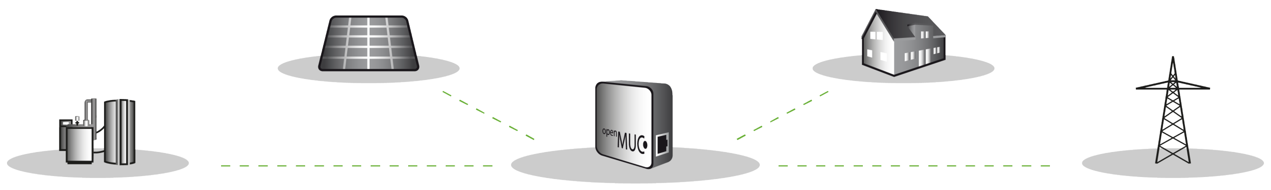


OpenMUC User Guide

Fraunhofer Institute for Solar Energy Systems ISE

openmuc.org

Table of Contents

1. Intro

OpenMUC is a software framework based on Java and OSGi that simplifies the development of customized monitoring, logging and control systems. It can be used as a basis to flexibly implement anything from simple data loggers to complex SCADA systems. The main goal of OpenMUC is to shield the application developer of monitoring and control applications from the details of the communication protocol and data logging technologies. Third parties are encouraged to create their own customized systems based on OpenMUC. OpenMUC is licensed under the GPL. If you need an individual license please contact us.

For a short overview of OpenMUC's goals and features please visit our overview page. This guide is a detailed documentation on how OpenMUC works and how to use it.

2. Architecture

The following image depicts the software layers of an OpenMUC system.

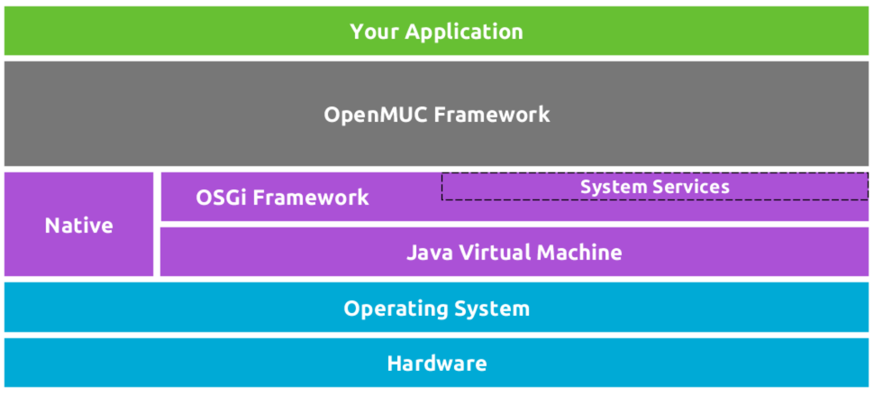


Figure 1. OpenMUC software layers

The OpenMUC framework runs within an OSGi environment which in turn is being run by a Java Virtual Machine. The underlying operating system and hardware can be chosen freely as long as it can run a Java 7 VM.

OpenMUC consists essentially of various software modules which are implemented as OSGi bundles that run in the OSGi environment and communicate over OSGi services. The following figure illustrates the main modules that make up OpenMUC.

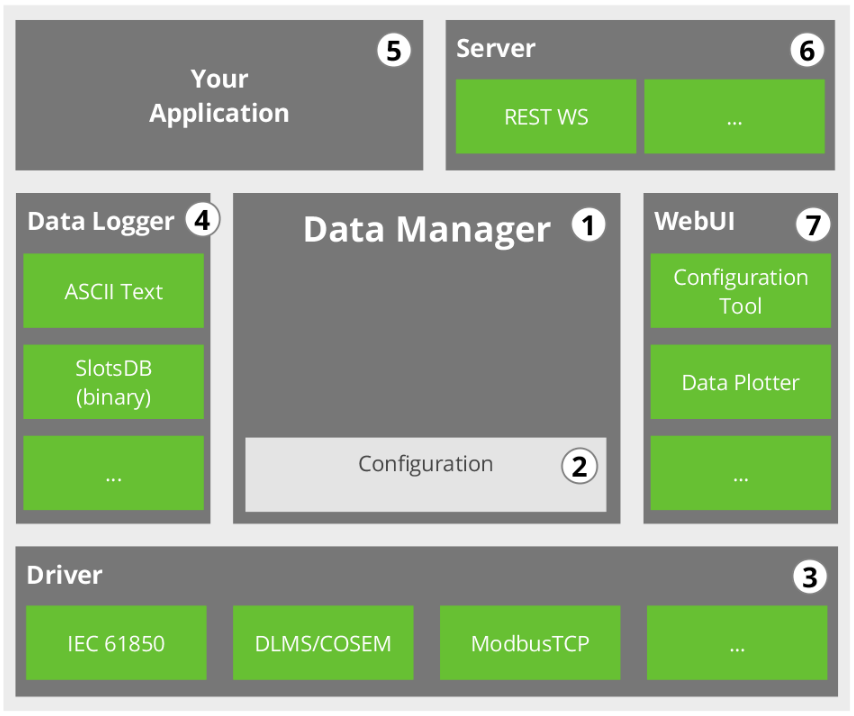


Figure 2. OpenMUC modules

All modules except for the data manager are optional. Thus by selecting the modules you need you can easily create your own customized and lightweight system.

The different modules in the picture are now further explained:

1. The **data manager** represents the core and center of OpenMUC. Virtually all other OpenMUC modules (e.g. drivers, data loggers, servers, applications and web interface plugins) communicate with it through OSGi services. The data manager gets automatically notified when new drivers or data loggers get installed. OpenMUC applications communicate with devices, access logged data or change the configuration by calling service functions provided by the data manager. It is therefore the data manager that shields the application programmer from the details of the communication and data logging technology. What the data manager does is mostly controlled through a central configuration.
2. The **channel configuration** holds the user defined data channels and its parameters. Data channels are the frameworks representation of data points in connected devices. Amongst others the channel configuration holds the following information:

   a. communication parameters that the drivers require
   b. when to sample new data from connected devices
   c. when to send sampled data to existing data logger(s) for efficient persistent storage.

   The configuration is stored in the file conf/channels.xml. You may add or modify the configured channels by manually editing the channels.xml file or through the channel configurator web interface.
3. A **driver** is used by the data manager to send/get data to/from a connected device. Thus a driver usually implements a communication protocol. Several communication drivers have already been developed (e.g. IEC

61850, ModbusTCP, KNX, DLMS/COSEM). Many drivers use standalone communication libraries (e.g. OpenIEC61850, jMBus) developed by the OpenMUC team. These libraries do not depend on the OpenMUC framework and can therefore be used by any Java application. New communication drivers for OpenMUC can be easily developed by third parties.

4. A **data logger** saves sampled data persistently. The data manager forwards sampled data to all available data loggers if configured to do so. Data loggers are specifically designed to store time series data for short storage and retrieval times. Note that this usually means that they are not SQL-based. OpenMUC currently includes two data loggers. The ASCII data logger saves data in a human readable text format while SlotsDB saves data in a more efficient binary format.
5. If all you want is sample and log data then you can use the OpenMUC framework as it is and simply configure it to your needs. But if you want to process sampled data or control a device you will want to write your own **application**. Like all other modules your application will be an OSGi bundle. In your application you can use the DataAccessService and the ConfigService provided by the data manager to access sampled and logged data. You may also issue immediate read or write commands. These are forwarded by the data manager to the driver. The configuration (when to sample and to log) can also be changed during run-time by the application. At all times the application only communicates with the data manager and is therefore not confronted with the complicated details of the communication technology being used.
6. If your application is located on a remote system (e.g. a smart phone or an Internet server) then the data and configuration can be accessed through an OpenMUC **server**. At the moment OpenMUC provides a RESTful web service for this purpose.
7. Finally the OpenMUC framework provides a web user interface (**WebUI**) for tasks such as configuration, visualization of sampled data or exporting logged data. The web interface is modular and provides a plug-in interface. This way developers may write a website that integrates into the main menu of the web interface. The WebUI is mostly for configuration and testing purposes. Most companies will want to create their own individual UI.

