OpenEMS

Open Energy Management System

Stefan Feilmeier (c) 2018 FENECON GmbH

Version 2018.8.0-SNAPSHOT

Table of Contents

1. Introduction

OpenEMS is a modular platform for energy management applications. It was developed around the requirements of controlling, monitoring and integrating energy storage systems together with renewable energy sources and complementary devices and services.

1.1. OpenEMS IoT stack

The OpenEMS 'Internet of Things' stack contains three main components:

- **OpenEMS Edge** runs on-site and actually controls the devices
- **OpenEMS UI** is the generic user interface
- **OpenEMS Backend** runs on a (cloud) server, connects the decentralized Edge systems and provides aggregation, monitoring and control via internet

1.2. Features

The OpenEMS software architecture was designed to leverage some features that are required by a modern and flexible Energy Management System:

- Fast, PLC-like control of battery inverters and other devices
- Easily extendable due to the use of modern programming languages and modular architecture
- Wide range of supported devices - (battery) inverters, meters, etc. - and protocols
- Modern web-based real-time user interface

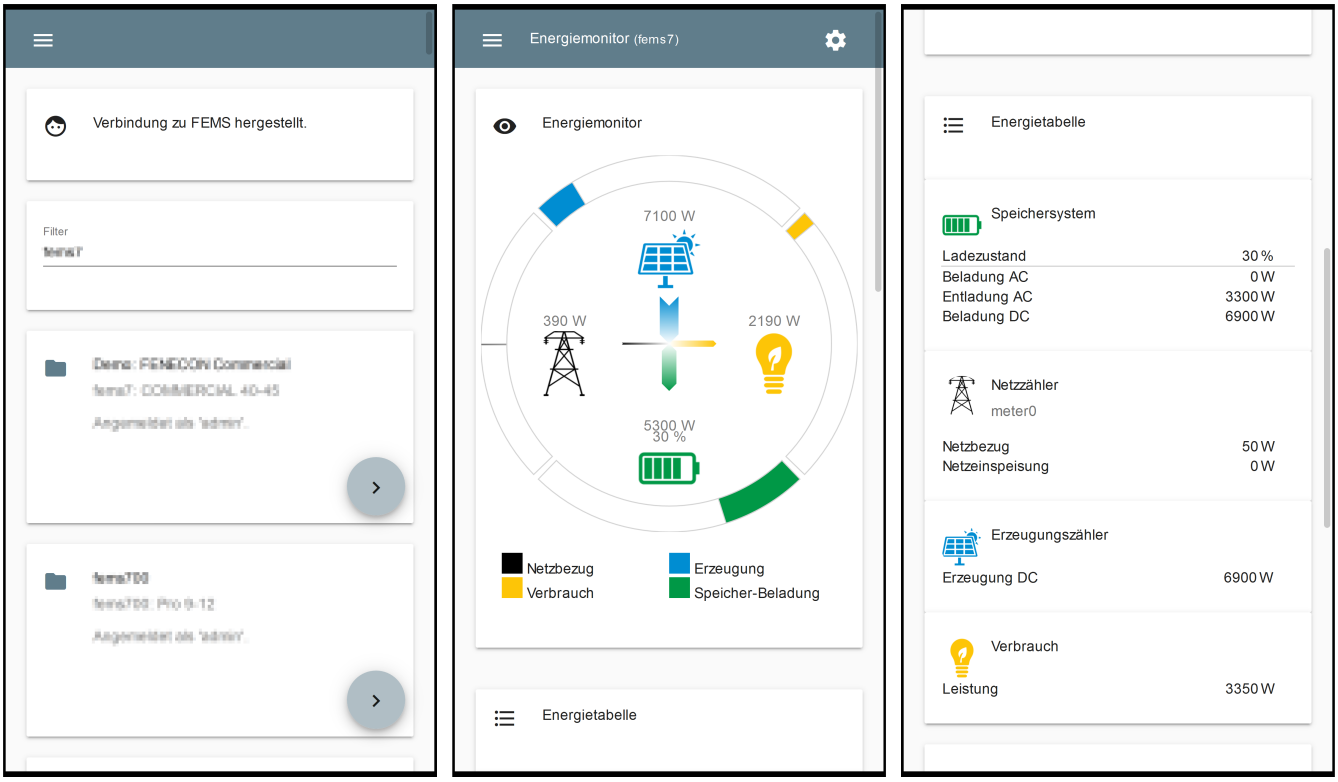

Figure 1. Screenshots of OpenEMS UI

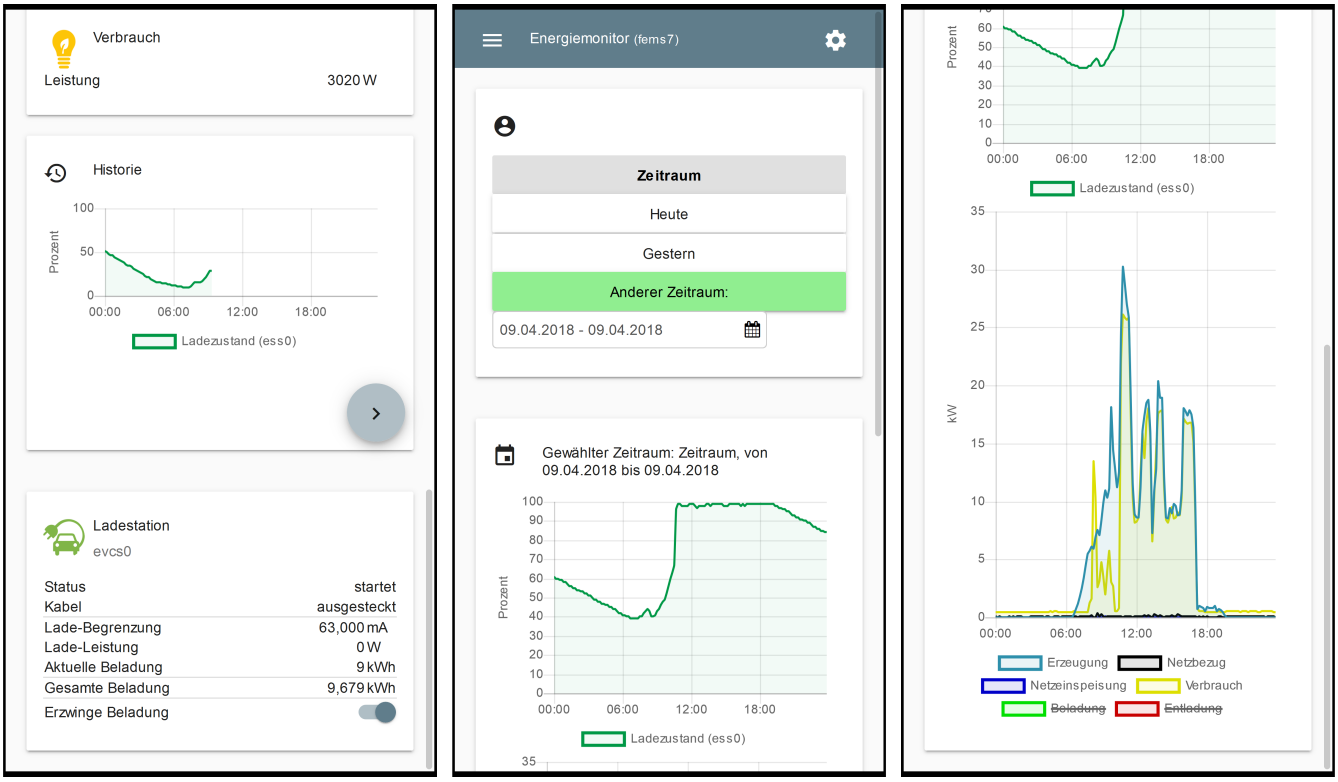

Figure 2. Screenshots of OpenEMS UI

1.3. Open Source philosophy

OpenEMS development was started by FENECON GmbH, a German company specialized in manufacturing and project development of energy storage systems. It is the software stack behind FEMS - FENECON Energy Management System and widely used in private, commercial and industrial applications.

We are inviting third parties - like universities, hardware manufacturers, software companies, commercial and private owners,... - to use OpenEMS for their own projects and are glad to support them with their first steps. In any case if you are interested in OpenEMS our development team would be glad to hear from you at fems@fenecon.de.

OpenEMS is funded by several federal and EU funding projects. If you are a developer and you would like to get hired by one of the partner companies or universities for working on OpenEMS, please send your motivation letter to fems@fenecon.de.

1.4. License

- OpenEMS Edge
- OpenEMS Backend

This product includes software developed at FENECON GmbH: you can redistribute it and/or modify it under the terms of the [Eclipse Public License version 2.0](LICENSE-EPL-2.0).

