OpenEMS

- Getting Started
- DDSU666 (Direct chint meter) + mbpoll Command Reference
- Running OpenEMS Edge on a Raspberry PI
- DTSU666 3-Phase + mbpoll Command Reference
- SRNE ASP48120SH3 (Direct Hybrid Inverter) + mbpoll Command Reference
- SRNE ASP48120SH3 Modbus Register Map (validated)
- SRNE ASP48120SH3 Safe Settings Write: design, registers, and commissioning (#53)
- SRNE charge/discharge scheduling registers (self-decoded, #67)
- SRNE phase balancing: big consumers on one phase trip the inverter on battery
- OpenEMS Backend InfluxDB Data Model and Edge-ID Collision

Getting Started

ZeroTier Remote Access Guide

This guide explains how to connect to the NFE Raspberry Pi from anywhere in the world using ZeroTier. It covers both SSH access and troubleshooting common issues.

Table of Contents

1. Prerequisites
2. Network Information
3. Installing ZeroTier on Your Computer
4. Joining the Network
5. SSH Access to Raspberry Pi
6. Troubleshooting
7. Setting Up ZeroTier on a New Raspberry Pi

Prerequisites

Before starting, you need:

- A Mac or Windows laptop
- A ZeroTier account (free at https://my.zerotier.com)
- ZeroTier One installed on your computer
- Network authorization from the network administrator

Network Information

Network ID: `2873fd00f2d70904` **Network Name:** `my-first-network` **Raspberry Pi ZeroTier IP:** `10.135.127.86` **Raspberry Pi Username:** `nfetestpi2`

Installing ZeroTier on Your Computer

Mac:

1. Download ZeroTier One: https://www.zerotier.com/download/
2. Install it normally
3. After installation, the ZeroTier icon will appear in the top-right menu bar

Windows:

1. Download ZeroTier from https://www.zerotier.com/download/
2. Install and launch it
3. ZeroTier icon will appear in the system tray

Joining the Network

Method 1: Using the ZeroTier Menu (Mac - Recommended)

1. Click the ZeroTier icon in your menu bar
2. You'll see "My Address:" with your device ID
3. Click on the network ID `2873fd00f2d70904` if it's already listed
4. Or select "Join New Network..." and enter: `2873fd00f2d70904`
5. The status will show "REQUESTING_CONFIGURATION"

Method 2: Using Command Line

Mac/Linux:

```
sudo zerotier-cli join 2873fd00f2d70904
```

Windows (run as Administrator):

```
zerotier-cli join 2873fd00f2d70904
```

Authorization

After joining, you need to be authorized:

1. Contact the network administrator
2. Provide them with your device's MAC address or Device ID
3. They will authorize your device at https://my.zerotier.com
4. Once authorized, your device will receive an IP address like `10.135.127.xxx`

Verify Connection

Mac/Linux:

```
sudo zerotier-cli listnetworks
```

You should see:

```
200 listnetworks 2873fd00f2d70904 my-first-network ... OK PRIVATE ... 10.135.127.xxx/24
```

The status should show **OK** and you should have an IP address assigned.

SSH Access to Raspberry Pi

Once connected to the ZeroTier network, you can SSH to the Raspberry Pi:

```
ssh nfetestpi2@10.135.127.86
```

Enter the password when prompted.

Note: If you get a password prompt but it keeps failing, try using the `-v` flag for verbose output:

```
ssh -v nfetestpi2@10.135.127.86
```

Optional: Set Up SSH Keys (Recommended)

To avoid entering passwords every time:

1. Generate SSH key on your computer (if you don't have one):

```
ssh-keygen -t ed25519 -C "your-email@example.com"
```

2. Copy your public key:

```
cat ~/.ssh/id_ed25519.pub
```

3. Add it to the Pi's authorized keys (via SSH or Raspberry Pi Connect):

```
mkdir -p ~/.ssh
nano ~/.ssh/authorized_keys
# Paste your public key, save and exit

# Set correct permissions
chmod 700 ~/.ssh
chmod 600 ~/.ssh/authorized_keys
```

4. Now you can SSH without a password:

```
ssh nfetestpi2@10.135.127.86
```

Troubleshooting

Issue: ZeroTier shows "REQUESTING_CONFIGURATION"

Cause: Your device hasn't been authorized on the network yet.

Solution:

1. Go to https://my.zerotier.com
2. Log in and navigate to network `2873fd00f2d70904`
3. Click "Member Devices" tab
4. Find your device and check the "Auth" checkbox

Issue: ZeroTier shows "OFFLINE"

Cause: ZeroTier service isn't running properly.

Solution for Mac:

```
# Restart ZeroTier service
sudo launchctl unload /Library/LaunchDaemons/com.zerotier.one.plist
sudo launchctl load /Library/LaunchDaemons/com.zerotier.one.plist

# Verify it's online
sudo zerotier-cli info
```

You should see:

```
200 info <device-id> 1.16.0 ONLINE
```

Solution for Windows:

- Restart the ZeroTier service from Services (services.msc)
- Or restart the ZeroTier One application

Issue: Can't ping or SSH to Raspberry Pi

Symptoms:

```
ping 10.135.127.86
# Shows: "No route to host" or "Request timeout"
```

Solutions:

1. **Check if ZeroTier is running on both devices:**

```
sudo zerotier-cli info
# Should show: ONLINE
```

2. **Verify both devices are on the same network:**

```
sudo zerotier-cli listnetworks
# Both should show network 2873fd00f2d70904 with status OK
```

3. **Try changing WiFi networks:** Sometimes the initial WiFi network blocks ZeroTier's peer-to-peer connections. Try connecting to a different WiFi network or mobile hotspot.
4. **Check for RELAY connection:**

```
sudo zerotier-cli peers
```

If the Pi shows as "RELAY" instead of "DIRECT", there's a NAT traversal issue. Try:

- Restarting ZeroTier on both devices
- Leaving and rejoining the network
- Changing WiFi networks

5. **Restart ZeroTier on both devices:**

Mac:

```
sudo launchctl unload /Library/LaunchDaemons/com.zerotier.one.plist
sudo launchctl load /Library/LaunchDaemons/com.zerotier.one.plist
```

Raspberry Pi (via Raspberry Pi Connect):

```
sudo systemctl restart zerotier-one
```

6. **Leave old networks:** If you have multiple networks joined, leave unused ones:

```
# List networks
sudo zerotier-cli listnetworks

# Leave old network
sudo zerotier-cli leave <old-network-id>
```

Issue: SSH password keeps failing

Solutions:

1. Try verbose SSH to see what's happening:

```
ssh -v nfetestpi2@10.135.127.86
```

2. Make sure you're using the correct username (`nfetestpi2`, not `pi`)
3. Set up SSH keys instead (see SSH Keys section above)

Setting Up ZeroTier on a New Raspberry Pi

If you need to set up ZeroTier on a new Raspberry Pi, follow these steps:

Prerequisites

- Raspberry Pi with Raspberry Pi OS installed
- Internet connection
- Access to the Pi (via Raspberry Pi Connect, monitor/keyboard, or local SSH)

Installation Steps

1. **Install ZeroTier on the Raspberry Pi:**

```
curl -s https://install.zerotier.com | sudo bash
```

2. **Join the network:**

```
sudo zerotier-cli join 2873fd00f2d70904
```

3. **Verify the Pi joined:**

```
sudo zerotier-cli listnetworks
```

You'll see status as "ACCESS_DENIED" initially.

4. **Authorize the Pi:**

- Go to https://my.zerotier.com
- Log in and navigate to network `2873fd00f2d70904`
- Click "Member Devices" tab
- Find the new Pi device (you can identify it by the MAC address or hostname)
- Check the "Auth" checkbox
- Note the "Managed IP" assigned to the Pi (e.g., `10.135.127.xxx`)

5. **Verify connection:**

```
sudo zerotier-cli listnetworks
```

Should now show:

```
200 listnetworks 2873fd00f2d70904 my-first-network ... OK PRIVATE ztxxxxxx 10.135.127.xxx/24
```

6. **Enable SSH (if not already enabled):**

```
sudo systemctl enable ssh
sudo systemctl start ssh
```

7. **Make ZeroTier start on boot:**

```
sudo systemctl enable zerotier-one
```

8. **Test connection from another device:**

```
# From your computer (already on ZeroTier network)
ping <new-pi-ip>
ssh <username>@<new-pi-ip>
```

9. **Update this documentation** with the new Pi's IP address!

