Automation Manual

ENERGIS
10IN Managed PDU

Revision: 1.1.0

DMT

Revision History

A revision is a new edition of the document and affects all sections of this document.

Version	Date	Responsible	Modification
1.0.0	12.12.2025	David Sipos	Initial creation of the document
1.1.0	13.01.2026	David Sipos	Added Software version 1.1.0 changes

Content

DMT

1. Overview

The ENERGIS managed PDU is a rack-mount power distribution unit that allows you to switch eight AC outlets individually and monitor voltage, current, power and uptime for each channel. The unit is built around a RP2040 microcontroller running FreeRTOS and communicates over Ethernet and serial. All relays are electromechanical and each channel is designed to handle 16A. HLW8032 power-metering chips measure the AC lines, individually per channel.

The PDU can be controlled locally via the front panel, remotely via a built-in HTTP server and SNMP agent, or through a USB serial console. This manual explains how to connect the unit, configure it and use these interfaces to integrate the PDU into any automation system.

1.1 Installation and Setup

1.1.1 Electrical and Network Connections

1. **Mounting and power** – Install the PDU into a 10-inch equipment rack. Connect the supplied AC power cord to the mains and attach your load devices to the eight IEC outlets. Each relay is rated 16A. The PDU inlet is 16A max, so the sum of all channels must not exceed 16A
2. **Networking** – Plug an Ethernet cable into the RJ-45 jack. By default, the PDU is programmed to use static IP Address, which can be found on the bottom of the device. The network settings can be read and changed via the web interface or the serial console (see below).
3. **USB/serial** – The PDU exposes its console over USB-CDC on the USB-C port. The serial interface operates at **115 200 baud, 8-N-1**. Connect the USB-C port to a host computer and open a terminal program (PuTTY, screen, minicom, etc.); the PDU enumerates as a CDC device without drivers. Note: *The internal HLW8032 power meter uses UART0 at 4 800 baud and it is NOT exposed externally.*

1.1.2 First Boot

On first boot the PDU performs a self-test, initializes the display and relays and loads configuration from its internal EEPROM. The internal power monitor performs a self-check; this makes the Power LED turns on. The eight outlet LEDs will play a pattern and the network LED will come on once a link is detected.

2. Local Operation

2.1 Front Panel Buttons and LEDs

When the device is powered, any interaction with any buttons (Left, Set, Right) will start a 10s selection window, in which the currently selected channel is shown by a blinking orange light. After 10 seconds of inactivity the selection times out and no channel is highlighted.

Each channel has two indicators: a green LED showing whether the channel is on or off. An orange LED showing which channel is currently selected. Within the selection window, the user is able to turn on and off any channel.

Status LEDs:

- **PWR** LED indicates that the PDU is energized, and the internal power is within the expected range.
- **ETH** LED lights when a link is established on the Ethernet port.
- **ERR** LED lights if an internal error is detected; clear it with a long press of the **Set** button.

2.2 Button functions:

- **Left** / **Right**: Within the selection window, use these buttons to move the selection highlight across the eight channel status LEDs. The highlight wraps around after channel 8.
- **Set:**
  - Outside the selection window, a short press on any button only opens the 10 s window (no step, no toggle).
  - Inside the selection window, a short-press toggles the currently selected channel on/off.
  - Anytime, a long press (>1 s) clears the fault LED.
- **PWR**:
  - Long press (>2s) when the device is powered initiates the low power sleep mode. The PWR LED blinks every second. Short press in active mode has no effect.
  - Short press, when the device is in sleep mode wakes up the device and puts it back in active mode. Long press in sleep mode has no effect.

3. Command-Line Interface - Serial

A comprehensive serial console is available over the USB-C port (USB-CDC). After connecting with a terminal program at 115200 baud basic system health information can be seen, emitted every 60 seconds. Type `help` to list commands.

3.1 Command list

Command	Description	Example
GENERAL COMMANDS		
HELP	Show available commands and syntax	
SYSINFO	Show system information	
GET_TEMP	Show MCU temperature	
REBOOT	Reboot the PDU. Changes to network settings take effect after reboot.	
BOOTSEL	Put the MCU into boot mode	
CLR_ERR	Clears all errors	
RFS	Restore Factory Settings	
OUTPUT CONTROL AND MEASUREMENT		
SET_CH <ch> <STATE>	Set channel (1–8) to (0|1|ON|OFF|ALL)	SET_CH 1 1
GET_CH <ch>	Read current state of channel <ch> (1-8|ALL)	GET_CH 3
OC_STATUS	Show overcurrent protection status	
OC_RESET	Manually clear overcurrent lockout	
READ_HLW8032	Read power data for all channels	
READ_HLW8032 <ch>	Read power data for (1-8)	READ_HLW8032 2
CALIBRATE <ch> <V> <I>	Start single (1-8) calibration on given voltage and current	CALIBRATE 2 0 0
AUTO_CAL_ZERO	Zero-calibrate all channels.	
AUTO_CAL_V <voltage>	Voltage-calibrate all channels.	AUTO_CAL_V 230
AUTO_CAL_I <current> <ch>	Current-calibrate single channel (1–8). Requires known load	AUTO_CAL_I 0.5 2
SHOW_CALIB <ch>	Show calibration data (1-8|ALL)	SHOW_CALIB 5
NETWORK SETTINGS		
NETINFO	Display the current IP, subnet mask, gateway and DNS addresses.	
SET_IP <ip>	Set a static IP address. Requires reboot.	SET_IP 10.10.0.67
SET_SN <mask>	Set the subnet mask.	SET_SN 255.255.255.0
SET_GW <gw>	Set the default gateway.	SET_GW 192.168.1.1
SET_DNS <dns>	Set the DNS server.	SET_DNS 8.8.8.8
CONFIG_NETWORK <ipsngw$dns>	Configure all network settings.	CONFIG_NETWORK 254.176.0.2 $255.255.255.0$254.176.0.1 $1.1.1.1