- OpenEMS UI

1.5. Development guidelines

Development follows the Agile Manifesto and is driven by the Scrum methodology. The source code is available online at http://openems.io and on GitHub. New versions are released after every Scrum Sprint and tagged accordingly. Version numbers are built using the pattern `year.number of sprint`, e.g. version `2018.4` is the result of the fourth sprint in 2018. Git development follows the Gitflow Workflow, so the master branch always holds the stable release, while active development is happening on the develop branch or in separate feature branches.

For Edge and Backend Java development we recommend the Eclipse IDE. For the UI (TypeScript + Angular.io) we recommend Visual Studio Code. The documentation is generated using AsciiDoc. For handling git we recommend Sourctree by Atlassian.

1.6. System architecture

OpenEMS is generally used in combination with external hardware and software components (the exception is a simulated development environment - see Getting Started) As a brief overview, this is how OpenEMS is used in production setups:

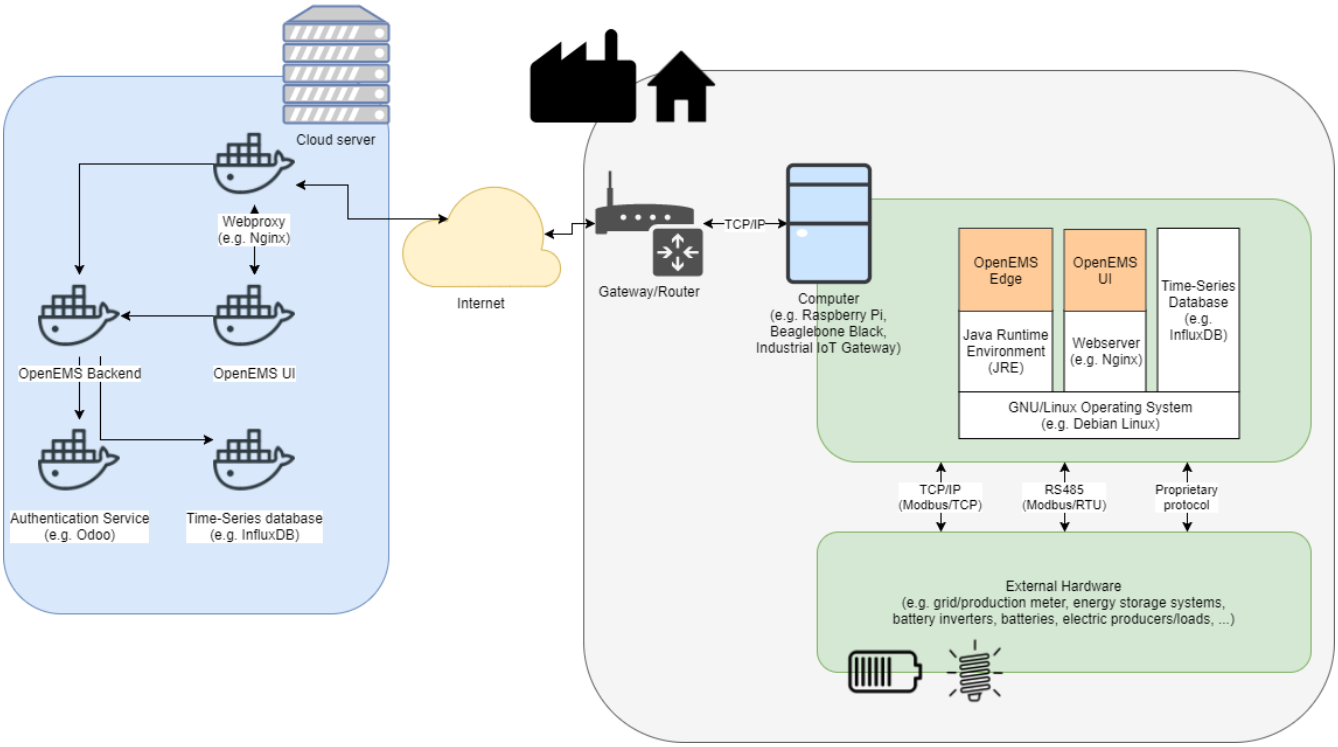

Figure 3. OpenEMS system architecture

2. Getting Started

This quick 'Getting Started' should help you setup up a complete development environment. On

finishing you will have a working instance of OpenEMS Edge, with simulated energy storage and photovoltaic system, as well as an OpenEMS UI for monitoring the simulator inside your web browser.

2.1. Download the source code

1. Download any git client and install it. Our recommendation is Sourctree by Atlassian
2. Clone the OpenEMS git repository
   a. In Sourcetree:
      i. press **[File]** → **[Clone]**
      ii. enter the git repository path `https://github.com/OpenEMS/openems.git`
      iii. select a target directory, for example `C:\Users\your.user\git\openems`
      iv. open **[Advanced Settings]**
      v. select the branch **[develop]**
      vi. and press **[Clone]**.

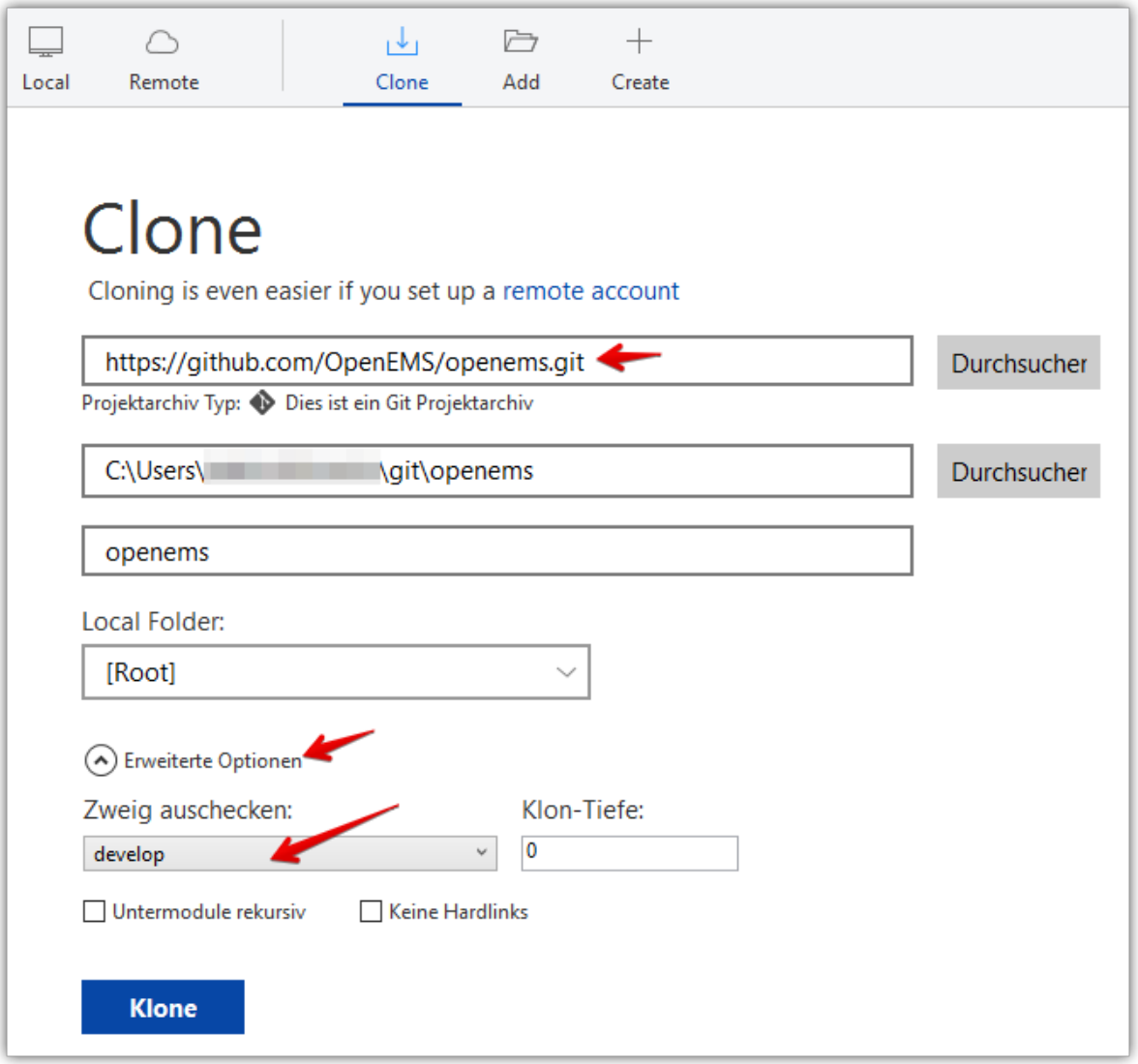

Figure 4. Cloning the git repository using Sourcetree

b. Alternatively: with the git command line utility

   i. open a console

   ii. change to the target directory

   iii. execute `git clone https://github.com/OpenEMS/openems.git --branch develop`

3. Git is downloading the complete source code for you.

2.2. Setup Eclipse IDE for OpenEMS Edge and Backend

1. Prepare Eclipse IDE

   a. Download Java SE Development Kit 8 and install it

   b. Download Eclipse for Java, install and start it

   c. On first start you will get asked to create a workspace. Select a directory - for example `C:\Users\your.user\git\openems-workspace` - and press **[Lauch]**. *The directory needs to be different from your source code directory selected above.*