Advanced: VNC Access (Remote Desktop)

If you need graphical access to the Raspberry Pi:

1. **Enable VNC on the Pi:**

```
sudo raspi-config
# Navigate to: Interface Options → VNC → Enable
```

2. **Install VNC Viewer on your computer:**
   https://www.realvnc.com/en/connect/download/viewer/
3. **Connect using the ZeroTier IP:**

- Open VNC Viewer
- Enter: `10.135.127.86`
- Enter Pi username and password

Quick Reference

Useful Commands

```
# Check ZeroTier status
sudo zerotier-cli info
```

```
# List joined networks
sudo zerotier-cli listnetworks

# Join a network
sudo zerotier-cli join <network-id>

# Leave a network
sudo zerotier-cli leave <network-id>

# Check peer connections
sudo zerotier-cli peers

# SSH to Raspberry Pi
ssh nfetestpi2@10.135.127.86
```

Network Details

- **Network ID:** `2873fd00f2d70904`
- **Network Name:** `my-first-network`
- **Pi IP:** `10.135.127.86`
- **Pi Username:** `nfetestpi2`

Last updated: 2026-03-28

DDSU666 (Direct chint meter) + mbpoll Command Reference

Version: 1.0
Prepared For: Field & Deployment Teams
Platform: Raspberry Pi / Linux
Tool: mbpoll

1. Purpose

This document explains how to communicate with the CHINT DDSU666 Direct Smart Meter using Modbus RTU and the mbpoll tool.

It covers:
• Reading electrical parameters
• Setting meter addresses
• Verifying communication
• Preparing for OpenEMS integration

2. Hardware & Software Requirements

Hardware
• DDSU666 (Direct Version)
• RS485 → USB Converter
• Raspberry Pi / Linux PC
• Correct RS485 wiring (A(converter)↔24(meter com-port terminal), B(converter)↔25(meter com-port terminal), GND recommended)

Software
Install mbpoll:

> “
> sudo apt update
> sudo apt install mbpoll

Check serial port:

```
ls /dev/ttyUSB*
```

Example output:

```
/dev/ttyUSB0
```

3. Communication Parameters (Confirmed)

Parameter	Value
Protocol	Modbus RTU
Baud Rate	9600
Data Bits	8
Parity	None
Stop Bits	2
Format	8N2
Float Order	Big Endian
Addressing	0-Based

These parameters must always be used.

4. Standard mbpoll Format

All commands follow this format:

```
mbpoll -m rtu -b 9600 -P none -s 2 -t 4:float -B -0 -r <register> -c <count>
/dev/ttyUSB0 -a <id> -1
```

Where:

- <register> = Modbus register
- <count> = Number of values
- <id> = Meter address (NO.)

5. Electrical Parameters

Summary:

```
```

> Voltage: 0x2000
> Current: 0x2002
> Active Power: 0x2004
> Power Factor: 0x200A
> Frequency: 0x200E
> Energy: 0x4000

5.1 Voltage (V) — Register 0x2000

> “
> **Meter ID = 1**
> mbpoll -m rtu -b 9600 -P none -s 2 -t 4:float -B -0 \
> -r 0x2000 -c 1 /dev/ttyUSB0 -a 1 -1
> **Meter ID = 2**
> mbpoll -m rtu -b 9600 -P none -s 2 -t 4:float -B -0 \
> -r 0x2000 -c 1 /dev/ttyUSB0 -a 2 -1

5.2 Current (A) — Register 0x2002

> “
> **Meter ID = 1**
> mbpoll -m rtu -b 9600 -P none -s 2 -t 4:float -B -0 \
> -r 0x2002 -c 1 /dev/ttyUSB0 -a 1 -1
> **Meter ID = 2**
> mbpoll -m rtu -b 9600 -P none -s 2 -t 4:float -B -0 \
> -r 0x2002 -c 1 /dev/ttyUSB0 -a 2 -1

5.3 Active Power — Register 0x2004

> “
> **Meter ID = 1**
> mbpoll -m rtu -b 9600 -P none -s 2 -t 4:float -B -0 \
> -r 0x2004 -c 1 /dev/ttyUSB0 -a 1 -1
>
> **Meter ID = 2**
> mbpoll -m rtu -b 9600 -P none -s 2 -t 4:float -B -0 \
> -r 0x2004 -c 1 /dev/ttyUSB0 -a 2 -1

5.4 Power Factor — Register 0x200A

> **Meter ID = 1**
> mbpoll -m rtu -b 9600 -P none -s 2 -t 4:float -B -0 \
> -r 0x200A -c 1 /dev/ttyUSB0 -a 1 -1
> **Meter ID = 2**
> mbpoll -m rtu -b 9600 -P none -s 2 -t 4:float -B -0 \
> -r 0x200A -c 1 /dev/ttyUSB0 -a 2 -1

5.5 Frequency (Hz) — Register 0x200E

> **Meter ID = 1**
> mbpoll -m rtu -b 9600 -P none -s 2 -t 4:float -B -0 \
> -r 0x200E -c 1 /dev/ttyUSB0 -a 1 -1
> **Meter ID = 2**
> mbpoll -m rtu -b 9600 -P none -s 2 -t 4:float -B -0 \
> -r 0x200E -c 1 /dev/ttyUSB0 -a 2 -1

5.6 Energy (kWh) — Register 0x4000

> **Meter ID = 1**
> mbpoll -m rtu -b 9600 -P none -s 2 -t 4:float -B -0 \
> -r 0x4000 -c 1 /dev/ttyUSB0 -a 1 -1
> **Meter ID = 2**
> mbpoll -m rtu -b 9600 -P none -s 2 -t 4:float -B -0 \
> -r 0x4000 -c 1 /dev/ttyUSB0 -a 2 -1

6. Reading Multiple Values

Use **-c** option to read multiple registers.

Example (Voltage + Current Together)

>

```
mbpoll -m rtu -b 9600 -P none -s 2 -t 4:float -B -0 \
-r 0x2000 -c 2 /dev/ttyUSB0 -a 1 -1
```

7. Meter Address

> **Register: 0x0006**
>
> ```
> mbpoll -m rtu -b 9600 -P none -s 2 -t 4:int16 -0 \
> -r 0x0006 -c 1 /dev/ttyUSB0 -a 2 -1
> ```
>
> **Example Output:**
> [6]: 2

8. Changing Address

- Only one meter connected.
- Power cycle after change.

> **Change Address (2 → 1)**
> ```
> mbpoll -m rtu -b 9600 -P none -s 2 -t 4:int16 -0 \
> -r 0x0006 /dev/ttyUSB0 -a 2 -W -- 1
> ```
> **Power Cycle**
> Turn OFF → ON the meter.
> **Verify**
> ```
> mbpoll -m rtu -b 9600 -P none -s 2 -t 4:int16 -0 \
> -r 0x0006 -c 1 /dev/ttyUSB0 -a 1 -1
> ```

9. Health Check

Voltage should be 220–240V.

> ```
> mbpoll -m rtu -b 9600 -P none -s 2 -t 4:float -B -0 \
> -r 0x2000 -c 1 /dev/ttyUSB0 -a 1 -1
> ```

10. Common Issues

Timeout Errors

- Duplicate IDs
- Wrong wiring
- Wrong stop bits
- Multiple meters during setup

Incorrect Float Values

Use -B option.

Current / Power = 0

Normal when:

- No load
- Bench testing

11. Deployment Workflow

For large installations:

- Connect one meter
- Convert to Modbus
- Set address
- Test voltage
- Install
- Repeat
- Never deploy duplicate addresses.

