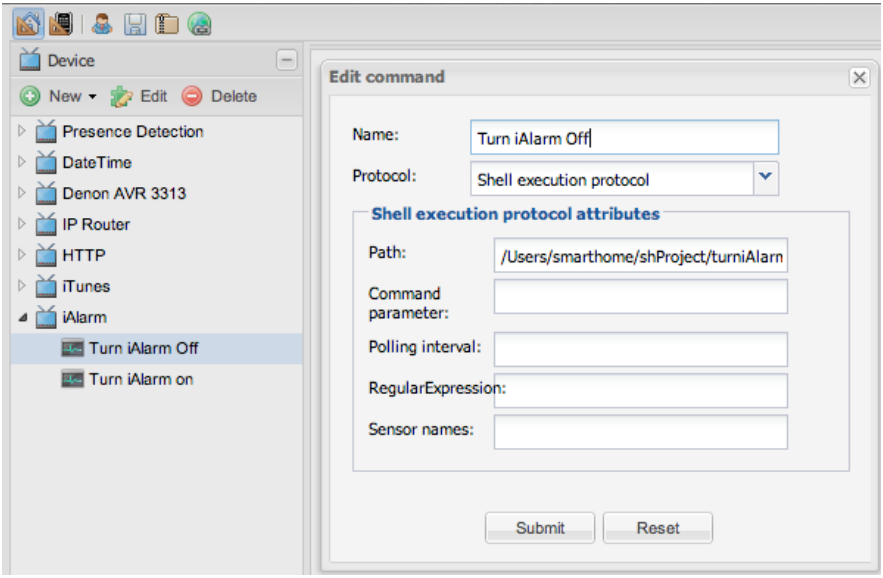


Figure 9.1 OpenRemote Command Definition Turn iAlarm Off for OS X

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).

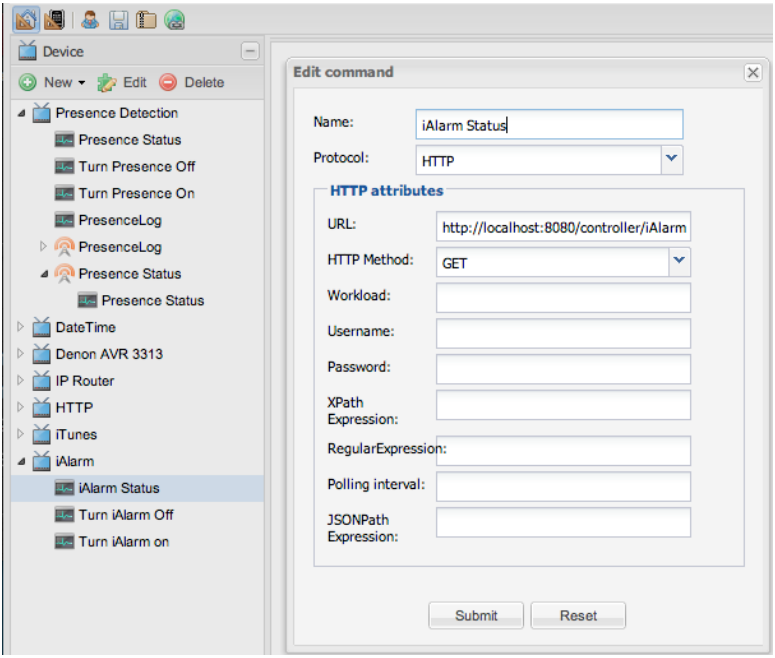


Figure 9.2 OpenRemote Command Definition for iAlarm Status using HTTP GET

Now we can create the sensor *iAlarm Status* using the command *iAlarm Status* we just defined. (Figure 9.3)

