Get Ready for Activity – Ambient Day Scheduling with Dementia

Applicable software components

Deliverable Name:	D2.2 - Applicable softrware components
Deliverable Date:	31.01.2018
Classification:	Report / public
Authors:	Quirino Nardin, Walter Ritter, Tom Ulmer, Sandro Emmenegger, Beat Sauter,
Document Version:	V1.0
Project Coordinator:	University of Applied Sciences Vorarlberg (FHV), Austria

Project Partners: Bartenbach GmbH
Fachhochschule St. Gallen
Apollis – Institut für Sozialforschung und Demoskopie O.H.G.
Intefox GmbH
Altersheim Stiftung Griesfeld
EMT – energy management team AG
CURAVIVA Schweiz
Tirol Kliniken GmbH – Hall

The project GREAT no AAL-2016-023 is funded through the AAL program of the EU

Preface

This document forms part of the Research Project "Get Ready for Activity – Ambient Day Scheduling with Dementia (GREAT)" funded by the AAL 2016 "Living well with dementia" funding program as project number AAL-2016-023. The GREAT project will produce the following Deliverables:

D1.1 Medical, psychological, and technological framework

D2.1 Applicable hardware components

D2.2 Applicable software components

D2.3 Field tested hardware components

D2.4 Field tested software components

D3.1 Implementation report

D3.2 Field test report

D4.1 Communication strategy

D4.2 Stakeholder management report

D5.1 Report on market analysis

D5.2 Dissemination plan

D5.3 Final business plan

The GREAT project and its objectives are documented at the project website http://uct-web.labs.fhv.at. More information on GREAT and its results can also be obtained from the project consortium:

Prof. Dr. Guido Kempter (project manager), University of Applied Sciences Vorarlberg (FHV), Phone: + 43 5572 792 7300, Email: guido.kempter@fhv.at

Hermann Atz, Institute for Social Research and Opinion Polling OHG (APOLLIS), Phone: +39 0471 970115, Email: hermann.atz@apollis.it

Mag. Wilfried Pohl, Bartenbach GmbH, Phone: +43-512-3338-66, Email: wilfried.pohl@bartenbach.com

Quirino Nardin, Intefox GmbH, Phone: +43 699 1900 8889, Email: info@intefox.com

Dr. Marksteiner Josef, Tirol Kliniken Hall, Phone: +43 (0)50504 33000, Email: josef.marksteiner@tirol-kliniken.at

Mag. Tom Ulmer, University of Applied Sciences St. Gallen (FHS), Phone: +41 71 226 17 41, Email: tom.ulmer@fhsg.ch

Beat Sauter, energy management team ag (emt), Phone: +41 71 660 02 86, Email: beat.sauter@emt.ch

Anna Jörger, CURAVIVA Schweiz, Phone: +43 (0)31 385 33 45, Email: a.joerger@curaviva.ch

Cornelia Ebner, Stiftung Griesfeld, ÖBPB – APSP, Phone: +39 (0) 471 82 63 43, Email: cornelia.ebner@griesfeld.it

Content

1. Great Prototype Software Components

1.1 System Overview

The GREAT system should be usable in widely varying environments. Therefore, a highly modular approach has been chosen. Individual components like light, sound, and scent modules can be used individually or in combination with one another. The system also gathers data from motion detectors and physiology sensors worn by caregivers to detect potential activity/relaxation levels of persons in a room. The system can be controlled via a mobile app manually as well as dedicated hardware buttons, that can be added to the system based on local requirements (see Figure 1 for an overview).

One important principle of the GREAT system is that users must always be in full control of the system, meaning that they will always be able to start/stop actions manually.

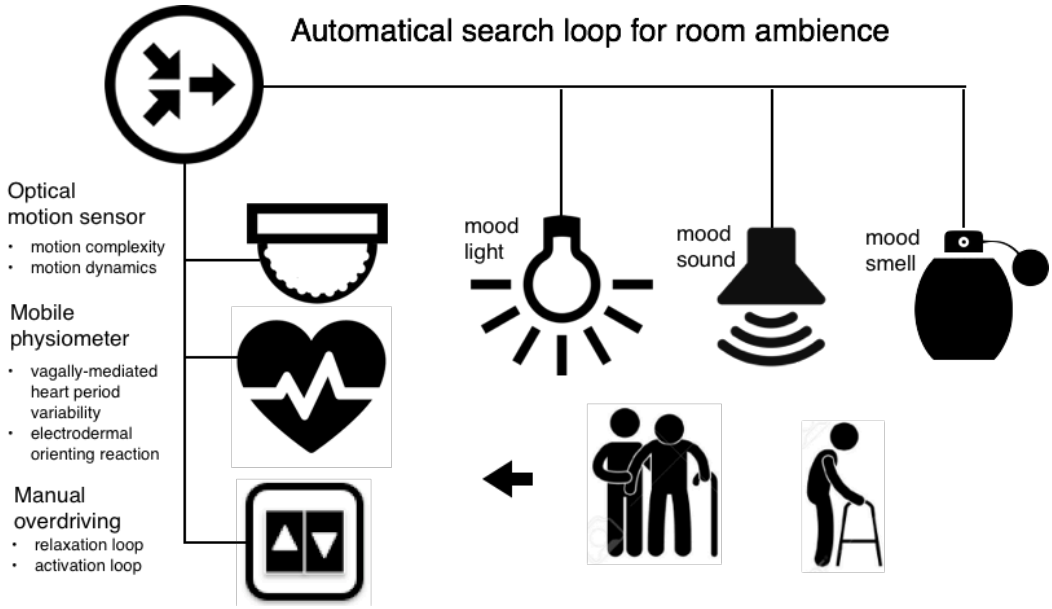


Figure 1: GREAT Components Overview, Source: GREAT consortium.

In a first phase, the system gathers data from motion detectors and physiology sensors as foundation for an analysis of typical patterns. Caregivers are also able to

give their current impression of the patient's state (e.g. whether they are very relaxed up to highly activated). During this time caregivers can manually trigger activation- or relaxation-cycles. For learning purposes, the system logs every activity.
In a second phase, the system recommends to caregivers the triggering of activation-/relaxation-cycles, when it detects certain situations. The actual triggering of these cycles however is still up to the caregivers.
By the end of the field tests, the system should have gathered enough data to trigger activation/relaxation cycles automatically.

1.2 System Architecture

The GREAT system is comprised of a main controller, light-, scent-, and sound-modules, sensors for room-based motion activity and physiological data capturing for selected caregivers, as well as a cloud based storage and configuration layer (see Figure 2).

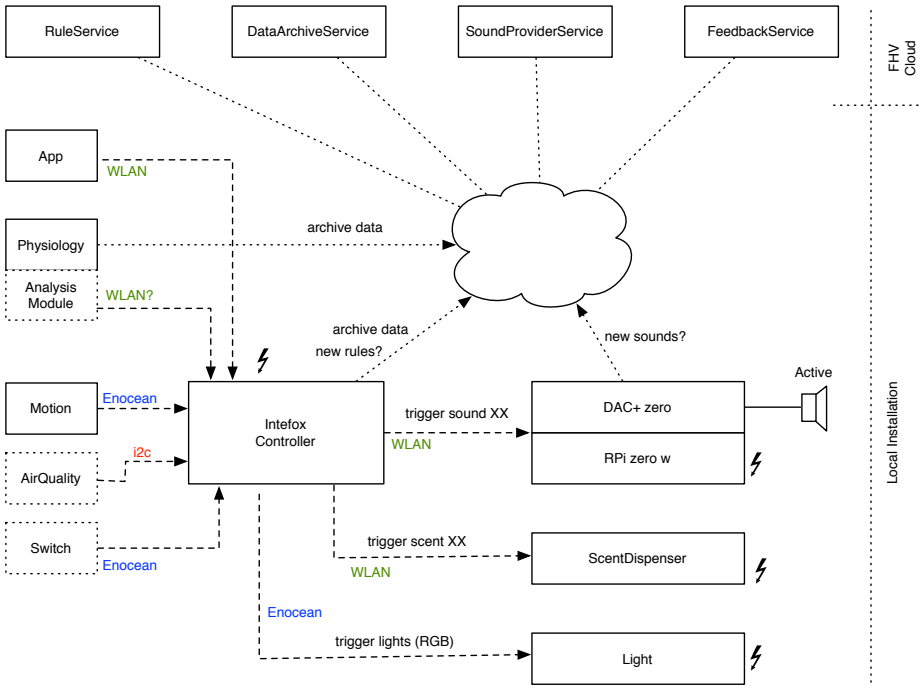


Figure 2: GREAT Distributed System Overview, Source: GREAT consortium.

The main component of the GREAT system is the local controller, that acts as a coordinator among the different other components. It runs an already available middleware solution for smart buildings control (provided by Intefox) with built-in support for a wide range of common building automation protocols (e.g. DALI,

EnOcean, KNX,…). The middleware system is also highly extensible to allow for integration of new components, either based on standard protocols or application specific ones.

Multiple GREAT systems can be used at the same time in multiple locations, each of them can be tailored to local requirements.

The controller and the modules are connected over wireless links (EnOcean, WLAN, optionally Bluetooth LE). A WLAN is provided by the GREAT controller itself to allow for efficient connection of components. For internet connection, an Ethernet port with publicly accessible Internet is required. Alternatively, also a USB WLAN adapter can be used to connect the system to an existing WLAN.

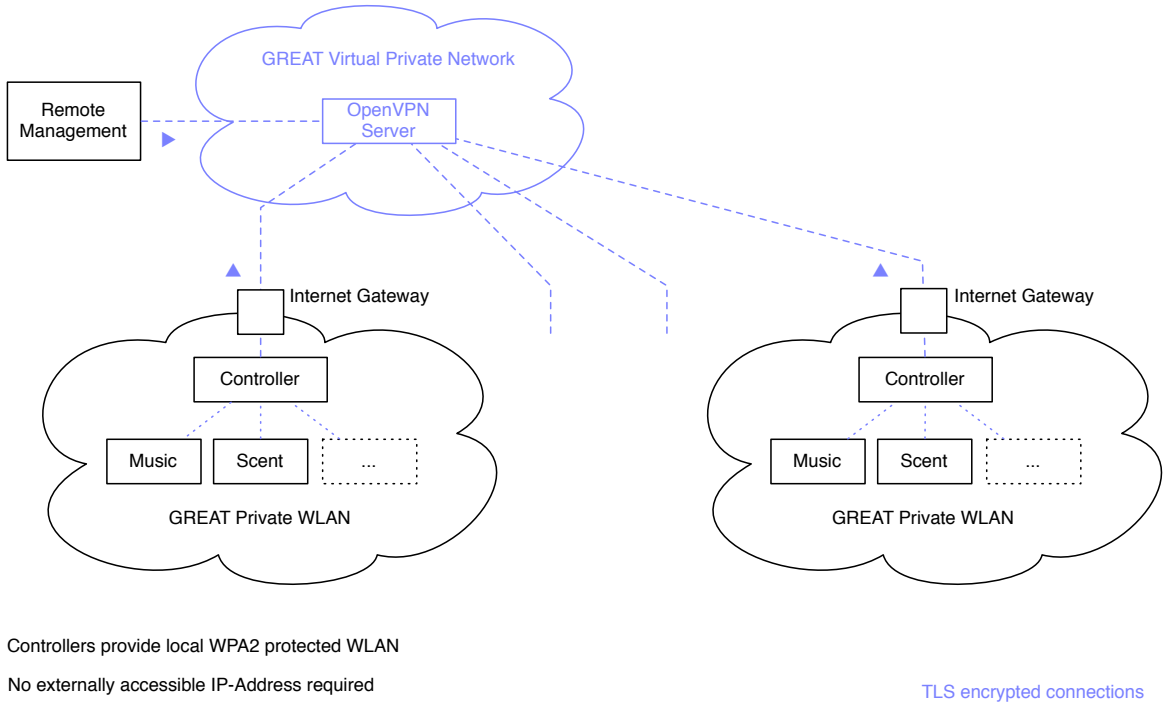


Figure 3: Distributed GREAT installations connected over VPN.

All communication from the controller to the Internet is encrypted. To allow for remote administration of the system, individual controllers are connected into a virtual private network, thus not requiring an externally accessible IP-address (see Figure 3).

2. Controller

The software stack of the GREAT controller is based on the open source Raspbian Jessie Linux distribution for Raspberry PI single board computers. The three main software modules of the controller are:

- Intefox Middleware stack for building automation
- Wireless Access Point functionality including routing

- OpenVPN Client for remote management

2.1 Intefox Middleware

The main purpose of the middleware is to hide the details of the individual component communications und provide unified access to individual objects/components. In this way, e.g. additional lights could be easily added into the GREAT system, even if they used some different building automation standard like KNX or DALI. At the GREAT software layer, they would be treated the same. In addition to this hardware abstraction layer it also provides a broad list of features that allow for rapid prototyping (drag & drop configurations), ready-made mobile apps, and event logging functionalities.

The basic architecture of the Intefox middleware software is based on an OSGi (originally Open Services Gateway initiative) software layer (see Figure 4).

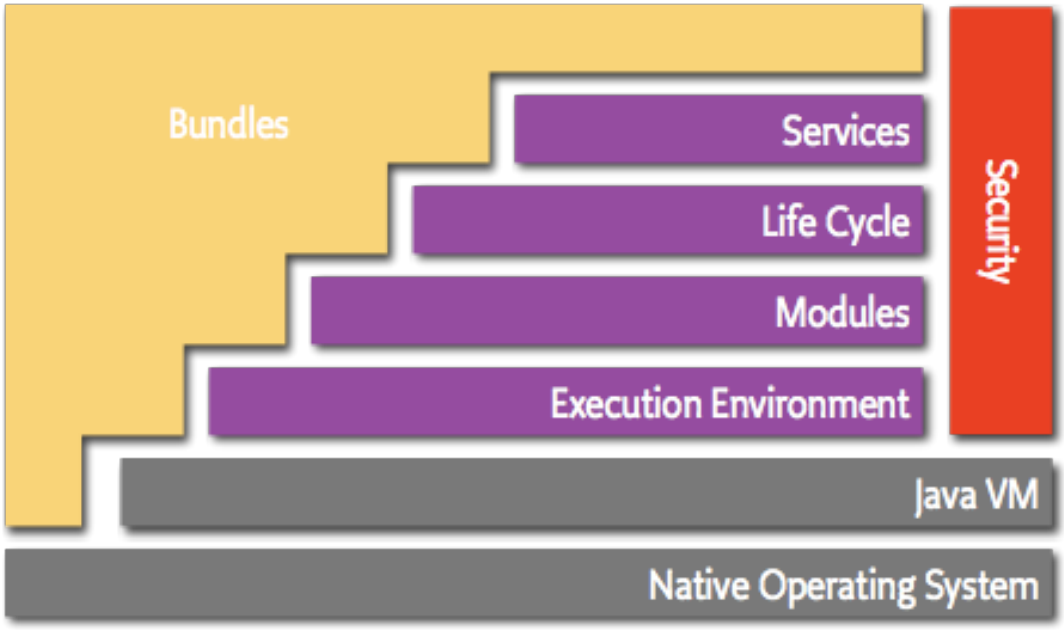


Figure 4: OSGi Architecture Diagram, Source: OSGi Alliance, https://www.osgi.org/developer/architecture

By using the OSGi dynamic component architecture, the functionality of the controller-software can be extended even at runtime. The Intefox system allows for easy extension of the system by means of so-called bundles. These bundles typically feature inputs, where they listen for incoming events, and outputs, where they can send events. So, for example in GREAT, a new bundle for the sound component was created, that manages encrypted communication with the sound component. It provides a set of inputs that can be used for controlling the sound module, and a set of outputs that provide other components with information (e.g. the status of the sound component).

In the GREAT setup, the Intefox middleware software runs on a Raspbian Linux operating system, but could be run on other Linux-/macOS- or Windows-based systems too, if a Java Runtime environment is available.

The configuration of the controller is edited using a graphical configurator software based on the open source Eclipse rich client platform (RCP) that is available for multiple platforms (e.g. Linux, macOS, or Windows) (see Figure 5). Connections between inputs and outputs of modules can be made via drag & drop.

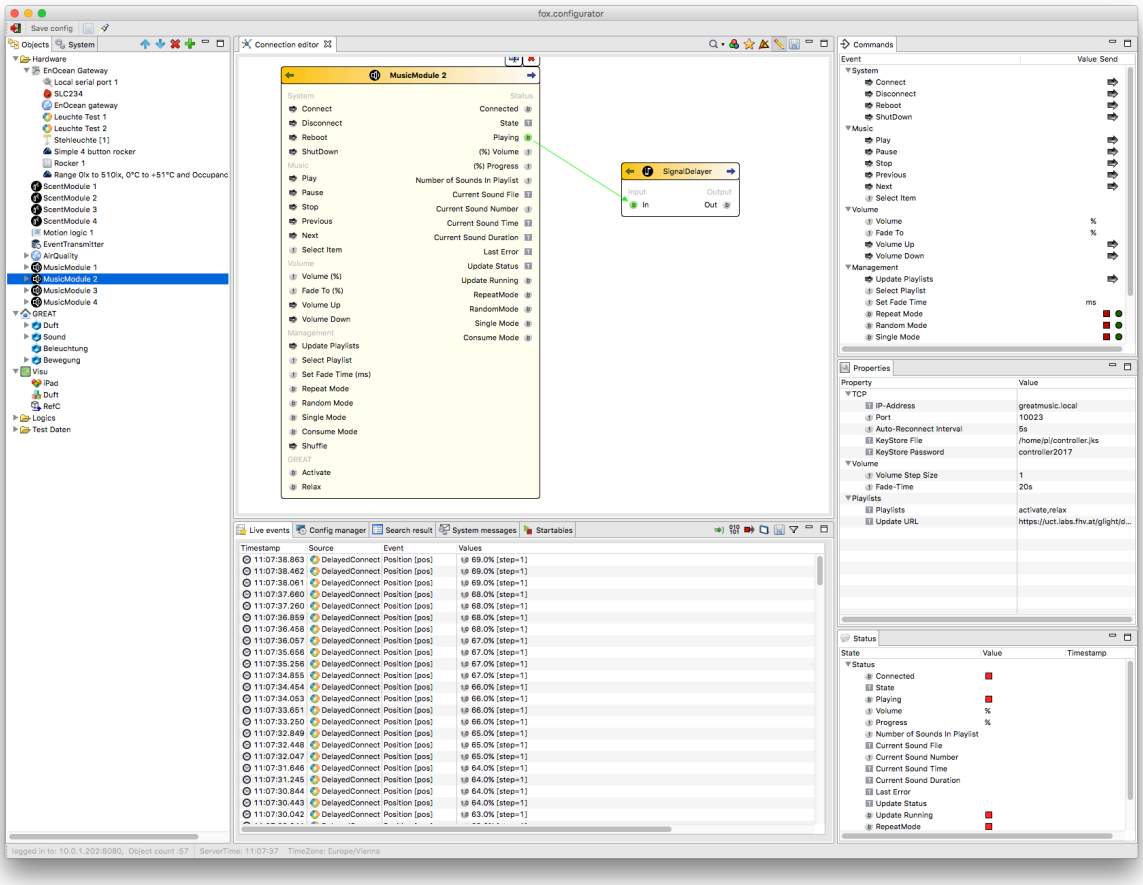


Figure 5: Screenshot of the Intefox configuration software showing connections between individual elements.