> “
> Connect → Configure → Test → Install

12. System Status

✔ Modbus Enabled
✔ 9600 8N2 Confirmed
✔ Big-Endian Floats
✔ Multi-meter Bus Working
✔ Ready for OpenEMS

Running OpenEMS Edge on a Raspberry PI

This guide explains how to:

1. Install and run **OpenEMS Edge on a Raspberry Pi**
2. Configure WebSocket + simulation
3. Run **OpenEMS UI on a macOS machine (not on the Pi)**
4. Connect the UI to the Edge over the internet

The structure is intentionally split into two clear parts:

- **PART A — Raspberry Pi (Edge)**
- **PART B — macOS Machine (OpenEMS UI)**

==========

PART A — Raspberry Pi (Edge)

==========

1. Raspberry Pi OS Setup (Headless)

Use Raspberry Pi Imager:

- Device: Raspberry Pi 4
- OS: Raspberry Pi OS Lite (64-bit)
- Configure:

    - Hostname
    - Username & password
    - Enable SSH
    - Configure Wi-Fi
    - Enable Raspberry Pi Connect

Boot the Pi and connect via:

- SSH
- OR Raspberry Pi Connect

---

2. Install Java 21 (Temurin ARM64)

On the Pi:

```
sudo apt update
sudo apt install -y wget apt-transport-https gpg

wget -qO - https://packages.adoptium.net/artifactory/api/gpg/key/public \
 | gpg --dearmor | sudo tee /etc/apt/trusted.gpg.d/adoptium.gpg > /dev/null

echo "deb https://packages.adoptium.net/artifactory/deb \
$(awk -F= '/^VERSION_CODENAME/{print $2}' /etc/os-release) main" \
 | sudo tee /etc/apt/sources.list.d/adoptium.list

sudo apt update
sudo apt install -y temurin-21-jdk
```

Verify:

```
java -version
```

---

3. Install OpenEMS Edge

```
mkdir -p ~/downloads
cd ~/downloads
wget https://github.com/OpenEMS/openems/releases/download/2025.11.0/openems-edge.jar
sudo chmod +x openems-edge.jar
```

```
sudo mkdir -p /usr/lib/openems
sudo mv openems-edge.jar /usr/lib/openems/
sudo mkdir -p /etc/openems.d
```

---

4. Configure systemd Service

Create:

```
sudo nano /etc/systemd/system/openems.service
```

Paste:

```
[Unit]
Description=OpenEMS Edge
After=network.target

[Service]
User=root
Group=root
Type=notify
WorkingDirectory=/usr/lib/openems
ExecStart=/usr/bin/java -Dfelix.cm.dir=/etc/openems.d/ \
  -jar /usr/lib/openems/openems-edge.jar
SuccessExitStatus=143
Restart=always
RestartSec=10
WatchdogSec=60

[Install]
WantedBy=multi-user.target
```

Enable & start:

```
sudo systemctl daemon-reload
sudo systemctl enable openems
sudo systemctl start openems
```

Check:

```
systemctl status openems
```

---

5. Access OpenEMS Config Manager

On the Pi, open:

```
http://localhost:8080/system/console/configMgr
```

---

6. Add Required Components

In Config Manager:

1. Add a **Scheduler** (any default scheduler)
2. Add **Controller.Api.Websocket**
   - Port: 8085
   - Enabled: true

Restart Edge:

```
sudo systemctl restart openems
```

Verify WebSocket is listening:

```
sudo ss -lntp | grep 8085
```

You should see Java listening on port 8085.

---

7. (Optional) Add Simulation Components

In Config Manager, add:

- Simulator ESS
- Simulator Grid Meter
- Simulator PV

Save and restart OpenEMS.

==========

==========

PART B — macOS Machine (OpenEMS UI)

==========

==========

1. Install Docker Desktop

```
brew install --cask docker
open -a Docker
```

Wait until Docker reports "Docker is running".

Verify:

```
docker version
```

You must see both Client and Server.

---

2. Clone OpenEMS Source

```
git clone -b 2025.11.0 https://github.com/OpenEMS/openems
cd openems
```

3. Build OpenEMS UI Image

```
docker build . \
  -t openems_ui \
  -f tools/docker/ui/Dockerfile.edge
```

4. Run OpenEMS UI

Replace YOUR_PI_IP with:

- Public IP
- OR VPN IP
- OR Public DNS

Example:

```
docker container run \
  -e WEBSOCKET_HOST=YOUR_PI_IP \
  -p 80:80 \
  -p 443:443 \
  --restart unless-stopped \
  --name openems_ui_container \
  openems_ui
```

5. Open the UI

On your Mac:

```
http://localhost/login
```

Default login:

- Username: admin
- Password: admin

If UI shows "disconnected":

- Confirm port 8085 is listening on the Pi
- Confirm WebSocket controller exists
- Confirm WEBSOCKET_HOST is correct

Why UI Runs on macOS and Edge Runs on Pi

- Edge interacts with hardware and benefits from native systemd management
- UI behaves like a stateless web application and is ideal for containerization
- Separating them improves stability and flexibility

Troubleshooting

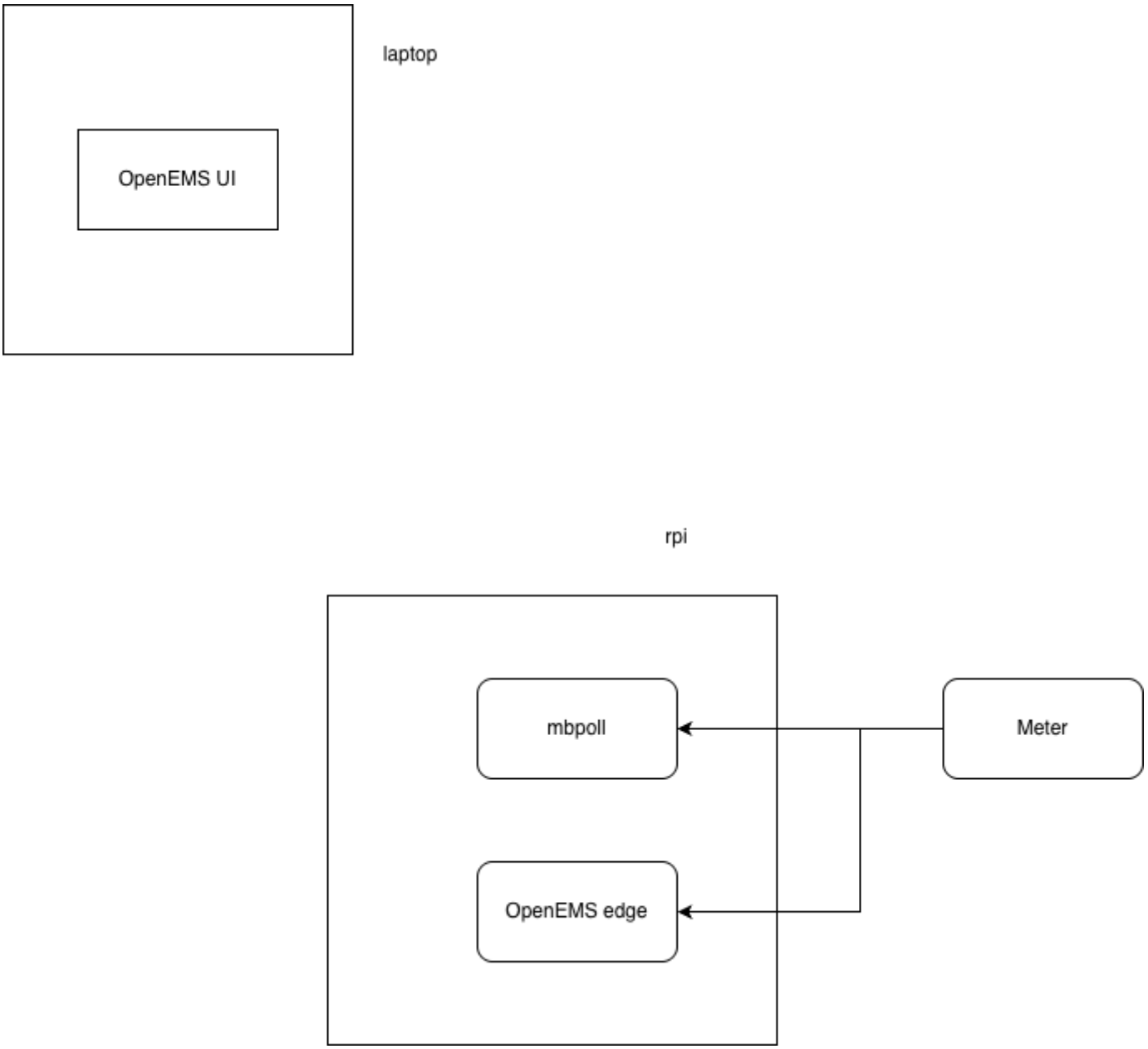
- UI loads but no connection → check port 8085
- Nothing listening on 8085 → WebSocket not configured
- Docker errors on Mac → ensure Docker Desktop is running

Security Notes

- Prefer VPN/tunnel (Tailscale/WireGuard) over port forwarding
- Use SSH key-based authentication
- Do not expose ConfigMgr (8080) publicly