3. Quick Start

This chapter will give you an idea of how OpenMUC works by showing you how to run and adjust the demo framework which is part of the OpenMUC distribution. OpenMUC requires Java 7 or higher, therefore make sure it is installed on your machine.

3.1. Framework Files

The folder "framework" contains a configured OpenMUC framework that can be used as a basis to create your own customized OpenMUC framework for your task. The framework folder contains the following important files and folders:

felix

The main Apache Felix OSGi jar which is run to start OpenMUC.

bin

Run scripts for Linux/Unix and Windows.

bundle

Contains all bundles that are started by the Felix OSGi framework. Note that this folder does not contain all available OpenMUC bundles but only a subset for demonstration purposes.

log

Log files produced by the running framework.

conf

Various configuration files of the framework.

3.2. Starting the Demo

First open a terminal and go to the folder named "framework".

To start OpenMUC on Linux run:

```
./bin/openmuc start -fg
```

To start OpenMUC on Window run:

```
bin\openmuc.bat
```

This will start the Apache Felix OSGi framework which in turn starts all the bundles located in the "bundle" folder. Among the bundles that are started is the Apache Gogo shell. This shell is entered once you run OpenMUC.

The shell can be used to start, stop, and reload bundles among other things. You can stop and exit the OSGi framework any time by typing "ctrl+d" or "stop 0".

Now type "lb" to list all installed bundles.

You will see that among the active bundles are the Simple Demo App, the OpenMUC core bundles, two data loggers (ASCII and SlotsDB) as well as the CSV driver.

3.3. Configuration

One of the most important configuration files is conf/channels.xml. This file tells the OpenMUC Data Manager which data it shall sample, listen for, and log. This is done by configuring so called channels. Each channel represents a single data point whose value can be an integer, double, string or byte array.

The demo’s channels.xml configures among others channels whose values are read by the CSV driver. The channels are sampled/read and logged every 5 seconds.

For more information about the channels.xml see the configuration chapter.

3.4. Simple Demo App

The simple demo app demonstrates how you can access channels and their records from an application. The app reads data from channels of the CSV driver, calculates new values from them and writes them to other channels. The app can be used as starting point to create your own OpenMUC application.

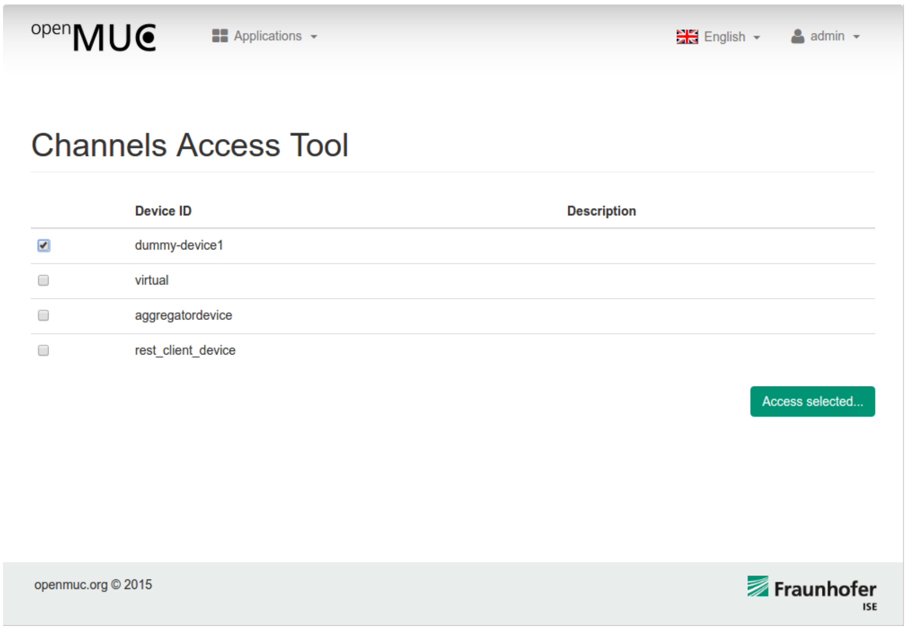
3.5. WebUI Walk Through

This section leads you through the framework’s WebUI.

Open a browser (works currently best with Google Chrome) and enter the URL "http://localhost:8888". This leads you to the login page. The default user is *admin* and the default password is *admin* as well.

After successful login the OpenMUC Dashboard opens, which provides various plugins for configuration and visualization. A full description of the plugins can be found in the chapter Web UI.

Let us first look at the Channel Access Tool which provides the current value of each channel and also enables you to write values. Click on Channel Access Tool to open this plugin. The next page lists all available devices which are currently configured in OpenMUC. Select the *home1* and proceed with *Access selected*.

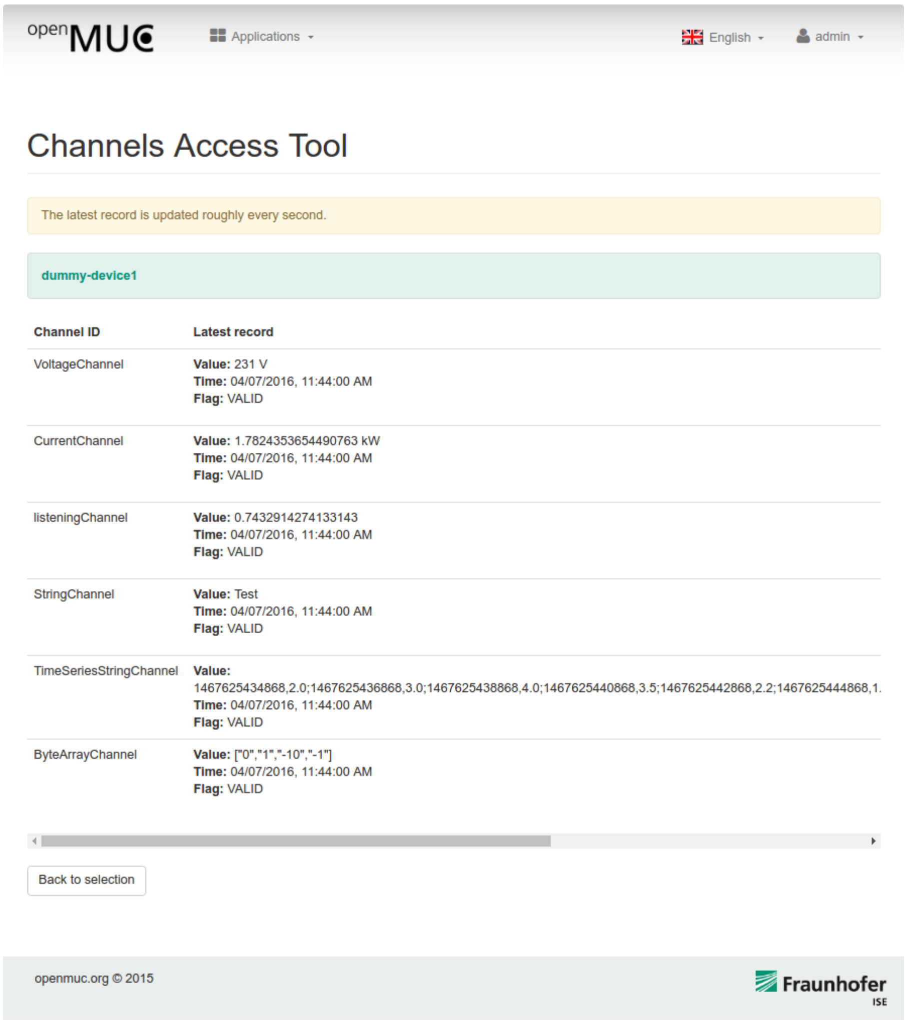


Channels Access Tool

	Device ID	Description
☑	dummy-device1	
☐	virtual	
☐	aggregatordevice	
☐	rest_client_device	

Figure 3. WebUI device selection

On the next page you will see the latest records of all channels of home1. Each record consists of a data value, a timestamp when it was sampled and a quality flag.