4. HTTP API

The embedded HTTP/1.1 server listens on **port 80** and serves both JSON endpoints under /api/... and a few HTML pages for browsers. All responses include:

- Connection: close
- Cache-Control: no-cache
- Access-Control-Allow-Origin: * (CORS enabled for any origin)

Status codes (summary)

- GET /api/status → 200 OK (JSON)
- GET /api/settings → 200 OK (JSON)
- POST /api/control → 200 OK with text body OK\n; 400 Bad Request if body missing; 503 Service Unavailable when overcurrent lockout blocks switching
- POST /api/settings → 204 No Content (changes are applied immediately; see notes)
- Preset & automation endpoints:
    - GET /api/config-presets → 200 OK (JSON; may include Retry-After: 1 while presets initialize)
    - POST /api/config-presets → 200 OK on success; 400 on malformed/missing fields; 500 on save error
    - POST /api/apply-config → 200 OK on success; 400 invalid/missing id; 503 overcurrent lockout; 500 apply failure
    - POST /api/startup-config → 200 OK on success; 400 invalid/missing parameters; 500 failure

> ***Notes***
> - JSON responses use Content-Type: application/json. The success body for POST /api/control is **text/plain** (OK\n).
> - Static HTML pages may be served **gzipped** with Content-Encoding: gzip

4.1 GET /api/status

Returns live telemetry and system state.

Response JSON shape:

- **channels**: array (8) of per-channel objects:
  ```
  {
   "voltage": float,
   "current": float,
   "uptime": uint,
   "power": float,
   "state": bool,
   "label": string
  }
  ```
- **internalTemperature**: float - device die temperature converted to the user's unit
- **temperatureUnit**: "°C" | "°F" | "K"
- **systemStatus**: "OK" | "WARNING" | "LOCKOUT" (derived from overcurrent sta te)
- **overcurrent**: object:
  ```
  {
      "state": "NORMAL"|"WARNING"|"CRITICAL"|"LOCKOUT"|"UNKNOWN",
      "total_current_a": float,
      "limit_a": float,
      "warning_threshold_a": float,
      "critical_threshold_a": float,
      "recovery_threshold_a": float,
      "switching_allowed": true|false,
      "trip_count": integer,
      "region": "EU"|"US"
    }
  ```

Example:

```
curl.exe -s http://<pdu-ip>/api/status
```

Response:

```
{
  "channels": [
    { "voltage": 229.987, "current": 0.120, "uptime": 12, "power": 27.598,
      "state": true,  "label": "CH1" },
    { "voltage": 229.954, "current": 0.000, "uptime": 0,  "power": 0.000,
      "state": false, "label": "" }
    /* ... 6 more ... */
  ],
  "internalTemperature": 26.970,
  "temperatureUnit": "°C",
  "systemStatus": "OK",
  "overcurrent": {
    "state":"NORMAL",
    "total_current_a":0.12,
    "limit_a":10.0,
    "warning_threshold_a":9.00,
    "critical_threshold_a":9.50,
    "recovery_threshold_a":8.50,
    "switching_allowed":true,
    "trip_count":0,
    "region":"EU"
  }
}
```

4.2 POST /api/control

Controls relay states **and channel labels** via form-urlencoded data.

Form fields

- **channelN** (N = 1..8):
  - "on" → turn channel **ON**
  - anything else or omitted → **OFF**
    (Omitting a channel means it is forced OFF.)
- labelN (N = 1..8): optional; set channel label (max **25** chars); empty value clears it.

Overcurrent guard

If any channel would transition from OFF→ON while overcurrent lockout is active, the request is **rejected** with **503 Service Unavailable** and a JSON error body.

Examples

Turn channels 1 and 3 ON (others forced OFF)

```
curl.exe -s -o $null -w "%{http_code}`n" -X POST "http://<pdu-ip>/api/control"
--data "channel1=on&channel3=on"
```

Turn channel 3 OFF (others remain OFF by omission):

```
curl.exe -s -o /dev/null -w "%{http_code}\n" -X POST "http://<pdu-ip>/api/
control” --data "channel3="
```

Set label for channel 1 and turn it ON:

```
curl.exe -s -o /dev/null -w "%{http_code}\n" -X POST "http://<pdu-ip>/api/
control" --data "channel1=on&label1=Router PSU"
```

Responses

- Success: 200 OK with body OK\n (text/plain)
- Missing body: 400 Bad Request with JSON error
- Overcurrent lockout: 503 Service Unavailable with JSON error

4.3 GET /api/settings

Returns the stored network configuration and user preferences plus device identity and limits).