You now have:

- OpenEMS Edge running on Raspberry Pi
- WebSocket enabled
- Optional simulation components
- OpenEMS UI running on macOS
- UI connected over internet


laptop
OpenEMS UI
rpi
mbpoll
Meter
OpenEMS edge

DTSU666 3?Phase + mbpoll Command Reference

? Standard Communication Settings (DTSU666)

Most DTSU666 meters use:

Protocol : Modbus RTU
Baud Rate : 9600
Data Bits : 8
Parity : None
Stop Bits : 2
Mode : Big-endian
Table : Input Registers (FC04)

Base communication block used in all commands:

-m rtu -b 9600 -P none -s 2

? Critical Flags (Important)

Flag	Meaning	Required?
`-t 3`	Input Registers (Function 04)	✅ Yes
`:float`	32-bit floating point	✅ Yes
`-B`	Big-endian word order	✅ Yes
`-0`	Zero-based addressing	✅ Yes
`-a <id>`	Slave address	✅ Yes

Flag	Meaning	Required?
`-1`	Poll once	Optional
`-l 5000`	Poll every 5 seconds	Optional

? Phase?to?Neutral Voltages (L1?N, L2?N, L3?N)

Register: `0x2006`

mbpoll -m rtu -b 9600 -P none -s 2 \
-t 3:float -B -0 \
-r 0x2006 -c 3 \
/dev/ttyUSB0 -a 1 -1

? Line?to?Line Voltages (L1?L2, L2?L3, L3?L1)

Register: `0x2000`

mbpoll -m rtu -b 9600 -P none -s 2 \
-t 3:float -B -0 \
-r 0x2000 -c 3 \
/dev/ttyUSB0 -a 1 -1

? Phase Currents (L1, L2, L3)

Register: `0x200C`

```
mbpoll -m rtu -b 9600 -P none -s 2 \
-t 3:float -B -0 \
-r 0x200C -c 3 \
/dev/ttyUSB0 -a 1 -1
```

? Total Active Power (kW)

Register: `0x2012`

```
mbpoll -m rtu -b 9600 -P none -s 2 \
-t 3:float -B -0 \
-r 0x2012 -c 1 \
/dev/ttyUSB0 -a 1 -1
```

? Frequency (Hz)

Register: `0x2044`

```
mbpoll -m rtu -b 9600 -P none -s 2 \
-t 3:float -B -0 \
-r 0x2044 -c 1 \
/dev/ttyUSB0 -a 1 -1
```

? Total Active Energy (kWh)

Register: `0x4000`

```
mbpoll -m rtu -b 9600 -P none -s 2 \
-t 3:float -B -0 \
-r 0x4000 -c 1 \
/dev/ttyUSB0 -a 1 -1
```

? Continuous Polling Example (Every 5 Seconds)

```
mbpoll -m rtu -b 9600 -P none -s 2 \
-t 3:float -B -0 \
-r 0x2006 -c 1 \
-l 5000 \
/dev/ttyUSB0 -a 1
```

? Change Slave Address (Only One Meter Connected)

⚠ Use Holding Registers (Function 03) for configuration.

Register: `0x002E`

```
mbpoll -m rtu -b 9600 -P none -s 2 \
-t 4:int16 -0 \
-r 0x002E \
/dev/ttyUSB0 -a 1 -W -- 5
```

Changes slave ID from 1 → 5.

? Quick Communication Test

Fastest health check:

```
mbpoll -m rtu -b 9600 -P none -s 2 \
-t 3:float -B -0 \
-r 0x2006 -c 1 \
/dev/ttyUSB0 -a 1 -1
```

If you see ~230V → communication confirmed.

? Troubleshooting Quick Guide

Symptom	Likely Cause
Timeout	Wrong address or wiring
4200+ volts	Reading wrong register
-8192 values	Wrong data type
Nonsense numbers	Missing `-B`
No `/dev/ttyUSB0`	USB-RS485 not detected

SRNE ASP48120SH3 (Direct Hybrid Inverter) + mbpoll Command Reference

Version: 1.1 **Prepared For**: Field & Deployment Teams **Platform**: Raspberry Pi / Linux **Tool**: mbpoll / pymodbus

1. Purpose

Complete reference for the SRNE ASP48120SH3 three-phase hybrid inverter over Modbus RTU: what can be READ, what can be WRITTEN, and the tested limits. Validated against the live unit at Nansana on 2026-07-21 (slave address 1).

2. Hardware & Software Requirements

- SRNE ASP48120SH3, RS485 to USB converter (CH340), Raspberry Pi / Linux PC.
- Wiring: A(converter) to WiFi-port RJ45 pin 7 (RS485-A), B to pin 8 (RS485-B), GND to pin 2. The WiFi port carries the monitoring bus; the RS485/CAN port is for the battery BMS.
- Install: `sudo apt install mbpoll` and/or `pip install pymodbus`.

3. Serial parameters and function codes

- 9600 baud, 8 data bits, No parity, 1 stop bit (8N1). Slave address 1.
- READ: function code 03 (holding registers). 16-bit registers with scale factors (not 32-bit floats).
- WRITE: BOTH FC06 (write single) and FC16 (write multiple) are supported and tested. Note: the CHINT meters reject FC06, the inverter does NOT.
- mbpoll read flags: `-b 9600 -P none -s 1 -t 4 -0`.

4. Reading values (exhaustive, all ranges respond)

Every documented range answers FC03. Ranges (SRNE V1.7 segment map, confirmed on this unit):

Product info (P00), 0x0000 to 0x0049

- 0x000B machine type = 4 (integrated inverter-controller)
- 0x0014 software version = 925
- 0x001A RS485 address = 1 (read only)

Controller / battery (P01), 0x0100 to 0x0122

- 0x0100 SOC (%), 0x0101 battery V (x0.1), 0x0102 battery I (x0.1 signed, + charge / - discharge), 0x0103 device temp (reads 0, unused)
- 0x0104 to 0x0106 DC load V/I/P (0, no DC load on this unit), 0x0107 to 0x0111 PV panels 1/2 (0, no PV), 0x010B charge state (1 = quick charge)

Inverter (P02), 0x0200 to 0x0237 (the main live telemetry, see the register-map page)

- state 0x0210, bus 0x0212, grid+inverter phase A 0x0213 to 0x0218, load 0x0219 to 0x021F, heat-sinks 0x0220 to 0x0223, PBus/NBus 0x0228/0x0229, 3-phase B/C V/I 0x022A to 0x022F, 3-phase B/C load 0x0230 to 0x0237.

Settings / parameters, 0xE001 to 0xE02D (writable, see section 5). Raw values present; battery voltage thresholds around 0xE006 to 0xE00E appear as 48V-system values (register/10 x 4), e.g. 144 = 57.6 V. Exact per-register meaning is TBD from the ASP settings doc.

Settings F, 0xF000 to 0xF01F: additional parameters (raw values present).

Control commands, 0xDF00 to 0xDF09: write-only, read back as 0.

Example reads (mbpoll):

```
# battery block 0x0100 (dec 256), 4 regs
mbpoll -m rtu -b 9600 -P none -s 1 -t 4 -0 -r 256 -c 4 -a 1 -1 <port>
# inverter block 0x0210 (dec 528), 32 regs
mbpoll -m rtu -b 9600 -P none -s 1 -t 4 -0 -r 528 -c 32 -a 1 -1 <port>
# settings 0xE001 (dec 57345), 45 regs
mbpoll -m rtu -b 9600 -P none -s 1 -t 4 -0 -r 57345 -c 45 -a 1 -1 <port>
```

5. Writing values (tested 2026-07-21)

The inverter accepts BOTH FC06 and FC16 and enforces a permission model:

- Read-only registers (telemetry, product info, e.g. 0x001A) REJECT writes with exception code 7 ("parameter is read only"). Tested.

- Settings registers (0xE0xx) ACCEPT writes (both FC06 and FC16), with no password or run-state lock on the tested registers (0xE015, 0xE01F, 0xE024). Tested via no-op write-back (wrote the current value, confirmed unchanged).
- Control commands (0xDF00 to 0xDF09) are write-only actions. NOT fired on the live unit because it feeds customers. Documented below.

Exception code table (SRNE V1.7): 1 illegal function, 2 illegal data address, 3 illegal data value, 4 operation failed, 5 password error, 7 read only, 8 cannot change while running, 9 password protection, 10 length error, 11 permission denied.