Channels Access Tool

The latest record is updated roughly every second.

dummy-device1

Channel ID	Latest record
VoltageChannel	**Value:** 231 V **Time:** 04/07/2016, 11:44:00 AM **Flag:** VALID
CurrentChannel	**Value:** 1.7824353654490763 kW **Time:** 04/07/2016, 11:44:00 AM **Flag:** VALID
listeningChannel	**Value:** 0.7432914274133143 **Time:** 04/07/2016, 11:44:00 AM **Flag:** VALID
StringChannel	**Value:** Test **Time:** 04/07/2016, 11:44:00 AM **Flag:** VALID
TimeSeriesStringChannel	**Value:** 1467625434868,2.0;1467625436868,3.0;1467625438868,4.0;1467625440868,3.5;1467625442868,2.2;1467625444868,1. **Time:** 04/07/2016, 11:44:00 AM **Flag:** VALID
ByteArrayChannel	**Value:** ["0","1","-10","-1"] **Time:** 04/07/2016, 11:44:00 AM **Flag:** VALID

Figure 4. WebUI channel access tool

Let’s have look at the *Data Plotter*. To get to the *Data Plotter* click on *Applications* next to the OpenMUC logo and select *Data Plotter*.

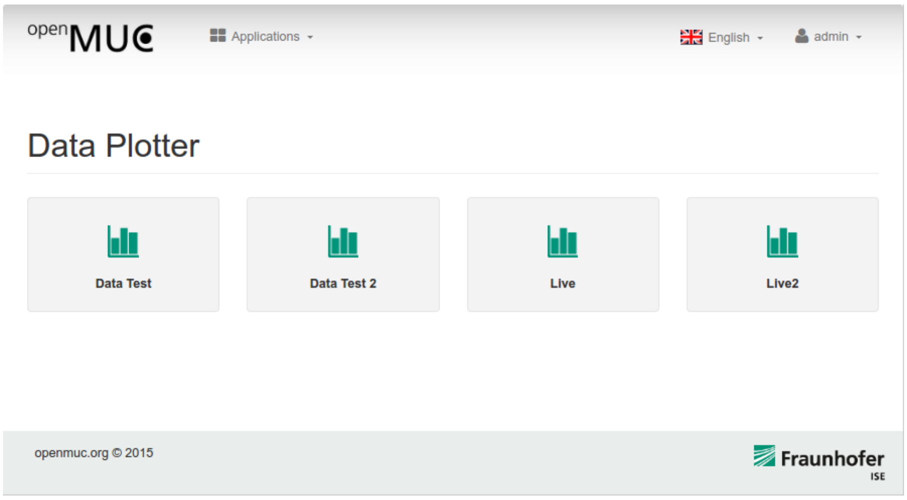


Figure 5. WebUI data plotter

Select the *Live* Data Plotter. To view the live data select the channels of your choice and click *Plot Data*.

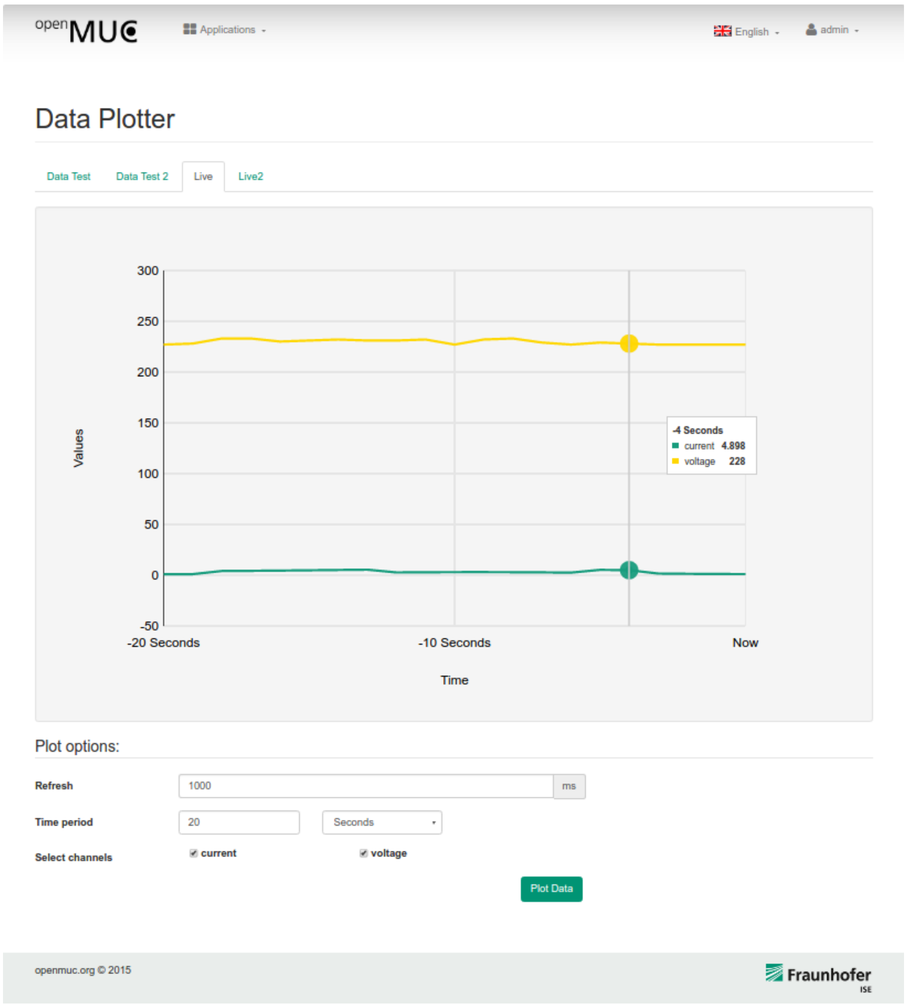


Figure 6. WebUI live plotter

3.5.1. Add a New Channel

All channels currently defined get their data using the CSV driver from the file "csv-driver/home1.csv". That file contains additional data. So let us now add a new channel to the OpenMUC configuration using the channel scan feature.

In the WebUI go to the Channel Configurator. Click the tab "Devices". In the row of device "home1" click on the search/scan icon. It shows you all the channels available in that device. Once the scan has completed a list of available channels is shown. In this tutorial we select the channel with address "pv_energy_production". Click "add channels".

Now the channel overview opens where we can find our selected channel. In the last step of the configuration we click on the edit icon of the channel and set the parameters *logging interval* and *sampling interval* to 5000 ms and change the unit to kWh.

You can now check that the new channel was added to the "conf/channels.xml" file.

After submitting the channel configuration we go back to the dashboard and open the Channel Access Tool. Here we select our home1 device and continue with *access selected*. Now we able to see the current values of the pv_energy_production channel.