Response JSON fields

```
{
  "ip": "a.b.c.d",
  "gateway": "a.b.c.d",
  "subnet": "a.b.c.d",
  "dns": "a.b.c.d",
  "mac": "AA:BB:CC:DD:EE:FF",
  "device_name": "ENERGIS-<ver>",
  "location": "Location",
  "temp_unit": "celsius" | "fahrenheit" | "kelvin",
  "temperature": <number in selected unit>,
  "serial_number": "<string>",
  "firmware_version": "<string>",
  "hardware_version": "<string>",
  "device_region": "EU" | "US" | "UNKNOWN",
  "current_limit": "<n.n A>",
  "warning_limit": "<n.nn A>",
  "critical_limit": "<n.nn A>"
}
```

```
curl.exe -s "http://<pdu-ip>/api/settings"
```

Example response:

```
{
  "ip":"192.168.0.12",
  "gateway":"192.168.0.1",
  "subnet":"255.255.255.0",
  "dns":"8.8.8.8",
  "mac":"AA:BB:CC:DD:EE:FF",
  "device_name":"ENERGIS-1.1.0",
  "location":"Location",
  "temp_unit":"celsius",
  "temperature":33.35,
  "serial_number":"<serial>",
  "firmware_version":"<fw>",
  "hardware_version":"<hw>",
  "device_region":"EU",
  "current_limit":"10.0 A",
  "warning_limit":"9.00 A",
  "critical_limit":"9.50 A"
}
```

4.4 POST /api/settings

Updates network parameters and user preferences. Fields include:

- ip, subnet, gateway, dns
- device_name, location
- temp_unit (celsius, fahrenheit, or kelvin)

Changes are saved to EEPROM; the device **responds 204 No Content and then reboots.**

```
curl.exe -s -o $null -w "%{http_code}`n" -X POST "http://<pdu-ip>/api/settings" ^
--data "ip=192.168.0.12&gateway=192.168.0.1&subnet=255.255.255.0&dns=8.8.8.8&device_name
=ENERGIS&location=Rack1&temp_unit=celsius"
```

4.5 Preset Management

The firmware supports saving, loading, and managing up to 5 relay configuration presets. These presets allow you to quickly recall and apply specific ON/OFF states to all 8 channels and optionally set one preset to be automatically applied at startup.

4.5.1 GET /api/config-presets

Purpose:
Retrieve the list of all saved relay presets and the currently selected startup preset (if any).

Response JSON:

```
{
  "presets": [
    { "id": 0, "name": "All Off", "mask": 0 },
    { "id": 1, "name": "Servers On", "mask": 3 }
    // ... up to 5 total
  ],
  "startup": null // or 0..4 if a startup preset is set
}
```

Each preset object contains:

- id: Preset slot number (0–4)
- name: User-defined name (max 25 characters)
- mask: 8-bit value, each bit representing ON/OFF for channels 1–8 (bit 0 = channel 1, bit 7 = channel 8)

During initial boot or if presets are not yet loaded, the response ma be:

```
{
  "presets": [],
  "startup": null,
  "pending": true
}
```

In this case, the HTTP response may also include a Retry-After: 1 header, indicating the client should retry after 1 second.

4.5.2 POST /api/config-presets

Purpose:
Create (save) a new preset or delete an existing one.

To save a preset:

- Send a form-encoded POST body with:
  - action=save
  - id=N (0–4)
  - name=<your name> (max 25 chars)
  - mask=<0..255> (bitmask for ON/OFF states)
- The field relay_mask is also accepted as an alias for mask.

Example (save a preset):

```
curl.exe -s -X POST "http://<pdu-ip>/api/config-presets" --data "action=
save&id=2&name=Night Mode&mask=1"
```

To delete a preset:

- Send:
  - action=delete
  - id=N (0–4)

Example (delete a preset):

```
curl.exe -s -X POST "http://<pdu-ip>/api/config-presets" --data "action=
delete&id=2"
```

Notes:

- If the name is longer than 25 characters, the request is rejected.
- If required fields are missing or invalid, the server returns 400 Bad Request.
- On EEPROM write failure, the server returns 500 Internal Server Error.
- On success, the response is 200 OK with {"success":true}.

4.5.3 POST /api/apply-config

Purpose:
Immediately apply a saved preset to the relay outputs.

How to use:

- Send a form-encoded POST body with:
  - id=N (0–4)

Example:

```
curl.exe -s -X POST "http://<pdu-ip>/api/apply-config" --data "id=1"
```

Behaviour:

- The relay states are set according to the preset's mask.
- If overcurrent lockout is active, the request is rejected with 503 Service Unavailable and a JSON error.
- If the id is missing or invalid, the server returns 400 Bad Request.
- On success, the response is 200 OK with {"success":true}.

4.5.4 POST /api/startup-config

Purpose:
Set or clear which preset (if any) should be automatically applied at device startup.