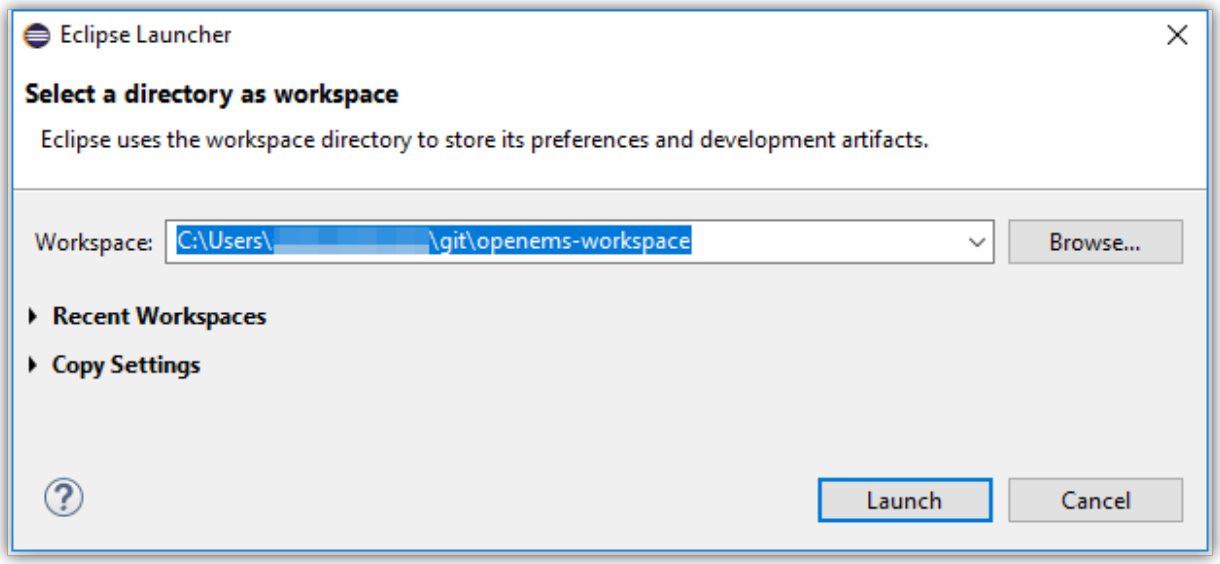

Figure 5. Creating a workspace in Eclipse IDE

   d. Install BndTools in Eclipse:

      Menu: **[Help]** → **[Eclipse Marketplace...]** → **[Find:]** → enter **[Bndtools]** → press **[Install]**

2. Import OpenEMS component projects (OSGi bundles):

   Menu: **[File]** → **[Import...]** → **[Bndtools]** → **[Existing Bnd Workspace]** → Root directory: **[Browse...]** → select the directory with the source code - for example `C:\Users\your.user\git\openems` → **[OK]** → **[Finish]** → "Switch to Bndtools perspective?" **[yes]**

3. Eclipse should have successfully built OpenEMS Edge and Backend, showing no entry in Problems.

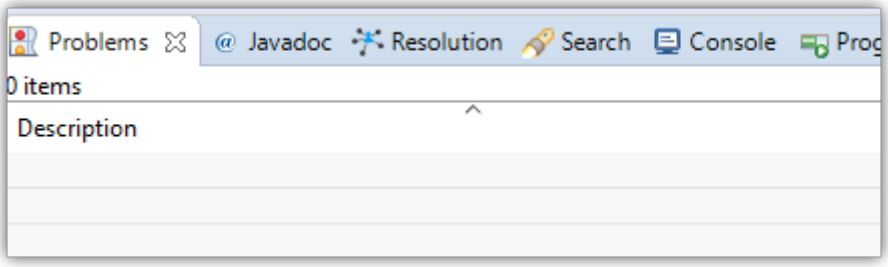

Figure 6. Eclipse IDE showing 'no problems'

2.3. Run OpenEMS Edge and start Simulator

1. Run OpenEMS Edge

   a. In Eclipse IDE open the project **[io.openems.edge.application]** and double click on **[EdgeApp.run]**.

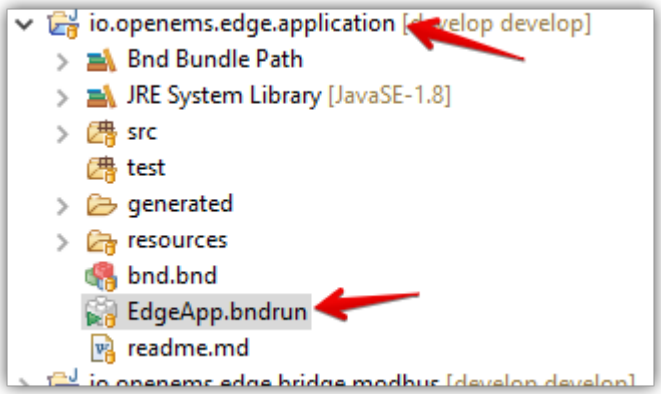

Figure 7. io.openems.edge.application project in Eclipse IDE

   b. Click on **[Resolve]** to resolve all dependencies and accept the 'Resolution Results' popup window with **[Finish]**.

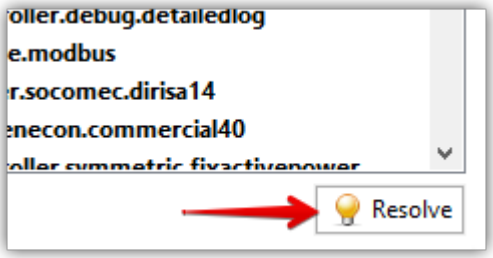

Figure 8. Resolve OSGi in Eclipse IDE

   c. Click on **[Run OSGi]** to run OpenEMS Edge. You should see log outputs on the console inside Eclipse IDE.

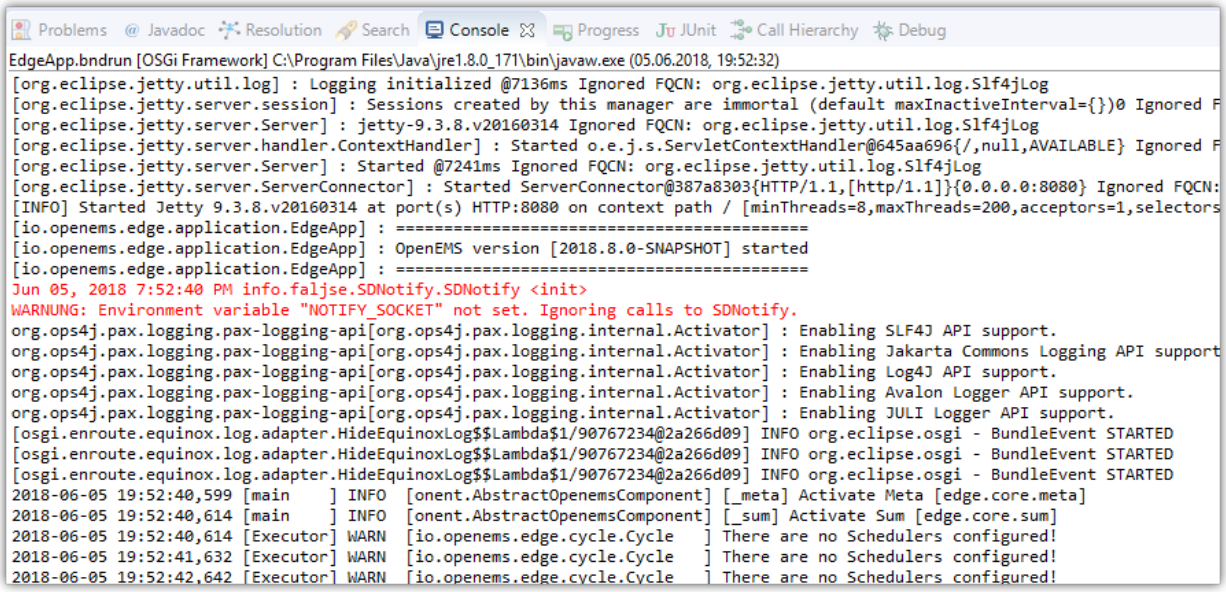

Figure 9. OpenEMS Edge initial log output

2. Configure and start the Simulator

   a. Open the Apache Felix Web Console Configuration.

