

FEMS App REST/JSON Write Access

Version:2023.6.1

Table of Contents

1. Introduction 2

2. Installing the app 2

3. FEMS App REST/JSON Write Access 2

3.1. Prerequisites. . . . 2

3.2. REST/JSON basics 2

3.3. Write access 2

3.3.1. Timeout 3

3.3.2. /channel endpoint 3

3.3.3. Data points 3

3.3.4. Example 1 — Active power specification: Python 3

3.3.5. Example 2 — Active power specification: Talend API Tester 4

4. Contact us 6

5. Directories 7

5.1. List of illustrations 7

5.2. List of tables 8

1. Introduction

Dear customer,

Thank you for choosing the "FEMS App REST/JSON Write Access". You are welcome to send us your suggestions so that we can further improve the quality of our products.

2. Installing the app

When you ordered the "FEMS App REST/JSON Write Access", you received a 16-digit license key. You can use this license key to redeem the app independently in the FEMS App Center.

Instructions on how to proceed can be found here.

3. FEMS App REST/JSON Write Access

These instructions describe write access to an FENECON electrical energy storage system using the REST/JSON API. The functionality of the interface is then explained.

3.1. Prerequisites

The device accessing the electrical energy storage system (e.g. notebook/PC) must have direct access to the IP address of the FEMS — e.g. be connected to the same physical network.

3.2. REST/JSON basics

The REST/JSON interface allows access to FEMS on the local network via an interface modeled after REST.

3.3. Write access

This app provides an interface modeled after REST that can be used to describe data points in the system.



This app is **not** included in the standard scope of delivery of the FEMS. However, it can be retrofitted at any time. Please contact us if you would like a retrofit.



It is not possible to use write access through a guest login. Instead, a separate customer login is required. The password "owner" must be used for this. As with read access, the user name is up to your preference.

All write accesses must be sent as `POST` requests.

3.3.1. Timeout

This app has a configurable timeout. By default, this is configured to 60 seconds. It ensures that a default value remains active for 60 seconds. As soon as a new default value is written, the new value is used. If no new default value is written within 60 seconds, the controller reverts to the lower-priority controller — e. g. specification of a "0" power or self-consumption optimization.

3.3.2. /channel endpoint

The endpoint `/channel` enables access to individual data points, so-called "channels", in the system.

The full address of the endpoint is:

http://x:<PASSWORD>@<IP>:80/rest/channel/<COMPONENT>/<CHANNEL>

- `<COMPONENT>` is the ID of the component.
- `<CHANNEL>` is the ID of the channel

3.3.3. Data points

The following data points of the component _*ess0* can be described:

Data point	Description	Unit
`SetActivePowerEquals`	Default charging or discharging power	Watt [W]
`SetReactivePowerEquals`	Set reactive power specification	VoltAmpere Reactive [var]
`SetActivePowerLessOrEquals`	Set maximum discharge power	Watt [W]
`SetReactivePowerLessOrEquals`	Set maximum reactive power	VoltAmpere Reactive [var]
`SetActivePowerGreaterOrEquals`	Set maximum charging power	Watt [W]
`SetReactivePowerGreaterOrEquals`	Set minimum reactive power	VoltAmpere Reactive [var]

Table 1. Data points of the component _*ess0*



The registers for reactive power specifications cannot currently be used for home systems.



You can find more information on the 'SetActivePowerEquals' channel and other channels for power setting at Glossary.

3.3.4. Example 1 — Active power specification: Python

1. g. to specify a discharge power of 5 kW for the first electrical energy storage system (or electrical energy storage cluster), send a `POST` request to the address `http://192.168.0.23:80/rest/`

`channel/ess0/SetActivePowerEquals` with the power specification in *JSON* format.

```
{
  "value": 5000
}
```

Positive values correspond to battery discharging — Negative values correspond to battery charging.