See chapter 3 for a more detailed description of relevant components of the middleware system.

2.2 Wireless Access Point, Router

Each GREAT controller provides its own GREAT wireless network, to allow for easy connection of WLAN based components or mobile app based remote controls, independent of the local availability of a WLAN. The GREAT WLAN is an encrypted WPA2 network for security reasons.

The wireless access point functionality provided by the GREAT controller is built using the open source packages **hostapd** and **dnsmasq**. These packages must be installed

using apt-get, as they are not part of the standard installation of Raspbian.

The hostapd provides the functionality for the actual access point, while dnsmasq is a light weight DHCP and DNS server and therefore hands out IP addresses to connected clients.

In the GREAT setup, we use the built in WiFi interface wlan0 as basis for our access point. To avoid that this interface is being used otherwise, it needs to be denied in the /etc/dhcpcd.conf above any other interface lines.

```
denyinterfaces wlan0
```

The actual interface definition then takes place in the /etc/network/interfaces file:

```
allow-hotplug wlan0

iface wlan0 inet static

   address 172.24.1.1

   netmask 255.255.255.0

   network 172.24.1.0

   broadcast 172.24.1.255
```

This defines a static IP address of the access point interface of 172.24.1.1. In a next step the hostapd is configured to provide a GREAT wireless network. Configuration of this network happens in /etc/hostapd/hostapd.conf

The typical config for GREAT looks like

```
# This is the name of the WiFi interface we configured above
interface=wlan0

# Use the nl80211 driver with the brcmfmac driver
driver=nl80211

# This is the name of the network
ssid=GREAT

# Use the 2.4GHz band
hw_mode=g

# Use channel 6
channel=6

# Enable 802.11n
ieee80211n=1

# Enable WMM
wmm_enabled=1

# Enable 40MHz channels with 20ns guard interval
ht_capab=[HT40][SHORT-GI-20][DSSS_CCK-40]

# Accept all MAC addresses
macaddr_acl=0

# Use WPA authentication
auth_algs=1

# Require clients to know the network name
ignore_broadcast_ssid=0
```

```
# Use WPA2
wpa=2

# Use a pre-shared key
wpa_key_mgmt=WPA-PSK

# The network passphrase
wpa_passphrase=**********

# Use AES, instead of TKIP
rsn_pairwise=CCMP
```

This basically sets up a WPA2 wireless network on the 2.4 GHz band with a network SSID of GREAT.

In the /etc/dnsmasq.conf file the details of the DHCP part of dnsmasq are configured. Specifically the DHCP range, name servers, interfaces and listen addresses are defined here. The specific settings used in GREAT are:

```
domain-needed

bogus-priv

server=8.8.8.8

interface=wlan0

listen-address=172.24.1.1

bind-interfaces

dhcp-range=172.24.1.50,172.24.1.150,12h
```

as well as mappings of hostnames for up to 5 sound- and scent-components.

```
dhcp-host=greatmusic,172.24.1.2,infinite

dhcp-host=greatscent,172.24.1.3,infinite

dhcp-host=greatmusic2,172.24.1.4,infinite

dhcp-host=greatscent2,172.24.1.5,infinite

dhcp-host=greatmusic3,172.24.1.6,infinite

dhcp-host=greatscent3,172.24.1.7,infinite

dhcp-host=greatmusic4,172.24.1.8,infinite

dhcp-host=greatscent4,172.24.1.9,infinite

dhcp-host=greatmusic5,172.24.1.10,infinite

dhcp-host=greatscent5,172.24.1.11,infinite
```

This allows for creating port forwarding rules to specific components for remote management tasks.

A final configuration step involves the iptables routing software. Here traffic from the eth0 interface is forwarded to the wlan0 interface. The routing information is loaded from a persistence file in the rc.local phase.

Excerpt of the iptabes-save persisted file that is loaded on startup:

```
# Generated by iptables-save
*filter
:INPUT ACCEPT [4823:503443]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [3401:1416793]
-A FORWARD -i eth0 -o wlan0 -m state --state RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -i wlan0 -o eth0 -j ACCEPT
COMMIT
# Completed on Fri Aug 18 09:09:28 2017
# Generated by iptables-save v1.4.21 on Fri Aug 18 09:09:28 2017
*nat
:PREROUTING ACCEPT [19:2901]
:INPUT ACCEPT [16:2709]
:OUTPUT ACCEPT [4:316]
:POSTROUTING ACCEPT [0:0]
-A PREROUTING -p tcp -m tcp --dport 33101 -j DNAT --to-destination 172.24.1.2:22
-A PREROUTING -p tcp -m tcp --dport 33102 -j DNAT --to-destination 172.24.1.3:22
-A PREROUTING -p tcp -m tcp --dport 33103 -j DNAT --to-destination 172.24.1.4:22
-A PREROUTING -p tcp -m tcp --dport 33104 -j DNAT --to-destination 172.24.1.5:22
-A PREROUTING -p tcp -m tcp --dport 33105 -j DNAT --to-destination 172.24.1.6:22
-A PREROUTING -p tcp -m tcp --dport 33106 -j DNAT --to-destination 172.24.1.7:22
-A PREROUTING -p tcp -m tcp --dport 33107 -j DNAT --to-destination 172.24.1.8:22
-A PREROUTING -p tcp -m tcp --dport 33108 -j DNAT --to-destination 172.24.1.9:22
-A PREROUTING -p tcp -m tcp --dport 33109 -j DNAT --to-destination 172.24.1.10:22
-A PREROUTING -p tcp -m tcp --dport 33110 -j DNAT --to-destination 172.24.1.11:22
-A POSTROUTING -o eth0 -j MASQUERADE
-A POSTROUTING -p tcp -m tcp --dport 22 -j MASQUERADE
COMMIT
# Completed
```

These rules also include network address translation to allow for direct reachability of the sound- and scent components over SSH for remote maintenance.

Since in some situations there are no wired network connections available, the GREAT controller also supports connecting to an existing WLAN via a USB WLAN stick mounted as interface wlan1. When a wlan1 interface is becoming available, a wlan1_up script adds rules to iptables to route from wlan1 to wlan0 and vice versa. When the stick is removed and the wlan1 interfaces is down, a wlan1_down script, removes the rules dynamically again.

Connections via WLAN tend to be prone to disconnecting. Therefore, a script checks every 5 minutes via cron, if the WLAN connection is still functional and if not, tries re-establishes the connection to the interface.

```
WLAN=wlan1
wlanExists=`ifconfig | grep ${WLAN}`
if [ $? -eq 0 ]
then
        router=`ip route | awk '/default/ {print $3;exit;}'`
        ping -I ${WLAN} -c2 $router > /dev/null
        #echo $router

        if [ $? !=  0 ]
        then
                ifdown --force ${WLAN}
                /bin/kill -9 `pidof wpa_supplicant`
                ifup --force ${WLAN}
        fi
fi
```

The onsite WLAN credentials can be easily set using a wpa_supplicant.conf file, that is placed into the root of the boot partion of the Raspberry PI microSD Card. The system then moves this file into the proper place automatically (/etc/wpa_supplicant/) and uses these credentials. Typically this text file includes the information:

```
country=AT
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
network={
        ssid="networkName"
        psk="networkPassword"
        key_mgmt=WPA-PSK
}
```

Instead of directly supplying a password, also the actual key can be supplied. This key for a given SSID can be generated using the wpa_passphrase command on the Raspberry.

2.3 OpenVPN Client

Since the GREAT prototypes will be used in various places in Austria, Italy and Switzerland, it's important to be able to update the systems via a remote connection. However, often it's not possible to get an externally reachable IP address at the institutions. To avoid the need of an externally reachable IP address, the GREAT controller connects itself into a remote management virtual private network hosted by the FHV (see figure XX) automatically when a network connection becomes available. On the client this is achieved by installing the openvpn-package and passing a client specific configuration (typically an ovpn-file, but this needs to be renamed to conf) to the service. It is then enabled by calling for example:

```
sudo systemctl enable openvpn@clientConfig1
```

The virtual private network is implemented using the open source VPN server OpenVPN. It is configured to apply TLS encryption to connections. Each GREAT controller and maintainer computer has its own certificate. Only clients with a valid certificate can connect to this network. The VPN network is configured to use TCP Port 443 for communication to avoid firewall issues on location.

In case of a remote management task, the maintainer connects a computer to the VPN and can then access GREAT controllers using their hostnames via SSH, as long as the controllers are connected to the Internet. The main controllers listen for SSH connections on port 33100. Submodules can then either be reached via SSH connections originating from the controller, or via port forwarding on the controller directly from the maintainer's computer.

3. Intefox - Middleware

3.1 Architecture and Components (foxcore)

The foxcore server runs on a java runtime and is using the OSGi standard to load and unload bundles during runtime and provides RESTful interfaces for configuration, control and visualization clients.

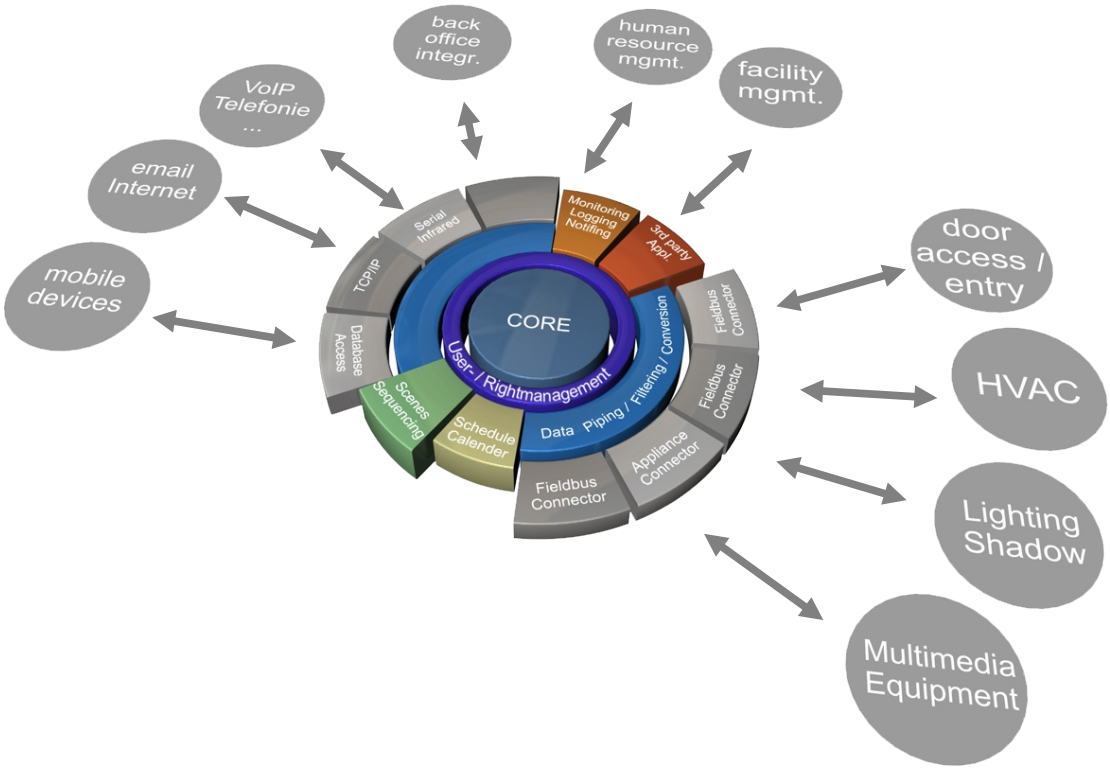


Figure 6: Basic architecture of the foxcore server

Figure 6 showsthe basic arichtecture oft he foxcore server, illustrating the layered model to provide abstraction for different technologies and interfaces.

3.2 Configurator software

The fox.configurator is used to configure and manage foxcore servers. The connection is established via TCP/IP and can therefore be used to either connect locally or remotely. Figure 7 shows the example of inserting a new light object into the system configuration.

Basic features:

- Managing bundles and updates
- Creating and managing objects
- Bundle activation (licensing)
- Managing configurations
- Server diagnostics
- User management
- Timer and Schedulers management
- Managing Cloud services (AutoBackup, Database, PushNotification, Alexa, etc)
- Enabling control and visulization interface
- Commissioning
- Live events

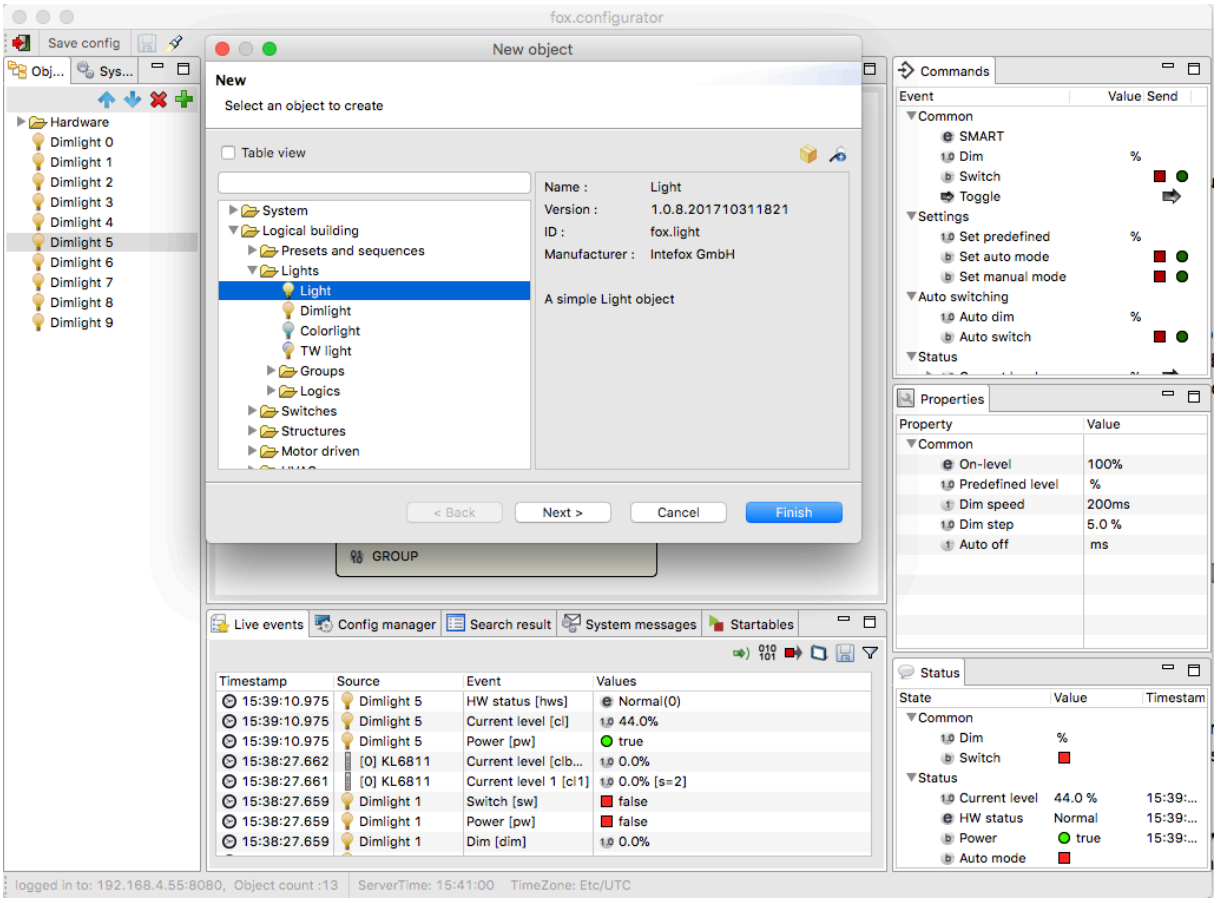


Figure 7: fox.configurator, example of adding a new light object

3.2.1 Online Bundle manager

The online bundle manager is fully integrated into the configurator software.

The bundles are managed through an online sharing platform where it allows a software developer to upload and share the bundle to others. In that way, it makes it very easy for everyone to install and update bundles on the controller. Figure 8 shows the bundle selection screen to extend the functionality of the core system.

