# Introduction

The project OpenHomeMatic implements the homematic driver for OpenMUC. It is based on the OGEMA project. For OGEMA we refer to the page <http://www.ogema.org/> and for OpenMUC to the page <https://www.openmuc.org/>. The most benefit we get from the OMEGA project, is that OGEMA contains a configuration and the devices for the messages, which are exchanged between the devices and the driver. Nevertheless we decide to refactor the OGEMA sources for our purpose, because we use the OpenMUC framework with the openmuc-core-api.jar and the openmuc-core-spi.jar, which are quite different to the OGEMA api part.

In case we use refactored OGEMA sources you will find them in the same packages as in the OGEMA project. Principly structure is the same.

The picture below shows the structure of the project. The blue filled rectangles belong to the homematic driver.

HomeMatic::

Ogema:

OpenMUC:

data:

O

settings

data

config

driver.spi

homematic

tools and config

manager

messages

devices

jrxtx

connection

# Package homematic

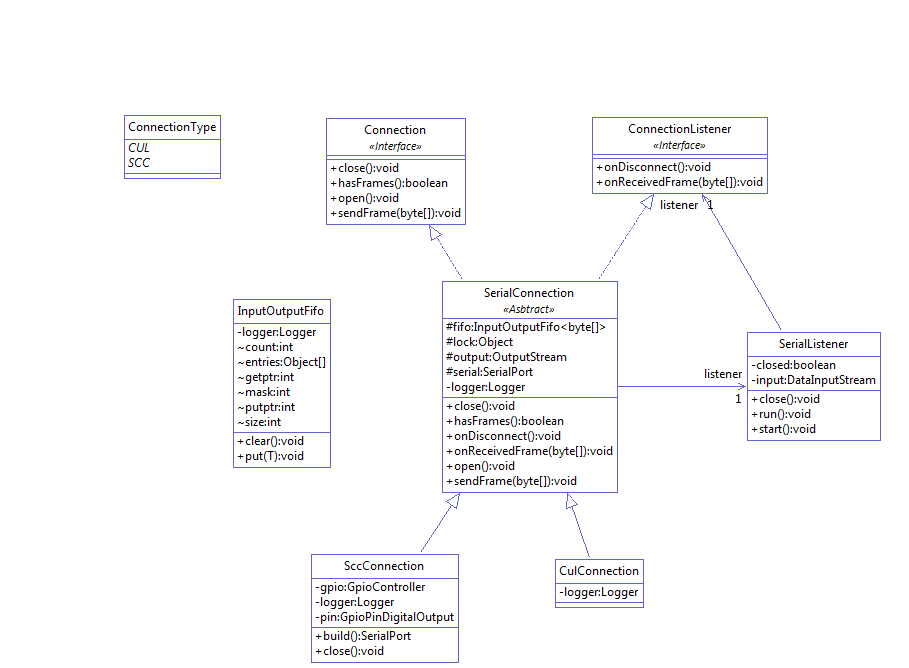
The package homematic is base of the driver. It instantiate the homematic manger to start up the jrxtx connection, the input handler and the output handler. It connects the driver with the OpenMUC framework. The OpenMUC framework calls the connect method of the homematic driver class to connect devices by its configuration. Therefore the connect method will get or create the device from the address and create a homematic connection. This connection is a connection to the OpenMUC framework and has nothing to do with jrxtx. The OpenMUC framework can call the method scanForDevices of the homeatic driver class to send a pairing request to the devices. In this way it is possible to pair devices with the OpenMUC framework. Only for paired devices one can get values from the channels of the device.

Dass die Klasse HomeMaticConnection eine Extension von HomeMaticDevice ist, ist wenig einleuchtend. Ich glaube es wäre besser, wenn die Klasse HomeMaticConnection eine Instanz von HomeMaticDevice halten würde! Warum braucht man überhaupt 2 unterschiedliche Klassen für Devices, nämlich Device und HomeMaticDevice? Am besten alles in die Device Klasse umziehen. In der HomeMaticConnection Klasse kommen die Devices PowerMeter und SwitchPlug vor. Das halte ich für unangemessen. Ich würde hier besser ein Interface von PowerMeter und SwitchPlug verwenden. Dann könnten andere Devices auch default states verwalten.