Control command registers (DANGEROUS on a live system, documented only):

- 0xDF00 power ON/OFF: 1 = on, 0 = off. OFF cuts inverter output, do NOT fire while it feeds customers.
- 0xDF01 reset: 1 = reset.
- 0xDF02 restore defaults: 0xAA wipes all settings.
- 0xDF03 clear current alarm: 1.
- 0xDF04 clear statistics: 1.
- 0xDF05 clear history: 1.
- 0xDF08 sleep/run: 0x5A5A sleep, 0xA5A5 run.

How to write:

```
# mbpoll FC06 (write single): value as the last argument
mbpoll -m rtu -b 9600 -P none -s 1 -t 4 -0 -r <dec_addr> -a 1 <port> <value>


# pymodbus
client.write_register(address=REG, value=V, device_id=1)     # FC06
client.write_registers(address=REG, values=[V], device_id=1) # FC16
```

6. Tested limits summary

- CAN: read every range over FC03; write settings 0xE0xx with FC06 or FC16; read-only registers are protected (code 7).
- CAUTION: control commands 0xDF00 to 0xDF09 are live actions (power, reset, restore). Documented, NOT fired on the customer-feeding unit. Only fire with an outage window and explicit sign-off.
- UNKNOWN / next: exact meaning of each 0xE0xx and 0xF0xx setting (needs the ASP settings doc or careful change-and-revert testing offline); machine-state codes other than 2 (verify by observing transitions).

7. Preparing for OpenEMS integration

The validated telemetry map is a declarative table in `src/inverter/registers.py` (nfe-modbus-energy-logger), grouped by OpenEMS nature (ElectricityMeter grid/backup, SymmetricEss battery, OffGridBatteryInverter state). It maps 1:1 to a GoodWe-style defineModbusProtocol (FC3ReadRegistersTask + word elements + SCALE_FACTOR). Control writes map to FC16WriteRegistersTask on the 0xDF00 area and 0xE0xx settings.

SRNE ASP48120SH3 Modbus Register Map (validated)

Validated Modbus register map for the SRNE ASP48120SH3 three-phase hybrid inverter, verified register-by-register against the live Nansana unit, including a **grid-off / grid-on transition test on 2026-07-27** that resolved the previously-unknown registers and corrected the battery-current sign.

Bus: RS485, 9600 8N1, function code 03 (read holding registers), slave address 1. **Source of truth (code)**: `src/inverter/registers.py` in nfe-modbus-energy-logger.

Valid register ranges (probed live)

- P00 product info: 0x0000 to 0x0049
- P01 battery / DC: 0x0100 to 0x0111
- P02 inverter telemetry: 0x0200 to **0x023F** (0x0240 and beyond return "Illegal data address")
- P03 control commands: 0xDF00 to 0xDF09 (write-only, read back 0)
- Settings E (writable params): 0xE001 to 0xE02D ; Settings F: 0xF000 to 0xF01F

Live telemetry, full per-phase - VERIFIED

quantity	phase A	phase B	phase C	notes
grid voltage	0x0213	0x022A	0x022B	x0.1 V (all go to 0 when grid is off)
grid current	0x0214	**0x0238**	**0x0239**	x0.1 A. B/C live in the extended block, CONFIRMED grid-side by the transition test
grid frequency	0x0215	-	-	x0.01 Hz
grid active power	0x023A	0x023B	0x023C	W (extended block, grid-side)
inverter voltage	0x0216	0x022C	0x022D	x0.1 V

quantity	phase A	phase B	phase C	notes
inverter current	0x0217	0x022E	0x022F	x0.1 A (rises when off-grid, inverter sources the load)
inverter frequency	0x0218	-	-	x0.01 Hz
load current	0x0219	0x0230	0x0231	x0.1 A (load-side, holds through the transition)
load active power	0x021B	0x0232	0x0233	W. No total register: sum the three phases
load reactive power	(derive)	0x0234	0x0235	var. Phase A: S=sqrt(P^2+Q^2) not given, derive
load apparent power	0x021C	(derive)	(derive)	VA. B/C: derive S = sqrt(P^2 + Q^2)
load ratio	0x021F	0x0236	0x0237	%

Other P02: state 0x0210, password status 0x0211, bus V 0x0212 (x0.1) = PBus 0x0228 + NBus 0x0229, inverter DC component 0x021D (mV), mains charge current 0x021E (x0.1 A), heat-sink temps 0x0220/0x0221/0x0222 (x0.1 C, 0x0223 bogus), PV charge current 0x0224 (0, no PV), inverter fault 0x0226, charge status 0x0227. Extended grid block tail 0x023D/0x023E/0x023F (0/221/245/309 on grid) is grid-side reactive/apparent, exact split still to confirm.

Battery / DC (P01) - VERIFIED

SOC 0x0100 (%), battery V 0x0101 (x0.1 V), battery current 0x0102 (x0.1 A, signed), device temp 0x0103 (reads 0, dead), DC load 0x0104-0x0106 (0), PV1 0x0107-0x0109 (0, no PV), charge state 0x010B, charge power 0x010E, PV2 0x010F-0x0111 (0).

IMPORTANT - battery current sign is POSITIVE = DISCHARGE on this unit (the OPPOSITE of the generic SRNE V1.7 doc). Proven by the 2026-07-27 off-grid test: with grid and PV both at 0, the battery is the only source, and while it was unambiguously discharging ~750 W to the loads it read **+14.2 A**; when the grid returned, discharge collapsed to +0.4 A (idle). So positive current = discharge, negative = charge.

Grid-off / grid-on transition test (2026-07-27) - what died vs held

- GRID-side (went to 0 when utility cut, back live when restored): grid V A/B/C (0x0213/0x022A/0x022B), grid I A (0x0214), grid freq (0x0215), and the whole extended block 0x0238 to 0x023F (grid I B/C + grid power). This is how the extended block was identified.
- LOAD-side (held through the transition, source-independent): load I/P/Q per phase (0x0219, 0x021B, 0x0230-0x0237).
- INVERTER-side (rose off-grid as it took the load): inverter I A/B/C (0x0217/0x022E/0x022F).
- STATE 0x0210: **2 = on-grid / mains-bypass, 3 = off-grid / inverter-powered** (both codes validated by the transition). Charge state 0x010B: 0 off-grid, 1 on-grid.

Corrections vs the generic SRNE V1.7 doc (do NOT trust the generic enums/signs)

- Battery current sign: **positive = discharge** (doc says positive = charge - WRONG here).
- Machine state 0x0210: 2 = running/mains-bypass, 3 = off-grid. NOT the generic "initialization/soft-start".
- Power factor 0x021A reads 0. Derive PF = active (0x021B) / apparent (0x021C).
- Device temperature 0x0103 reads 0. Use heat-sink temps 0x0220 to 0x0222.

For the OpenEMS driver (important)

- There is NO inverter-output-power and NO ESS/battery AC-power register. The ESS SymmetricEss.ACTIVE_POWER must be DERIVED from battery DC power (0x0101 x 0x0102). Do NOT map load power 0x021B (that is the loads, phase A only).
- **Dc/ESS power sign**: since positive battery current = discharge, DcDischargePower = +V x I (do NOT negate). A negated formula reports a real discharge as a charge.
- Grid meter can now have per-phase current and power: grid I A/B/C (0x0214/0x0238/0x0239) and grid active power A/B/C (0x023A/0x023B/0x023C).
- Consumption/backup meter power from the load registers (0x021B/0x0232/0x0233 active, 0x0234/0x0235 reactive).

Writes

Both FC06 and FC16 supported. Read-only registers reject with exception code 7. Settings 0xE0xx writable. Control 0xDF00 area (power/reset/restore) is dangerous on the live unit: documented, not fired.

SRNE ASP48120SH3 Safe Settings Write: design, registers, and commissioning (#53)

Purpose and scope

This page documents the SRNE ASP48120SH3 safe settings write feature (story #53): the ability to remotely change a bounded set of inverter parameters over Modbus, with every write guarded, verified by read back, and never retried. It records the design, the exact registers, the commissioning procedure, and, importantly, the current verification status so the work stays traceable and defendable.

Read this alongside the validated register map (page 126) and the mbpoll command reference (page 125).

What can and cannot be written

The feature can change eight settings only. Each is disabled by default (sentinel -1), bounded, applied only when the configured target differs from the live reading, and verified by a fresh read back.