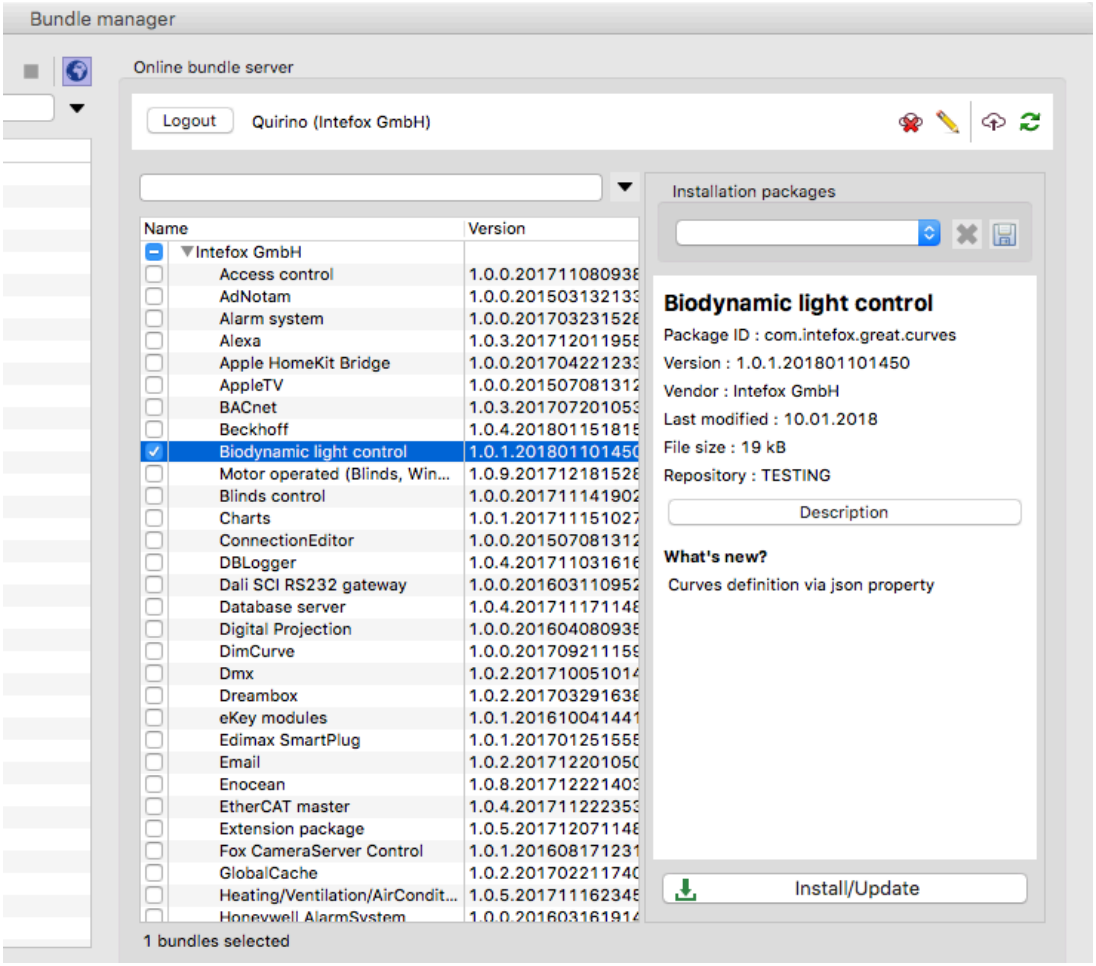


Figure 8: Online bundle manager

3.3 Logging and Data Access interface

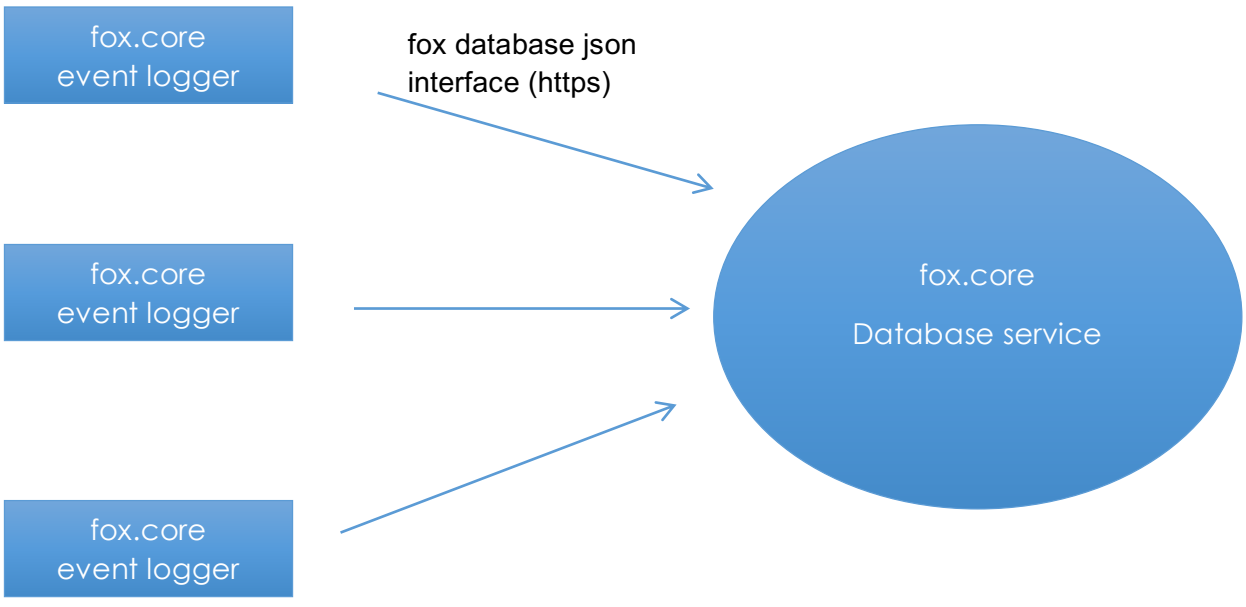


Figure 9: Basic structure of the event logging architecture

Figure 9 illustrates the basic event logging architecture that consists of a database service backend, as well as event logger objects.

Database service

The database service is available as a fox.core bundle and is designed to either run within the same local network or some hosted server on a fox.core based system.

Currently PostgreSQL is supported as database backend. Further database types might be added, as the need for it arises though.

Event Logger

An event logger processes defined events and sends them to the database service. It also takes care of buffering events locally, if the remote database service is not available at the time. It is available as a fox.core bundle and is designed to run on a fox.core based system.

Multiple instances are allowed on the same controller as well as on different controllers to send the data to the same database service.

Through the event logger it is also possible to compare the logged values from a local controller with values from other controllers live in charts views and reports if they are being logged to the same database.

3.3.1 Event logger configuration

Creating the Event logger

First, an event logger object needs to be created. Therefore, select the 'Database loggers' container within the System tree and select 'Add object'. Figure 10 and Figure 11 illustrate the process.

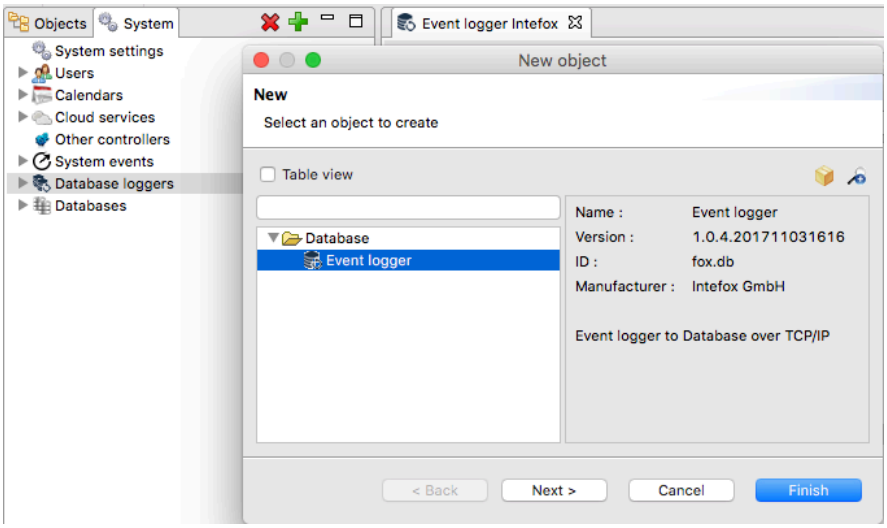


Figure 10: Select the created event logger and edit the properties (URL and Context)

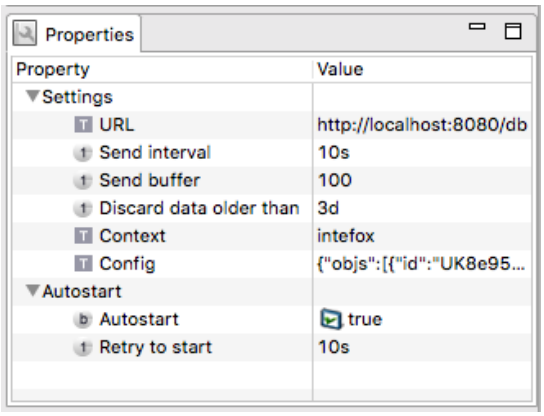


Figure 11: Properties of the event logger service

URL:

If the database runs on the same controller, the URL would be like this:

http://localhost:8080/db

Context:

the database name set in the database service configuration. (e.g. myDBname)

Send interval:

Defines the minimum interval to be used to send the queued data to the database service.

Send Buffer:

Defines the buffer size for queueing the data before sending to the database service.

If the queue is full, the data will be sent immediately.

Discard data older than

Defines, how long the data will be kept in the queue if the data could not been sent due to any communication error with the database service.

The event logger uses the interface (described in section 'interface') for the data exchange.

3.3.2 Log settings (Selecting the events to be logged)

With the event logger configuration editor (by double clicking the 'event logger object)

the events can be selected to be logged.

There are 2 ways to select the event:

- Type: all objects of this type will be logged with the defined settings
- Object: single objects will be logged with the defined settings

Figure 12 shows the configuration of an eventlogger to log the temperature output of all temperature sensors in the system, even if they might be added at a later point in time.

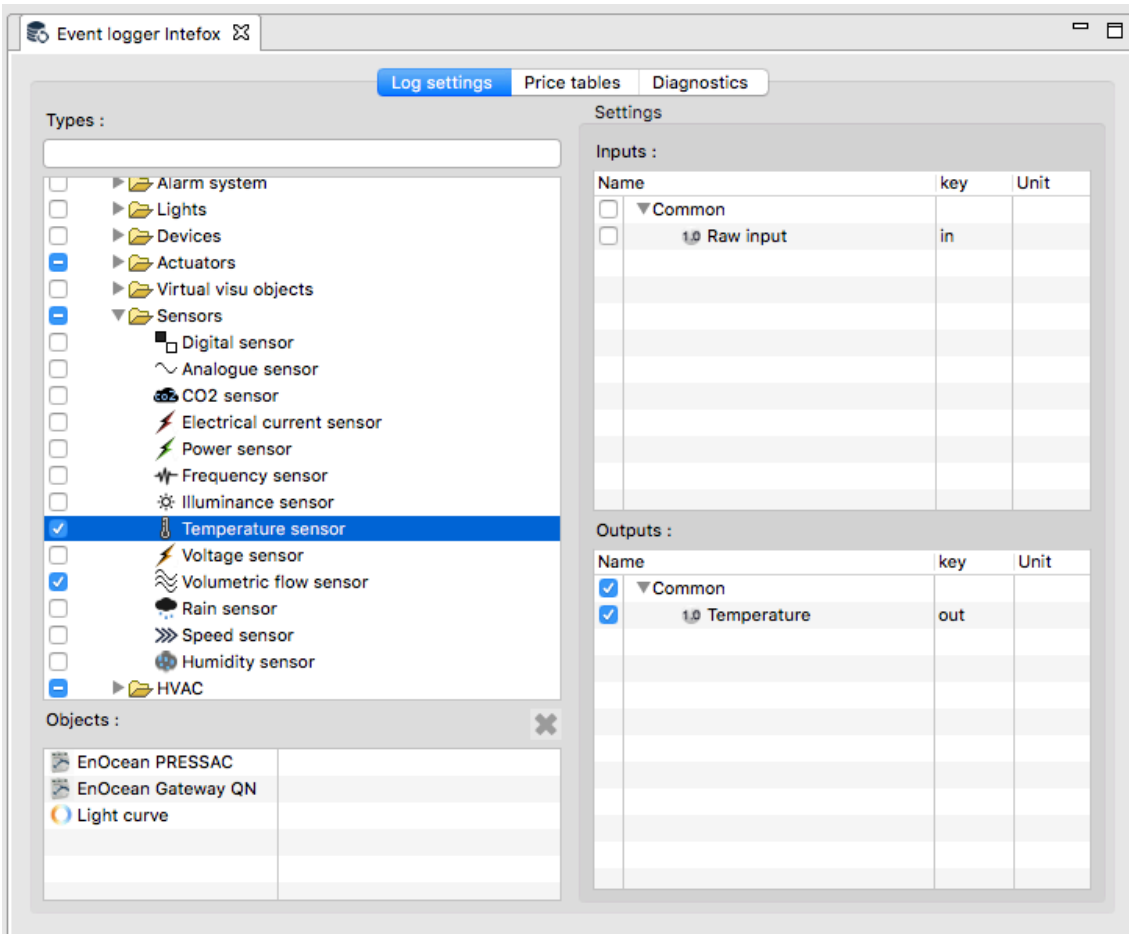


Figure 12: In our example, the 'Temperature' output of all 'Temperature sensors' will be logged, even if they are being created later.

3.3.3 Data access interface

The database service offers a REST-based interface to store and retrieve data as well as to read the logging configuration. Parameters are encoded using json format.

format: http://localhost:8080/db/<cmd>?params...

cmd: data

writes Event data to the Database

the data is being sent as post data in json format

Description with example data:

http://localhost:8080/db/data

< POST DATA AS JSON>

cmd: readconfig

retrieves the current foxEvents table with objectIds in json format

parameters:

id	database id
coreid	if not set, ALL events of all cores will be returned
statistics	includes the current amount of data logged of each event
space	includes the space being used by the DB and also the free space

cmd: getdata

request data in json format

parameter:

id	database id
eid	comma separated id's of foxevents (known when previously read from config)
	(or)
objId	id of FoxNode object
key	the in- or output key depending on dirout
dirout	0 = input key, 1 = output key
from	epoch format (milliseconds since 1.1.1970)
to	epoch format (milliseconds since 1.1.1970)
dur	duration in milliseconds (note: use either 'to' or 'dur')
	(or)
range	-->possible values: **keyword**[,back,count]
	(back = keyword units back, count = range in keyword units,
	e.g. month,6,3 = starting from 6 month's back with a range of 3 months
	thishour
	lasthour
	today

yesterday

thisweek

lastweek

thismonth

lastmonth

thisyear

lastyear

hour,back,count

day,back,count

week,back,count

month,back,count

year,back,count

maxpoints the maximum number of points returned per eventId. Default value = 100

debug if debug parameter is present, the returned values will include additional information of

the requested eventId's

units a string based unit map

➔ example value: K=°C,m/s=km/h
- will return all temperature values as '°C' and all velocity values as 'km/h'

unituser the id of a user object. If the user exists, the unit map will be taken from the user properties

rangeinfo if rangeinfo parameter is present, the returned values will include the requested 'from' and 'to' timestamp. This is helpful if the request is done by a range keyword

examples:

http://localhost:8080/db/getdata?id=qnhome&eid=3&range=lastweek&debug

--> returns a json with data of object with eventId=3 of last week, also includes additional debug informations

http://localhost:8080/db/getdata?id=qnhome&eid=3,5&range=month,6,3

--> return a json with data of objects with eventId = 3 and 5 from 6 month back with a range of 3 months

http://localhost:8080/db/getdata?id=qnhome&objId=7jgbd7kx&key=out&dirout=1&from=1498984980000&duration=360000

type line (default), sum, day, week, month, year

type sum:

retrievs the total value only plus price if a pricetable is assigned

example: (analog values)

sum (value per hour): 345 kWh

price (through price table) 123,23 EUR

example: (digital values)

count (impulses), true count (boolean)

type day:

retreives the total value of 1 day and 24 hour values

type week

retreives the total value of 1 week and 7 day values

type month

retrieves the total value of 1 month and 28-31 day values

type year

retreives the total value of 1 year and 12 month values

3.4 Controller & Visualization interface

3.4.1 LiveCycle of a Visualization client

1) Login (retrieves last version id to check if a reload of descriptions is necessary)
2) getDescriptions
3) getStructure (mixed with requested Style)
4) poll (receive Status updates)
5) Send commands asynchronously

HTTP request, format:

http://host:port/json/session_id/key

return format is json

3.4.2 Login

key: login

Table 1: Login Parameters

user	the user name for the visu user
pwd	the password for the visu user note: the login procedure will be changed in future to a more secure method
style	optional: customized style properties for each single object
appid	optional – see PushNotification support
devtoken	optional – see PushNotification support
id	optional – see PushNotification support

Figure 13: Example login request

3.4.3 Request object descriptions

key: descs

Retrieves a description of each object type which is represented in the current visu configuration for the user logged in. Additionally, the object type and the current values of each single object will be added at the end.