   Login with username **admin** and password **admin**.

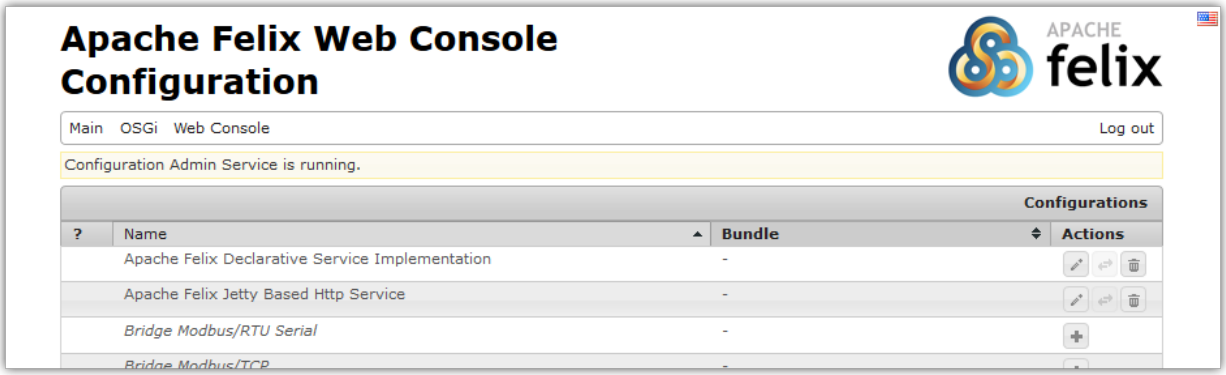

Figure 10. Apache Felix Web Console Configuration

   b. Configure a Scheduler

> The Scheduler is responsible for executing the control algorithms (Controllers) and defines the OpenEMS Edge application cycle

      i. Click on "Scheduler All Alphabetically"

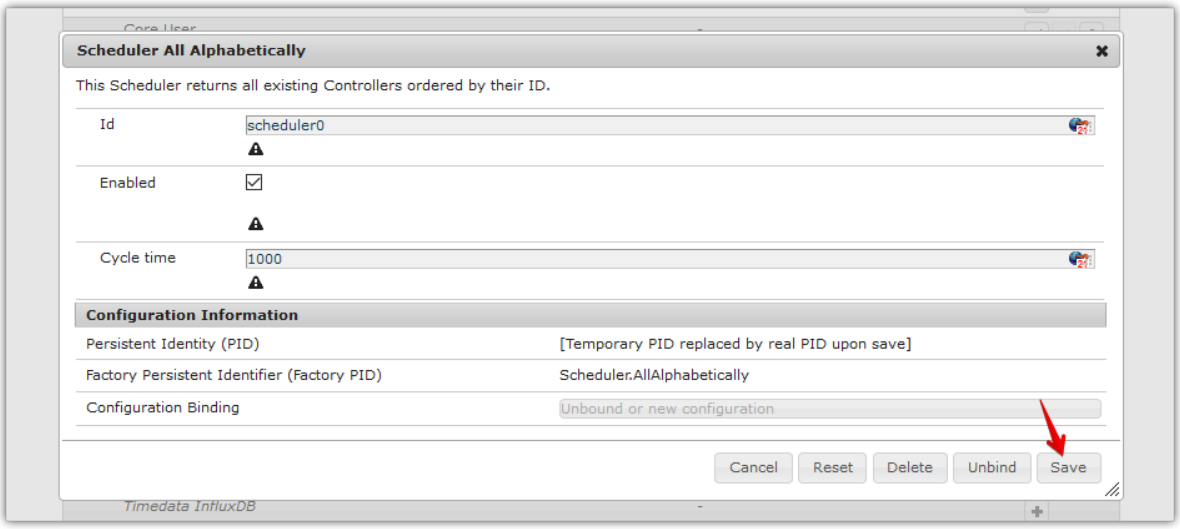

Figure 11. Configuration of All Alphabetically Scheduler

ii. Accept the default values and click **[Save]**

iii. You created your first instance of an OpenEMS Component with ID "scheduler0". The log shows:

```
INFO [onent.AbstractOpenemsComponent] [scheduler0] Activate AllAlphabetically [edge.scheduler.allalphabetically]
```

Add any other OpenEMS Components in the same way:

c. Configure debug outputs on the console: "Controller Debug Log". The default values can be accepted without changes.

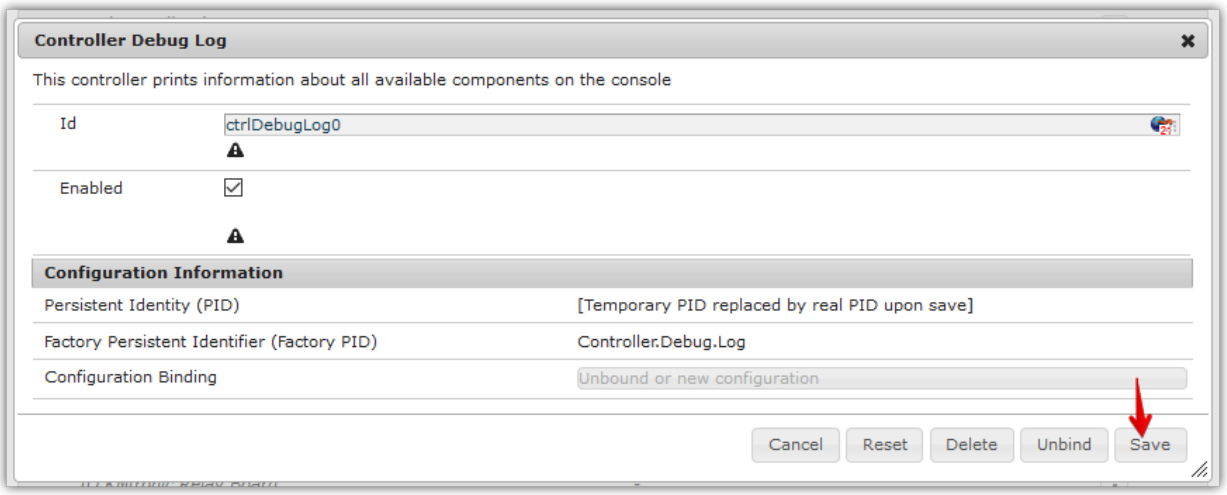

Figure 12. Configuration of Controller Debug Log

The log shows:

```
INFO [onent.AbstractOpenemsComponent] [ctrlDebugLog0] Activate DebugLog [edge.controller.debuglog],
```

followed once per second by

```
INFO [e.controller.debuglog.DebugLog] [ctrlDebugLog0] _sum[Ess SoC:0 %|L:0 W Grid L:0 W Production L:0 W Consumption L:0 W].
```

It is once per second because you accepted the default value of "1000 ms" for "Cycle time" in the Scheduler configuration.

d. Configure the standard-load-profile datasource: "Simulator DataSource: Standard Load Profile". The default values can be accepted without changes.