#	Setting	Register (FC16)	Bounds	Units / raw multiplier
0	Discharge cutoff SoC	0xE00F	0 to 100	percent, x1
1	Stop charge current	0xE01C	0 to 100 A	amps, x10 (deci-amp raw)
2	Stop charge SoC	0xE01D	0 to 100	percent, x1
3	Low SoC alarm	0xE01E	0 to 100	percent, x1
4	Switch to line (grid) SoC	0xE01F	0 to 100	percent, x1
5	Switch to battery SoC	0xE020	0 to 100	percent, x1
6	AC charge current limit	0xE205	0 to 100 A	amps, x10

#	Setting	Register (FC16)	Bounds	Units / raw multiplier
7	Max total charge current limit	0xE20A	0 to 100 A	amps, x10

The current ceiling is MAX_BATTERY_CURRENT_A = 100 A, the continuous rating. The datasheet peak is 120 A for 3 seconds, so the continuous value is the correct, conservative cap.

It explicitly cannot: change charge or discharge time windows, command a specific charge or discharge power, change operating or source priority modes, start or stop the inverter, or reset or clear data.

Safety design

Every write passes through the same defended path (component SrneBatteryInverterImpl, helper SafeWriteHandler in the io.openems.edge.ess.srne bundle):

1. Default off. Nothing writes unless the operator sets controlEnabled true in the gateway config. Default is false.
2. Machine state gate. A write is only queued when the inverter reports a verified stable state. Register 0x0210 value 2 (on-grid, mains bypass) is the only verified state; every other state, including off-grid (3), transitioning, fault, and undefined, blocks all writes.
3. Bounds. Each setting is range checked against the table above. An out of range target is rejected before any Modbus write and marked failed (terminal).
4. One shot per activation. The write state machine is IDLE, then QUEUED, then AWAITING_READBACK, then VERIFIED or FAILED. A setting is queued at most once per component activation; VERIFIED and FAILED are terminal, so there is no re-write and no storm even though reconciliation now runs every cycle.
5. Read back verification. After the FC16 write executes, a fresh read of the same register must equal the target to reach VERIFIED. A mismatch goes to FAILED.
6. Bounded wait, no retry. The read back is allowed up to 30 Edge cycles; on timeout the state is FAILED. A failed or mismatched write is never retried automatically.
7. Thread safe. The handler methods are synchronized so the cycle thread (reconcile) and the Modbus worker thread (execute and read back) transition state atomically.

The aggregate SAFE_WRITE_STATE channel surfaces the highest severity handler state so an operator can watch commissioning progress.

The standalone topology fix (openems PR #23)

During #53 commissioning, gateway config v18 enabled control and targeted Low SoC alarm 15 to 16, but no FC16 write occurred and the read back stayed at 15. Root cause: gw-aaron runs a standalone Srne.Ess topology, so the generic ManagedSymmetricBatteryInverter run() path was never invoked, and reconciliation was hung off run(). The gateway was restored to the healthy v17 read only profile; the inverter setting did not change.

The fix moves the reconcile trigger to the Edge TOPIC_CYCLE_AFTER_PROCESS_IMAGE event, which fires every cycle in all topologies. It drives both the read back timeout advance and the queue, so the failure path works too. All guards above are unchanged; only the trigger moved.

Commissioning procedure (staged)

The commissioning is deliberately low risk and staged, with a pre staged rollback:

1. v17: read only baseline, control disabled.
2. v18: enable control, target exactly one setting (Low SoC alarm 15 to 16), leave the other seven at -1 (disabled). Verify a fresh read back of 16 and SAFE_WRITE_STATE VERIFIED, with stable telemetry and no other writes.
3. v19: restore Low SoC alarm 16 to 15, keep control enabled, all other targets at -1.
4. Rollback profile (microgrid PR #33): a break glass config that targets only Low SoC alarm 15, used if v18 changes the inverter but verification fails.

Gate: the v19 enable step (microgrid PR #34) must not be deployed until v18 has been deployed and verified with SAFE_WRITE_STATE VERIFIED on the real inverter.

Verification status (as of 2026-08-08)

Be precise here, because the live unit has not yet confirmed a write.

- Code reviewed at source level, including an adversarial second pass and mutation testing of the safety gates.
- Unit and integration tested in Java on JDK 21. Coverage includes: the write queues via the new event path in the standalone topology; control disabled never queues; a non verified machine state never queues (both mutation proven, that is, removing the guard makes the test fail with a real queued write); the one shot property, no re-queue across extra cycles; and, at unit level, the read back mismatch and timeout paths with no retry.
- Not yet verified on the live inverter. No setting has been successfully written and read back on the physical unit. The only live result so far is the v18 non write above, which exposed the topology bug. Live verification is pending: merge openems #23, build a new openems-edge image, then repeat the v18 15 to 16 commissioning gate on the real inverter.

- There is no Python live test harness for this. Writes are exercised through OpenEMS plus the gateway commissioning config and verified by Modbus read back, not by a standalone script.

Traceability

- Story: nfe-modbus-energy-logger issue #53.
- Code: openems PR #21 (guarded write infrastructure, merged), openems PR #23 (standalone topology fix plus gate tests, in review).
- Deployment: microgrid-gateway PR #32 (v18 commission, merged), #33 (rollback pre stage), #34 (v19 enable). Gated as above.
- Registers: SRNE ASP48120SH3 Modbus Register Map, validated (page 126). mbpoll command reference (page 125).

Live read test results (2026-08-08)

First live read of the eight settings on the pilot inverter (gw-aaron), via direct Modbus (mbpoll, FC03) with the OpenEMS edge and the nfe-monitor process stopped to free the RS485 bus. Read only, no writes. Accessed over Raspberry Pi Connect plus ZeroTier (nfetestpiaaron, 10.135.127.37).

Setting	Register	Raw	Decoded
Discharge cutoff SoC	0xE00F	5	5 percent
Stop charge current	0xE01C	30	3.0 A
Stop charge SoC	0xE01D	100	100 percent
Low SoC alarm	0xE01E	15	15 percent
Switch to line (grid) SoC	0xE01F	10	10 percent
Switch to battery SoC	0xE020	90	90 percent
AC charge current limit	0xE205	1200	120.0 A
Max total charge current limit	0xE20A	1000	100.0 A

Context registers: machine state 0x0210 = 3 (off-grid, inverter-powered), SoC 0x0100 = 92 percent.

Findings:

1. Low SoC alarm reads 15, matching the gateway config. This confirms the v18 15 to 16 commissioning write did not change the inverter (the standalone topology bug fixed in openems #23); the live value is still 15. This is the live evidence behind that finding.

2. Machine state is 3 (off-grid). The write feature only permits writes in the verified state (2, on-grid mains bypass), so a write attempted right now would be correctly blocked by the machine-state gate. A live write test must wait for the inverter to be on-grid.
3. Current scaling is x0.1 (deci-amp): AC charge limit 120.0 A, max charge 100.0 A, stop-charge current 3.0 A. SoC and percent settings are x1.
4. Bus collision confirmed live: both the OpenEMS edge and the nfe-monitor (F3) process were found mastering /dev/ttyUSB0 at the same time. The bus only freed after stopping both. F3 must not run a second Modbus master on this Pi.
5. The block read of 0xE001 to 0xE02D (count 45) returns Modbus exception 0x0A (gateway path unavailable) on this unit even with the bus free; read the settings individually or in small blocks.

Raw mbpoll output:

```
[57359]  5      (0xE00F discharge cutoff SoC)
[57372]  30     (0xE01C stop charge current)
[57373]  100    (0xE01D stop charge SoC)
[57374]  15     (0xE01E low SoC alarm)
[57375]  10     (0xE01F switch to line SoC)
[57376]  90     (0xE020 switch to battery SoC)
[57861]  1200   (0xE205 AC charge current limit)
[57866]  1000   (0xE20A max charge current limit)
[528]    3      (0x0210 machine state)
[256]    92     (0x0100 SoC)
```

Status: reads are now live-verified as above. The live WRITE test (Low SoC alarm 15 to 16, verify by fresh readback, restore to 15) is still pending, and requires both openems #23 deployed and the inverter in the verified on-grid state (it was off-grid at read time).

Commissioning results — live on gw-aaron (2026-08-10)

The guarded settings-write path was commissioned end to end on the live Aaron pilot inverter (SRNE ASP48120SH3), using the low-risk representative setting LowSocAlarm 15 -> 16 -> 15.