The json is splitted in 2 parts:

- descs — contains possible commands to send status values returned
  - i — id of object type (e.g. fox.light, fox.blinds)
  - c — commands to be send to the core
  - s — status to be received from the core
    - n — display name of the command or status
    - k — key of the command or status to be used for identication
    - t — value type (see table below)
    - u — unit of each value (e.g. %, ms, °C)
- objs — contains object id and current status values
  - i — id of the object (unique id per object)
  - t — object type (this value refers to the id (i) in descs (see Table 2)
  - v — current values of the object (see Table 3)

Table 2: Request object types (t)

none	Trigger (no value)
1	Boolean (0 = false, 1 = true)
2	Integer (signed value, uses internaly 4 bytes)
3	Double (floating point value, uses internally 8 bytes)
4	String (Text)
5	Enumeration (Integer value)
6	Bytes (starting with 0x, example: 0xa1bb4c83)
7	Long (signed value, uses internally 8 bytes)
8	Float (floating point value, uses internally 4 bytes)
9	Tristate (0 = false, 1 = true, 2 – undefined)

Table 3: Request values (v)

#	The ‚#' key refers to the main value of the command or status
<key>	If a command or status has more values, sub values are identicated by a key word

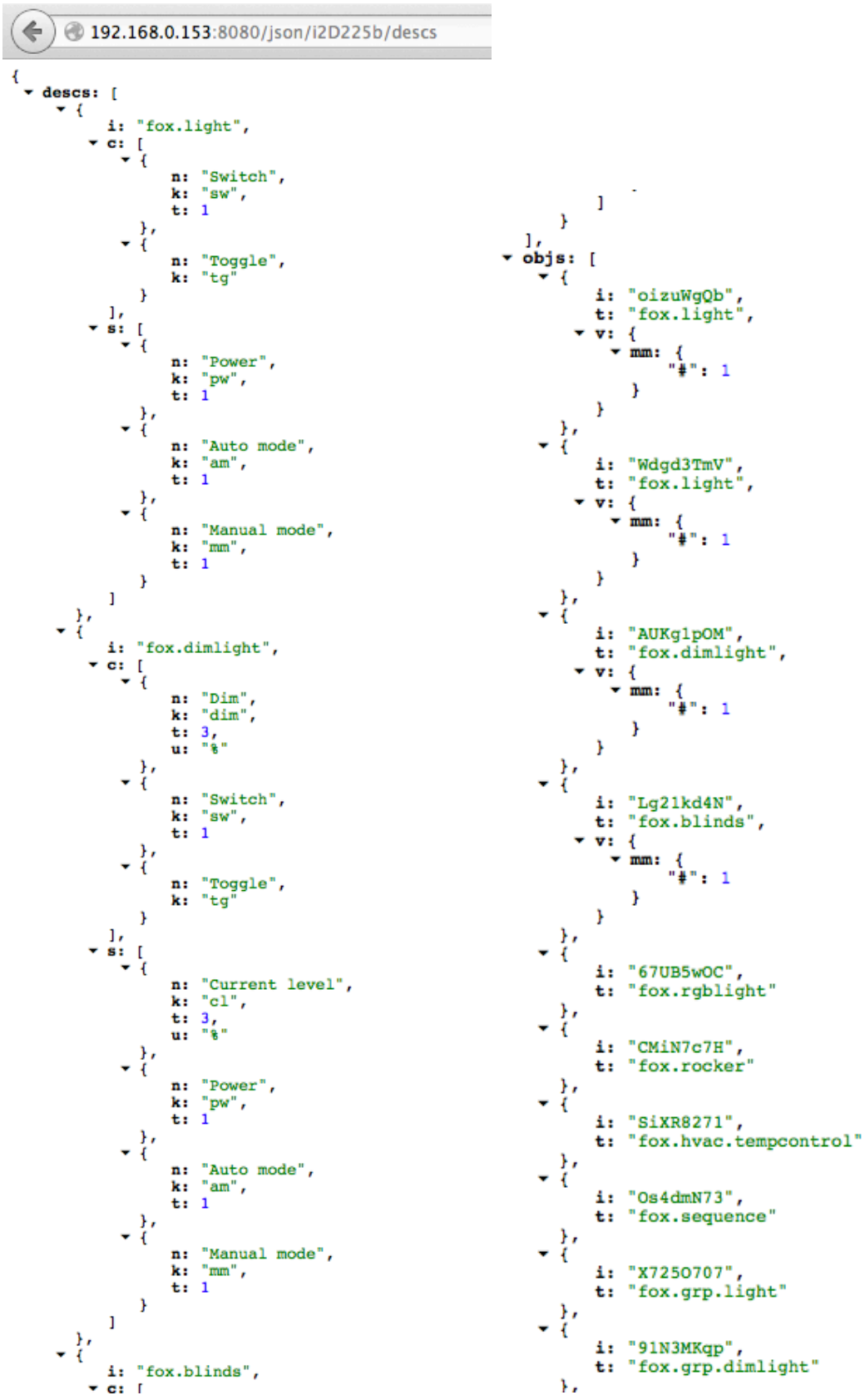


Figure 14: Example description request

3.4.4 Request structure

key: structure

No parameters.

Retrieves the hirarchical structure of the visualization configured for the user logged in.

Json key description:

- contains the structure in hirarchical order
  - p page id – a page can either be a ‚page' object or a container object (room, floor etc).
  - r page id – this is a ‚reference container object' (building, floor, room) which might contain additional pages
  - c category id – used to manage style properties in categories
    - l listed objects in category
    - t category type
    - i short id, refers to ‚c' properties in ‚page' or container objects
  - o contains additional objects or pages
  - n display name for the object or page
  - i object id to refer to in previous request ‚objs'
  - s style properties (customizable)

192.168.0.153:8080/json/i2D225b/structure


```
[
  {
    p: "BXYnFbmm",
    n: "Visu",
    o: [
      {
        r: "14KCrL2O",
        n: "my Home",
        o: [
          {
            p: "121Y3MYZ",
            n: "Ground floor",
            o: [
              {
                p: "lTWvA60X",
                n: "Living room",
                o: [
                  {
                    i: "oizuWgQb",
                    n: "Light Living room"
                  },
                  {
                    i: "Wdgd3TmV",
                    n: "Light Kitchen"
                  },
                  {
                    i: "AUKg1pOM",
                    n: "Dimight TV"
                  },
                  {
                    i: "Lg21kd4N",
                    n: "Blinds Living room"
                  },
                  {
                    i: "67UB5wOC",
                    n: "Colorlight LED"
                  },
                  {
                    i: "CMiN7c7H",
                    n: "Rocker 1"
                  },
                  {
                    i: "SiXR8271",
                    n: "TempControl 1"
                  },
                  {
                    i: "Os4dmN73",
                    n: "Sequence 1"
                  },
                  {
                    i: "X725O707",
                    n: "Light group 1"
                  },
                  {
                    i: "91N3MKqp",
                    n: "Dimlight group 1"
                  }
                ]
              },
              {
                p: "S0Ud697x",
                n: "Storage",
                o: [
                  {
                    i: "GwssmIUF",
                    n: "Light Storage"
                  }
                ]
              },
              {
                p: "2a5vPQ5U",
                n: "Office",
                o: [
                  {
                    i: "eVwZ44lg",
                    n: "Light Office"
                  },
                  {
                    i: "wHFZlehe",
                    n: "Blinds office"
                  }
                ]
              }
            ]
          },
          {
            p: "ALghuhOX",
            n: "1st floor",
            o: [
              {
                p: "9fHn6AZR",
                n: "Bedroom",
                o: [
                  {
                    i: "GsW9S4HF",
                    n: "Dimight Bedroom"
                  },
                  {
                    i: "GMe1Js3B",
                    n: "Blinds Bedroom"
                  }
                ]
              },
              {
                p: "XHmvV80i",
                n: "Bathroom",
                o: [
                  {
                    i: "3jloO6g8",
                    n: "Light Bathroom"
                  },
                  {
                    i: "ZXOtSrw1",
                    n: "Blinds Bathroom"
                  }
                ]
              }
            ]
          }
        ]
      },
      {
        p: "V640993i",
        n: "Favorites",
        o: [
          {
            p: "183kzp4X",
            n: "Extra page",
            o: [
              {
                i: "Os4dmN73",
                n: "Sequence 1"
              }
            ]
          },
          {
            i: "oizuWgQb",
            n: "Light Living room"
          },
          {
            i: "X725O707",
            n: "Light group 1"
          }
        ]
      }
    ]
  }
]
```


Figure 15: Example structure request

3.4.5 LongPoll request

key: poll

No parameters

Retrieves value changes.

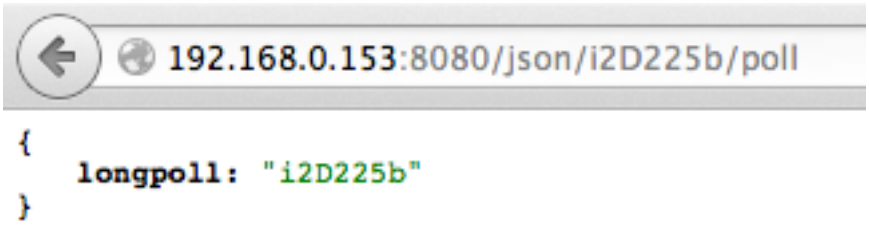


Figure 16: Example long poll request with no changes

If no value has changed, the core will respond after 10 seconds with a longpoll message (see Figure 16).

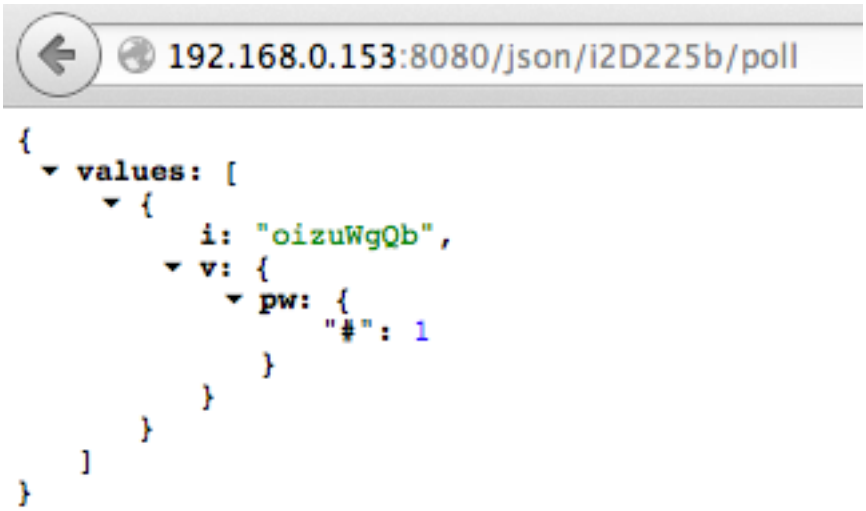


Figure 17: Example long poll request with changes

When a value got changed, the core will respond immediately with the changed values (see Figure 17)

Json key description:

- values contains the values in order of occurence
  - i id of the object which has changed values
  - v the value – contains the status key and it's new values

3.4.6 Send commands

key: cmd

Table 4: Command (cmd) parameters

key	the key of the command to be send
objIds	the id(s) of the objects where the command shall execute if more than 1 id is used, the id's must be seperated by semicolon (;)
value	the new value . the value should be UTF-8 encoded
subKeys	if subValues are submitted, seperated by semicolon (;)
subValues	add subvalues in the same order as subKeys, seperated by semicolon(;) The values should be UTF-8 encoded

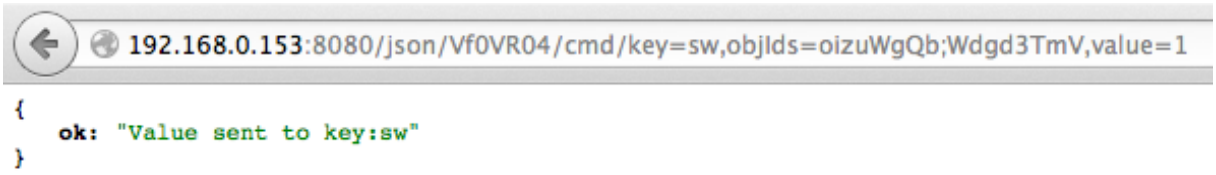


Figure 18: Example cmd request to switch two lighs on

Figure 18 shows an example to switch on two lights.

4. Light

4.1 Biodynamic Light Extension Bundle

The biodynamic light objects regulate the light instensity and color temperature during a day period with the possibilities of different interventions (see Figure 19,Figure 20, Figure 21).

The light curve objects send values to 3 separate output channels to set the following values calculated by a predefined curve in json format:

- dim level (0-100%)
- Color temperature in Kelvin
- fadetime in seconds (0 - 180 sec)

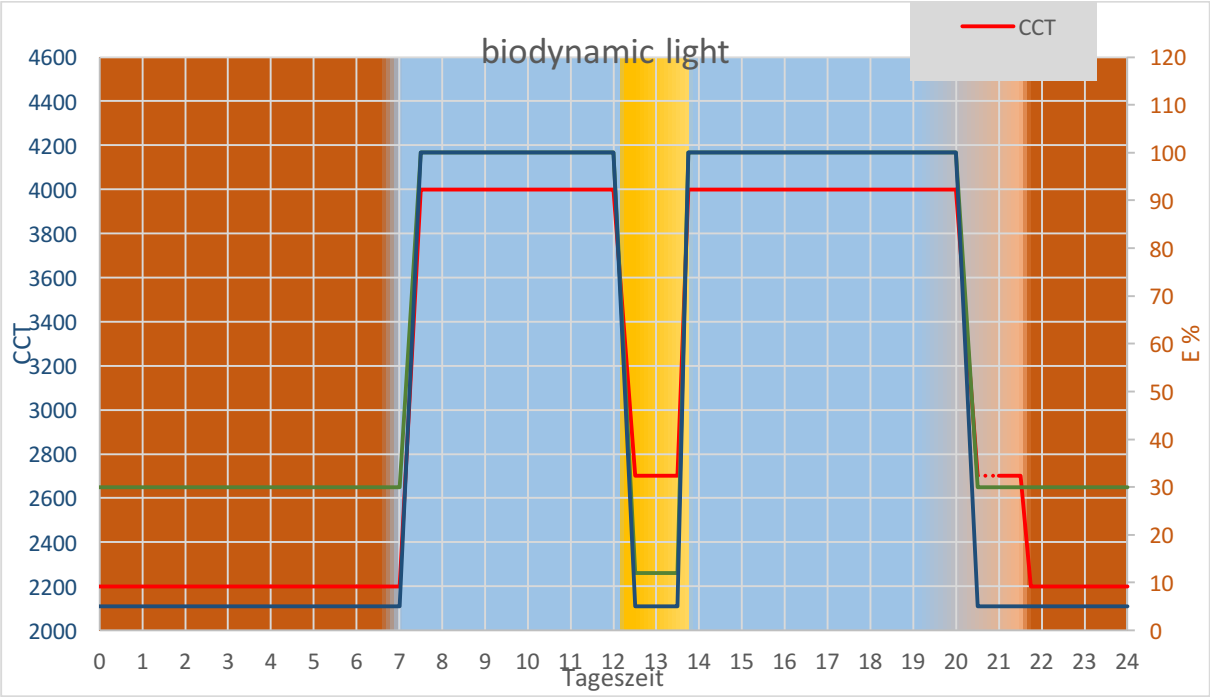


Figure 19: Example biodynamic light definition

Interventions:

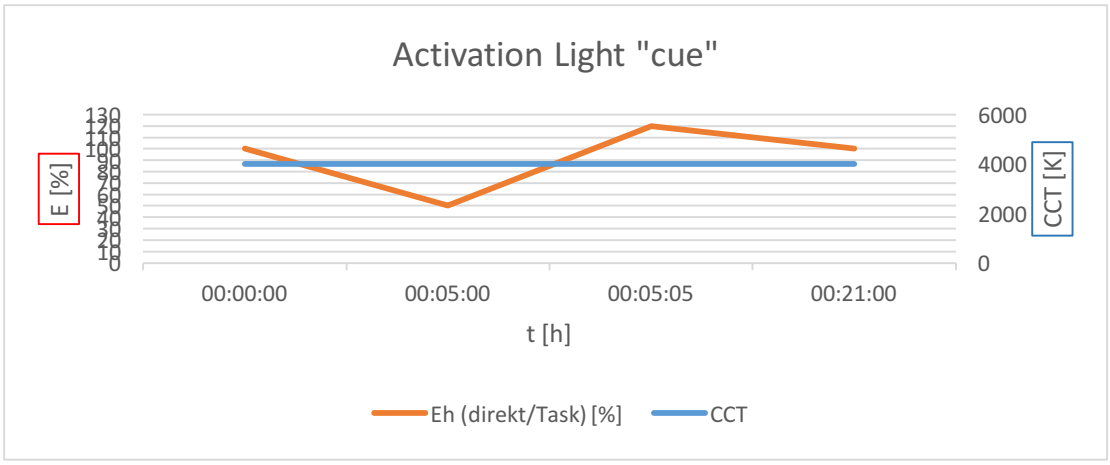


Figure 20: Example: Activation Light 'cue' definition

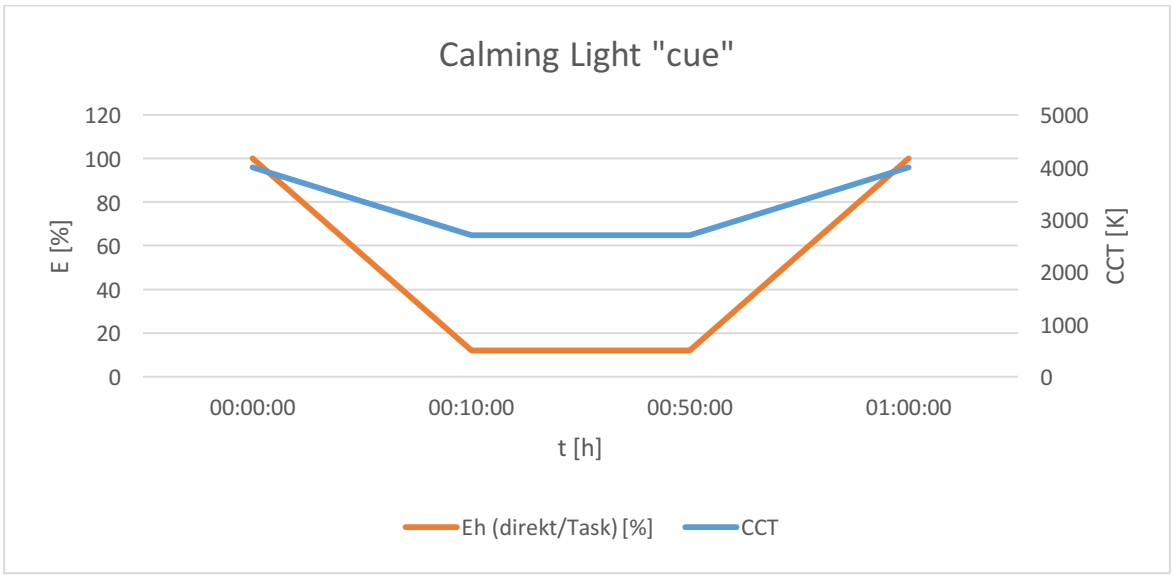


Figure 21: Example: Calming light 'cue' definition

4.1.1 Light curve object description

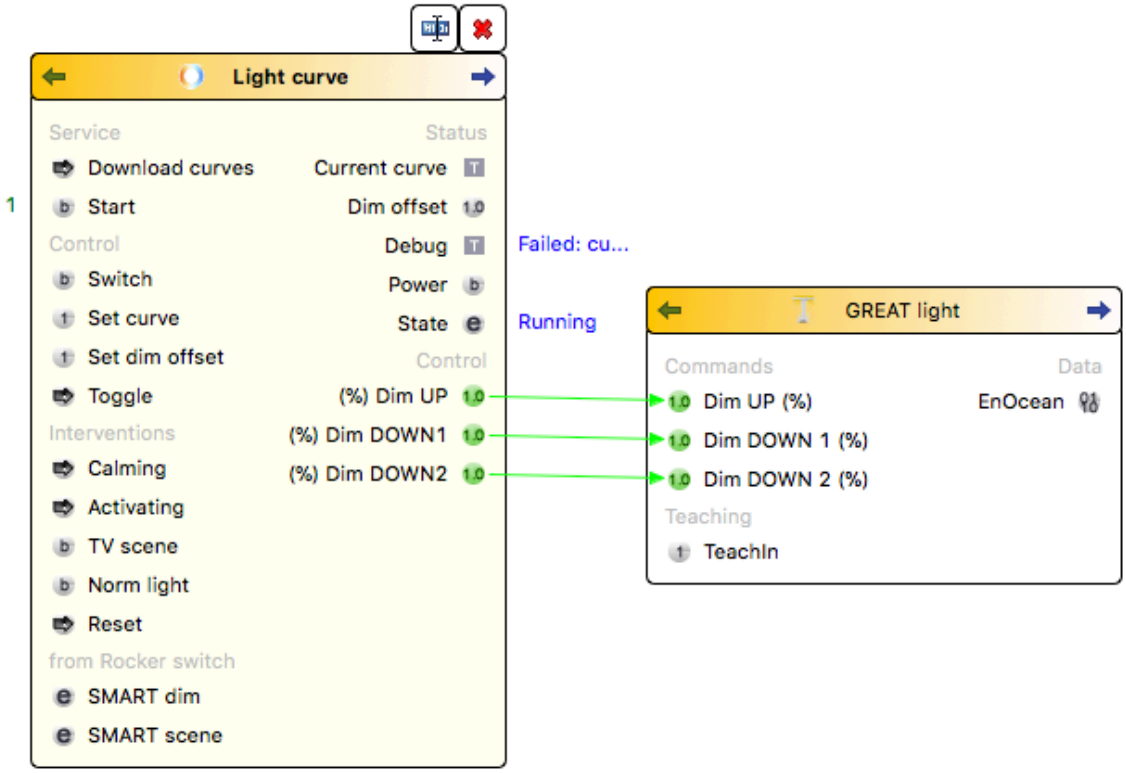


Figure 22: Configuration of the Light curve object in conjunction with the light device