Each OpenMUC driver device can have one or more channels. The OpenMUC framework can call the scanForChannels method of the homematic connection. There the ChannelScanInfo of the OpenMUC framework is filled to allow the user to select the channel for which he wants to get values, write values or listen for values. The OpenMUC framework can call the read method of the homematic connection to set the value to the container. With the write method of the homematic connection the framework of OpenMUC has the possibility to write values to the channel, which are forwarded to the real hardware device. It is also possible for the OpenMUC framework to listen to channels. Therefore the OpenMUC has to register itself with a listener by the channel calling the startListening method of homematic connection.

# Package config

# Package connection



Class diagram of package connection

# Package data

Because the OpenMUC framework don’t support an object value class to hold an arbitrary object, therefore the package contains the ObjectValue class. This class should be moved to data package of the OpenMUC framework.

# Package manager

# C:\Users\gb\Workspace\OpenHomeMatic\projects\ogema-driver\manager.png

Class diagram package manager and sub package devices

The base class in this package is the HomeMaticManager. When the HomeMaticManager is created, it creates the MessageHandler, a map of devices and the DeviceDesriptor, which is part of the package tools. The MessageHandler contains the InputTread and OutputThreads in a map. When the messageHandler is created, it creates the connection object of the configured connection type and opens the connection.

The InputHandler is a thread and is started after creation. Then the Connection is opened. The InputHandler asks the manager for incoming messages and handle this messages. Only incoming messages of kind (first Byte of the message) “V” for Version message and “A” or “a” for status messages are handled. If we receive a pairing message, a new device is created and added to the map of devices in the HomeMaticManager. Then the device send the pairing messages over the SerialConnection. Otherwise it is checked if the device is contained in the map of devices of the HomeMaticManager and if so the HomeMaticManager is called with the receiving message. The HomeMaticManager calls the OutputHandler that a message is received.

If the OutputHandler is called to send a message it creates for each device a thread and add the message to send to this thread. (Achtung diese Threads werden nicht beendet, wenn die Verbindung geschlossen wird. Vielleicht sollte der OutputHandler auch den InputHandler (Thread erzeugen) und beim Stop seine Threads beenden. Asserdem dann den OutputHandler in MessageHandler umbenennen.)

# Package devices

The device (type) classes in this package are all derived from the abstract Device(Type)Handler class of the package manager. For each device type class the available device attributes und device commands are defined by the method configure channels. It parses the messages. There are two types of messages one containing the configuration of the device and one containing values of the device. The message containing the configuration is parsed in the method parseConfig of the Device(Type)Handler class. The messages containing the values of the devices is parsed in the method parseValue of each device (type) class. The method channelChanged of the device (type) classes sets the parameters for sending the device command and trigger the sending.

# Package messages

The class StatusMessage parse the header of the incoming message and hold the content of the message as byte array. The CommandMessage class builds the command to send to the device. The CommandMessage class is the only class which extends the abstract Message class.

Da CommandMessage die einzige Klasse ist die Message erweitert, könnte man Message Klasse in der CommandMessage Klasse implementieren und entfernen.

The Message class holds the device to get the destination where to send the message and to get the message number or to increment the message number.

# Package settings

In the settings package are classes, which are used for reading the configuration for OpenMUC (see below Device Configuration). The configuration concerns the devices (DeviceSettings). Each device can have one or more channels, which will be configured with attributes of ChannelSettings. To control the device scan the attribute ignoreExisting is used (DeviceScanSettings).

# Package tools

The package contains two classes which are the same as in OGEMA. Only the instantiation of the logger is adapted. The class DeviceDescriptor is used to load the deviceTypes configuration json file (see below Device Types Configuration). From the Class DeviceDescriptor one can get the name, the type and the channels of a device, if one knows the key of the device. In the class Converter we found static methods for converting data types as byte array, hex string, char, byte, int, long to byte array, hex string, int, long.

# Configuration

# System Property Configuration

The homematic driver uses following system properties as configuration.