The logged data can be found in *openmuc/framework/data/ascii/<currentdate>_5000.dat*

4. Distribution

The distribution contains the following important files and folders:

build/libs-all

    All modules/bundles that make up the OpenMUC framework

dependencies

    Information on the external dependencies of the OpenMUC framework. Also contains the RXTX library (repacked as a bundle) which is needed by many OpenMUC drivers based on serial communication.

projects

    All sources of the OpenMUC framework. You can easily change and rebuild OpenMUC using Gradle.

framework

    A ready to use OpenMUC demo framework that is introduced next.

5. Running OpenMUC

To start OpenMUC on Linux run:

```
./bin/openmuc start
```

This runs OpenMUC as a background process. If you want to run OpenMUC in the foreground run:

```
./bin/openmuc start -fg
```

To start OpenMUC under Window run:

```
bin\openmuc.bat
```

The Linux start script is a sophisticated bash script that can be used to start, stop, restart OpenMUC. The Windows run script is a simple bat file that starts OpenMUC. The following explanations will focus on using OpenMUC in a Linux environment as it is the more common scenario.

Starting OpenMUC really means running the Felix OSGi Framework by executing

java -jar felix/felix.jar

The Felix OSGi Framework will then start all bundles located in the "bundle" folder.

When you start OpenMUC in the foreground you will enter the Felix Gogo shell. From the shell you can start, stop and reload bundles among many other things. To quit the shell and stop the framework press ctrl+d or enter "stop 0".

If you ran OpenMUC as a background process you can access the Gogo shell using the telnet:

```
netcat 127.0.0.1 6666
```

or using the openmuc run script

```
./bin/openmuc remote-shell
```

Pressing ctrl+d will exit the remote shell but not stop the openmuc framework.

To stop OpenMUC run:

```
./bin/openmuc stop
```

6. Install a Driver

When you want to use a new driver you have to copy the corresponding jar file from the folder "build/libs-all/" to the "bundle" folder of the framework. Many drivers are "fat jars" which include their dependencies. An exception is the RXTX library which cannot be packed with the jars.

6.1. Use a Driver with Serial Communication

When you need to use a driver that uses serial communication you have to copy the RXTX bundle to the frameworks "bundle" folder.

```
cp ../dependencies/rxtx/rxtxcomm_api-2.2pre2.jar ./bundle/
```

Additionally you need to install librxtx-java:

```
sudo apt-get install librxtx-java
```

The serial ports /dev/tty* are only accessible to members belonging to the group dialout. We therefore have to add our user to that group. E.g. using:

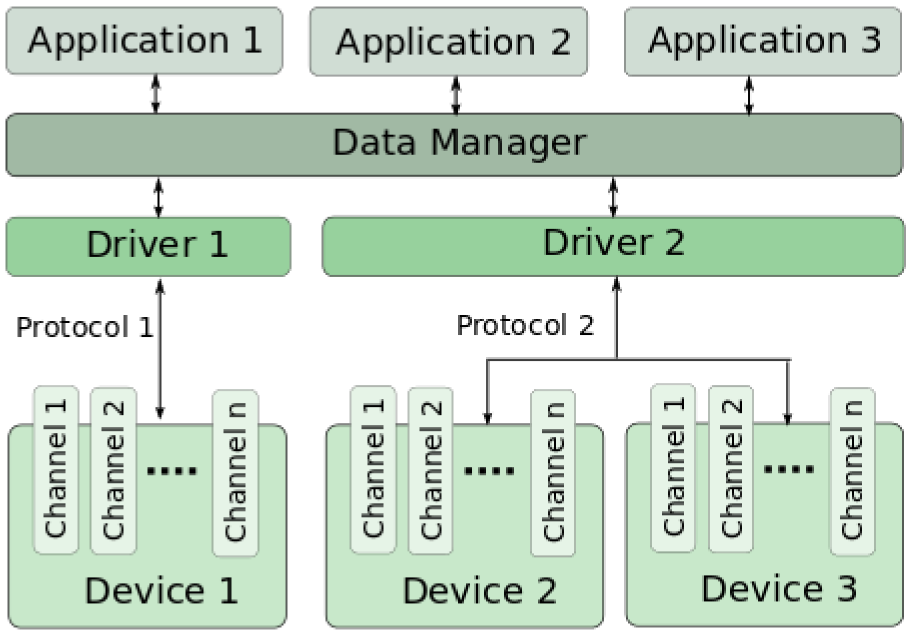
```
sudo adduser <yourUserName> dialout
```

7. Devices and Channels

OpenMUC works on the basis of channels. A channel basically represents a single data point. Some examples for a channel are the metered active power of a smart meter, the temperature of a temperature sensor, any value of digital or analog I/O module or the some manufacture data of the device. Thus a channel can represent any kind of data point.

The following picture illustrates the channel concept.

OpenMUCs Channel Concept



7.1. Configuration

The **conf/channels.xml** file is the main configuration file for OpenMUC. It tells the OpenMUC framework which channels it should log and sample. It contains a hierarchical structure of drivers, devices and channels. A driver can have one or more devices and devices can have one or more channels. Following listing shows a sample configuration to illustrate the hierarchical structure. The driver, device and channel options are explained afterwards.

Listing 1. channels.xml structure

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration>
    <logger>loggerId</logger>

    <driver id="driver_x">
        <!-- driver options -->
        <device>
            <!-- device options -->
            <channel>
                <!-- channel options -->
            </channel>
            <channel>
                <!-- channel options -->
            </channel>
        </device>
    </driver>
</configuration>
```

The available driver settings, device settings and channel settings can be found in the Javadoc of DriverConfig.java, DeviceConfig.java and ChannelConfig.java respectively.

Default Data Logger

You can define a default data logger by adding a logger element with the id of a data logger to the configuration. If available, that data logger is used to read logged values. The ids of data loggers shipped with the OpenMUC Framework are defined in the "Data Loggers" chapter. If no logger with the defined id is available, or the logger element is missing from the configuration, an arbitrary available logger is used to read logged values. Only one default logger may be defined. If multiple logger elements exists, only the first one is evaluated.

This configuration only affects reading of already logged values. Channels are still logged by all available loggers.

7.2. Sampling, Listening and Logging

- **sampling** is when the data manager frequently asks a driver to retrieve a channel value.
- **listening** is when the driver listens on a channel and forwards new values to the data manager.
- **logging** is when the data manager forwards the current sampled value to the data loggers that are installed. The data loggers then store the data persistently

The following examples will give you a better understanding of these three settings.

Listing 2. Example 1: Just Sampling

```xml
<channel>
    <id>channel1</id>
    <channelAddress>dummy/channel/address/1</channelAddress>
    <samplingInterval>4s</samplingInterval>
</channel>
```

In example 1 the channel is sampled every 4 seconds which means the data manager requests every 4 seconds the current value from the driver.

Listing 3. Example 2: Sampling and Logging

```xml
<channel>
    <id>channel2</id>
    <channelAddress>dummy/channel/address/2</channelAddress>
    <samplingInterval>4s</samplingInterval>
    <loggingInterval>8s</loggingInterval>
</channel>
```

Example 2 extends example 1 by an additional logging. The logging interval is set to 8 seconds which means that every 8 seconds the last sampled value is stored in the database. In this case every second sampled value is stored because the sampling interval is 4 seconds. To log every sampled value the sampling interval and logging interval need to be the same.

Listing 4. Example 3: Just Listening

```
<channel>
    <id>channel3</id>
    <channelAddress>dummy/channel/address/3</channelAddress>
    <listening>true</listening>
</channel>
```

In example 3 listening instead of sampling is used. This means that the driver reports a new channel value to the data manager when the value has changed for example.

Listing 5. Example 4: Listening and Logging