Commands:

Service:

Download curves:

Downloads a new curve configuration from an external server

Start

Starts/Stopps the service

Manual control:

Switch

To switch the light on/off manually (e.g. by an external switch)

Set curve

Sets the current curve to use for the calculation of the light values

Set dim offset (+/- 0-100%)

adds a dim offset to the current light values

Toggle

To switch the light on/off manually (e.g. by an external switch)

Interventions:

Calming

Sets a calming intervention (1 hr)

Activating

Sets the activating interventions (20 min)

TV scene

Starts the TV scene (1 hr)

Norm light

Switches to the Norm light curve

Reset

Switches back to the biodynamic light control

from Rocker switch (Manual control via external switches)

SMART dim

Dims the light up/down

SMART scene

Calls predefined scenes

Status:

Current curve

Shows the name of the current selected curve

Dim offset

Shows the current dim offset added to the current output values

Debug

Shows some detailed debug information

Power

Shows if the light is on or off

State

Shows if the service has started (Running, Stopped, Failed)

Control

3 independent control channels to set the value on the connected lights

Properties:

Config

curve in json format

Curve server url

The URL where the curve is being fetched from during automatic update

e.g. https://uct.labs.fhv.at/glight/greatcurves/getCurveData.php?key=greatDemo

Curve server update

Off	Automatic updates are disabled
Every hour	Updates the curve every full hour
Midnight	Updates the curve every day at midnight

Simulation mode

Off	Normal operation	
24h in 1 min	Simulates 24h in 1 min	(1440 times faster)
24h in 5 min	Simulates 24h in 5 min	(288 times faster)
24h in 30 min	Simulates 24h in 30 min	(58 times faster)

Scale % max

defines the highest % value set within the curves config.

Example: in an 'Activating Intervention' it we like to overrule the regular 100% value and activate 120% for a short time. In that case, the output values will be scaled to 0-100% for all calculated values and therefore the regular 100% would be scaled to 83.3% to be sent to the hardware.

Send delay

Send delay in milliseconds between the cmds. when multiple commands are sent at the same time

if not set or 0 means NO pause between commands

Autostart

Defines wheter this object will be started (true) after System startup or not (false)

Retry to start

Time to wait after the startup failed to restart the object.

4.1.2 Protokoll data exchange between light and controller

The controller acts as master to the lights. All settings and commandes will be sent by the controller.

4.2 DATA: Controller to light

The light sets the requested color and brightness values in the requested fade time by its own with predefind correction values.

Basis: Enocean 4BS Telegram: A5-38-09 / modified

Byte	Description	Bit pos	Function	Values
0x03	OPTION FLAG	0.7	Reserve, Future Use	0
	FARBTEMPERATUR	0.6…0.0	Color Temperature	0 -> 2000K 1 -> 2050K (INC 50K) 127 -> 8350K (1)
0x02	BRIGHTNESS	0.7…0.0	Definition Brightness	0 (OFF) 1 -> Minimalwert 255 -> Maximalwert
0x01	FADETIME	0.7…0.0	Definition Fade Time	0….180 -> 1Sec 181 -> 4Min 255 -> 75Min
0x00	FUNCTION (Receiver)	0.7...0.4	UPLIGHT DOWNLIGHT FLÄCHE DOWNLIGHT SPOT	7 6 5
	LEARN BIT	0.3	Data telegram Learn Controller	1 Default 0
	SEND STATUS	0.2	No Status Send Status	1 0 Default
	STORE FINAL VALUE	0.1	Yes No	1 Default 0
	SERVICE MODE FLAG	0.0	Service function Normal operation	1 0 Default

(1) Received color temperature values will be adapted to the defined light color. (Range 2200K..5000K)

4.3 Statusresponse from light to controller

Ist Statusflag in FUNCTION is at *Send Status*, a Status response with the current values will be sent 5.5 sec after the last command.

Byte	Description	Bit pos	Function	Values
0x03	OPTION FLAG	0.7	Reserve, Future Use	0
	CURRENT VALUE COLOR TEMP	0.6…0.0	Current Color Temperature	0 -> 2000K 1 -> 2050K (INC 50K) 127 -> 8350K (1)
0x02	CURRENT VALUE BRIGHTNESS	0.7…0.0	Current brightness	0 (OFF) 1 -> min value 255 -> max value
0x01	ERROR FLAG	0.7	Error driver (auto reset)	0 -> No error 1 -> Error
	TEMPERATURE VALUE	0.6..0.0	Temperature of the light	0 -> 0°C 100 -> 100°C Max 101..124 reserved, Errorspecific. 125 -> Error Temp. Sensor 126 -> No Temp. Sensor 127 -> Temp. Measure in process, Initial value (2)
0x00	FUNCTION (Absender)	0.7...0.4	UPLIGHT	7
			DOWNLIGHT FLÄCHE	6
			DOWNLIGHT SPOT	5
	LEARN BIT	0.3	Data telegram	1
			Learn Controller	0
	SEND STATUS	0.2	No Status	1
			Send Status	0
	STORE FINAL VALUE	0.1	Yes	1
			No	0
	SERVICE MODE FLAG	0.0	Service function	1
			Normal operation	0

(2) Not all light elements have integrated temperature sensors. Values from 101 to 125 will be used for Status and error specifications.

4.4 Teach In

After pressing a button or by manually activating the teachIn function on the lgiht, the light will switch to the teachIn mode. This will be shown through a flashing light.

The TeachIn mode will be activated with a maximum duration of 1 minute. After that the light will be switched back to normal operation. A reactivating of the TeachIn mode will can be switched immediately.

If the Controller sents a Telegram with the TeachIn bit set within this time, the light will be teached in.

Maximum 5 controllers can be teached in. A long press of the button of at least 5 seconds will delete all teachedIn devices.

4.5 Definition Factory default

In factory default state, the following functions and values are predefined:

- Each controllr can control the light
- The light is preset to 15% light and 3000K for up-Lights and down-lights.

Timing of commands

The minimum time between commands shall be at least 250ms.

Dimming control

The new values must be at least 100K or 5 brightness points alter to the previous value.

After receiving of the last value and the 'status flag' has been set, a status response will be sent to the controller after 5.5 sec.

5. Sound Module

The sound module offers sound playback for the GREAT system. It is built on top of existing open source software with a minimal abstraction layer to safely connect the sound module to the GREAT system.

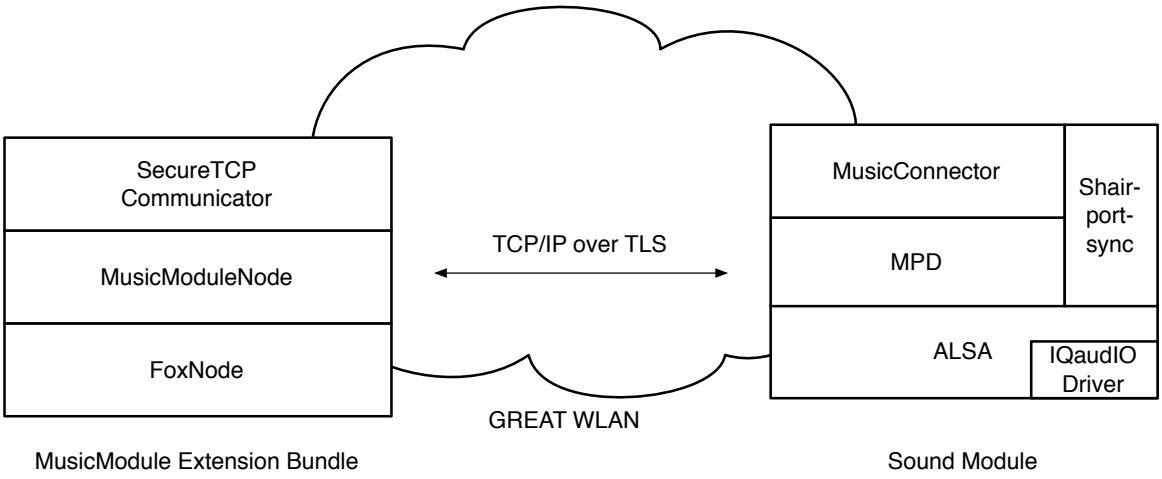


Figure 23: Communication between the Intefox music module extension and the sound module.

The software stack of the sound module is based on the Raspbian Stretch Lite Linux distribution for Raspberry Pi. For sound playback, the open source music player daemon (MPD) is used in combination with the mpc client tool for control (see Figure 23).

A thin abstraction layer on top offers secured communication with the GREAT system and remote playlist updating.

In addition to the playlist based sound playback offered by mpd, also AirPlay music streaming is supported using the open source shairport-sync package. This allows for using the GREAT sound module also as independent wireless speakers.

The sound module is connected to the GREAT system via WLAN. The connection to the network is monitored and if the network connection is lost (typical reasons could be power outage at the main controller, wireless signal interferences…), reconnection attempts will be made periodically until they succeed. This also allows for automatic connection of the module during installation and after network dropouts.

Availability of the sound module inside the GREAT network is announced via the open source avahi-daemon package.

5.1 Image preparation

5.1.1 Basics

The basis for the software stack is the Raspbian Stretch Lite distribution. Once this is installed, the other software packages used by the sound module must be installed.

First the driver for the Pi-DACZero sound card needs to be activated. This is done using a dtoverlay-entry in the /boot/config.txt file:

```
dtoverlay=iqaudio-dacplus
```

Optionally the onboard audio can be disabled by commenting out the dtparam for audio-on:

```
#dtparam=audio=on
```

Setting up the music player functionality involves installing the mpd and mpc packages:

```
sudo apt-get install mpd mpc
```

In the mpd.conf file, the user group should be adjusted to the audio group

```
group   "audio"
```

and the audio output needs to be adjusted to use the hardware output devices provided by the IQaudIO driver

```
audio_output {
        type            "alsa"
        name            "My ALSA Device"
        device          "hw:0,0" # optional
        mixer_type      "hardware"      # optional
        mixer_device    "hw:0"    # optional
        mixer_control   "Digital"           # optional
        mixer_index     "0"                 # optional
}
```

Other than that, the default settings are fine.

For the update functionality, the python requests package is required:

```
sudo apt-get install python-requests
```

Also, a sounds directory and a tmp directory need to be created that are used for music storage or temporary files during updates.

```
mkdir /home/pi/sounds
mkdir /home/pi/tmp
```

For the TLS connections to work, the required certificates and key files need to be copied to /home/pi/certs on the PI:

ca-chain.cert.pem, great_music.cert, great_music_nopw.key

Finally, the files musicModuleServer.py, ThreadedPlaylistUpdater.py and startMusicConnector.sh need to be copied to the system and an entry to the crontab needs to be made to automatically start the music connector on startup:

@reboot sh /home/pi/startMusicConnector.sh

5.2 Shairport-Support (for AirPlay functionality)

To build and install the shairport-sync package, follow the instructions, given by the author at https://github.com/mikebrady/shairport-sync:

Install necessary packages to build the shairport-sync daemon:

sudo apt-get install build-essential git xmltoman

sudo apt-get install autoconf automake libtool libdaemon-dev libpopt-dev libconfig-dev

sudo apt-get install libasound2-dev

sudo apt-get install avahi-daemon libavahi-client-dev

sudo apt-get install libssl-dev

sudo apt-get install libsoxr-dev

Then checkout the latest version of the shairport-sync source code, configure, make and make install it:

git clone https://github.com/mikebrady/shairport-sync.git

cd shairport-sync

autoreconf -i -f

./configure --sysconfdir=/etc --with-alsa --with-avahi --with-ssl=openssl --with-metadata --with-soxr --with-systemd

make

sudo make install

Then create a shairport-sync directory in /var/run and change the owner to shairport-sync:shairport.sync

sudo mkdir /var/run/shairport-sync

sudo chown shairport-sync:shairport.sync /var/run/shairport-sync

Then create a shairport-sync.conf file in /etc/tmpfiles.d to allow shairport to create temporary files. The content should be:

```
d /var/run/shairport-sync 0755 shairport-sync shairport-sync -
```

Finally adjust the shairport-sync.service file to run as a daemon by editing /lib/systemd/system/shairport-sync.service and add a '-d' to the ExecStart and adding a Type = forking line.

```
ExecStart=/usr/local/bin/shairport-sync -d
Type=forking
```

In /etc/shairport-sync.conf the name of the speakers (in general section) and quality parameters can be set. If no name is set, the hostname will be used.

Finally enable the service:

```
sudo systemctl enable shairport-sync
```

5.3 WLAN Setup

Each sound module is prepared to connect to the GREAT wireless network. This is done via the wpa_supplicant.conf file.

```
network={
   ssid="GREAT"
   psk=87b61f54914c527c67f87766167db5f9626c31a5c7a0d9e30cfe5024be6fa1ec
   key_mgmt=WPA-PSK
}
```

Instead of supplying the password in clear text, the wpa_passphrase tool can be used to generate the psk. For this the ssid and password need to be supplied as parameters.

The interfaces configuration file /etc/network/interfaces sets up the wlan0 interface to use the wpa_supplicant.conf file.

The interface definition of the interfaces file in /etc/network/ contains:

```
source-directory /etc/network/interfaces
auto lo
iface lo inet loopback
```

```
iface eth0 inet manual

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

To watch for network disconnects, a wlanConnector script is run periodically to check if the gateway is still available. If it's not available anymore, the wlan0 interface is shut down and re-enabled to try to reconnect again:

```
#!/bin/bash
# ping the router, no need to hit google for this.
SERVER=172.24.1.1
#specify wlan interface
WLANINTERFACE=wlan0

# Only send two pings, sending output to /dev/null
ping -I ${WLANINTERFACE} -c2 ${SERVER} > /dev/null

# If the return code from ping ($?) is not 0 (meaning there was an error)
if [ $? != 0 ]
then
# Restart the wireless interface
/sbin/ifdown --force ${WLANINTERFACE}
/sbin/ifup ${WLANINTERFACE}
fi
```

The wlanConnector script is run every two minutes as a cron job with the following definition in crontab:

```
*/2 * * * *   /home/pi/wlanConnector.sh
```

5.4 Music Module Extension Bundle

The music module system bundle provides functionality of the sound module inside the Intefox automation system. It communicates with the sound module over a TLS secured TCP connection.

The music module bundle is implemented using the OSGi based plugin architecture provided by the Intefox system.

The music module bundle provides a range of input-connections that can be connected to outputs of other event sources. It also provides output events that can be connected to inputs of other components (see Figure 24). In the GREAT context, the output events are mainly used for logging the system state. See Table 5 for a description of the input events, Table 6 for a description of the output events and Table 7 for a description of the parameters.

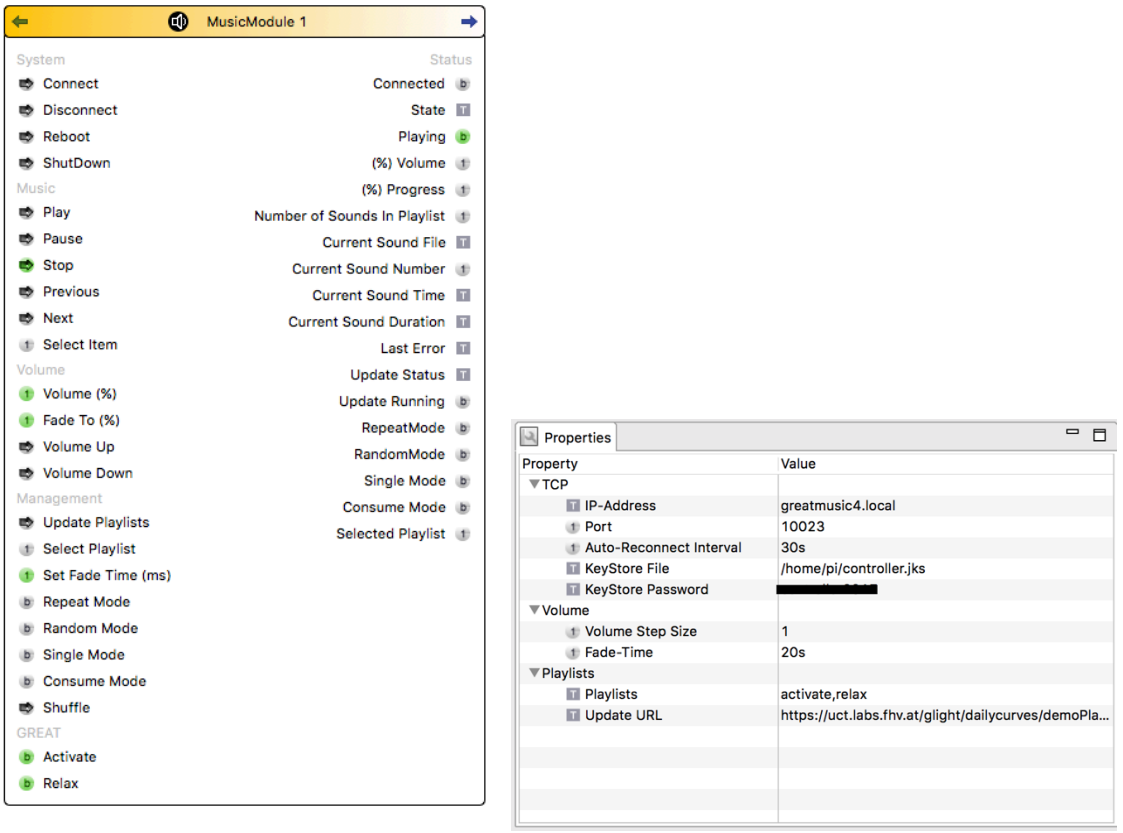


Figure 24:The available connections for the music module extension for the Intefox system and its parameter settings.

Table 5: Description of input events of the music module extension.

Input	Description
Connect	Tries to connect the node to the module at the address given in the URL parameter.
Disconnect	Disconnects from the module and closes all connections
Reboot	Causes the Sound module device to reboot.
ShutDown	Causes the Sound module device to safely shut down.
Play	Starts playback of the current song
Pause	Pauses playback of the current song
Stop	Stops playback of the current song
Previous	Jump to the previous song in the playlist

Next	Jump to the next song in the playlist
Select Item	Selects the song at the given number in the playlist (e.g. the first song would be 1)
Volume	Immediately sets the volume to the given percentage level (volume control is logarithmic)
Fade To	Fades to the given percentage level within the Fade Time that is provided in the parameters.
Volume Up	Increases the volume by the step size defined in the parameters
Volume Down	Decreases the volume by the step size defined in the parameters
Select Playlist	Selects the playlist at the given index (zero means the first playlist defined in the Playlists parameter settings)
Update Playlists	Causes the sound module to update its playlists. The playlist definition is downloaded from the URL that is defined in the parameters
Set Fade Time	Sets the fade time to the given time in ms
Repeat Mode	Turns the repeat mode on/off.
Random Mode	Turns the random mode on/off.
Single Mode	Turns the single mode on/off.
Consume Mode	Turns the consume mode on/off.
Shuffle	Shuffles the order of the current playlist.
Activate	Selects and starts the activate playlist on True, stops playback on False
Relax	Selects and starts the relax playlist on True, stops playback on False

Table 6: Description of output events of the music module extension.

Output	**Description**
Connected	True if connected to the sound module, False if not
State	Possible values: Playing, Paused, Stopped, Finished
Playing	True if currently playing, False otherwise
Volume	Current volume level in %
Progress	Progress of the current sound playback in %
Number of Sounds in Playlist	Number of sounds in current playlist
Current Sound File	The currently selected file for playback
Current Sound Number	The number of the currently selected file inside the currently selected playlist
Current Sound Time	The current time of the currently playing sound
Current Sound Duration	The total time of the currently playing sound
Last Error	The last received error message
Update Status	The status of the update process.
Update	True if the update is currently running, False otherwise

Running	
Repeat Mode	Status of the Repeat Mode
Random Mode	Status of the Random Mode
Single Mode	Status of the Single Mode
Consume Mode	Status of the Consume Mode
Selected Playlist	Index of the selected playlist (zero based)

Table 7: Description of paramters of the music module extension.

Parameter	Description
IP-Address	IP-Address or link local name of the sound module
Port	The port on which the sound module listens (default 10023)
Auto-Reconnect-Interval	Interval in seconds for automatic reconnection if the connection gets lost. 0 means automatic reconnect is disabled.
KeyStore File	The path to the keystore file containing the certificate for connection to the sound module.
KeyStore Password	The password for the keystore file
Volume Step Size	The step size for volume up and volume down commands
Fade Time	The time in seconds to transition to the new volume set by the Fade To command
Playlists	Comma separated list of playlist names for the mapping between Index (zero based) and playlist name.
Update URL	The URL that provides the playlist information for the sound module. This URL is queried when the Update command is issued.

5.6 Sound Module Protocol Description

The communication between the music module extension for the Intefox system and the sound module hardware is based on TLS secured TCP/IP streams. Messages are exchanged in a text based form and its commands are based on the open source mpc tool for controlling the mpd music player daemon that is used for the actual sound playback. A separate musicConnector acts as a wrapper to mpc/mpd that provides a secured communication layer to the Intefox extension and communicates with the mpd via the mpc tool.

musicModuleExtension | < - > | musicConnector -> mpc -> mpd

The music connector receives commands from the music module extension via an TLS secured TCP/IP stream. It parses the commands and sends them on to the mpc tool that controls the mpd.

The musicConnector queries the mpc tool for the status of the mpd and sends the status messages back to the music module extension on the Intefox system.

The music connector acts as a server where the music modules extensions connects to. Only one client can connect to the server (a music module extension on the Intefox system has exclusive access).

The music module extension and the sound module must authenticate themselves using a TLS-Certificate. Only communication between verified peers is allowed.

Messages are sent using a text based stream. If multiple parameters are sent along, they are delimited by spaces or tabs. Each message is terminated by a newline character.

cmd [param]><NL>

If the command is accepted by the musicConnector, a simple ack message is sent back. If a command cannot be handled, a nack message is sent back.

5.6.1 Commands from client to server:

nop

This is a keep alive message with no other purpose.

play <itemNumber>

Plays the item at the specified number. This invokes the mpc play command. The item number parameter is 1 based (1 means the first item).

stop

Stops the playback. This invokes the mpc stop command.

next

Switches to the next track in the playlist. This invokes the mpc next command.

prev

Moves to the previous track in the playlist. This invokes the mpc prev command.

pause

Pauses playback of the current track. This invokes the mpc pause command.

shuffle

Shuffles the current playlist. This invokes the mpc shuffle command.

random on|off

Sets the random playback mode of the mpd. This invokes the mpc random command with the supplied argument.

single on|off

Sets the single playback mode of the mpd. This invokes the mpc single command with the supplied argument.

repeat on|off

Sets the repeat mode of the mpd. This invokes the mpc repeat command with the supplied argument.

consume on|off

Sets the consume mode of the mpd. This invokes the mpc consume command with the supplied argument.

volume <targetVolume>

Sets the playback volume of mpd. This invokes the mpc volume command. If a fade operation is ongoing at the moment, it is terminated and the volume is immediately set to the targetVolume.

fadeTo <targetVolume> <fadeTime>

Fades the volume from the current level to the target volume level within the specified fadeTime (in milliseconds). This repeatedly invokes the mpc volume command with intermediate steps until the target volume is reached.

playlist <playlistName>

Switches to the specified playlist. This invokes the mpc clear command followed by the mpc load command with the specified playlist name. Only if playback was active at the time, the playback for the new playlist will be started automatically. In this case the mpc play command will be invoked for the first item.

startPlaylist <playlistName>

Switches to the specified playlist and starts playback. This invokes the mpc clear command followed by the mpc load command with the specified playlist name followed by the mpc play command for the first item.

update url

Invokes an asynchronous update of the music playlists. A the server at the URL is expected to return a playlist description in json format. Then, for each listed item a mpd compatible m3u playlist file is generated and the respective files are downloaded from the sources specified in the json description, if they are not already present on the system.

json-format for update command:

```
{
        <playlistName> = {
                [
                        “soundFile”=<“destination path of sound”>,
                        “sourceURL”=<“download url for sound”>
                ],…
        },
        <playlistName> = {
                [
                        “soundFile”=<“destination path of sound”>,
                        “sourceURL”=<“download url for sound”>
                ],…
        },…
}
```

The reply contains playlists (in the GREAT context playlist names are supposed to be “activate” and “relax”). Each playlist contains an array of items, where each item is specified by the soundFile property that specifies the target location on the music module system, and a sourceURL property that specifies where the sound is available for download in case it doesn’t already exist on the system.

Status info of the asynchronous update process are sent to the music module extension while the process is running.

system shutdown|restart

Shuts down or restarts the music module.

5.6.2 Commands from server to client:

Note, in contrast to the client-server messages, parameters in server-client messages are delimited by tab, as params can potentially contain spaces in their values.

status finished | playing | stopped | paused

The current playback status. If finished, the playlist has been played to the end. If playing the playlist is currently playing, if stopped, the playlist is currently stopped, if paused, the playlist is currently paused.

volume <currentLevel>

The current volume level in %.

repeat on | off

The current repeat setting of mpd.

random on | off

The current random setting of mpd.

single on | off

The current single setting of mpd.

consume on | off

The current consume setting of mpd.

currentSound <soundName>

The name of the currently playing sound.

ack

The command was received and executed

nack

The command was received but not understood. No action has been taken.

5.7 Relevance/Reuse-potential outside GREAT

The developed wrapper to mpd/mpc and the extension for the Intefox system can be used in any scenarios that involve mpd-based music playback systems that need to be controlled from home automation systems.

Outside Intefox powered smart buildings the music module can also be easily integrated into other smart building middleware systems, due to its lightweight and open protocol.

The music module can be used independently of any home automation system as an AirPlay speaker system. The music connector automatically handles airplay sessions.

6. Scent Module

The scent module is a remote controllable 2-channel scent dispenser for GREAT. It offers a simple TCP/IP based remote interface for easy integration into home automation systems.

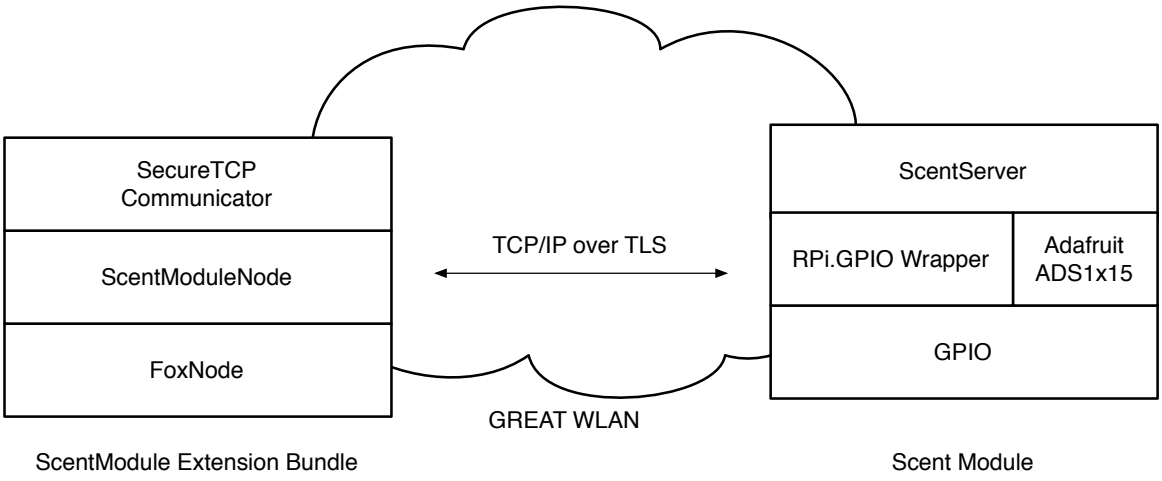


Figure 25: Integration of the scent module into the GREAT controller system based on Intefox.

The software stack of the sound module is based on the Raspbian Stretch Lite Linux distribution for Raspberry Pi which runs on a Raspberry Pi Zero W hardware. For controlling the scent pump-drive motors, the GPIOs of the Raspberry are used which are controlled via the Python based GPIO wrapper RPi.GRPIO (see Figure 25). The current that's flowing through the motors is measured via a shunt and an I2C based analog digital converter (see hardware description). To sample and access these values, the Adafruit_ADS1x15 library is used. A simple peak detection algorithm on these motor current values is then used to identify the end of a pump cycle.

The scent module is connected to the GREAT system via WLAN. The connection to the network is monitored and if the network connection is lost (typical reasons could be power outage at the main controller, wireless signal interferences…), reconnection attempts will be made periodically until they succeed. This allows for automatic connection of the module during installation or network dropouts.

Availability of the scent module inside the GREAT network is announced via the open source avahi-daemon package.

6.1 Image preparation

The basis for the software stack is the Raspbian Stretch Lite distribution. Once this is set up, the following packages need to be installed:

```
sudo apt-get install git build-essential python-dev

sudo apt-get install python-pip

sudo pip install adafruit-ads1x15

sudo apt-get install python-smbus
```

Each scent module should get a unique and meaningful hostname. In the GREAT context, this should be in the format greatscentXX, where xx is a continuous number. The hostname is adjusted by editing the files /etc/hostname and /etc/hosts by replacing the default "raspberrypi" with the new name.

For the TLS connections to work, the required certificates and key files need to be copied to
/home/pi/certs on the PI:

ca-chain.cert.pem, great_scent.cert, great_scent_nopw.key

The WLAN needs to be set up in a similar way as described for the sound module, so the scent module connects itself to the GREAT network and attempts to reconnect itself if the network connection is dropped.

Finally, the files scentServer.py, ThreadedPowerReader.py and startScentServer.sh need to be copied to the system and an entry to the crontab needs to be made to automatically start the scent server on startup:

```
@reboot sh /home/pi/startScentServer.sh >> /home/pi/scent.log 2>&1
```

6.2 Scent Module Extension Bundle

The scent module extension bundle provides functionality of the scent module inside the Intefox automation system. It communicates with the scent module over a TLS secured TCP connection.

The scent module bundle is implemented using the OSGi based plugin architecture provided by the Intefox system.

The scent module bundle provides a range of input-connections that can be connected to outputs of other event sources. It also provides output events that can be connected to inputs of other components. In the GREAT context, the output events are mainly used for logging the system state. See Table 8 for a description of the input events, Table 9 for a description of the output events and Table 10 for a description of the parameters.

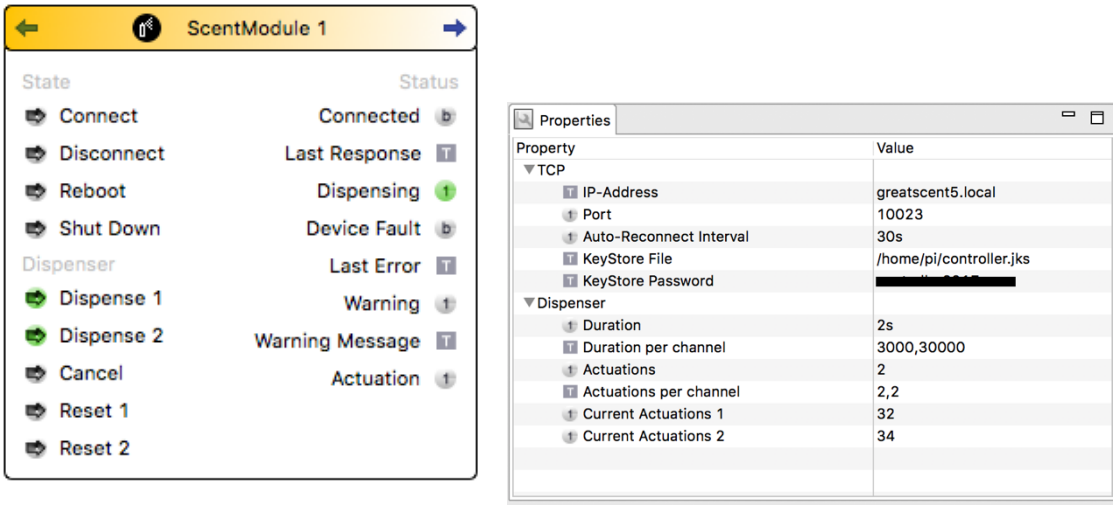


Figure 26: Input and output events (left) and parameter settings (right) of the scent module extension.

Table xx: Description of input events of the scent module extension

Table 8: Description of input events of the scent module extension.

Input	Description
Connect	Tries to connect the node to the module at the address given in the URL parameter.
Disconnect	Disconnects from the module and closes all connections
Reboot	Causes the Sound module device to reboot.
ShutDown	Causes the Sound module device to safely shut down.
Dispense 1	Dispenses scent in slot 1
Dispense 2	Dispenses scent in slot 2
Cancel	Stops scent dispensing immediately
Reset 1	Resets the dispense counter for slot 1
Reset 2	Resets the dispense counter for slot 2

Table 9: Description of output events of the scent module extension.

Output	Description
Connected	True if connected to the scent module, False if not
Last Response	The last response sent from the scent module
Dispensing	Dispensing state of the scent module: 0…ready, 1…dispensing slot 1, 2…dispensing slot 2
Device Fault	True if the driver module reports a fault state, False otherwise
Last Error	The last error that occurred.
Warning	The last warning code that was received.
Warning Message	The last warning message that was received.
Actuation	Reports a pump cycle on the respective slot.

Table 10: Description of output events of the scent module extension.

Parameter	Description
IP-Address	IP-Address or link local name of the scent module
Port	The port on which the scent module listens (default 10023)
Auto-Reconnect-Interval	Interval in seconds for automatic reconnection if the connection gets lost. 0 means automatic reconnect is disabled.
KeyStore File	The path to the keystore file containing the certificate for connection to the scent module.
KeyStore Password	The password for the keystore file
Duration	The max. duration of a dispense action
Duration per channel	CSV list of max duration in ms, e.g. 3000,1000 would mean 3 seconds for slot 1, 1 second for slot 2
Actuations	Number of actuations (pump cycles) per dispense action
Actuations per channel	CSV list of number of actuations per channel, e.g. 2,3 would mean 2 pumps for slot 1, and 3 pumps for slot 2
Current Actuations 1	Current number of actuations for slot 1 since the last reset
Current Actuations 2	Current number of actuations for slot 2 since the last reset

6.3 Scent Module Protocol Description

The communication between the scent module extension for the Intefox system and the scent module hardware is based on TLS secured TCP/IP streams. Every message received from the client is confirmed by a response from the server running on the scent module hardware. The server also sends status messages to the client.

The scent module extension and the scent module server must authenticate themselves using a TLS-Certificate. Only communication between verified peers is allowed.

Messages are sent using a text based stream. If multiple parameters are sent along, they are delimited by spaces or tabs. Each message is terminated by a newline character.

<cmd [param]><NL>

If the command is accepted by the scent module server, a simple ack message is sent back. If a command cannot be handled, a nack message is sent back.

6.3.1 Commands from client to server:

nop

This is a keep alive message with no other purpose. If no nop message is received within 1 minute, the TCP connection is assumed to be lost.

dispense <channel> <duration> [<actuations>]

Triggers a dispense operation for the channel given (1 means first slot, 2 means second slot) with the specified duration in ms. Optionally also an actuation count (how many pumps should be applied) can be passed. In this case, the duration parameter sets the maximum time until the operation should be stopped even if the actuation count is not reached (e.g. in cases where there is no bottle, or the motor is blocked).

cancel

Immediately stops the current operation.

system shutdown|restart

Shuts down or restarts the music module.

6.3.2 Commands from server to client:

Note, in contrast to the client-server messages, parameters in server-client messages are delimited by tab, as params can potentially contain spaces in their values.

ack [<status> <channel>]

The command was received and executed. If the command related to a dispense operation, the status and the channel involved are included.

Available states:

done: the action has completed for channel channel

nack

The command was received but not understood. No action has been taken.

input fault <val>

Signals whether the fault flag is active or inactive (val=1: fault flag set, val=0: fault flag reset)

warning <type> <message>

Signals a warning to the client. The following warning types exist:

1: No motor connected?

2: No bottle inserted?

3: Motor blocked?

4: Device busy (e.g. when already dispensing on another channel)

event <type> <channel>

Signals an event of type at the specified channel.

Available types:

pumped: one pump action took place on the specified channel

6.4 Relevance/Reuse-potential outside GREAT

The developed scent module can be easily used within Intefox powered smart buildings, but also allows for easy integration into various other smart building middleware systems, due to its lightweight and open protocol.

7. Sensors

7.1 PIR Sensor

The PIR sensor is use for motion detection with a simple Boolean value, either the sensor detects moving persons or not. Changes to this state are communicated via EnOcean interface to the Intefox controller and will be logged for further analysis. See Table 11 for the specification of the sensor used.

Motion detection: Standard Thermokon PIR EnOcean, battery powered

Manual switches: Standard EnOcean rockers

Table 11: Thermokon "EasySens" SR-MDS BAT specification

Vendor	Thermokon Sensortechnik GmbH (Germany)
Series	EasySense
Type	SR-MDS BAT
Technical design	Wireless
Wireless technology	EnOcean ISO/IEC 14543-3-10

Radio frequency	868,3 MHz
Functions	Motion detection and brightness measurement
Motion detection	Passive infrared
Detection area	360°; 105° conical (ceiling installation)
Detection radius (2,5 m room height)	3,25 m
Power source	3 x Battery 3,6V 1/2 AA LS14250
Brightness (Accuracy)	0-510 Lux (+/- 30 Lux)
Sleep time interval (modified)	1s – 1000s (for GREAT set to 1 second)

7.2 Biovotion Everion Sensor

The Everion device will be worn by one caregiver at each field test location. The device is worn at the upper arm and can measure either raw data and/or vital parameters. The raw data mode produces a huge amount of data and is used only for research and development purposes. The vitals parameters are calculated by functions on the sensor device itself and cause less data and traffic to handle. Currently the following vitals can be measured and streamed with a frequency of 1Hz:

- Heart rate*
- Blood Oxygenation or SPO2*
- Skin temperature*
- Skin blood perfusion*
- Steps / Motion*
- Blood pulse wave
- Heart Rate Variability
- Activity
- Respiration Rate
- Energy Expenditure
- Electrodermal activity/ galvanic skin response
- Barometric pressure

*) clinical grade

Since we are mainly interested in stress detection, the company did implement a stream to gather the IBI (Inter-Beat-Interval in ms) as shown Figure 27. The IBI is used to calculate the HRV (Heart Rate Variability) and the HRV itself is one parameter to detect stress.

Figure 27: Heart Inter-Beat-Interval

7.3 Stress detection

For the stress index calculation, we rely on results from a past research project where individual calibrations were made during a learning phase. These individual adaptions were based on personal feedbacks of the test persons. This will be missing in this project and we must analyse how the quality will be influenced.

There are no standard values or threshold recommendations for some physiological parameters (esp. heart rate variability HRV). The approach from earlier projects to identify individual ranges between low and high stress based on personal user feedback is not possible within the GREAT setup. We need to develop algorithms to identify stress thresholds based on historical data und on smart combinations of additional data based on analytical approaches and data mining approaches (e.g. plausibility calculations, combination physiological data with PIR data, time of the day, accelerometer activity profiles etc.).

An additional challenge in this project will be to find the shortest timeframe with enough data to allow a qualitatively good detection of stress. Figure 28 depicts stress timeframes of 1 hour during the day. In our feedback loop we might continuously measure and calculate the relative stress of the caregiver:

$$Relative\ highstress\ within\ timeframe\ = \frac{\#\ \text{stress index values} > 2}{\#\ \text{stress index values}}$$

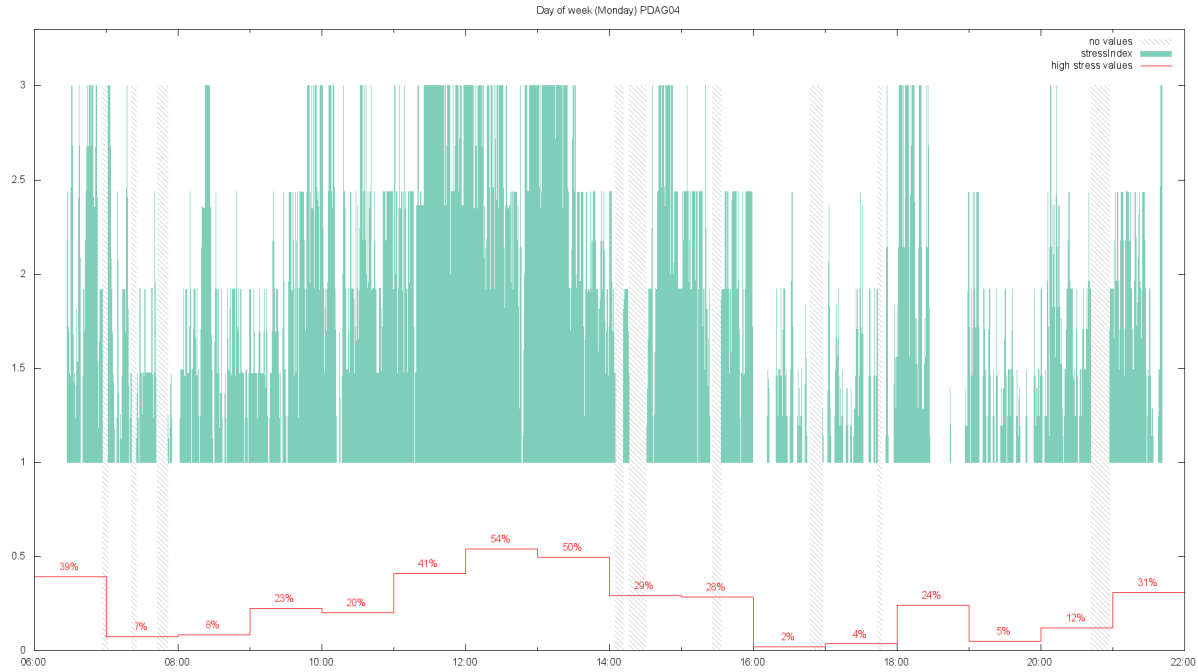


Figure 28: Stress index values and 1-hour phases

7.4 Architecture

Customers and users of the Everion device use an App (Android or iOS) to receive and analyze vital data. For our purpose, we must further process the data and can therefore not use this software directly. For our purpose, we use software provided by Biovotion for research and development tasks. One is the VSM tool to configure the sensor and another one is the Streamer tool to continuously stream data. Unfortunately, both tools are only available for Windows. To keep costs low and to fit into the hardware family used within the GREAT system we tried several setups with the raspberry pi as the hardware device receive streamed data from the Everion sensor. The most successful option was to use an Android version (Emteria) for the raspberry pi and to adapt the Android app of Biovotion. The source code was provided by Biovotion and first adaption were successful. But for a complete adapted software version to many development tasks were left. Therefore, we did decide to freeze these efforts and continue with the Intel Compute Stick version and Windows 10 home as shown in Figure 29.

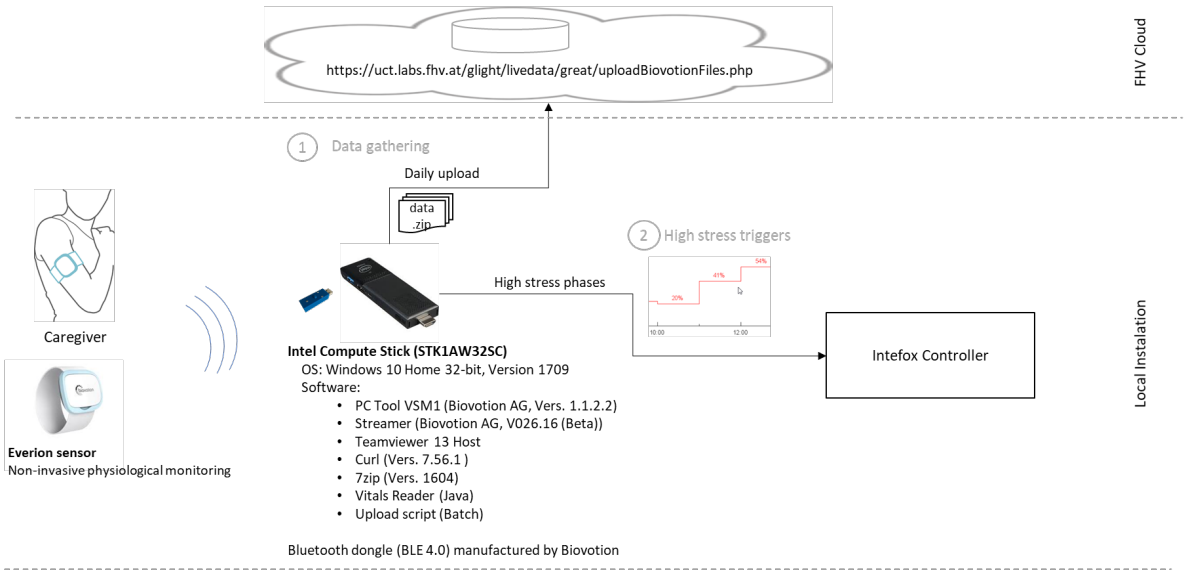


Figure 29: Everion sensor setup

7.5. Gathering vital data (test phase 1)

In a first test phase, we just gather the vital data (1) for further analysis.

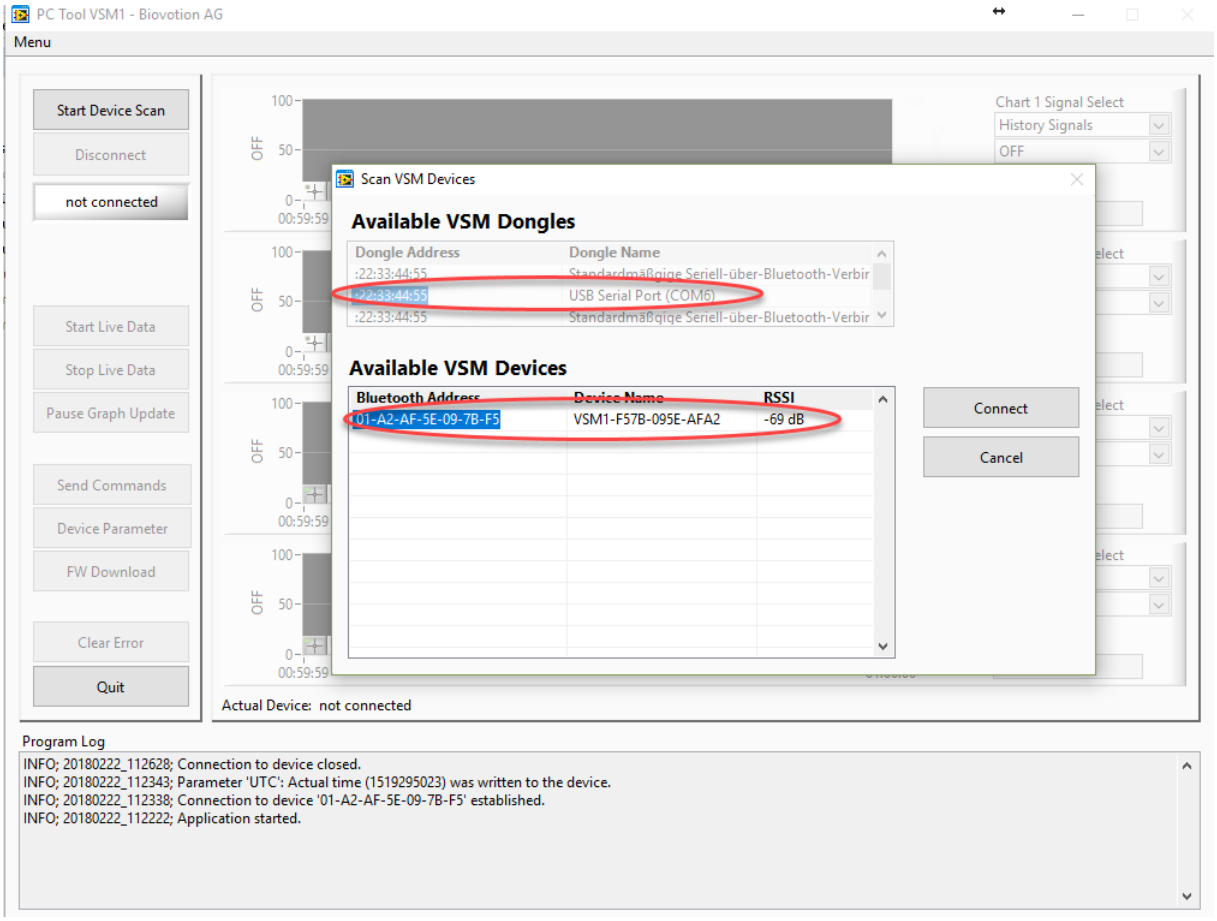


Figure 30: PC Tool VSM1 pairing with sensor

7.5.1 Setup

Follow these steps to initialize the Everion for vital parameter measurements and streaming.

1. Put the Everion device on the charger
2. Connect the Bluetooth dongle from Biovotion with the PC
3. Start the PC Tool VSM1
4. Start scanning the COM port belonging to the dongle
5. The device with its MAC address and the connection signal strength should appear in the list as shown in Figure 30. Select the sensor and connect.
6. Under the menu item *Device Parameter,* the sensors configuration can be changed or just read. The most important parameter is the algo mode. The following modes are available:

   0=VITAL MODE (Light Skin Mode)
   1=VITAL_CAPPED_MODE (SPO2)
   2=HR_ONLY_MODE (Dark Skin Mode)
   4=RAW_DATA_VITAL_MODE
   5=RAW_DATA_HR_ONLY_MODE
   6=SELF_TEST_MODE
   7=RAW_DATA_OFF_MODE
   8=RAW_DATA_FAST (64 Hz)
   9=MIXED_VITAL_RAW
   10=VITAL_MODE_AUTO_DATA
   11=GREEN_ONLY_MODE (Battery Saving Mode)
   12=RAW_DATA_FIX_CURRENT
   13=SHORT_SELF_TEST_MODE
   14=MIXED_VITAL_RAW_SILENT
   Depending on the algo mode the memory on the device is restructured. Therefore, this might take around a minute. The modes differ regarding the data measured and collected. For our use case we use the mode 0 (vital mode resp. light skin mode).

   Galvanic skin response (GSR_ON) shall be set to 1.

   The Table 12 lists the vitals measured and streamed in algo mode 0.

7. Close PC Tool VSM1
8. Open Streamer tool and set the configuration as depicted in
9. Close the tool or run the streamer if you want to start the measuring

Table 12 Vitals of light skin algo mode

Stream	Len	Type	Counter	Timestamp									
		0	1-4	5-8	9	10	11	12	13	14	15	16	1
Algo1	19	**9**		T(u)	HR(u)	HRQ(u)	SpO2 (u)	SpO2Q (u)	PI	Act. Class	Act.	Act. Class Q	Ste (u)
Algo2	15	**10**	C(u)	T(u)	HRV(u)	HRV Q (u)	RR (u)	RR Q (u)	Energy (u)	Energy Q (u)			
Raw board	19	**17**	C(u)	T(u)	Impedance low byte (u)	Impedance high byte (u)	Local Temp	Local Temp	Obj Temp	Obj Temp	Bar. Temp	Bar. Temp	m bar
IPI DB	10+ N*2	**22**	C(u)	T(u)									

Table 13: Value specifications

Name	Def	Data Type	Bytes	min (received)	max (received)	[offset;]conversion factor	Description
Heart Rate	value quality	uint8 uint8	1 1	30 0	240 100	1 1	[Bpm] [%]
SpO2	value quality	uint8 uint8	1 1	60 0	100 100	1 1	[%] [%]
Blood Pulse Wave	value quality	uint8 uint8	1 1	0 0	255 100	1/50 1	without units [%]
Perfusion Index	value quality	uint8 uint8	1 1	0 0	255 100	1/50 1	[% swing] [%]
Activity/Motion	value	uint8	1	0	255	100/255	without units
Activity Classification	value quality	uint8 uint8	1 1	0 0	255 100	enum 1	undefined 0 resting 1 walking_flat 2 running_flat 3 biking_flat 4 walking_up 5

							running_up 6 biking_up 7 rowing 8 other 9 biking 10 running 11 walking 12
Steps	value	uint8	1	0	255	1	[steps/second]
Energy Expenditure	value quality	uint8 uint8	1 1	0 0	255 100	2 1	[cal/s] not impl
HRV	value quality	uint8 uint8	1 1	0 0	255 100	1 1	[ms] (rMSSD) [%]
Respiration Rate	value quality	uint8 uint8	1 1	0 0	255 100	1 1	[Bpm] [%]
GSR-Sensor	value [Impedance]	uint16	2	0	65535	1/3000	ampl[kOhm]
Inter Pulse Interval (IPI)	value	uint16 B12-15 B0-11		0 0	15 4095	100/15 1	Quality value [%] Time in [ms]

The Inter Pulse Interval (IPI) value is a 2-byte value whereas the first 4 bits provide the quality and the left 12 bits the interval in milliseconds.

Example:

Entry in CSV data file:

22,240840,2017/12/08 22:53:31,,62333,
22,240841,2017/12/08 22:53:31,,62382,
22,240842,2017/12/08 22:53:31,,58281,

Leads to the following IBI values

62333 : IPI_Q 15 / IPI 893 mS
62382 : IPI _Q 15 / IPI 942 mS
58281 : IPI _Q 14 / IPI 937 mS

The sum over all IPI values of one day should always be 86400 seconds. The timestamp is set only once for each page therefore the counter should be considered for the order.

Table 14: Quality value specification

0	No IPI values detected, the time will be filled artificially to reach the 86400 seconds over a day.
1-7	Quality not sufficient (equal to the <50 quality values of vitals)
8-15	Quality is good

7.5.2 Pairing malfunction tipps

If the device does not appear in a list of the PC Tool VSM1 or the Streamer, then close the software, remove the dongle and plug it in again. This causes a **reset of the dongle**.

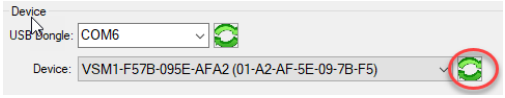
If connection problems receide, it might be due to a pairing problem. A pairing issue arises also, if the device was connected to a smartphone and shall be connected to the PC or visversa. In all this case **unpair** the device with the following steps (all data will be deleted):

1. Put the device on charger (If it is on the charger already, remove it, wait until the device vibrates twice and put the device on the charger again)
2. Wait, until the blue led is off (now you have 30 seconds to proceed with step 3)
3. Press the button until the device vibrates (as on the picture below)
4. Remove the finger
5. First, a double vibrate indicates that the unpairing was successful. (If there is no double vibration repeat the steps 1-4), then data deletion is executed. It will take approx. 40 seconds until your able to reconnect with a mobile device.

7.5.3 Run data gathering

The measuring of vital data can be started with the Streamer tool:

1. Start the tool and open the settings. 
2. Pair (connect) it with the sensor by selecting the correct COM port and pressing the green arrows of the *device* until the device MAC address appears and is selectable.



3. Save settings
4. Then start measuring

5. Check if the pairing and streaming start was successful

6. Every time the device is worn at the upper arm, the measurement continues and the tool streams the values into a file in the users documents and VSM data folder.

   Data file example:

   9,137277,2018/02/22 15:36:49,,75,82,60,0,10,1,2,98,0,65,
   10,137277,2018/02/22 15:36:49,,56,53,16,83,16,100,
   17,137277,2018/02/22 15:36:49,,231,219,3028,2978,2950,9285,
   22,219684,2018/02/22 15:33:48,,62315,

   The filename follows the pattern as shown in Figure 31. The device id follows the pattern VSM1-<MAC-Adress>.

7. Create a Windows task to run the upload batch file every evening. The files of the day will be zipped and uploaded for further analysis to the cloud server of FHV.

   uplodNewFiles.bat