To set a startup preset:

- Send:
  - id=N (0–4)

Example:

```
curl.exe -s -X POST "http://<pdu-ip>/api/startup-config" --data "id=1"
```

To clear the startup preset:

- Send:
  - action=clear

Example:

```
curl.exe -s -X POST "http://<pdu-ip>/api/startup-config" --data "action=
clear"
```

Behaviour:

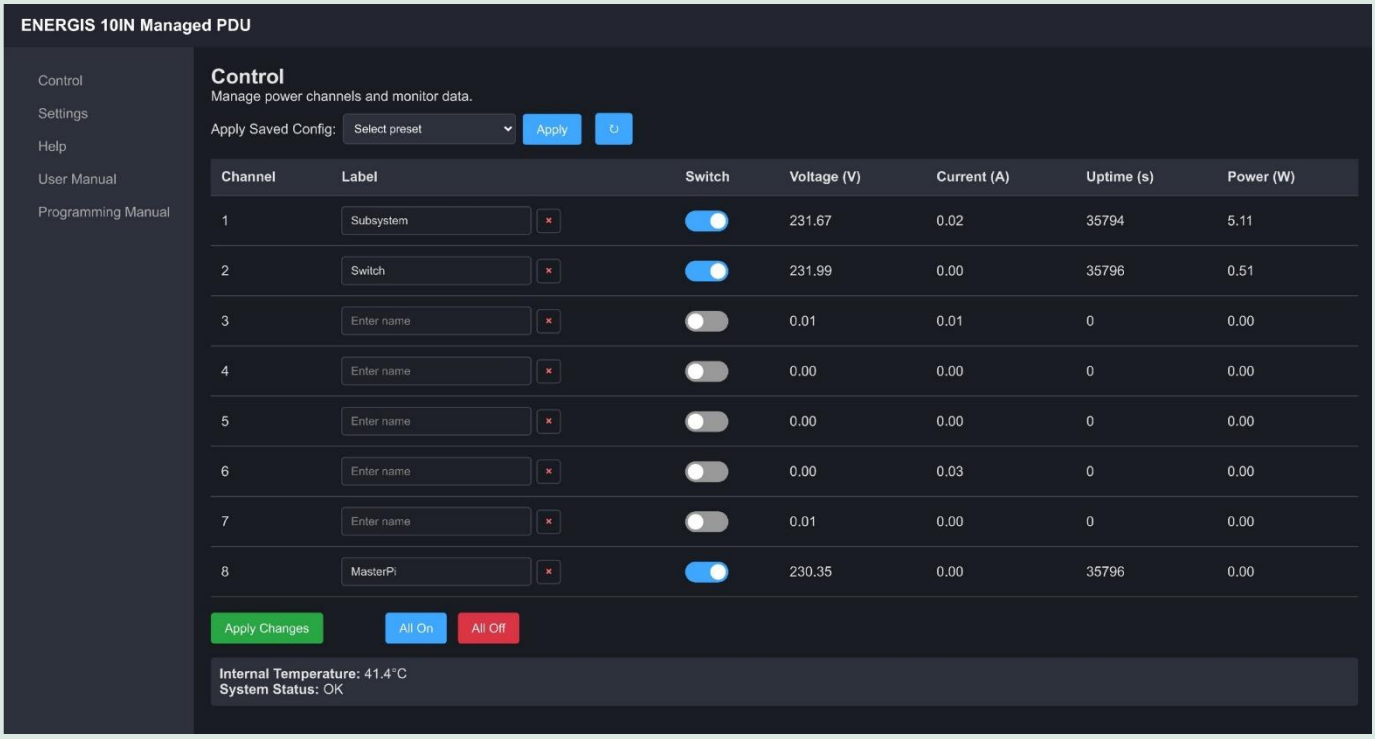
- Only one startup preset can be set at a time.
- If the request is successful, the response is 200 OK with {"success":true}.
- If parameters are missing or invalid, the server returns 400 Bad Request.
- On EEPROM write failure, the server returns 500 Internal Server Error.

Summary Table

Endpoint	Purpose	Required Fields	Success Response	Error Codes
GET /api/config-presets	List all presets & startup	–	200 OK (JSON)	–
POST /api/config-presets	Save or delete a preset	action, id, (name, mask)	200 OK (JSON)	400, 500
POST /api/apply-config	Apply a preset immediately	id	200 OK (JSON)	400, 503, 500
POST /api/startup-config	Set/clear startup preset	id or action=clear	200 OK (JSON)	400, 500