Sequence:

- OpenEMS PR #23 merged (writes run from the Edge cycle so the standalone Srne.Ess topology schedules the FC16). Merge 485c075b.
- Corrected edge image nfe-v0.2.2 built and pinned by digest sha256:3828fc838a67045bc2b88f4347e1424ca6a284dde0602dce52d9d4256dfc1d1f.
- Gateway PR #36 (v20 settings-commissioning) deployed. Reconciler wrote 15 -> 16.

Evidence (edge logs):

```
Queued one-shot settings write [LOW_SOC_ALARM] from [15] to [16]
Settings write [LOW_SOC_ALARM] state [VERIFIED] after readback [16]
```

- Confirmed by a bus-exclusive mbpoll read: register 0xE01E = 16.
- Restored 16 -> 15; verified by three consecutive stable direct reads = 15.
- Final resting config: gateway PR #37 (v21) = read-only (controlEnabled=false, all targets -1) on nfe-v0.2.2; supersedes v20 so a redeploy cannot re-arm. Edge healthy, backend connected, machine state 2 Running (mains/bypass).

Two operational lessons banked:

- The write gate only fires in a live-validated verified machine state (state 2 here). While the flaky grid is down the inverter sits in state 3 (off-grid, unverified) and the guard correctly defers the write; that is not a failure, just wait for the grid.
- To read the RS485 bus directly with mbpoll you must stop both openems-edge and the mbpoll container and wait ~5 s for the bus to settle; reading sooner returns garbage (a spurious 1200 was observed).

Process note: settings changes must go through git -> deploy-gateway.sh, never hand-edited on the box. A read-only resting config (v21) is the safe default between commissioning windows.

SRNE charge/discharge scheduling registers (self-decoded, #67)

Summary

We self-decoded the SRNE ASP48120SH3 settings registers (V1.7 protocol PDF + a live read-scan of the pilot inverter, firmware v925, 2026-08-09). The key result: **time-of-use charge/discharge scheduling is implemented and writable**, even though real-time active-power dispatch is not. This is the arbitrage lever, and it is not firmware-blocked.

Aaron's SRNE email (2026-08-03) reported that `0xE047-0xE04D` (timed charge/discharge MAX POWER) and `0xE400/E401/E402` (grid active/PF/reactive power) return Modbus "Illegal data address" on our firmware. That is confirmed. But those are a different feature (instantaneous power dispatch). The actual **time-window schedule registers `0xE026-0xE032` are RW and implemented.**

Charge / discharge schedule registers (RW, hour*256 + minute)

Register	Function	Live value	Decoded
0xE026	1-section charge start	0	00:00
0xE027	1-section charge stop	1536	06:00
0xE028 / 0xE029	2-section charge start / stop	0 / 0	unused
0xE02A / 0xE02B	3-section charge start / stop	0 / 0	unused
0xE02D / 0xE02E	1-section discharge start / stop	0 / 0	unused
0xE02F / 0xE030	2-section discharge start / stop	0 / 0	unused
0xE031 / 0xE032	3-section discharge start / stop	0 / 0	unused

Encoding: value = hour * 256 + minute. Max 5947 (23*256 + 59 = 23:59). So `1536 = 6*256 + 0 = 06:00`.

Current live config: charge window 1 = 00:00 to 06:00 (off-peak), no scheduled discharge. To do TOU arbitrage, set a discharge window over the peak (18:00 to 24:00 = start 4608, stop 6144-ish, i.e. 24:00 as 23:59 = 6143 or 0).

RTC / current time: `0xE034` (year+month) `0xE035` (day+hour) `0xE036` (minute+second) RW; read via `0x020C/020D`. The inverter's clock must be correct for the schedule to fire at the right wall-clock time - verify it.

Mode / priority registers (RW)

Register	Function	Live	Notes
0xE204	Output priority	3	0/1/2 documented (2 = hybrid, mains+PV, PV preferred); reads 3 on this unit
0xE20F	Charge priority	(def 2)	0-3
0xE216	Start charge time (hour)	1	01:00, "valid for some custom model"
0xE217	Start discharge time (hour)	12	12:00, "valid for some custom model"
0xE205	Mains charge current limit	80	x0.1 A? doc says A; range 0-100
0xE20A	Maximum charge current	80	0-150

Stopping a night charge: the timer is not enough (20 Nov 2026)

- **Disarming the charge timer (0xE02C = 0) does not stop a grid charge while charge priority 0xE20F = 2 (hybrid).** With E20F = 2 the unit grid-charges at any hour while it is on the line.
- Seen on Sezibwa gw-pi2, night of 19 to 20 Nov (v38h): the night stop latched at 85% about 01:17 EAT and E02C = 0 read back, but the battery kept charging at about 6.2 kW until 01:35 and ended at 99%.
- So the night stop must also move **E20F to 3 (PV only)** when it latches. That is openems #53.

- Whether E20F = 3 stops every grid charge is still a planned test (N1 in the proven behaviour register, page 235). The first night after #53 checks: E20F reads 3 after the latch, battery current falls to about 0 within a few minutes, SoC stays near the stop value, and E02C is re-armed after 01:45.

Full settings decode (0xE001-0xE039, from V1.7 + live)

Voltage/battery: E002 nominal capacity, E003 system voltage (48), E004 battery type, E005 over-voltage (15.0V), E006 limited-charge (14.4V), E007 equalize, E008 boost/overcharge, E00A boost-return (13.2V-ish), E00B over-discharge-return, E00C under-voltage warn, E00D over-discharge (load cut), E00E limited-discharge, E00F discharge-cutoff SoC (5%), E01B mains-switching voltage, E022 inverter-switching voltage. Times: E010 over-discharge delay (s), E011 equalize time, E012 boost time (min), E023 equalize timeout. The 8 settings the OpenEMS write feature (#53) uses are a subset: E00F, E01C, E01D, E01E, E01F, E020, E205, E20A.

NOT implemented on this firmware (Illegal data address, confirmed live)

`0xE047-0xE049` timed discharge max power, `0xE04A-0xE04C` timed charge max power, `0xE04D` timed charge source, `0xE400` grid active-power set, `0xE401` PF, `0xE402` reactive, `0xE42A` battery grid participation, `0xE42C` zero-export. So OpenEMS-style instantaneous power dispatch is not available; TOU scheduling is the achievable control.

What this unlocks for NFE

TOU arbitrage via scheduling: set the charge window to off-peak (already 00:00-06:00) and add a discharge window over the peak tariff, using `0xE026-0xE032`. All RW, all validated readable live. Next: a careful readback-verified live write test of one discharge window (write, verify, restore), then integrate scheduled writes (guarded, like #53) so NFE can shift the battery by tariff without a site visit.

Method / traceability

Live read-scan via mbpoll (edge + nfe-monitor stopped to free the RS485 bus), firmware v925, 2026-08-09. Decoded against `SRNE_Hybrid_Inverter_MODBUS_protocol_V1.7.pdf` (repo: NFE/CUSTOM_CODE/srne-products). Related: register map (page 126), mbpoll reference (page

125), safe settings write (page 128, #53). Issue logger #67.

SRNE phase balancing: big consumers on one phase trip the inverter on battery

Problem

When the site runs on battery (grid outage, typically evening or night), switching on a large single-phase load such as a heater or fan trips the inverter off until the load is removed. This causes an outage for the whole site.

Evidence (2026-08-11)

Per-phase active power read directly off the SRNE ASP48120SH3 (bus-quiet mbpoll on gw-aaron):

Phase	On mains (state 2)	On battery (state 3)
L1	102 W	145 W
L2	230 W	327 W
L3	212 W	291 W
Total	544 W	763 W

L1 consistently carries about half of L2 and L3. Total load is small (under 1 kW against the inverter rating of 12 kW / 24 kVA, roughly 4 kW / 8 kVA per phase), so the trip is not a total-capacity overload. It points to the big consumers being concentrated on one phase: a large load landing on an already heavier phase, plus its switch-on inrush, can overload that single phase or trip protection while the other phases sit nearly idle.

Action

1. Rebalance customer phase assignments toward an even split. Prefer balancing by measured load rather than by simple 4-3-3 count.
2. Identify the big consumers and move them off the loaded phases (L2 and L3) onto the lighter phase (L1).
3. Re-measure per-phase load after rebalancing to confirm the spread has narrowed.