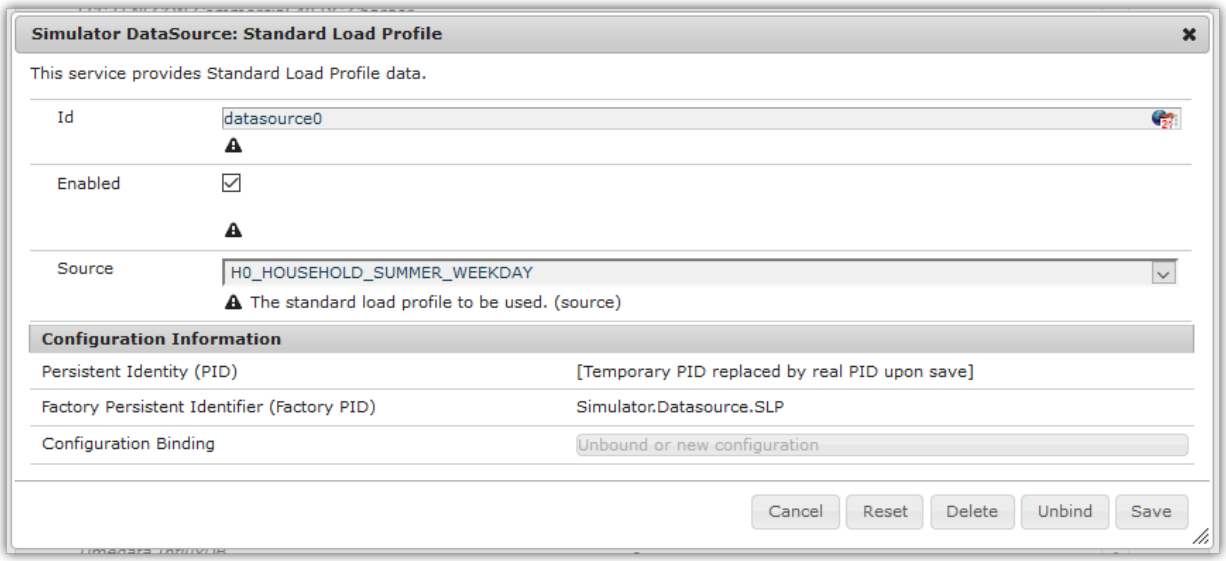

Figure 13. Configuration of Simulator DataSource: Standard Load Profile

The log shows:

```
INFO [onent.AbstractOpenemsComponent] [datasource0] Activate StandardLoadProfileDatasource [edge.simulator.datasource.standardloadprofile],
```

The data source was configured with the OpenEMS Component ID "datasource0" which will be used in the next step as reference.

e. Configure a simulated grid meter: "Simulator GridMeter Acting". Configure the Datasource-ID "datasource0" to refer to the data source configured above.

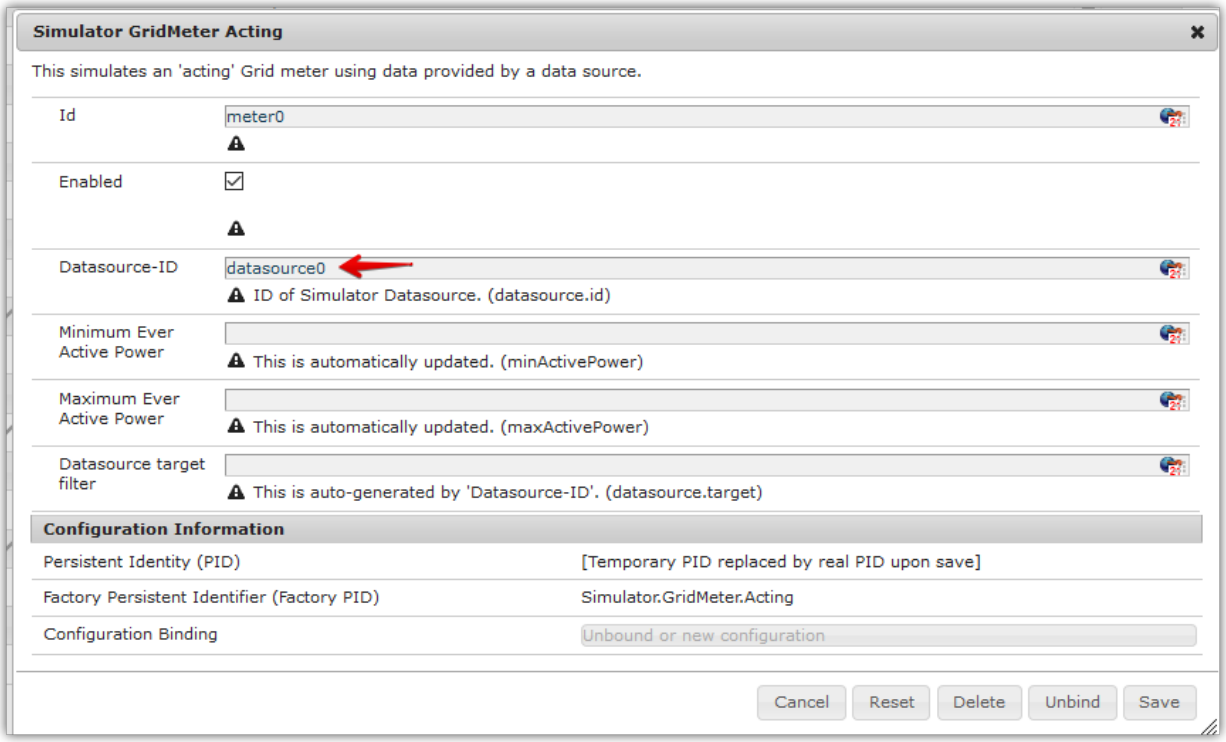

Figure 14. Configuration of Simulator GridMeter Acting

This time some more logs will show up. Most importantly they show, that the Grid meter now shows a power value.

```
INFO  [onent.AbstractOpenemsComponent] [meter0] Activate GridMeter
[edge.simulator.meter.grid.acting]
[onent.AbstractOpenemsComponent] [meter0] Deactivate GridMeter
[edge.simulator.meter.grid.acting]
[onent.AbstractOpenemsComponent] [meter0] Activate GridMeter
[edge.simulator.meter.grid.acting]
[e.controller.debuglog.DebugLog] [ctrlDebugLog0] _sum[Ess SoC:0 %|L:0 W Grid
L:1423 W Production L:0 W Consumption L:1423 W] meter0[1423 W]
```

This setup causes the simulated grid-meter to take the standardized load-profiles data as input parameter.

'Acting' referrs to the fact, that this meter actively provides data - in opposite to a 'Reacting' device that is reacting on other components: for example the 'Simulator.EssSymmetric.Reacting' configured below.

f. Configure a simulated reacting energy storage system: "Simulator EssSymmetric Reacting". The default values can be accepted without changes.

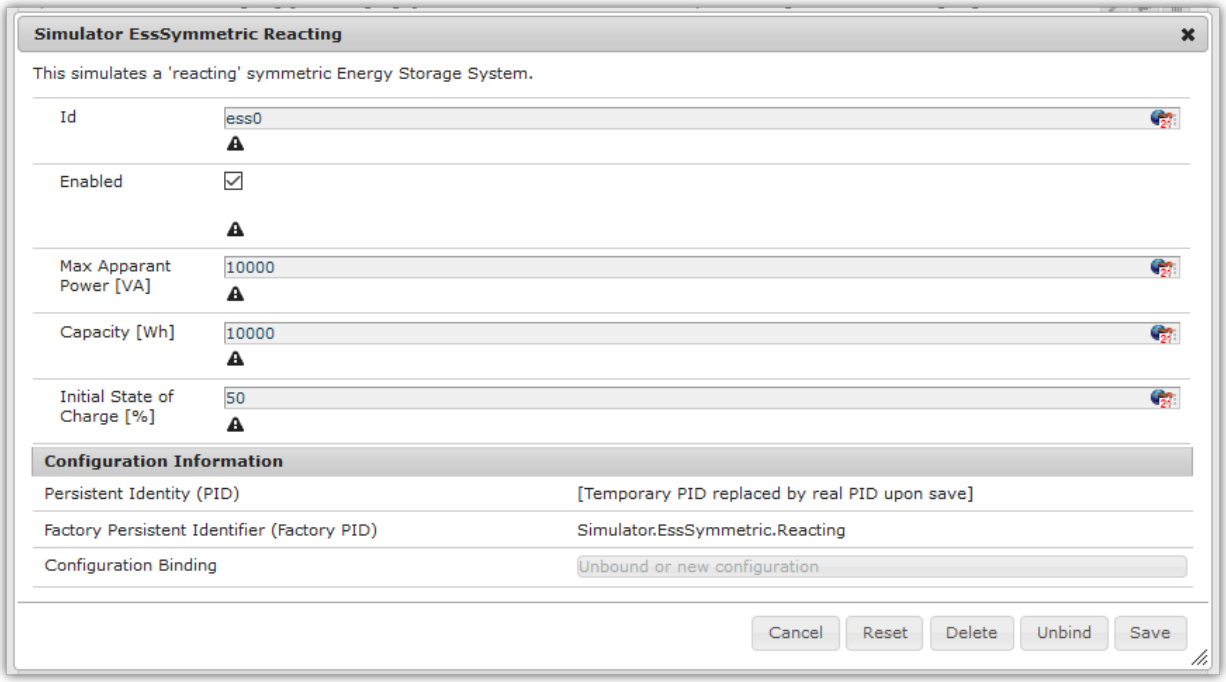

Figure 15. Configuration of Simulator EssSymmetric Reacting

The log shows:

```
INFO [e.controller.debuglog.DebugLog] [ctrlDebugLog0] _sum[Ess SoC:50 %|L:0 W Grid L:864 W Production L:0 W Consumption L:864 W] ess0[SoC:50 %|L:0 W|OnGrid] meter0[864 W]
```

Note, that the DebugLog now shows data for the battery, but the charge/discharge power stays at "0 W" and the state of charge stays at "50 %" as configured. Next step is to configure a control algorithm that tells the battery to charge or discharge.

g. Configure the self-consumption optimization algorithm: "Controller Balancing Symmetric". Configure the Ess-ID "ess0" and Grid-Meter-ID "meter0" to refer to the components configured above.