5. HTML pages

5.1 GET /control.html

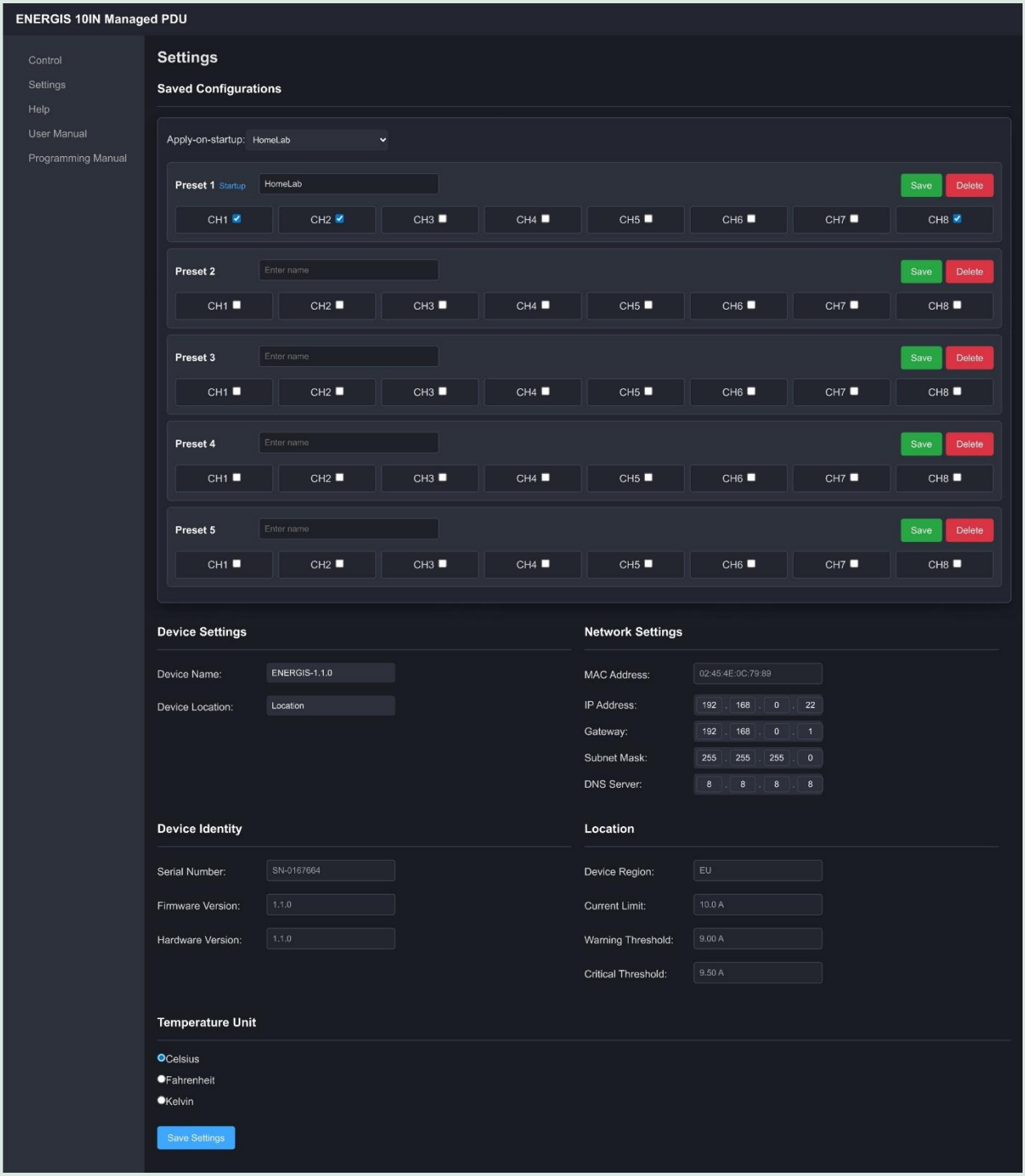


This is the main control UI.

- It loads in a browser,
- displays the 8 channels,
- and uses /api/status (for live data) and /api/control (for switching/ labels) in the background via JavaScript.

The page is served as HTML (gzipped), with Content-Type: text/html, Cache-Control: no-cache, and the connection is closed after the response.

5.2 GET /settings.html



Serves an HTML settings page for use in a browser. It displays current network values and allows updating them via a form. Posts to /api/settings. Served as gzipped HTML.

5.3 GET /help.html, /user_manual.html, /automation_manual.html

These endpoints serve small HTML pages.

- **/user_manual.html** and **/automation_manual.html**: simple stubs that link to the PDFs hosted on GitHub Pages:
  - User Manual
    https://dvidmakesthings.github.io/HW_10-In-Rack_PDU/Manuals/User_Manual.pdf
  - Programming & Interfacing Manual
    https://dvidmakesthings.github.io/HW_10-In-Rack_PDU/Manuals/Automation_Manual.pdf

  Internet access is **required** to load those PDFs.
- **/help.html**: a fast help page that is served fully from the device. It does not depend on internet access and is meant for quick reference and basic troubleshooting when you're offline.

HTTP behavior: All three return 200 OK, Content-Type: text/html, Cache-Control: no-cache, and the server closes the connection after sending the body.

What's on /help.html? (fast help, offline-safe)

- Front panel cheatsheet
- Network quick checks
- Web API short guide
- SNMP sanity
- Offline troubleshooting

Why keep Help?

- It gives operators something usable when the site has no internet.
- It points to the full PDFs when WAN is available.
- It reduces support by solving the common "why doesn't it switch / where's my IP / why are values zero" cases directly on the box.

6. OpenMetrics / Prometheus endpoint

6.1 Overview

The ENERGIS PDU exposes a `/metrics` HTTP endpoint that provides runtime telemetry in OpenMetrics/Prometheus text format (version 0.0.4). This plugs directly into Prometheus, Grafana, and similar tools.

6.2 Configuration - Accessing the Endpoint

```
curl.exe http://<pdu-ip-address>/metrics
```

Example response

```
# HELP energis_up 1 if the metrics handler is healthy.
# TYPE energis_up gauge energis_up 1
# HELP energis_build_info Build and device identifiers.
# TYPE energis_build_info gauge energis_build_info{version="1.1.0",se
rial="SN-0167663"} 1
# HELP energis_uptime_seconds_total System uptime in seconds.
# TYPE energis_uptime_seconds_total counter energis_uptime_seconds_to
tal 27 ...
# HELP energis_channel_state Relay state (1=ON, 0=OFF).
# TYPE energis_channel_state gauge energis_channel_state{ch="1"} 1
# HELP energis_channel_telemetry_valid 1 if cached telemetry for chan
nel is fresh.
# TYPE energis_channel_telemetry_valid gauge energis_channel_telemetr
y_valid{ch="1"} 1
# HELP energis_channel_voltage_volts Channel voltage.
# TYPE energis_channel_voltage_volts gauge energis_channel_voltage_vo
lts{ch="1"} 0.000
# HELP energis_channel_current_amps Channel current.
# TYPE energis_channel_current_amps gauge energis_channel_current_amp
s{ch="1"} 0.000
# HELP energis_channel_power_watts Active power per channel.
# TYPE energis_channel_power_watts gauge energis_channel_power_watts
{ch="1"} 0.000
# HELP energis_channel_energy_watt_hours_total Accumulated energy per
 channel.
# TYPE energis_channel_energy_watt_hours_total counter energis_channe
l_energy_watt_hours_total{ch="1"} 0.000
```