```
 <channel>
    <id>channel4</id>
    <channelAddress>dummy/channel/address/4</channelAddress>
    <listening>true</listening>
    <loggingInterval>8s</loggingInterval>
</channel>
```

Example 4 extends example 3 by an additional logging.

When listening is true and additional a sampling interval is defined then the sampling is ignored.

8. Drivers

8.1. Modbus

Modbus Homepage: http://www.modbus.org
Modbus Protocol Specifications: http://www.modbus.org/specs.php
Modbus Master Simulator modpoll: http://www.modbusdriver.com/modpoll.html

The Modbus driver supports RTU, TCP and RTU over TCP.

Table 1. Configuration Synopsis

	TCP (ethernet)	**RTU (serial)**	**RTUTCP (serial over ethernet)**
ID	modbus		
Device Address	<ip>[:<port>]	<serial port>	<ip>[:<port>]
Settings	<type>	<type>:<encoding>:<baudrate>:<databits>:<parity>:<stopbits>:<echo>:<flowControlIn>:<flowControlOut>	<type>
Channel Address	<UnitId>:<PrimaryTable>:<Address>:<Datatyp>		

DeviceAddress

For TCP and RTUTCP

The DeviceAddress is specified by an IP address and an optional port. If no port is specified, the driver uses the modbus default port 502.

For RTU:

The DeviceAddress is specified by a serial port like /dev/ttyS0.

The driver uses the jamod library which itself uses the rxtx library for serial communication. Therefor the librxtx-java package needs to be installed on the system. Furthermore the user needs to be in the groups dialout and plugdev

Settings

Table 2. Settings

Config	Description/ Values
<type>	RTU|TCP|RTUTCP
<encoding>	SERIAL_ENCODING_RTU
<baudrate>	Integer value: e.g.: 2400, 9600, 115200
<databits>	DATABITS_5, DATABITS_6, DATABITS_7, DATABITS_8
<parity>	PARITY_EVEN, PARITY_MARK, PARITY_NONE, PARITY_ODD, PARITY_SPACE
<stopbits>	STOPBITS_1, STOPBITS_1_5, STOPBITS_2
<echo>	ECHO_TRUE, ECHO_FALSE
<flowControlIn>	FLOWCONTROL_NONE, FLOWCONTROL_RTSCTS_IN, FLOWCONTROL_XONXOFF_IN
<flowControlOut>	FLOWCONTROL_NONE, FLOWCONTROL_RTSCTS_OUT, FLOWCONTROL_XONXOFF_OUT

Listing 6. Example Settings

```
<channelAddress>
RTU:SERIAL_ENCODING_RTU:38400:DATABITS_8:PARITY_NONE:STOPBITS_1
:ECHO_FALSE:FLOWCONTROL_NONE:FLOWCONTROL_NONE
</channelAddress>
```

ChannelAddress

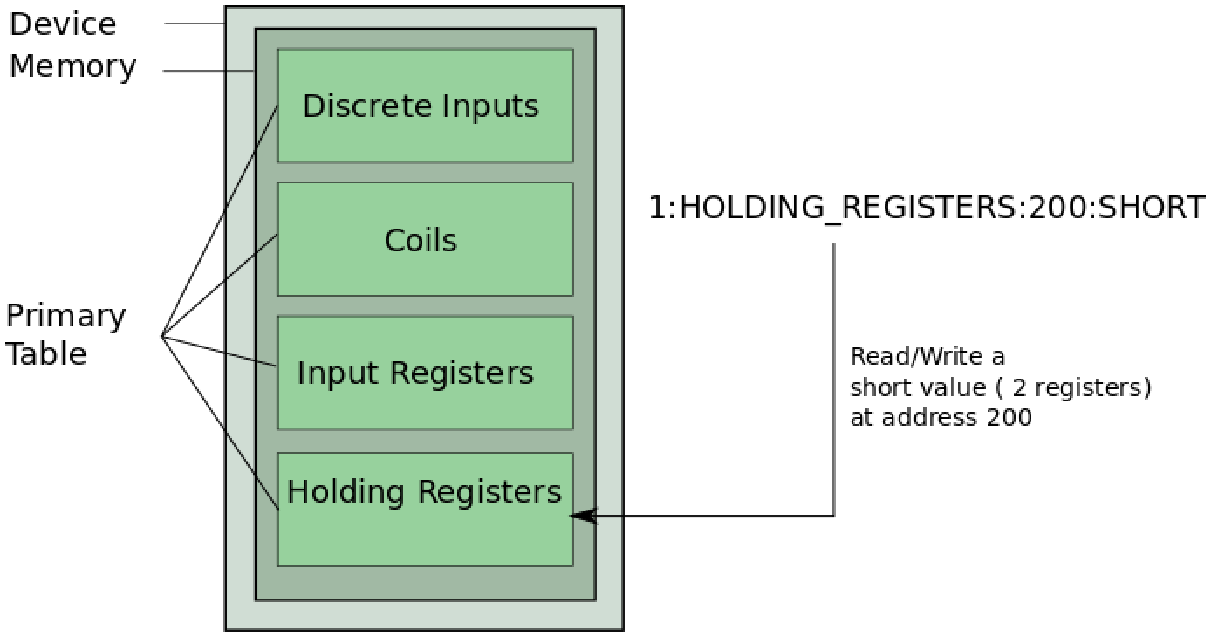
The ChannelAddress consists of four parts: UnitId, PrimaryTable, Address and Datatyp which are explained in detail in the following table.

Table 3. Parameter Description

Parameter	Description
UnitId	In homogenious architecture (when just MODBUS TCP/IP is used) On TCP/IP, the MODBUS server is addressed by its IP address; therefore, the MODBUS Unit Identifier is useless. The value 255 (0xFF) has to be used. In heterogeneous architecture (when using MODBUS TCP/IP and MODBUS serial or MODBUS+) This field is used for routing purpose when addressing a device on a MODBUS+ or MODBUS serial line sub-network. In that case, the “Unit Identifier” carries the MODBUS slave address of the remote device. The MODBUS slave device addresses on serial line are assigned from 1 to 247 (decimal). Address 0 is used as broadcast address. Note: Some MODBUS devices act like a bridge or a gateway and require the UnitId even if they are accessed through TCP/IP. One of those devices is the Janitza UMG. To access data from the Janitza the UnitId has to be 1.

Parameter	Description
PrimaryTable	PrimaryTable defines the which part of the device memory should be accessed. Valid values: COILS, DISCRETE_INPUTS, INPUT_REGISTERS, HOLDING_REGISTERS
Address	Address of the channel/register. Decimal integer value - not hex!
Datatyp	Valid values: BOOLEAN, SHORT, INT, FLOAT, DOUBLE, LONG, BYTEARRAY[n] (n = number of REGISTERS not BYTES, 1 Register = 2 Bytes!)

Primary Tables and Channel Address



Valid Address Parameter Combinations

Since COILS and DISCRETE_INPUTS are used for bit access, only the data type BOOLEAN makes sense in combinations with of one of these. INPUT_REGISTERS and HOLDING_REGISTERS are used for register access. There is also a difference between reading and writing. Only COILS and HOLDING_REGISTERS are readable and writable. DISCRETE_INPUTS and INPUT_REGISTERS are read only. The following table gives an overview of valid parameter combinations of PrimaryTable and Datatyp.

Table 4. Valid Address Parameters for reading a channel

Primary Table	BOOLEAN	SHORT	INT	FLOAT	DOUBLE	LONG	BYTEARRAY[n]
COILS	x	-	-	-	-	-	-
DISCRETE_INPUTS	x	-	-	-	-	-	-
INPUT_REGISTERS	-	x	x	x	x	x	x
HOLDING_REGISTERS	-	x	x	x	x	x	x

Table 5. Valid Address Parameters for writing a channel

Primary Table	BOOLEAN	SHORT	INT	FLOAT	DOUBLE	LONG	BYTEARRAY[n]
COILS	x	-	-	-	-	-	-
DISCRETE_INPUTS	-	-	-	-	-	-	-

Primary Table	BOOLEAN	SHORT	INT	FLOAT	DOUBLE	LONG	BYTEARRAY[n]
INPUT_REGISTERS	-	-	-	-	-	-	-
HOLDING_REGISTERS	-	x	x	x	x	x	x

Listing 7. Examples for valid addresses