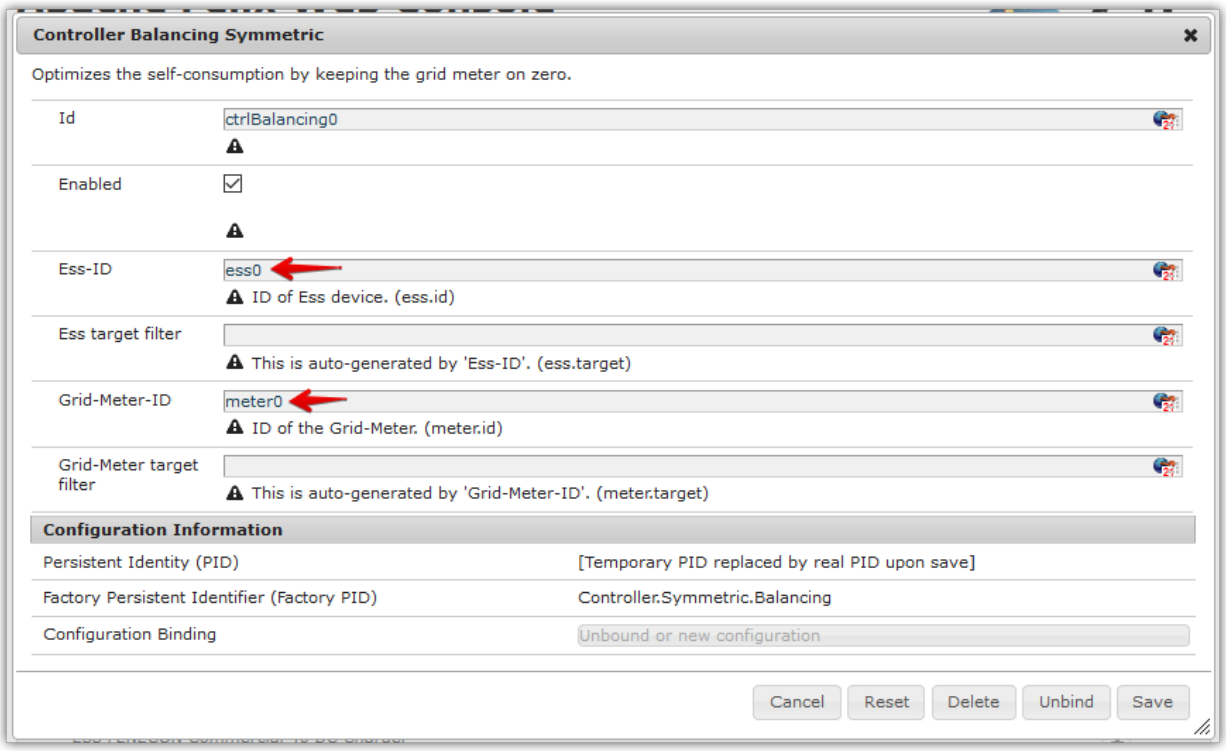

Figure 16. Configuration of Symmetric Balancing Controller

The log shows:

```
INFO [e.controller.debuglog.DebugLog] [ctrlDebugLog0] _sum[Ess SoC:49 %|L:1167 W Grid L:-39 W Production L:0 W Consumption L:1128 W] ess0[SoC:49 %|L:1167 W|OnGrid] meter0[-39 W]
```

Note, how the Controller now tells the battery to discharge (`Ess SoC:49 %|L:1167 W`), trying to balance the Grid power to "0 W" (`Grid L:-39 W`):

h. Configure the websocket Api Controller: "Controller Api Websocket". The default values can be accepted without changes.

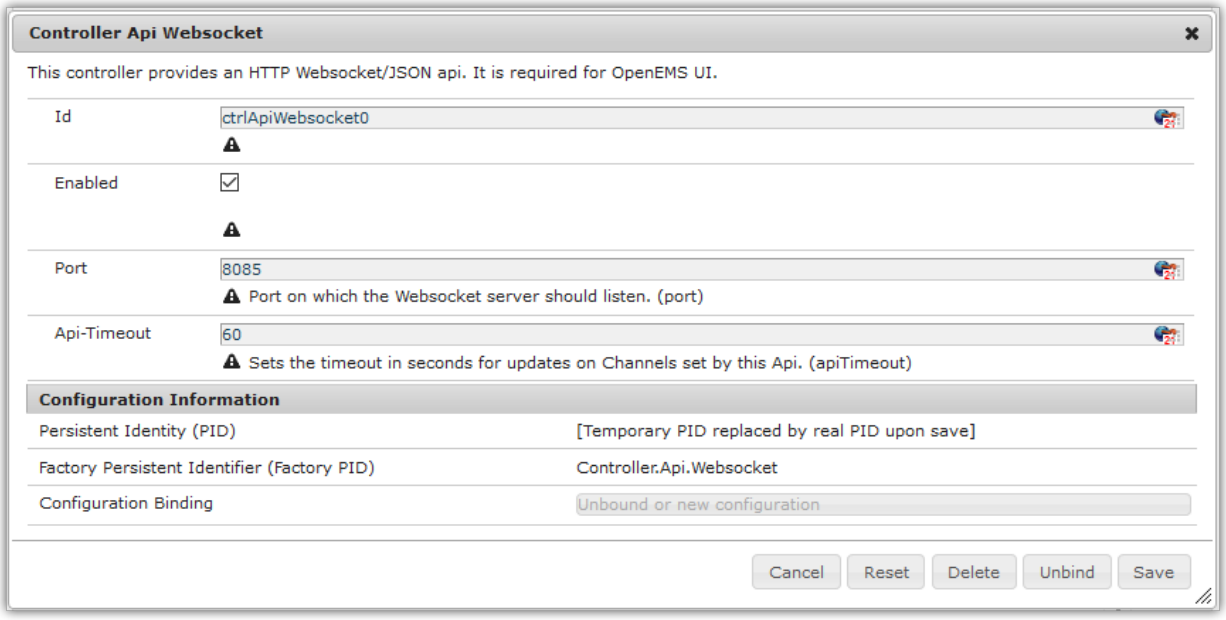

Figure 17. Configuration of Controller Api Websocket

The log shows:

```
INFO  [onent.AbstractOpenemsComponent] [ctrlApiWebsocket0] Activate WebsocketApi
[edge.controller.api.websocket]
INFO  [ler.api.websocket.WebsocketApi] [ctrlApiWebsocket0] Websocket-Api started
on port [8085].
```

The Controller Api Websocket is required to enable access to OpenEMS Edge by a local OpenEMS UI.

2.4. Setup Visual Studio Code for OpenEMS UI

1. Download node.js LTS and install it.
2. Download Visual Studio Code, install and start it.
3. Open OpenEMS UI source code in Visual Studio Code:

   Menu: **[File]** → **[Open directory...]** → Select the `ui` directory inside the downloaded source code (for example `C:\Users\your.user\git\openems\ui`) → **[Select directory]**

4. Open the integrated terminal:

   Menu: **[Show]** → **[Integrated terminal]**

5. Install Angular CLI:

   ```
   npm install -g @angular/cli
   ```

6. Resolve and download dependencies:

   ```
   npm install
   ```

2.5. Run OpenEMS UI

1. In Visual Studios integrated terminal type...

   ```
   ng serve
   ```

   The log shows:

   ```
   NG Live Development Server is listening on localhost:4200, open your browser on
   http://localhost:4200/
   ```

2. Open a browser at http://localhost:4200
3. You should see OpenEMS UI. Log in as user "guest" by clicking on the tick mark. Alternatively type "admin" in the password field to log in with extended permissions.