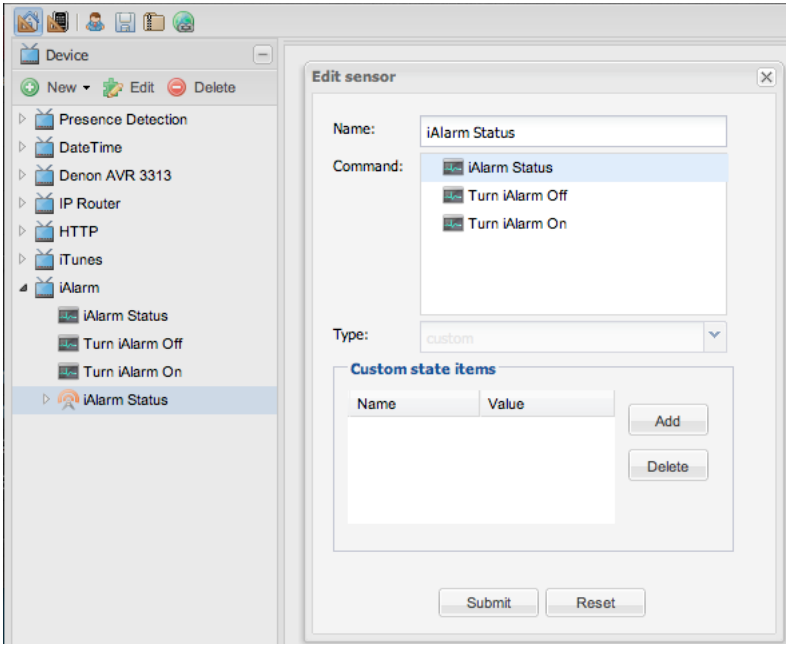


Figure 9.3 OpenRemote Sensor Definition for iAlarm Status

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.

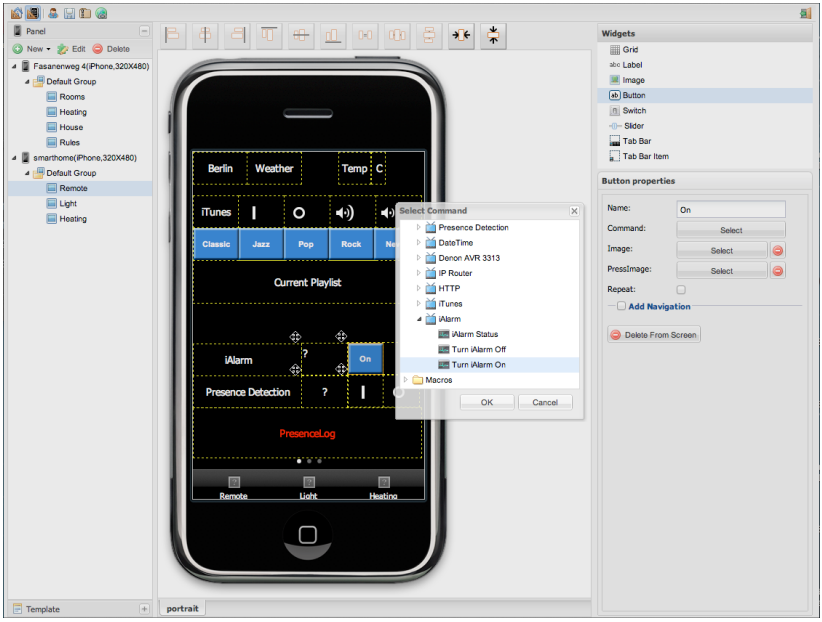


Figure 9.4 OpenRemote GUI design for iAlarm control

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.. 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

seconds	0-59 or * for any value
minutes	0-59 or * for any value
hours	0-23 or *
day-of-month	1-31, L,?,*
month	1-12 or JAN-DEC
day-of-week	1-7 or SUN-SAT