```
<channelAddress>255:INPUT_REGISTERS:100:SHORT</channelAddress>
<channelAddress>255:COILS:412:BOOLEAN</channelAddress>
```

Listing 8. Examples for invalid addresses

```
<channelAddress>255:INPUT_REGISTERS:100:BOOLEAN</channelAddress> (BOOLEAN
doesn't go with INPUT_REGISTERS)
<channelAddress>255:COILS:412:LONG</channelAddress> (LONG does not go with
COILS)
```

Function Codes (more detailed information about how the driver works)

The driver is based on the Java Modbus Library (jamod) which provides read and write access via modbus. Following table shows which modbus function code is used to access the data of the channel.

Table 6. Relation between function code and channel address

jamod Method	Modbus Function Code	Primary Table	Access	Java Data Type
ReadCoilsRequest	FC 1 Read Coils	Coils	RW	boolean
ReadInputDiscretesRequest	FC 2 Read Discrete Inputs	Discrete Inputs	R	boolean
ReadMultipleRegistersRequest	FC 3 Read Holding Registers	Holding Registers	RW	short, int, double, long, float, bytearray[]
ReadInputRegistersRequest	FC 4 Read Input Registers	Input Registers	R	short, int, double, long, float, bytearray[]
WriteCoilRequest	FC 5 Write Single Coil	Coils	RW	boolean
WriteMultipleCoilsRequest	FC 15 Write Multiple Coils	Coils	RW	boolean
WriteMultipleRegistersRequest	FC 6 Write Single Registers	Holding Registers	RW	short, int, double, long, float, bytearray[]
WriteMultipleRegistersRequest	FC 16 Write Multiple Registers	Holding Registers	RW	short, int, double, long, float, bytearray[]

Example

```
<channelAddress>255:INPUT_REGISTERS:100:SHORT</channelAddress> will be accessed
via function code 4.
```

8.1.1. Modbus TCP and Wago

Till now the driver has been tested with some modules of the Wago 750 Series with the Fieldbus-Coupler 750-342

If you want to use the Modbus TCP driver for accessing a Wago device you first need to know how the process image is build. From the process image you can derive the register addresses of your Wago modules (AO, AI, DO, DI). You find detailed information about the process image in WAGO 750-342 Manual on page 46 and 47.

The following Examples are based on figure Wago 750-342 Process Image

*Example 1: Read AI 2 from first (left) 472-module (Register Address 0x0001)

```
<channelAddress>255:INPUT_REGISTERS:1:SHORT</channelAddress>
```

Example 2: Read DI 3 from first (left) 472-module (Register Address 0x0003)

```
<channelAddress>255:DISCRETE_INPUTS:3:BOOLEAN</channelAddress>
```

Example 3: Write AO 1 from first (left) 550-module (Register Address 0x0000/0x0200)

For writing only the +0x0200 addresses should be used! Since the driver accepts only a decimal channelAddress 0x0200 must be converted to decimal. The resulting address would be:

```
<channelAddress>255:HOLDING_REGISTERS:512:SHORT</channelAddress>
```

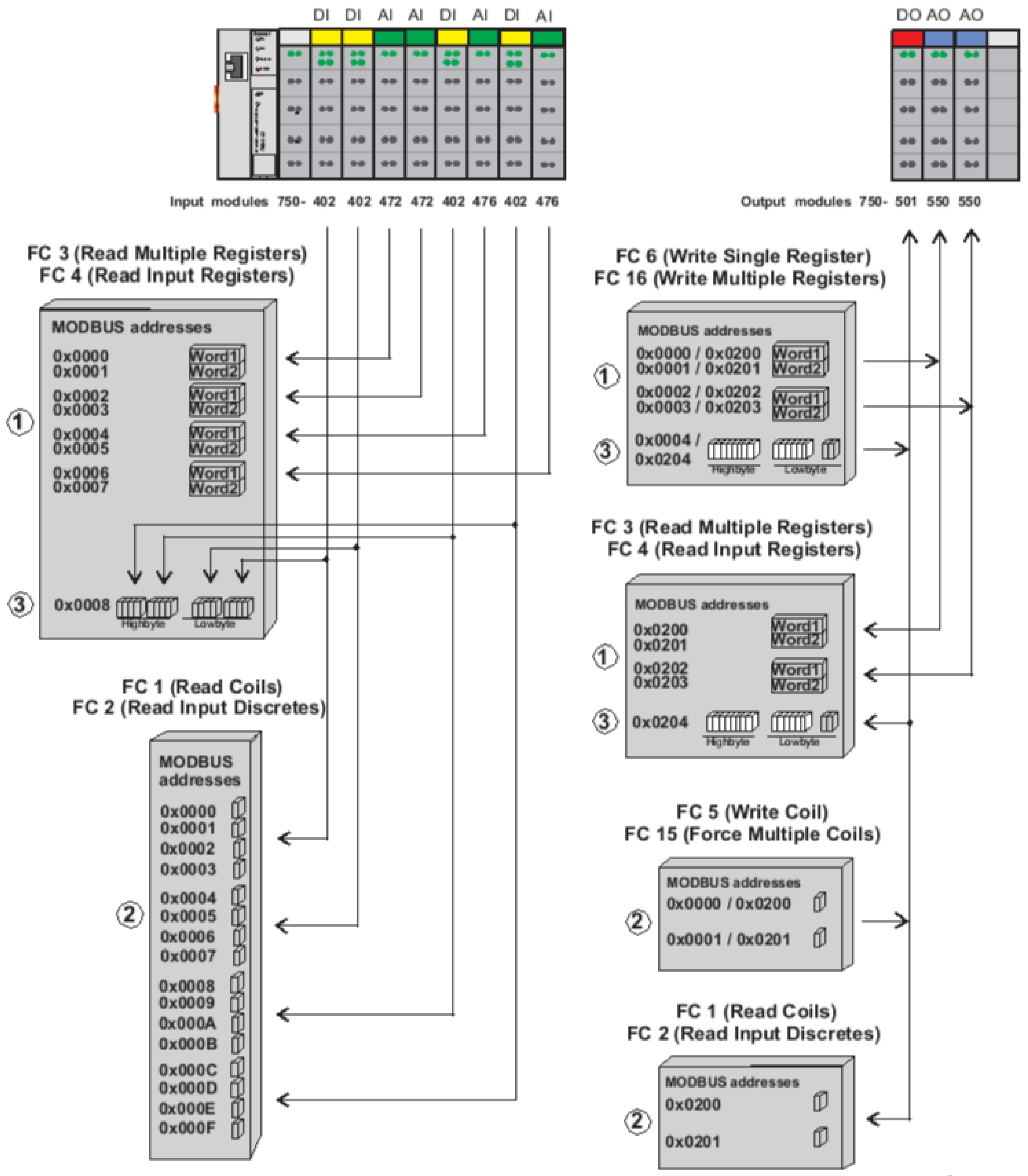
Example 4: Write DO 2 from 501-module (Register Address 0x0000/0x0201)
For writing only the +0x0200 addresses should be used! Since the driver accepts only a decimal channelAddress 0x0201 must be converted to decimal. The resulting address would be:

```
<channelAddress>255:COILS:513:BOOLEAN</channelAddress>
```

Example 5: Read back DO 2 from 501-module (Register Address 0x0201)