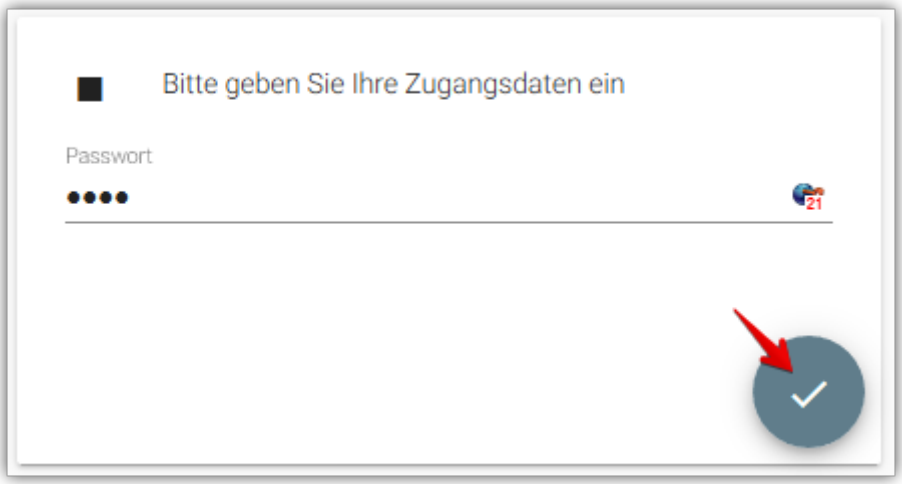

Figure 18. OpenEMS UI Login screen

4. Change to the Energymonitor by clicking on the arrow.

Figure 19. OpenEMS UI Overview screen

5. You should see the Energymonitor showing the same data as the DebugLog output on the console.

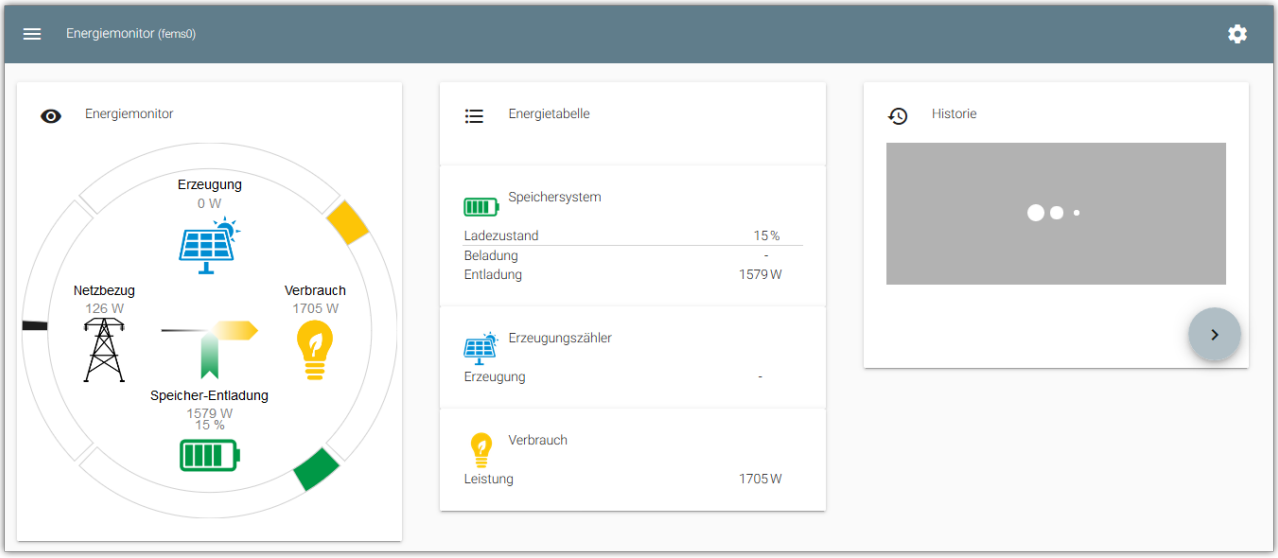

Figure 20. OpenEMS UI Energymonitor screen

OpenEMS UI will complain that "no timedata source is available". Because of this the historic chart is not yet functional.

3. Core concepts & terminology

This chapter describes some of the core concepts and commonly used terms in OpenEMS:

3.1. Bundle

OpenEMS Edge is using the OSGi platform to provide a completely modular and dynamic service oriented system.

Logical groups of source code are put into one OSGi Bundle. Every directory in the source code root directory starting with 'io.openems.*' is a bundle.

3.2. Component

OpenEMS Edge is built of Components, i.e. every main component implements the OpenemsComponent interface </>.

By definition each Component has a unique ID. Those **Component-IDs** are typically:

- `ess0` for the first storage system or battery inverter
- `ess1` for the second storage system or battery inverter
- ...
- `meter0` for the first meter in the system
- ...

If you receive your OpenEMS together with a FENECON energy storage system, you will find the following Component-IDs:

- FENECON Pro
  - `ess0`: FENECON Pro Ess
  - `meter0`: Socomec grid meter
  - `meter1`: FENECON Pro production meter
- FENECON Mini
  - `ess0`: FENECON Mini
  - `meter0`: FENECON Mini grid meter
  - `meter1`: FENECON Mini production meter

3.3. Channel

Each OpenemsComponent provides a number of Channels. Each represents a single piece of information. Each Channel implements the Channel interface </>. By definition each Channel has a unique ID within its parent Component.

3.4. Nature

Natures extend normal Java interfaces with 'Channels'. If a Component implements a Nature it also needs to provide the required Channels. For example the Energy Storage System (ESS) Simulator Simulator.EssSymmetric.Reacting </> implements the Ess interface </> and therefor needs to provide a `Soc` Channel that provides the current 'State of Charge' of the battery.

Controllers are written against Nature implementations. Example: A Controller can be used with any ESS, because it can be sure that it provides all the data the Controller requires for its algorithm.

3.5. Channel Address

By combining the unique **Component-ID** and **Channel-ID** each Channel in the system can be addressed by a distinct 'Channel Address' in the form `Component-ID/Channel-ID`.

Example: the state of charge ("Soc") of the first energy storage system ("ess0") has the channel address `ess0/Soc`.

3.6. Scheduler

_see Scheduler and Controller below

3.7. Controller

The actual business logic or algorithms are wrapped as 'Controllers'. i.e. they implement the Controller interface </>. Each Controller holds one specific, encapsulated task.

4. OpenEMS Edge

OpenEMS Edge is the core component of the energy management that runs on-site and is responsible for communicating with and controling of external hardware like battery systems, inverters, meters and so on.

4.1. Architecture

The OpenEMS Edge software architecture is carefully designed to abstract device communication and control algorithms in a way to provide maximum flexibility, predictability and stability, while simplifying the process of implementing new components.

4.1.1. Input-Process-Output

OpenEMS Edge is built around the well-known IPO (input-process-output) model which defines the internal execution cycle.

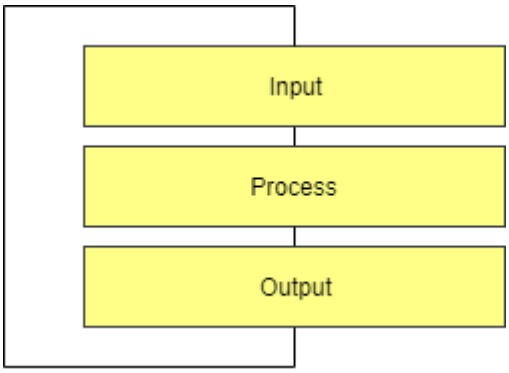

Figure 21. Input-Process-Output model

Input

During the input phase all relevant information - e.g. the current 'state of charge' of a battery - is collected and provided as a **process image**. This process image is guaranteed to never change during the cycle.

Process

The process phase runs algorithms and tasks based on the process image - e.g. an algorithm uses the 'state of charge' information to evaluate whether a digital output should be turned on.

Output

The output phase takes the results from the process phase and applies it - e.g. it turns the digital output on or off.

4.1.2. Scheduler and Controller

During the 'process' phase different algorithms (Controllers) might try to access the same resources - e.g. two Controllers try to switch the same digital output. It is therefor necessary to prioritize their execution and restrict access according to priority.

OpenEMS Edge uses Scheduler implementations to receive a sorted list of Controllers. The

Controllers are then executed in order. Later executed Controllers are not allowed to overwrite a previously written result.