Year	1970-2199

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 (`/`). 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)
```

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.

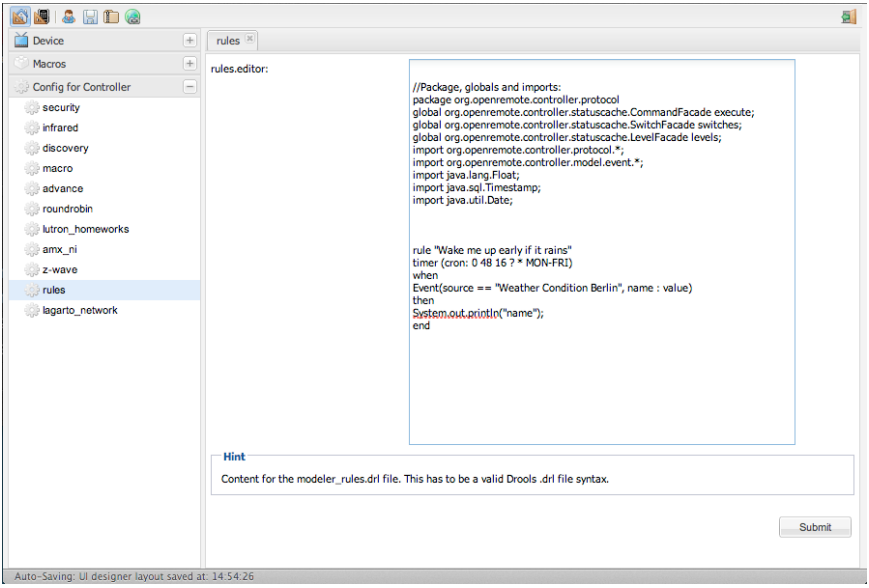


Figure 9.5 OpenRemote rules definition editor

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 playist 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....

14. Cold Start: Launch Automation

If the case of the smart home controller reboots (due to a power outage or due to maintenance downtime), we need to ensure, we have a running system in a defined operating state in place once the system is back up. For this purpose we need to have a script in place that automatically restarts the OpenRemote controller after the reboot of the system and documents the restart in a log file. For this purpose, under OS X /Linux we will use `crontab` under Windows `Task Scheduler`.

14.1 Windows Task Scheduler

Under Windows scheduling a task to be executed at start-up 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.exe`, enter `run` in the Add arguments field, and select Next again, to finish the creation of the task. (Figure 14.1)

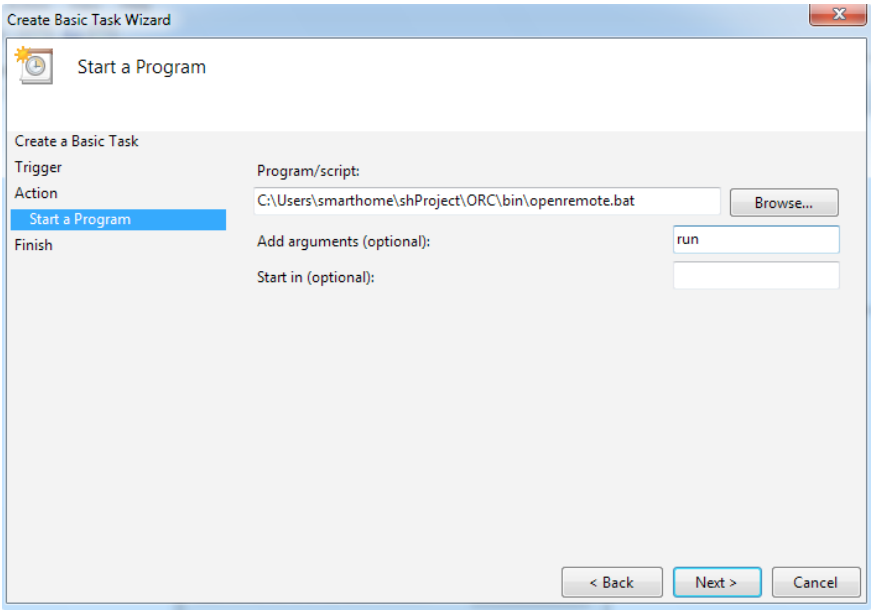


Figure 14.1 Task Scheduler configuration under Windows 7

14.2 OS X Crontab

Crontab is a popular tool under OS X and Linux that is used for scheduling tasks at specific times or at reboot. Although the preferred method on OS X for running automated jobs recently has become `launchd`, for our simple schedule task we will use `crontab`. It is still widely used and easier to understand than the XML based `launchd`.
It consists of the special file *crontab*, which can be created under root for the entire system or for a specific user. In any case for the initial creation of either file type super user status (using the `sudo` command) is required. The crontab commands are:

`crontab [-u account] -e`	create or edit crontab file
`crontab -l`	display the content of crontab file
`crontab -r`	remove the crontab file

In order to create a crontab file for our user smarthome we open a terminal window and type:

```
sudo crontab -u smarthome -e
```

to get the following response:

```
crontab: no crontab for smarthome - using an empty one

crontab: installing new crontab
```

Once the file is created, we can use the crontab commands without the preceding `sudo`. After entering