HTTP Response:

- 200 OK on success, with Content-Type: text/plain; version=0.0.4; charset=utf-8
- 503 Service Unavailable if the output would overflow the static buffer (encourages retry)
- 404 Not Found if the metrics feature is disabled at compile time

6.3 Available Metrics

6.3.1 System

energis_up

Health indicator (gauge; 1 if handler responds)

energis_build_info{version,serial}

Build and device info (gauge; always 1)

energis_uptime_seconds_total

Uptime since boot in seconds (counter)

energis_internal_temperature_celsius

Internal MCU temperature, calibrated (gauge)

energis_temp_calibrated

1 if temperature calibration is applied (gauge)

energis_temp_calibration_mode

Calibration mode (0=none, 1=1pt, 2=2pt) (gauge)

energis_vusb_volts

USB rail voltage (gauge)

energis_vsupply_volts

12 V rail voltage (gauge)

energis_http_requests_total

Total HTTP requests served (counter)

6.3.2 Per-Channel

energis_channel_state{ch}

Relay state: 1=ON, 0=OFF (gauge)

energis_channel_telemetry_valid{ch}

1 if cached telemetry for channel is fresh (gauge)

energis_channel_voltage_volts{ch}

Channel voltage [V] (gauge)

energis_channel_current_amps{ch}

Channel current [A] (gauge)

energis_channel_power_watts{ch}

Active power [W] (gauge)

energis_channel_energy_watt_hours_total{ch}

Accumulated energy in watt-hours (counter)

Values are served from cached snapshots owned by MeterTask. No direct sensor I/O in the handler. Temperature uses the active calibration. If no calibration is stored, defaults are used.

6.3.3 Overcurrent protection

energis_ocp_state {state}

Overcurrent protection state (0=NORMAL,1=WARNING,2=CRITICAL,3=LOCKOUT)

energis_ocp_switching_allowed {state}

Switching allowed (1=yes,0=no)

energis_ocp_total_current_amps {value}

Total measured current value in Amps (A)

energis_ocp_limit_amps {value}

Configured current limit

- EU Version (IEC/ENEC): 10.0A maximum
- US Version (UL/CSA): 15.0A maximum

energis_ocp_warning_threshold_amps {value}

Warning threshold, Limit - 1A

energis_ocp_critical_threshold_amps {value}

Critical threshold Limit - 0.5A

energis_ocp_recovery_threshold_amps {value}

Recovery threshold Limit – 1.5A

energis_ocp_trip_count_total {cnt}

Overcurrent trips since boot

energis_ocp_last_trip_ms {t}

Timestamp of last trip (ms since boot)

6.4 Prometheus Integration

Add to `prometheus.yml`:

```
scrape_configs:
  - job_name: 'energis-pdu'
    scrape_interval: 15s
    scrape_timeout: 10s
    metrics_path: /metrics
    static_configs:
      - targets: ['192.168.0.22:80']
        labels:
          location: 'rack-1'
          device: 'pdu-main'
```

Example Alerts

```
groups:
- name: energis
  rules:
  - alert: EnergisHighTemp
    expr: energis_internal_temperature_celsius > 70
    for: 2m
    labels: {severity: warning}
    annotations:
      summary: "ENERGIS MCU hot >70C"

  - alert: EnergisVsupplyLow
    expr: energis_vsupply_volts < 10.8
    for: 1m
    labels: {severity: critical}
    annotations:
      summary: "ENERGIS 12V rail low"

  - alert: EnergisNoTelemetry
    expr: sum(energis_channel_telemetry_valid) < 8
    for: 2m
    labels: {severity: warning}
    annotations:
      summary: "ENERGIS channel telemetry not fully valid"
```

6.5 Performance

- Static buffer ~8 KB. No heap allocations.
- Non-blocking, minimal CPU load.
- Typical response time < 50 ms
- If the output would overflow the buffer, the handler returns **503** to encourage a retry instead of serving a partial payload.

6.6 Implementation Notes

- Reads from MeterTask_GetTelemetry() and MeterTask_GetSystemTelemetry()
- Uses atomic counters for request accounting.
- OpenMetrics text format compatible with Prometheus 0.0.4.

7. SNMP interface

This firmware exposes a compact **SNMPv1** agent (community public). A static table binds each OID to a getter (and sometimes a setter). The table itself carries the SNMP type and the maximum response length; the getters in the submodules return the bytes to match that declaration. Identity (MIB-II system) lives alongside a private enterprise subtree at **1.3.6.1.4.1.19865** for

- outlets,
- power telemetry,
- rails/temperature, and
- a read-only mirror of network settings.