The *requests* library, which must be imported at the beginning, can be used for this:

```
import requests

url = 'http://192.168.0.23:80/rest/channel/ess0/SetActivePowerEquals'

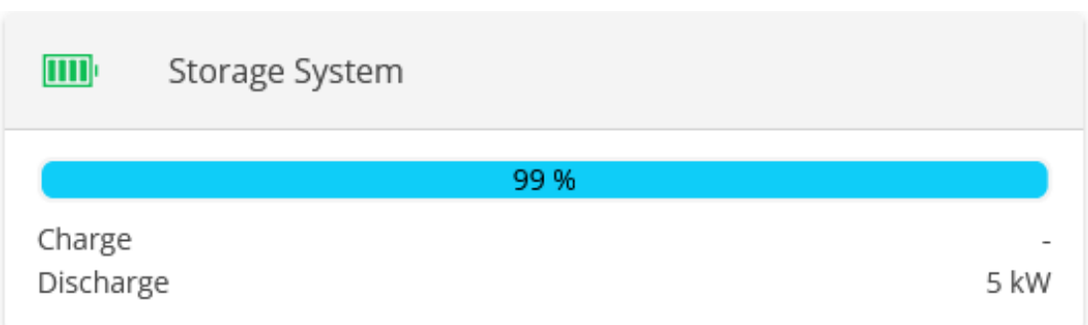
user = 'x'
password = 'owner'

session = requests.Session()
session.auth = (user, password)

data = {"value": 5000}

response = session.post(url, json = data)
response.raise_for_status()
```

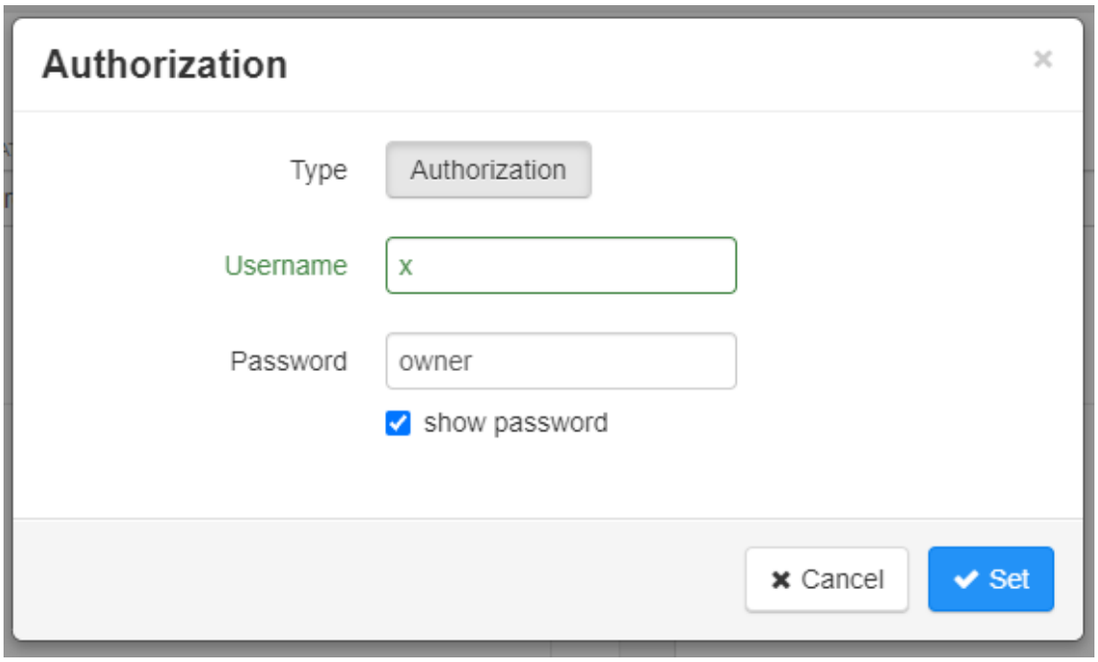
The correct execution of the request can be checked via a subsequent `GET` request or via Online Monitoring (see below).