You have to decide which connection you want to use for this driver. The possible connections are the CulConnection used for connection with a CUL Stick and SccConnection with a SCC board. Both are serial connections. With the property “org.openmuc.framework.driver.homematic.interface” you can decide which connection will be established. The default value is “SCC”. If you want to establish a connection with CulConnection you have to set the property to “CUL”.

The property “org.openmuc.framework.driver.homematic.connection.port” for the CulConnection is used to define the port to be used. The default value of the port is “/dev/ttyUSB0”. The property “org.openmuc.framework.driver.homematic.connection.port” for the SccConnection is used to define the port to be used. The default value of the port is “/dev/ttyAMA0”.

The AskSin protocol contains a destination address of 6 character length to identify the driver. With the property “org.openmuc.framework.driver.homematic.id” you can set this address. The address can be an arbitrary string of 6 characters. The default value is “F11034”.

# Device Configuration

The options for the device configuration for OpenMUC are defined in the options.xml. You have to identify each device by a unique address. The “Identifier” is of type String, can have an arbitrary length and has no default value. This parameter is mandatory. For each device you also have to set the type of the device. You can select the “Type” from a selection box containing all types of devices which are supported by this driver. The device type is normally somewhere printed on the hardware device. This parameter is mandatory and of type String. The String decodes the device type in hex code. Therefore it is not recommended to edit directly. Some device types has the possibility to have default values so it can be switched to the same state as they are before for example a reboot. The parameter “Default State” is of type boolean, is not mandatory and has no default value.

Each device can have one or more channels. For each channel it is mandatory to set an “Identifier”. The “Identifier” is of type String, can have an arbitrary length and has no default value. For each channel you have to define the type of the channel. The channel can be of type “ATTRIBUTE” or of type “COMMAND”. From attribute channels values can be read. To command channels values can be written. You can select it from a selection box. The parameter is of type String, is mandatory and the default value is “ATTRIBUTE”

In the scanForDevices method of the homematic driver it is possible to decide whether already configured and paired devices should be ignored. If disabled, all already paired and registered devices will be returned as well. The name of the parameter is “Ignore Existing”. The parameter is of type Boolean, is not mandatory and has the default value “true”.

# Device Types Configuration

This configuration a json file and interpreted by the tools class DeviceDescriptor. The first part of the json file is an array of keys, described by a hex code. For each key of a device the descriptor allows to fetch the properties name, type and channels. We build a key value pair containing the name for each key in the device configuration (see above). This key value pairs are members of the selection box. We use the device key only internally and the name of the device in the user interface.

The property type of the device type we use to create the device (type) handler class of package devices.

The property channels are only used in the remote (type) handler class of package devices. It adds the two attributes short pressed button and long pressed button, if in the channels “Sw” or “Btn” is configured.

All other properties as “cyc”, “lst” or “rxt” are not used in this driver.

combines the two different software releases OGEMA and OpenMUC. For OGEMA we refer to the page <http://www.ogema.org/> and for OpenMUC to the page <https://www.openmuc.org/>. We have used the OGEMA package “homematic-driver” as base for our development. But we refactored this package in our package “ogema-homematic-driver”. This package uses the OGEMA package “api” in the version 2.1.2 especially the sub-packages of the “core.channelmanager”. Because the OGEMA package “api” and the pendant package “openmuc-core-api” uses different data classes, we introduced the package “ogema” containing all port converters between the two worlds. The package “openmuc-driver-homematic-cc1101” binds the packages “ogema” and “ogema-driver-homematic” to store data in the OpenMUC world and to excute commands of OpenMUC. The diagram below shows the dataflow.

openmuc-driver-homematic-cc1101

ogema-driver-homematic

ogema

OpenMUC

OGEMA (api: package org.ogema.core.channelmanager)

Devices (Smartplugs, Switches, Thermostats)

jrxt

framework

As interface to the devices as smartplugs, switches or thermostats we use the serial protocol jrxt of OpenMUC. The data structure which are exchanged is based on the RF protocol ASKSIN (aka BidCos(R)) of “culfw” from FHEM (s. page <http://culfw.de/commandref.html#X>).