```
crontab -e
```

the crontab file opens in the vi editor, the ancient UNIX editor. Unfortunately there is no real easy way to avoid working with vi for crontab files, but for the few lines we enter, we will survive.

If you have never worked with vi, for what we are doing here, you just need three commands:

`i`	command to switch to editing (insert) mode
`escape`	command to switch from editing mode to command mode
`:wq`	save file and quit

Before you start working with vi you need to understand that there are two main operating modes of the editor:

– the command mode

– the insert (editing) mode

When the crontab file opens in vi, you see its content, but you are still in command mode. Before you can make changes you need to type the command `i` (character `i`). Now you can move around with the cursor using the arrows and make any changes. Once you are done you hit the `escape` key and you are back in command mode. Here you enter `:wq` and your edits are saved and you have quit vi.

The crontab syntax is quite easy. Each line corresponds to a so called cron job and has the following format:

```
{activation parameters} {command #1} ; {optional command #2} ;
{optional command #3} ; {etc, commands end on the line ending}
```

The activation parameters contain the information about when to run the job. They use the following convention:

`@reboot`	job runs only when the system boots
`1 2 3 4 5`	five digit time specifier

- where –

`1`	minute [0-59]
`2`	hour [0-23]
`3`	day [1-31 ... 29,30,31 may not activate during all months]
`4`	month [1-12 or 3-letter names “Jan,Feb,Mar”]
`5`	weekday [0-7 or 3-letter names “Mon,Tue,Wed”]
`*`	every
`* * * * *`	would schedule every minute
`5 * * * *`	five minutes after every full hour
`10 20 * * *`	every day at 8:10 p.m.

To get our first cronjob up and running using a `hello world` example is always a good idea in order to validate we are doing things right. The plan is to set up a job which prints the string `hello world` every minute. Since the cron service sends its output to `stderr` and `stdout`, we redirect `stderr` to `stdout` (command `2>&1`) and both to the file

`crontest.log`:

```
>> ./shProject/crontest.log 2>&1
```

Without the redirection to `crontest.log`, crontab would send the output of `stdout/stderr` to the users mailbox. We now enter our crontab command

```
* * * * * echo "hello world!" >> ./shProject/crontest.log 2>&1
```

and save it. The message

```
crontab: installing new crontab
```

confirms, that our job is scheduled and after about a minute, we should see the log file `crontest.log` in our home directory. Looking at its content with the `cat` command we see our string `hello world`. We now know what we are doing is working and can delete the test job we just created. We can now start to create the command which starts the OpenRemote controller

`./openremote.sh run`

in the directory

`./shProject/ORC/bin`.

Since OpenRemote needs to be started from the OpenRemote `/bin` directory, we create a small batch file called *orBoot.sh,* which contains just two lines: The first one, which changes the directory to the OpenRemote `/bin` directory, and a second line with the actual start command:

```
#!/bin/sh

cd /Users/smarthome/shproject/ORC/bin

./openremote.sh run
```

Don't forget to add execute permission to the file after its creation (`chmod +x orBoot.sh)`. Now, we can create the crontab entry, which calls the above shell script whenever the system boots up. Since some processes of the OpenRemote startup sequence require specific permissions, we start our crontab job as super user and type:

```
sudo crontab -e
```

In the actual crontab file we just have to type a single line:

```
@reboot /Users/smarthome/orBoot.sh > /Users/smarthome/orreboot.
log 2>&1
```

The command `@reboot` indicates that the job will only run at reboot. Then, the shell script of which we want to start including the file path follows. And finally, we redirect `stderr` to `stdout` (`2>&1`) and `stdout` to `/Users/smarthome/orReboot.log`. We are done.

In case you run into problems, in general the two most common root causes for problems with crontab files are:

– the script which will be started does not have the required permissions

– the environment variables required to run the script are unavailable. (Crontab is ignorant of environment variables, which is why they need to be specified as part of the script that will be executed)

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.)

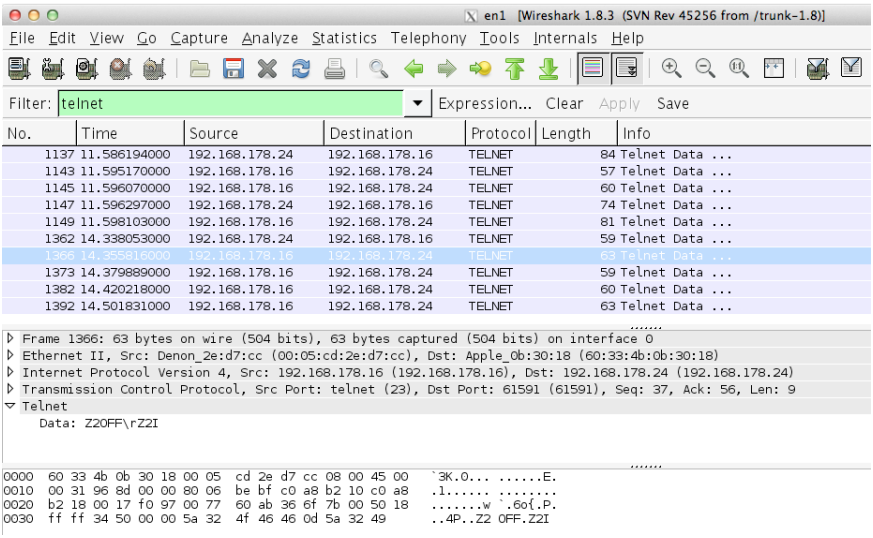


Figure 15.1 Wireshark protocol trace of the Denon 3313 Telnet response to a "Z2?" command

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, which periodically (e.g. every minute) writes a value (e.g. 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, so that the watchdog 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.

As an example, we want to monitor our key process, the OpenRemote controller. The file *watchdog.html* contains the watchdog counter. The script *watchdog.sh* reads the content of *watchdog.html* and sends an alarm message if the value equals to zero. If the value is unequal to zero, it decrements the value by ten and stores the result as the new watchdog value to *watchdog.html*. Then we define the OpenRemote command *setWatchdog*, which sets the content of *watchdog.html* to 100. And finally we create an OpenRemote rule, which calls the command setWatchdog every minute:

```
//Rule to execute command "setWatchdog" every minute

rule "OR Watchdog"

timer (cron: 1 * * * ?)

execute.command("setWatchdog");

System.out.println("!!!Watchdog set to 100!!!");

end
```

Now, as long as the OpenRemote process works properly, every minute the content of *watchdog.html* is set to 100. If OpenRemote stops working, after 10 minutes the watchdog value will reach zero, and the watchdog will set an alarm. In addition to an alarm message the watchdog could report the event to a log a file and try to restart the OpenRemote process.

16. Appendix

16.1 ETS4 Installation on a Mac Using Parallels and Windows 7

Non-English versions of Windows-7 might have to overcome some installation challenges when the ETS4 install procedure tries to install Microsoft SQL Server 2008. If your ETS4 installation stops at the Microsoft SQL Server installation step you need to open the Windows Registry Editor by typing regedit in the Windows start field and selecting the regedit application. You then go to:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\
Perflib\
```

where you will see, that e.g. in \007 (code for German) there are no values for counter and help, other than in \009 (code for English). You need to create both counter entries as new under 007 and copy the values over from \900. Your installation should then finish without problems.

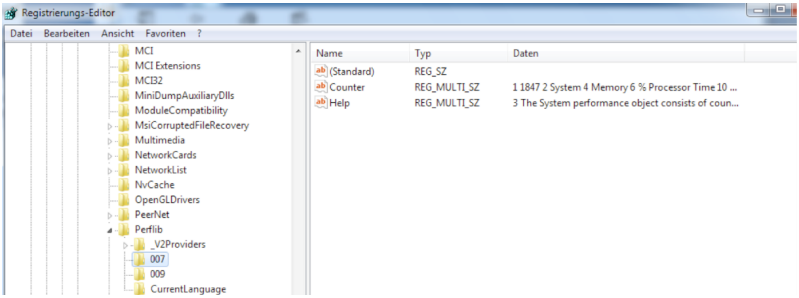


Figure 16.1 Editing the Windows 7 Registry for Microsoft SQL Server installation on a Mac using Parallels and Windows 7

17. 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". U.S. 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/UmweltoekonomischeGesamtrechnungen/EnergieRohstoffeEmissionen/Tabellen/EnergieverbrauchHaushalte.html

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

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 radiocommunication 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

Christian Paetz, Serguei Polterak. “ZWay Manual”, Z Wave.Me, 2011

http://en.z-wave.me/docs/zway_manual_en.pdf