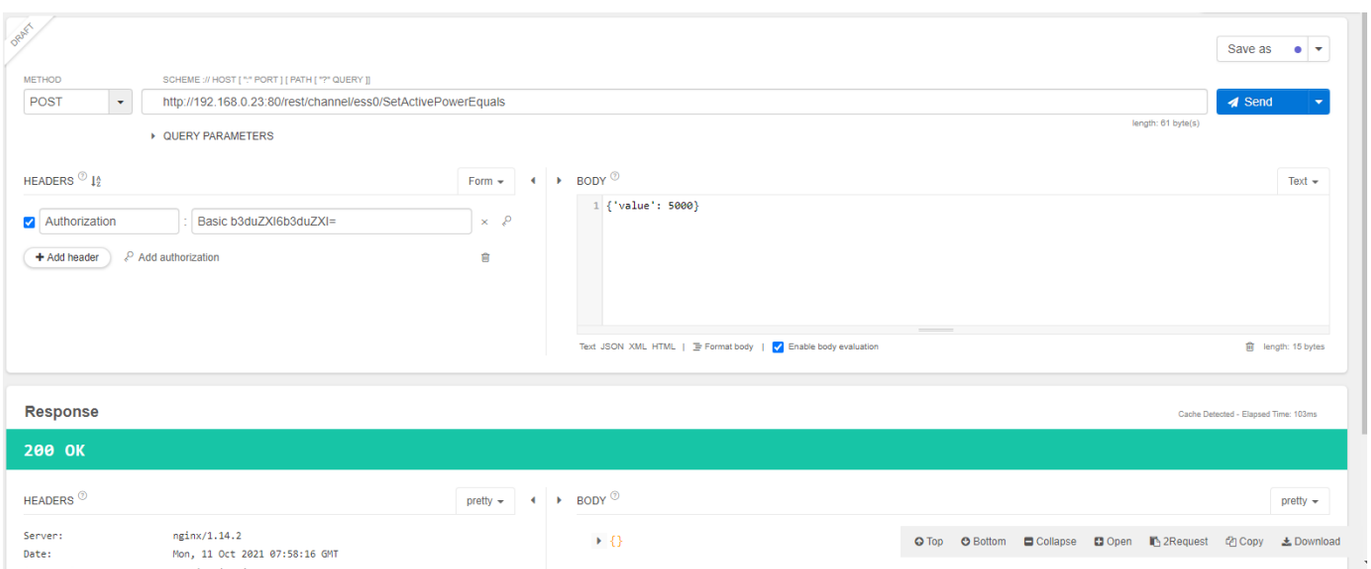
3.3.5. Example 2 — Active power specification: Talend API Tester

Talend API Tester is a Google Chrome extension that allows you to test REST APIs.

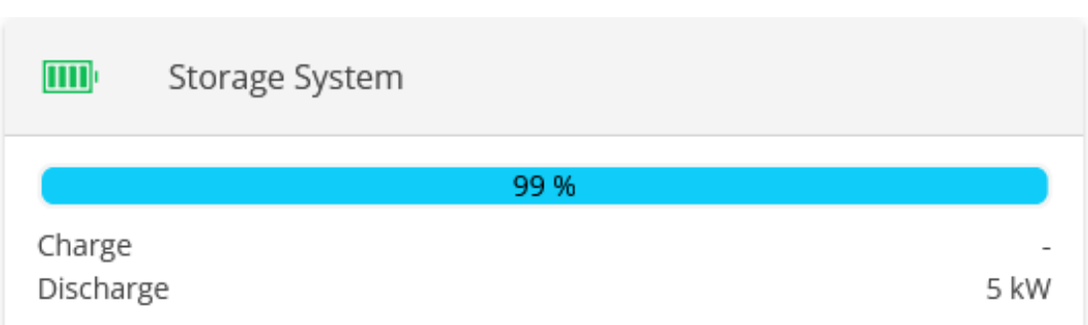
First, an *Authorization* header must be added:



The POST request can then be executed.



The correct execution of the request can be checked via a subsequent GET request or via Online Monitoring (see below).



4. Contact us

For support, please contact:

FENECON GmbH
Gewerbepark 6
94547 Iggensbach

Phone — Service: +49 (0) 9903 6280 0
E-mail — Service: service@fenecon.de

5. Directories

5.1. List of illustrations

5.2. List of tables

Table 1. Data points of the component _ess0