Use the backup-phase-dashboard phase-balance analysis (reads the customer meter CSVs) to plan the reassignment.

Also check (a separate, compounding cause)

Protection and power-save settings may make the trip worse on battery. Read (read-only) the power-save level E207, eco mode E20C, overload auto-restart E20D, and overload bypass E212. Off-grid there is no mains to bypass to, so an overload just shuts the inverter down.

Tracking

GitHub issue: Nearly-Free-Energy/nfe-modbus-energy-logger #70

OpenEMS Backend InfluxDB Data Model and Edge-ID Collision

> [!warning] Verification status
> **The storage-key collision vulnerability is confirmed in source code, and the repository contains a real configured collision.** However, the vulnerability alone does not prove that Hillary data contaminated the current Sezibwa Pi2 history. Historical causation must be established from production InfluxDB data, connection periods, and gateway logs before any records are rewritten or deleted.
> **Update, 2 November 2026:** no two edges on the production Backend share a tag any more, but history already stored under tag `1` is unchanged. See Status on 2 November 2026.

Executive conclusion

- The configured backend Timedata.InfluxDB component writes edge data into a shared InfluxDB measurement, currently `data`.
- It does not use the complete edge ID as the storage discriminator. It parses the trailing number and stores that number in the OEM edge tag, normally `edge=<number>`.
- Two edge IDs with the same parsed number are indistinguishable to this timedata store and its history queries.
- The repository contains `hillary-test-pi-1` and `aaron-pi-1`; both map to `edge=1`. This is a genuine collision risk.
- The current Sezibwa Pi2 edge definition is `sezibwa-rentals-gw-pi2`, which maps to `edge=2` and does not collide with Hillary.
- `nfetestpi1` is a hostname/deployment user in this repository, not the backend edge ID, and should not be used as evidence of an edge-tag collision.

Confirmed backend data model

The relevant implementation is:

`io.openems.backend.timedata.influx/src/io/openems/backend/timedata/influx/TimedataInfluxDb.java`

For every accepted data timestamp, the backend constructs an InfluxDB point using:

```
Point.measurement(this.config.measurement())
    .addTag(this.oem.getInfluxdbTag(), String.valueOf(influxEdgeId))
    .time(timestamp, WritePrecision.MS);
```

The deployment configuration sets `measurement="data"`. “Measurement” is the correct InfluxDB term; it is analogous to, but not identical with, a relational database table. All edges handled by this configured timedata provider share that measurement unless another Timedata.InfluxDB instance or measurement is configured.

Channel addresses such as `meter0/ActivePower` and `_sum/EssSoc` are written as fields. History queries parse the requested edge ID through the same numeric function and filter using the resulting numeric tag.

Confirmed edge-ID parser behavior

`InfluxConnector.parseNumberFromName(edgeId)` uses:

```
\D++(\d++)$
```

This extracts the trailing numeric run rather than preserving the full edge ID:

Configured value	Role in this repository	Influx tag value	Finding
`hillary-test-pi-1`	Hillary test edge ID	`1`	Collides with `aaron-pi-1`
`aaron-pi-1`	Aaron edge ID	`1`	Collides with `hillary-test-pi-1`
`sezibwa-rentals-gw-pi2`	Current Sezibwa Pi2 edge definition	`2`	No suffix collision with Hillary
`nfetestpi1`	Hostname/deployment user	Not applicable as currently configured	Not the backend edge ID

If an edge ID contains no parseable trailing number, the timedata write is not stored. This is **not completely silent**: `TimedataInfluxDb.writeData()` logs a warning stating that it could not parse the numeric Influx Edge-ID, then returns without writing the notification.

What a collision does

InfluxDB identifies a point by measurement, tag set, and timestamp. Once two gateways share the same numeric edge tag:

- History queries cannot distinguish which gateway supplied a value.
- Different channel fields can appear together under the same edge series.
- If both gateways write the same field at the same timestamp, later writes can replace or combine with the existing point according to InfluxDB point-update behavior.
- Backend caches keyed by the parsed integer, such as timestamped-channel tracking, can also conflate the two edges.

The resulting history may therefore be mixed, overwritten, or both; “merge” should not be interpreted as guaranteed preservation of every value from both sources.

What is and is not proven about the incident

Proven: the implementation is collision-prone, and two repository edge configurations currently resolve to `edge=1`.

Not yet proven: that this caused the reported Hillary-to-Sezibwa contamination. If the affected Sezibwa history was requested under `aaron-pi-1`, the collision is a strong causal candidate. If it was requested under `sezibwa-rentals-gw-pi2`, the `1`-versus-`2` mapping means this specific mechanism does not explain it.

Confirm the affected edge ID and examine production `edge=1` data before finalizing the root-cause statement.

Immediate containment

1. Inventory every edge registered with or connecting to the production backend.
2. Compute the parsed numeric ID for each and identify duplicates before making changes.
3. Disconnect colliding test rigs from production or assign a globally unique temporary numeric suffix.
4. Back up InfluxDB and the backend/metadata configuration before renaming an edge.
5. Update all coupled references together: metadata, API keys, gateway configuration, monitoring, dashboards, automation, and operational documentation.
6. After each change, verify websocket connectivity, live channels, new timedata writes, history queries, and monitoring alerts.

> [!note] Unique numeric suffixes are containment, not the permanent design
> A naming convention reduces immediate risk but remains brittle and easy to violate as the fleet grows.

Status on 2 November 2026

The production Backend lists these edges (checked with the nfe-openems Claude skill, see Using OpenEMS with Claude):

Edge ID	Influx tag value	Status on 2 Nov 2026
`aaron-pi-1`	`1`	Offline, no message recorded
`edge0`	`0`	Offline, no message recorded
`hillary-test-pi-99`	`99`	Online
`sezibwa-rentals-gw-pi2`	`2`	Online

- **No current collision.** Hillary's test Pi is now registered as `hillary-test-pi-99`, and `hillary-test-pi-1` is no longer registered, so no two current edges share a tag. This is the containment above (a unique numeric suffix), not the permanent correction.
- **Stored history is unchanged.** Tag `1` still holds whatever `hillary-test-pi-1` and `aaron-pi-1` wrote while both used it. Treat older `aaron-pi-1` history as possibly mixed until the audit below is done.
- **The parser is unchanged.** The deployed Backend (OpenEMS 2026.10, NFE fork) still files history with `InfluxConnector.parseNumberFromName` and `\D++(\d++)$`, so the permanent correction below is still needed.
- **New collisions are flagged.** The nfe-openems skill warns whenever two current edges share a tag, so a new one shows up in Claude's answers about history and energy.

Permanent backend correction

1. Use the complete immutable edge ID as the InfluxDB discriminator, or resolve the edge to another stable, globally unique identifier maintained by backend metadata.
2. Validate uniqueness at edge registration and backend startup. Reject duplicate storage identifiers instead of accepting ambiguous writes.
3. Fail visibly for invalid identifiers, with actionable logs and operational alerts.
4. Add automated tests covering identical suffixes, multi-digit suffixes, IDs without numbers, reconnects, resends, and history-query isolation.
5. Review other structures keyed by the parsed integer, including timestamped-channel tracking and aggregated timedata, so the fix covers more than the point tag.

Migration requirements

Changing the tag key or value can make existing history unreachable to current queries. The implementation therefore needs an explicit migration plan, such as:

- dual-write the legacy numeric tag and the new full-ID tag for a controlled period;
- support query fallback across both schemas during migration;
- backfill only records that can be attributed confidently;
- retain an immutable backup and a tested rollback path; and
- define a cutover date after which the legacy numeric identity is no longer accepted.

Historical-data audit before cleanup

1. Identify exactly which logical Sezibwa edge displayed Hillary channels.
2. Determine when `hillary-test-pi-1` and `aaron-pi-1` were simultaneously connected to the same production backend and bucket.
3. Inspect `edge=1` by time range, unique channel inventory, device serial numbers, meter topology, and gateway connection logs.
4. Classify records as confidently Hillary, confidently Aaron/Sezibwa, or ambiguous.
5. Quarantine ambiguous records. Do not delete or rewrite them merely from channel-name assumptions.
6. Document any irrecoverable overlap where the same field and timestamp may have been overwritten.

Approval boundary

Approved direction: immediate containment, a reviewable backend fix, collision tests, and a documented migration design.

Requires separate approval: irreversible deletion, reassignment, or rewriting of production historical data. That work must follow the backup and attribution audit above.