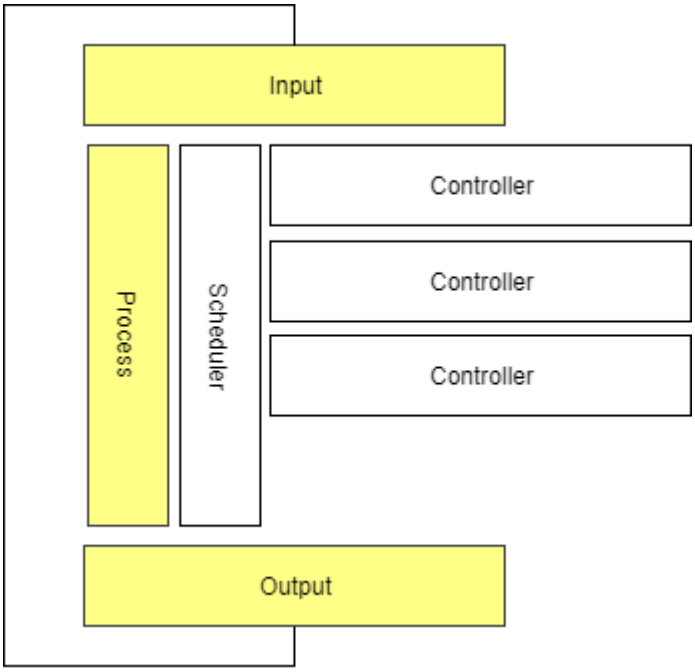

Figure 22. IPO model with Scheduler and Controllers

4.1.3. Cycle

The input-process-output model in OpenEMS Edge is executed in a Cycle - implemented by the Cycle component </>). It handles the setting of a process image in the input phase and executes the Controllers in the process phase. Furthermore it emits Cycle Events that can be used in other Components to synchronize with the Cycle.

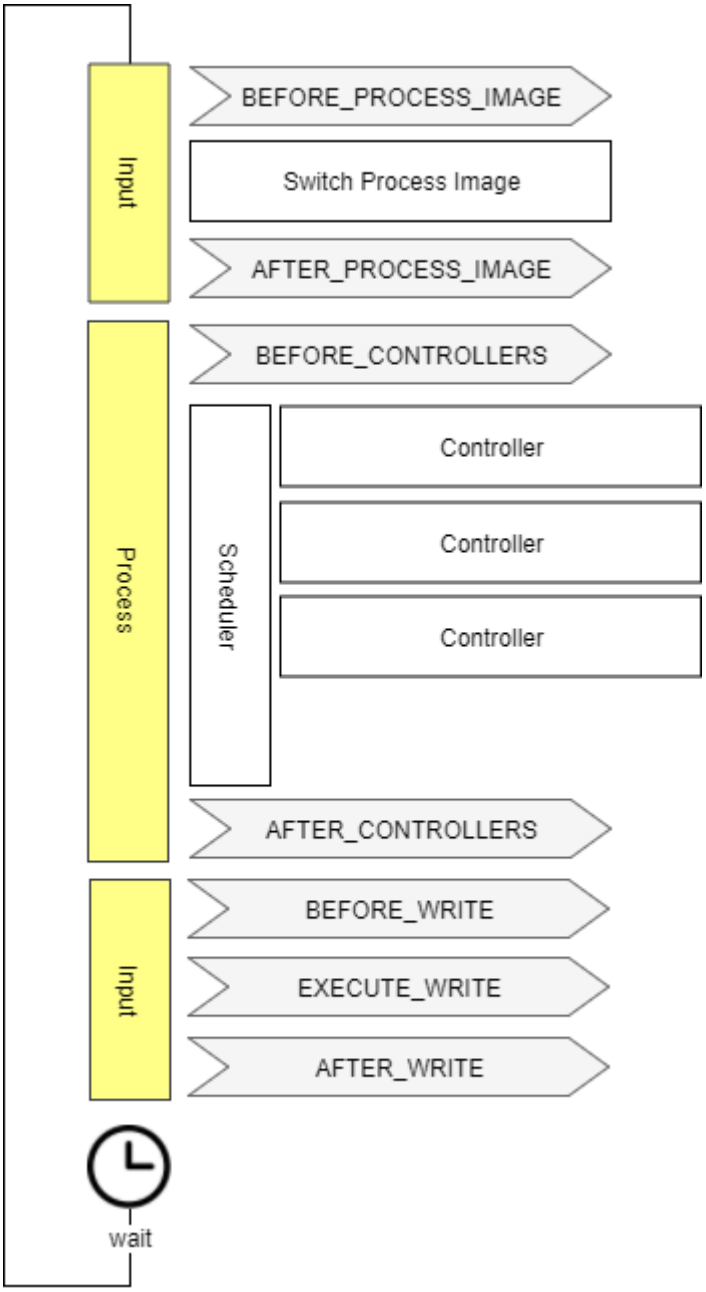

Figure 23. OpenEMS Edge Cycle

4.1.4. Asynchronous threads and Cycle synchronization

Communication with external hardware and services needs to be executed in asynchronous threads to not block the system. At the same time, those threads need to synchronize with the Cycle.

The following example shows, how the Modbus implementation </> uses Cycle Events to synchronize with the Cycle:

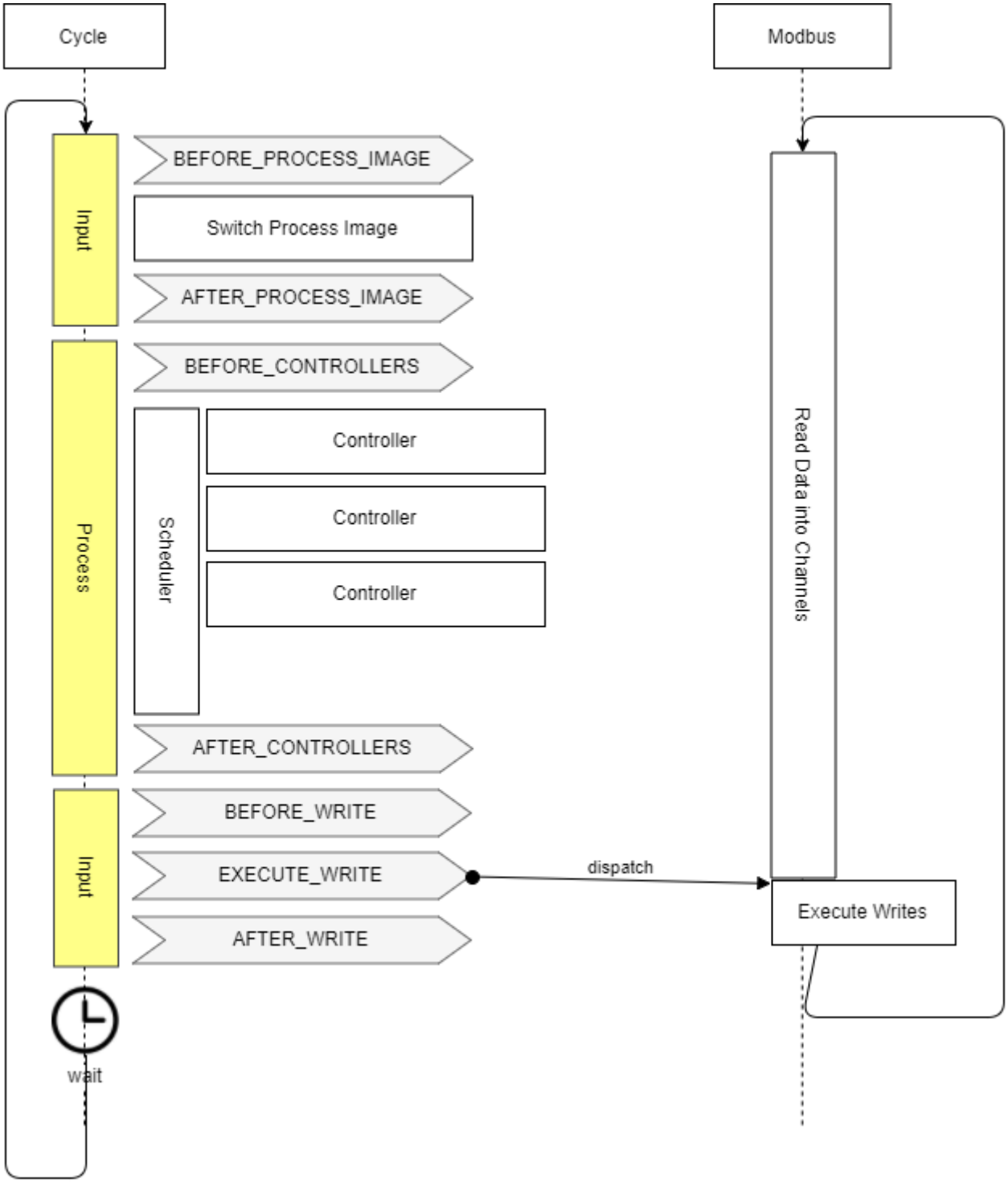

Figure 24. Synchronize Cycle with Modbus read/write

4.2. Configuration

4.3. Hardware

4.3.1. Natures

4.3.2. Bridges