No request performs heavy work: outlet writes toggle state immediately; reads return cached or computed values with fixed, short encodings.

Two encoding patterns matter:

- Outlet states are **INTEGER (4 bytes)**:
  - GET returns a 32-bit 0 or 1,
  - SET treats any non-zero as “on”.

  Group controls are also typed as INTEGER.
- Measurements (power, rails, temperature, etc.) and network values are **OCTET STRING**s containing ASCII like "230.12" or "192.168.0.50". Parse as text. Buffers are small (≤16 bytes).

7.1 Submodules

System & table layer

Owns the SNMP community (public), publishes the MIB-II system group (sysDescr, sysObjectID=enterprise 1.3.6.1.4.1.19865, sysUpTime, sysContact, sysSN, sysLocation, sysServices), and wires all private OIDs to domain getters/setters. It also defines an initial warmStart trap helper under the same enterprise. The table is “dumb”: it declares type/length and calls into the right function; logic lives in the domain modules.

Key OIDs (system):

> - 1.3.6.1.2.1.1.1.0 sysDescr (string)
> - 1.3.6.1.2.1.1.2.0 sysObjectID = 1.3.6.1.4.1.19865
> - 1.3.6.1.2.1.1.3.0 sysUpTime (TimeTicks)
> - 1.3.6.1.2.1.1.4.0 sysContact (string)
> - 1.3.6.1.2.1.1.5.0 sysSN (string)
> - 1.3.6.1.2.1.1.6.0 sysLocation (string)
> - 1.3.6.1.2.1.1.7.0 sysServices (INTEGER)

7.2 Outlet control (read/write)

Eight independent channels and two group triggers. Per-channel GET returns a 4-byte INTEGER (0/1). Per-channel SET: 0 → off, any non-zero → on. Group triggers **apply the action when you write any non-zero**, and their GET encodes whether **all** channels currently match that state. Internally, OID channels are 1..8; drivers use 0..7.

7.2.1 Per-channel state (INTEGER):

- **Format:**

```
.1.3.6.1.4.1.19865.2.<ch>.0 where <ch>=1..8
```

GET: *int32* 0 or 1
SET: ***i 0*** or ***i 1*** (non-zero treated as 1) Example (CH3):

```
snmpget -v1 -c public -Oqv <pdu-ip> 1.3.6.1.4.1.19865.2.3.0
```

7.2.2 Group controls (INTEGER):

- All OFF (trigger):

```
.1.3.6.1.4.1.19865.2.9.0
```

GET: '1' only if every channel is off;
SET: non-zero → force all off.

- All ON (trigger):

```
.1.3.6.1.4.1.19865.2.10.0
```

GET: '1' only if every channel is on;
SET: non-zero → force all on.

Implementation detail: per-channel GET always returns **exactly 4 bytes** to match the table's INTEGER declaration. Group GET encodes '0'/'1' via a one-byte helper but the table still declares INTEGER. Managers will still treat these as INTEGER because the table declares INTEGER, the single-byte helper is an internal optimization.

7.3 Power and energy telemetry

Six metrics per channel, all as **ASCII strings** in OCTET STRINGs, pulled from the **cached** metering snapshot (no blocking I/O on the SNMP path). If the cache isn't valid yet, you'll see zeros by design.

7.3.1 Per-channel telemetry:

- Voltage [V]: .1.3.6.1.4.1.19865.5.<ch>.1.0 → "%.2f"
- Current [A]: .1.3.6.1.4.1.19865.5.<ch>.2.0 → "%.3f"
- Power [W]: .1.3.6.1.4.1.19865.5.<ch>.3.0 → "%.1f"
- Power factor: .1.3.6.1.4.1.19865.5.<ch>.4.0 → "%.3f"
- Energy [kWh]: .1.3.6.1.4.1.19865.5.<ch>.5.0 → "%.3f"
- Uptime [s]: .1.3.6.1.4.1.19865.5.<ch>.6.0 → "12345" (decimal ASCII)

Quick sense of outputs for channel 1:

```
snmpget -v1 -c public -Oqv <pdu-ip> 1.3.6.1.4.1.19865.5.1.1.0   # "230.12"
snmpget -v1 -c public -Oqv <pdu-ip> 1.3.6.1.4.1.19865.5.1.2.0   # "0.534"
snmpget -v1 -c public -Oqv <pdu-ip> 1.3.6.1.4.1.19865.5.1.3.0   # "85.3"
```

Formatting is centralized (ftoa, u32toa) and capped at 16 bytes per response. Precision is uniform across channels and metrics.

7.4 Rails and temperature

On-die temperature and board rails, formatted as short ASCII. The module samples Pico ADC for the die sensor (with a tiny low-pass/settle step) and uses HAL helpers for external rails. Everything fits in 16 bytes.