```
forfiles /s /m "*.txt" /d %date% /c "cmd /c 7z a -tzip @FNAME.zip @PATH"
for /r %%f in (*.zip) do (
        curl -v -F "greatUserID=great" -F "greatPassword=<password>" -F
"upfile=@%%f"
https://uct.labs.fhv.at/glight/livedata/great/uploadBiovotionFiles.php
)
del /s *.zip
```

   Precondition: 7zip tool and curl installed.

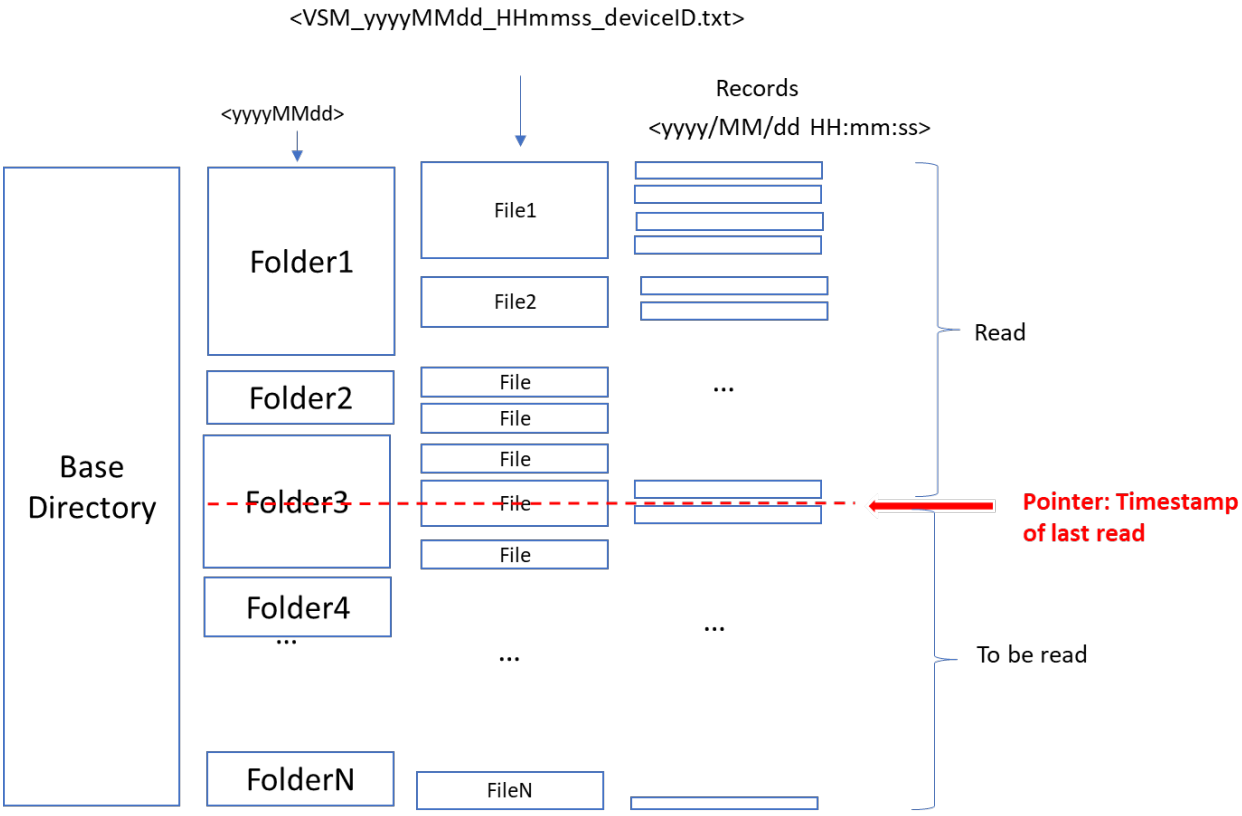


Figure 31: Data file structure

7.5.4 High stress triggers (testphase 2)

The collectd biodata will be analyzed together with the systems log files and feedbacks given by the caregivers. We assume to derive a pattern for the stress dedection and possible triggers to provide a suggestion for interventions to the caregivers.

The stress events or triggers shall be sent to the intefox controller for further dispatchment over the user interface. To allow a continuous measurement and stress calculation, we wrote a small Java program to constantly read the vitals from the files created by the Biovotion Streamer tool (see Figure 31).

8. User-Interface for Control

For the functional testing period a remote-control tool was created based on the existing Intefox mobile app that's available for iOS and Andorid devices. The configuration was created to allow for an easy control of the separate light/scent/sound-modules.

Figure 32 shows the main menu, the scent-, and sound-module offering control for starting an activation or relaxation session.

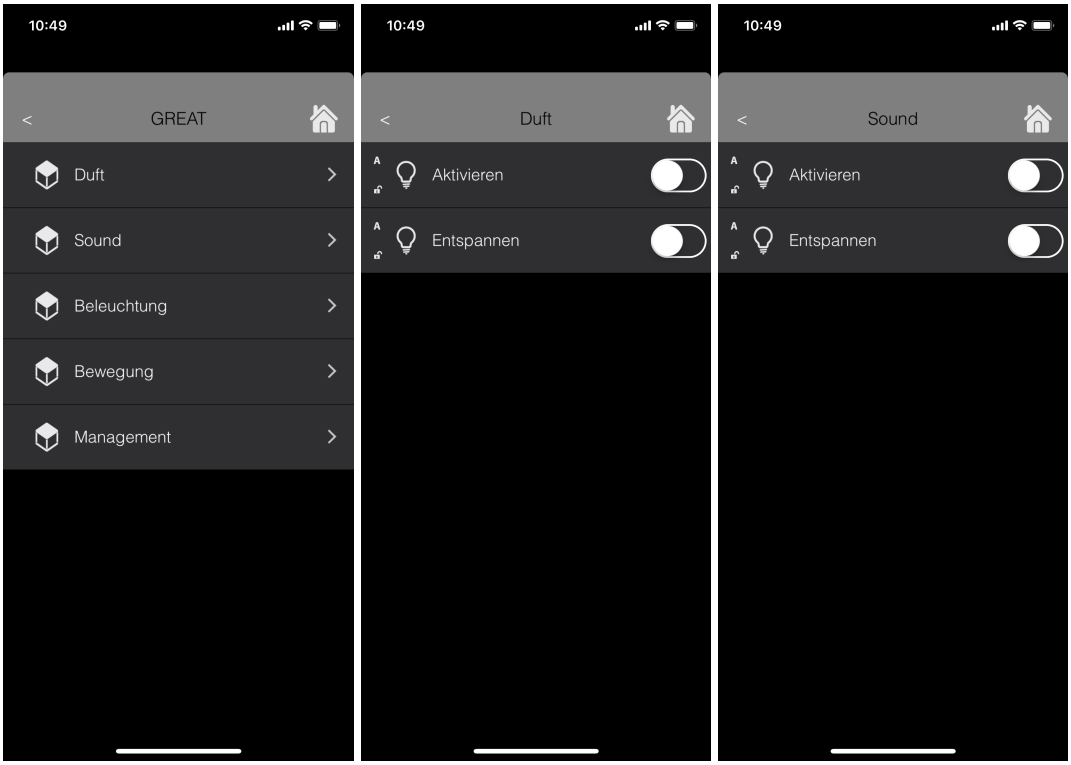


Figure 32: Screenshots of the main menu, the scent-, and sound-module offering control for starting an activation or relaxation session.

Figure 33 shows the control page for the light. When the light is switched on without further action, a biodynamic light is applied throughout the day. The interface allows to choose activation or relaxation interventions, as well as predefined light scenes for quickly applying a norm light or a scene fitted for watching TV.
The second screen shows the status of the PIR sensor, delivering brightness values (in lux) and motion status.

Using this remote interface, the individual components could be controlled manually, allowing for testing of the individual components in a typical setting.

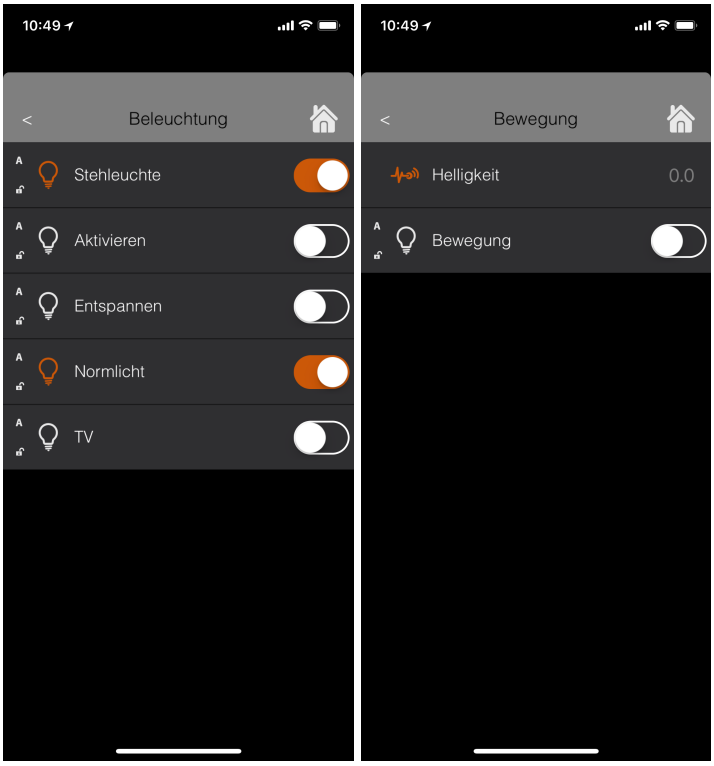


Figure 33: Screenshots of the light control page and the status view for the motion detector.

9. Outlook

For the field tests, an easy to use end user interface for controlling the system will be developed as an alternative to the existing Intefox mobile app. It will be based on standard technologies like HTML/JavaScript/CSS and will be delivered in a native app container, to allow for features like Push Notifications. Besides control functionality, also a user feedback option to help the learning system will be implemented.

For adjusting light interventions on an individual basis, a webbased front end will be developed, where new versions of intervention curves can be defined. The system will then automatically fetch the most recent definition from the server. A similar system is planned for the sound- and scent-components. These scheduled plans form the basis for an automated room ambience system.

The backend will be extended to perform daily analysis of the system's use and behaviour, and calculate statistics to generate individualized time- and effects profiles, which are the basis for the automatic room ambience system. An adaptive recommendation system that genereates suggestions for users to apply certain stimuli based on live data and history (embedded in these profiles) will be developed.

A convenient and secured web-based export option for the data gathered will be created for project members, to perform additional analysis already during the field test period.

Descriptions of these systems will be available in Deliverable D2.4.

10. References

Light:

[1] Lighting concept based on literature search see D1.1

Scent and Sound:

[2] AGÖF (2013). Orientierungswerte für flüchtige organische Verbindungen in der Raumluft (Aktualisierte Fassung November 2013), Online available: 30.11.2017, URL: http://www.agoef.de/orientierungswerte/agoef-voc-orientierungswerte.html

[3] Deutsches Umweltbundesamt (2007). Gesundheit und Umwelthygiene - Richtwerte für die Innenraumluft. Online available: 30.11.2017, URL: http://www.umweltbundesamt.de/gesundheit/innenraumhygiene/richtwerte-irluft.htm

[4] G. Ohloff (1990). Riechstoffe und Geruchssinn. Springer-Verlag, Berlin Heidelberg New York.

[5] V. Heitmann (2008). Teuflische Düfte. Online available: 30.11.2017. URL: https://www.forum-essenzia.org/downloads/080305daab2.pdf

Sensors:

[6] Wahl, F., Milenkovic, M., & Amft, O. (2012, December). A distributed PIR-based approach for estimating people count in office environments. In Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on (pp. 640-647). IEEE.

[7] Zappi, P., Farella, E., & Benini, L. (2007, September). Enhancing the spatial resolution of presence detection in a PIR based wireless surveillance network. In Advanced Video and Signal Based Surveillance, 2007. AVSS 2007. IEEE Conference on (pp. 295-300). IEEE.

[8] Luo, Xiaomu; Guan, Qiuju; Tan, Huoyuan; Gao, Liwen; Wang, Zhengfei; Luo, Xiaoyan (2017): Simultaneous Indoor Tracking and Activity Recognition Using Pyroelectric Infrared Sensors. In: Sensors (Basel, Switzerland) 17 (8). DOI: 10.3390/s17081738.

[9] Nef, Tobias; Urwyler, Prabitha; Büchler, Marcel; Tarnanas, Ioannis; Stucki, Reto; Cazzoli, Dario et al. (2015): Evaluation of Three State-of-the-Art Classifiers for Recognition of Activities of Daily Living from Smart Home Ambient Data. In: Sensors (Basel, Switzerland) 15 (5), S. 11725–11740. DOI: 10.3390/s150511725.

[10] Sereda, A., Moreau, J., Boulade, M., Olivéro, A., Canva, M., & Maillart, E. (2015). Compact 5-LEDs illumination system for multi-spectral surface plasmon resonance sensing. Sensors and Actuators B: Chemical, 209, 208-211.

[11] Beyer, C., & Ernib, D. (2014). Accurate Multiscale Skin Model Suitable for Determining the Sensitivity and Specificity of Changes of Skin Components. Computational Biophysics of the Skin, 353.

[12] Ulrich Reimer; Emanuele Laurenzi; Edith Maier; Tom Ulmer (2017): Mobile Stress Recognition and Relaxation Support with SmartCoping: User-Adaptive Interpretation of Physiological Stress Parameters Hilton Waikoloa Village, Hawaii, USA, January 4-7, 2017. In: 50th Hawaii International Conference on System Sciences, HICSS 2017, Hilton Waikoloa Village, Hawaii, USA, January 4-7, 2017: AIS Electronic Library (AISeL). Online verfügbar unter http://aisel.aisnet.org/hicss-50/hc/apps_for_health_management/5.

[13] Reimer, Ulrich; Emmenegger, Sandro; Maier, Edith; Zhang, Zhongxing; Khatami, Ramin (2017): Recognizing Sleep Stages with Wearable Sensors in Everyday Settings. In: Proceedings of the 3rd International Conference on Information and Communication Technologies for Ageing Well and e-Health. 3rd International Conference on Information and Communication Technologies for Ageing Well and e-Health. Porto, Portugal: SCITEPRESS - Science and Technology Publications, S. 172–179.

11. List of Figures

12. List of Tables