```
<channelAddress>255:COILS:513:BOOLEAN</channelAddress> or
<channelAddress>255:DISCRETE_INPUTS:513:BOOLEAN</channelAddress>
```

Wago 750-342 Process Image


DI DI AI AI DI AI DI AI
DO AO AO
Input modules 750- 402 402 472 472 402 476 402 476
Output modules 750- 501 550 550
FC 3 (Read Multiple Registers)
FC 4 (Read Input Registers)
MODBUS addresses
0x0000 0x0001 0x0002 0x0003 0x0004 0x0005 0x0006 0x0007
Word1 Word2
0x0008
Highbyte Lowbyte
FC 1 (Read Coils)
FC 2 (Read Input Discretes)
0x0000 0x0001 0x0002 0x0003 0x0004 0x0005 0x0006 0x0007 0x0008 0x0009 0x000A 0x000B 0x000C 0x000D 0x000E 0x000F
FC 6 (Write Single Register)
FC 16 (Write Multiple Registers)
0x0000 / 0x0200 0x0001 / 0x0201 0x0002 / 0x0202 0x0003 / 0x0203 0x0004 / 0x0204
0x0200 0x0201 0x0202 0x0203 0x0204
FC 5 (Write Coil)
FC 15 (Force Multiple Coils)
0x0000 / 0x0200 0x0001 / 0x0201
0x0200 0x0201
1 2 3

Source: Wago 750-342 Manual V 2.1.1

8.2. M-Bus (wired)

M-Bus is communication protocol to read out meters.

Table 7. Configuration Synopsis

ID	mbus
Device Address	<serial_port>:<mbus_address>
Settings	[<baudrate>][:timeout]
Channel Address	<dib>:<vib>

Device Address

<serial_port> - The serial port should be given that connects to the M-Bus converter. (e.g. /dev/ttyS0, /dev/ttyUSB0 on Linux).

<mbus_address> - The mbus adress can either be the the primary address or secondary address of the meter. The primary address is specified as integer (e.g. 1 for primary address 1) whereas the secondary address consits of 8 bytes that should be specified in hexadecimal form. (e.g. e30456a6b72e3e4e)

Settings

<baudrate> - If left empty the default is used: "2400"

<timeout> - Defines the read timeout in ms. Default is 2500 ms. Example: t5000 for timeout of 5 seconds

Channel Address

Shall be of the format <dib>:<vib> in a hexadecimal string format (e.g. 04:03 or 02:fd48)

8.3. M-Bus (wireless)

Wireless M-Bus is communication protocol to read out meters and sensors.

Table 8. Configuration Synopsis

ID	wmbus
Device Address	<serial_port>:<secondary_address>
Settings	<transceiver> <mode> [<key>]
Channel Address	<dib>:<vib>

Device Address

<serial_port> - The serial port used for communication. Examples are /dev/ttyS0 (Linux) or COM1 (Windows)

<secondary_address> - The secondary address consists of 8 bytes that should be specified in hexadecimal form. (e.g. e30456a6b72e3e4e)

Settings

<transceiver> - The transceiver being used. It can be *amber* or *rc* for modules from RadioCrafts.

<mode> - The wM-Bus mode can be S or T.

<key> - The key in hexadecimal form.

Channel Address

Shall be of the format <dib>:<vib> in a hexadecimal string format (e.g. 04:03 or 02:fd48)

8.4. IEC 61850

IEC 61850 is an international communication standard used mostly for substation automation and controlling distributed energy resources (DER). The IEC 61850 driver uses the client library from the OpenIEC61850 project.

ID	iec61850
Device Address	<host>[:<port>]
Settings	[-a <authentication parameter>] [-lt <local t-selector>] [-rt <remote t-selector>]
Channel Address	<bda reference>:<fc>

Channel Address

The channel address should be the IEC 61850 Object Reference and the Functional Constraint of the Basic Data Attribute that is to be addressed separated by a colon. Note that an IEC 61850 timestamp received will be converted to a LongValue that represents the milliseconds since 1970. Some information is lost during this conversion because the IEC 61850 timestamp is more exact.

Settings

The defaults for TSelLocal and TSelRemote are "00" and "01" respectively. You can also set either TSelector to the empty string (e.g. "-lt -rt"). This way they will be omitted in the connection request.

8.5. IEC 62056 part 21

The IEC 62056 part 21 driver can be used to read out meter via optical interface

Table 9. Configuration Synopsis

ID	iec62056p21
Device Address	<serial_port>
Settings	
Channel Address	<data_set_id>

Device Address

<serial_port> - The serial port should be given that connects to the M-Bus converter. (e.g. /dev/ttyS0, /dev/ttyUSB0 on Linux).

Channel Address

<data_set_id> - Id of the data set. It is usually an OBIS code of the format A-B:C.D.E*F or on older EDIS code of the format C.D.E.that specifies exactly what the value of this data set represents.

8.6. DLMS/COSEM

DLMS/COSEM is a international standardized protocol used mostly to communicate with smart meter devices. The DLMS/COSEM driver uses the client library developed by the jDLMS project. Currently, the DLMS/COSEM driver supports communication via HDLC and TCP/IP using Logical Name Referencing to retrieve values from the device.

Dependencies: rxtxcomm_api-2.1.7.jar (optional)

ID	dlms
Device Address	hdlc:<serial-port>[:<server-physical-port>]:<server-logical>:<client-logical> tcp:<server-ip>[:<server-port>]:<server-logical>:<client-logical>
Settings	[SendDisconnect=<disconnect>];[UseHandshake=<handshake>];[..]
Channel Address	<class-id>/<reference-id>/<attribute-id>

Interface Address

The interface address consists of all elements the driver needs to uniquely identify and address a physical smart meter and format depends on the used protocol. Refer to the following table for the format of the interface address.

Protocol	Physical-Connection	Example
hdlc	<serial-port>[:<physical-device-id>]	hdlc:ttyUSB0 or hdlc:ttyUSB0:16
tcp	<server-ip>[:<server-port>]	tcp:16:192.168.200.25 or tcp:16:192.168.200.25:4059

Settings

Settings are separated by a semi-colon. The available settings are determined by the used protocol, defined as first parameter of the device address. All possible settings with a short description and default values are listed in the following table.

Options	Protocol	Values	Default	Description
PW	all	string		Authorization password to access the smart meter device
SendDisconnect	all	*true/false*	true	Send a disconnect message at DLMS layer on disconnecting from device. Set this flag to false if the remote device is expecting the disconnect message at a lower layer (like HDLC)
UseHandshake	HDLC	*true/false*	true	Use initial handshake to negotiate baud rate
Baudrate	HDLC	*integer*		Maximum supported baud rate (0 = no maximum). If UseHandshake = false, this value will be used to communicate with the device and *has* to be set
ForceSingle	all	*true/false*	false	Forces every attribute to be requested individually. This option has to be enabled to support Kamstrup 382 smart meter devices

8.7. KNX

KNX is a standardised protocol for intelligent buildings. The KNX driver uses KNXnet/IP to connect to the wired KNX BUS. The driver supports group read and writes and is also able to listen to the BUS. The driver uses the calimero library.

Table 10. Configuration Synopsis

ID	knx
Device Address	knxip://<host_ip>[:<port>] knxip://<device_ip>[:<port>]
Settings	[Address=<Individual KNX address (e. g. 2.6.52)>];[SerialNumber=<Serial number>]
Channel Address	<Group Adress>:<DPT_ID>

Device Address

The device address consists of the host IP and the IP of the KNX tunnel or router.

Channel Address

The channel address consist of the group address you want to monitor and the corresponding data point ID. A data point consists of a main number and a subtype. For example a boolean would be represented by the main number 1 and a switch by the subtype 001, the DPT_ID of a switch is 1.001.

8.8. eHZ

OpenMUC driver for SML and IEC 62056-21

Dependencies: rxtxcomm_api-2.1.7.jar

Table 11. Configuration Synopsis

ID	ehz
Device Address	sml://<serialPort> or iec://<serialPort> e.g. sml:///dev/ttyUSB0
Settings	
Channel Address	<OBIScode> e.g. 10181ff (not 1-0:1.8.1*255)

scanForDevices() and scanForChannels will return the specific configuration.

8.9. SNMP

Simple Network Management Protocol (SNMP) is an Internet-standard protocol for monitoring and management of devices on IP networks.

Dependencies: snmp4j-2.2.5.jar

Table 12. Configuration Synopsis

ID	snmp
Device Address	IP/snmpPort
Settings	settings string
Channel Address	SNMP OID address

Device Address

IP address and available SNMP port of the target device should be provided as Device Address.

Example for Device Address:

```
192.168.1.1/161
```

Settings

All settings are stored in "SnmpDriverSettingVariableNames" enum.

Table 13. Setting Parameters

SNMPVersion	"SNMPVersion" enum contains all available values
USERNAME	string
SECURITYNAME	string
AUTHENTICATIONPASSPHRASE	is the same COMMUNITY word in SNMP V2c
PRIVACYPASSPHRASE	string

SNMPVersion

SNMPVersion is an enum variable containing valid SNMP versions. (V1, V2c, V3)

Example for valid settings string:

```
SNMPVersion=V2c:USERNAME=public:SECURITYNAME=public:AUTHENTICATIONPASSPHRASE=pas
sword
```

In order to read specific channel, corresponding SNMP OID shall be passed.

Example for SNMP OID:

```
1.3.6.1.2.1.1.1.0
```

For scanning SNMP enabled devices in the network, range of IP addresses shall be provided. This functionality is implemented only for SNMP V2c.

8.10. Aggregator

The Aggregator which performs aggregation of logged values from a channel. It uses the DriverService and the DataAccessService. It is therefore a kind of OpenMUC driver/application mix. The aggregator is fully configurable through the channels.xml config file.

Table 14. Configuration Synopsis

ID	aggregator
Device Address	virtual device e.g "aggregatordevice"
Settings	
Channel Address	<sourceChannelId>:<aggregationType>[:<quality>]

Channel Address

<sourceChannelId> - id of channel to be aggregated

<aggregationType> -

- AVG: calculates the average of all values of interval (e.g. for average power)
- LAST: takes the last value of interval (e.g. for energy)

- DIFF: calculates difference of first and last value of interval
- PULS_ENERGY,<pulses per Wh>,<max counter>: calculates energy from pulses of interval (e.g. for pulse counter/meter). Example: PULSE_ENERGY,10,65535

<quality> - Range 0.0 - 1.0. Percentage of the expected valid/available logged records for aggregation. Default value is 1.0. Example: Aggregation of 5s values to 15min. The 15min interval consists of 180 5s values. If quality is 0.9 then at least 162 of 180 values must be valid/available for aggregation. NOTE: The missing/invalid values could appear as block at the beginning or end of the interval, which might be problematic for some aggregation types

Example:

Channel A (channelA) is sampled and logged every 10 seconds.

```
<channelid="channelA">
  <samplingInterval>10s</samplingInterval>
  <loggingInterval>10s</loggingInterval>
</channel>
```

Now you want a channel B (channelB) which contains the same values as channel A but in a 1 minute resolution by using the *average* as aggregation type. You can achieve this by simply adding the aggregator driver to your channel config file and define a the channel B as follows:

```
<driver id="aggregator">
  <device id="aggregatordevice">
    <channelid="channelB">
      <channelAddress>channelA:avg</channelAddress>
      <samplingInterval>60s</samplingInterval>
      <loggingInterval>60s</loggingInterval>
    </channel>
  </device>
</driver>
```

The new (aggregated) channel has the id channelB. The channel address consists of the channel id of the original channel and the aggregation type which is channelA:avg in this example. OpenMUC calls the read method of the aggregator every minute. The aggregator then gets all logged records from channelA of the last minute, calculates the average and sets this value for the record of channelB. NOTE: It’s recommended to specify the samplingTimeOffset for channelB. It should be between samplingIntervalB - samplingIntervalA and samplingIntervalB. In this example: 50 < offset < 60. This constraint ensures that values are AGGREGATED CORRECTLY. At hh:mm:55 the aggregator gets the logged values of channelA and at hh:mm:60 respectively hh:mm:00 the aggregated value is logged.

```
<driver id="aggregator">
  <device id="aggregatordevice">
    <channelid="channelB">
      <channelAddress>channelA:avg</channelAddress>
      <samplingInterval>60s</samplingInterval>
      <samplingTimeOffset>55s</samplingTimeOffset>
      <loggingInterval>60s</loggingInterval>
    </channel>
  </device>
</driver>
```

9. The WebUI

9.1. Plugins

Plotter

  Plugin which provides plotter for visualisation of current and historical data

Channel Access Tool

  Plugin to show current values of selected channels. Provides possibility to set values.

Channel Configurator

  Plugin for channel configuration e.g. channel name, sampling interval, logging interval

Media Viewer

  Plugin which allows to embed media files into OpenMUC’s WebUI

User Configurator

  Plugin for user configuration

9.2. HTTPS

You can access the WebUI over https as well: https://localhost:8889. To make the framework more secure you could disable http by setting org.apache.felix.http.enable in the conf/system.properties file to false.

10. REST Server

The openmuc-server-restws bundle manages a RESTful web service to access all registered channels of the framework. The RESTful web service is accessed by the same port as the web interface mentioned in Chapter 2.

The address to access the web service using the provided demo/framework folder is *http://localhost:8888/rest/*

10.1. Requirements

In order to start the RESTful web service, the following bundle must be provided:

- Bundle that provides an org.osgi.service.http.HttpService service. In the demo framework, that service is provided by the org.apache.felix.http.jetty bundle.

This bundles is already provided by the demo framework. The RESTful web service will start automatically with the framework without additional settings.

10.2. Accessing channels

The latest record of a single channel can be accessed, by sending a GET request at the address: *http://server-address/rest/channels/{id}* where {id} is replaced with the actual channel ID. The result will be latest record object of the channel encoded in JSON with the following structure:

Listing 9. Record JSON

```
{
  "timestamp" : time_val, /*milliseconds since Unix epoch*/
  "flag" : flag_val,      /*status flag of the record as string*/
  "value" : value_val     /*actual value. Omitted if "flag" != "valid"*/
}
```

You can access logged values of a channel by adding */history?from=fromTimestamp&until=untilTimestamp* to the channel address, fromTimestamp and untilTimestamp are both milliseconds since Unix epoch (1970-01-01 00:00:00). The result is a collection of records encoded as JSON.

Additionally, the records off all available channels can be read in one go, by omitting the ID from the address. The result is a collection of channel objects encoded in JSON using this structure:

Listing 10. ChannelCollection JSON

```
[
    {
        "id" : channel1_id,  /*ID of the channel as string*/
        "record" : channel1_record /*current record. see Record JSON*/
    },
    {
        "id" : channel2_id,
        "record" : channel2_record
    }
    ...
]
```

New records can be written to channels by sending a PUT request at the address that represents a channel. The data in the put request is a record encoded as specified in Record JSON above.

If HTTPS is used to access the REST server then HTTP basic authentication is required. The login credentials are the same as the one used to log into the web interface of the OpenMUC Framework.

11. Authors

Developers:

- Stefan Feuerhahn
- Marco Mittelsdorf
- Dirk Zimmermann
- Albrecht Schall

Former developers:

- Michael Zillgith
- Karsten Müller-Bier
- Simon Fey
- Frederic Robra
- Philipp Fels