OIDs:

- Die sensor (voltage): .1.3.6.1.4.1.19865.3.1.0 → "%.5f"
- Die temperature [°C]: .1.3.6.1.4.1.19865.3.2.0 → "%.3f"
- V_SUPPLY [V]: .1.3.6.1.4.1.19865.3.3.0 → "%.3f"
- V_USB [V]: .1.3.6.1.4.1.19865.3.4.0 → "%.3f"
- V_SUPPLY divider tap: .1.3.6.1.4.1.19865.3.5.0
- V_USB divider tap: .1.3.6.1.4.1.19865.3.6.0
- Core VREG [V]: .1.3.6.1.4.1.19865.3.7.0 → "%.2f"
- Core VREG status: .1.3.6.1.4.1.19865.3.8.0 → "OK" | "Overload" | "Hi-Z" | "Unknown"
- Bandgap ref: .1.3.6.1.4.1.19865.3.9.0
- USB PHY rail: .1.3.6.1.4.1.19865.3.10.0
- IO rail: .1.3.6.1.4.1.19865.3.11.0

7.5 Network configuration mirror

Reflects the persisted network config (EEPROM) as dotted-decimal strings. These are not writable via SNMP in this build.

OIDs:

- IP address: .1.3.6.1.4.1.19865.4.1.0 → "A.B.C.D"
- Subnet mask: .1.3.6.1.4.1.19865.4.2.0 → "A.B.C.D"
- Gateway: .1.3.6.1.4.1.19865.4.3.0 → "A.B.C.D"
- DNS server: .1.3.6.1.4.1.19865.4.4.0 → "A.B.C.D"

7.6 Overcurrent

Overcurrent protection status is exposed under the enterprise subtree .1.3.6.1.4.1.19865.6.*.0. It reports the **live OCP status snapshot** (no blocking I/O on the SNMP path). Integer values use SNMP INTEGER (4 bytes). Floating values are returned as **ASCII strings in OCTET STRINGs** (fits in 16 bytes). If OCP is not initialized yet, values fall back to safe defaults (typically zeros / NORMAL).

7.6.1 OCP status OIDs

- **OCP state:** .1.3.6.1.4.1.19865.6.1.0 → INTEGER
  Values:
  - 0 = NORMAL
  - 1 = WARNING
  - 2 = CRITICAL
  - 3 = LOCKOUT
- **Total current [A]:** .1.3.6.1.4.1.19865.6.2.0 → "%.3f" (ASCII)
- **Current limit [A]:** .1.3.6.1.4.1.19865.6.3.0 → "%.2f" (ASCII)
- **Warning threshold [A]:** .1.3.6.1.4.1.19865.6.4.0 → "%.2f" (ASCII)
- **Critical threshold [A]:** .1.3.6.1.4.1.19865.6.5.0 → "%.2f" (ASCII)
- **Recovery threshold [A]:** .1.3.6.1.4.1.19865.6.6.0 → "%.2f" (ASCII)
- **Last tripped channel:** .1.3.6.1.4.1.19865.6.7.0 → INTEGER
  Values:
  - 1..8 = outlet index that was switched OFF by OCP
  - 0 = not available / fallback action used
- **Trip count:** .1.3.6.1.4.1.19865.6.8.0 → INTEGER Monotonic counter since boot.
- **Last trip time [ms since boot]:** .1.3.6.1.4.1.19865.6.9.0 → INTEGER
- **Switching allowed:** .1.3.6.1.4.1.19865.6.10.0 → INTEGER
  Values:
  - 0 = switching ON blocked (lockout)
  - 1 = switching allowed

7.6.2 Manual OCP reset (SNMP SET)

A write-only control is provided to clear OCP lockout:

- **Reset/clear lockout:** .1.3.6.1.4.1.19865.6.11.0 → INTEGER (SET) Any non-zero value triggers a lockout clear. Use only after reducing load, otherwise the device may immediately trip again.

7.7 Failure modes and things worth knowing

- **Type discipline**: outlet nodes are 4-byte INTEGERs; telemetry and network are ASCII strings. If your NMS expects numbers everywhere, it will complain on the string nodes. Parse them.
- **"All on/off" asymmetry**: those two reports aggregate state and execute batch writes on any non-zero input; 0 is ignored. Verify final state with per-channel GETs if you need certainty.
- **Small buffers, fixed precision**: string responses are capped at 16 bytes, with formats like "%.2f", "%.3f". If you widen precision in firmware, update the table lengths or you'll truncate.
- **Cached telemetry**: power data is a snapshot from the MeterTask cache. Immediately after enabling a channel, it may read "0.000" until the next update. This path never blocks waiting for the meter.
- **Network fallback**: if EEPROM is blank or inaccessible, the network mirror returns compiled defaults. If your reads show defaults, the storage flow didn't load.
- **Table is authoritative**: per-channel outlet GET always returns exactly 4 bytes to match the table's INTEGER definition, even if the internal code path could compute a one-byte value. Keep your client side tolerant.

7.8 Summary

One static table, four domains. Outlets are writable INTEGER scalars at **.1.3.6.1.4.1.19865.2.<ch>.0**, with two INTEGER triggers for "all off/on."

Power telemetry is six ASCII string metrics per channel at **.1.3.6.1.4.1.19865.5.<ch>.<metric>.0**, pulled from a non-blocking cache.

Rails/temperature live under .1.3.6.1.4.1.19865.3.*.0 as short strings.

Network settings are read-only, dotted-decimal strings under **.1.3.6.1.4.1.19865.4.*.0**, with EEPROM-to-default fallback.

Respect types and 16-byte limits, assume cached updates for measurements, and rely on per-channel reads to verify group actions.

8. Support

For firmware updates and source code, visit the project’s GitHub repository. For technical support or to report bugs, contact the maintainer listed in the SNMP *sysContact* field (dvidmakesthings@gmail.com).