SolarSync P1 Meter

API Integration Guide

Connect your devices to real-time home energy data

Firmware V012 | DSMR 5.0

For developers and system integrators

BattPulse
battpulse.com

1. Overview

The SolarSync P1 Meter provides a simple HTTP API that lets any device on the same network read real-time energy data from a Dutch or Belgian DSMR 5.0 smart meter. This guide shows you how to integrate with it from ESP32, Arduino, Raspberry Pi, Python, Home Assistant, or any HTTP-capable device.

1.1 What You Get

- Real-time solar export and grid import power (kW)
- Cumulative energy per tariff (kWh)
- Gas meter reading (m^3)
- Meter status: tariff, power failures, voltage sags, timestamp
- Device info: IP, MAC, hostname, firmware version, WiFi signal

1.2 Network Discovery

The device is reachable by:

- IP address (e.g., 192.168.68.106) — shown on the dashboard footer
- mDNS hostname (e.g., http://SolarSync_E77308.local) — works from browsers and most OS platforms

> **Important:** ESP32 HTTPClient has unreliable mDNS resolution. See Section 5 for the correct pattern using setReuse(true).

2. API Endpoints

Endpoint	Method	Content-Type	Description
`/power`	GET	text/plain	Minimal JSON: Solar2Grid and Grid2Home only. Designed for ESP-to-ESP polling.
`/api/data`	GET	application/json	Full JSON with all meter data, phase info (if available), device status, diagnostics.
`/info`	GET	application/json	Device identification: type, model, firmware, IP, MAC.
`/version`	GET	text/plain	Returns firmware version string (e.g., "012").

3. Primary Endpoint: /power

This is the endpoint you should use for device-to-device integration. It returns the minimum data needed to make energy decisions.

3.1 Request

```
GET /power HTTP/1.1
Host: 192.168.68.106
```

3.2 Response

```
{ "Solar2Grid": 0.780, "Grid2Home": 0.000 }
```

3.3 Fields

Field	Type	Description
`Solar2Grid`	float	Net solar power being exported to the grid (kW). If this is > 0, the home is producing more solar than it consumes.
`Grid2Home`	float	Net power being imported from the grid (kW). If this is > 0, the home is consuming more than solar produces.

> **Note:** Only one of these values will be non-zero at any time. When Solar2Grid > 0, Grid2Home is 0 (and vice versa). They represent the net balance, not the raw meter values.

3.4 Timing

- The P1 meter sends a new telegram approximately every 1 second.
- The /power values update on every telegram.
- Recommended polling interval: 500ms to 1000ms.
- Polling faster than 500ms gives no benefit (same data) and wastes TCP connections.

4. Full Data Endpoint: /api/data

Returns all parsed meter data plus device diagnostics. Use this for dashboards, logging, or detailed monitoring.

4.1 Response Fields

Field	Type	Description
`netSolar`	float	Net solar export (kW), same as Solar2Grid
`netGrid`	float	Net grid import (kW), same as Grid2Home
`pwrImp`	float	Raw instantaneous import from OBIS 1-0:1.7.0 (kW)
`pwrExp`	float	Raw instantaneous export from OBIS 1-0:2.7.0 (kW)
`hasPhase`	bool	true if meter provides per-phase OBIS codes (3-phase meters only)
`pwrDelL1/L2/L3`	float	Per-phase power delivered grid→home (kW). Zero if hasPhase=false.
`pwrRetL1/L2/L3`	float	Per-phase power returned home→grid (kW). Zero if hasPhase=false.
`voltL1/L2/L3`	float	Per-phase voltage (V). Zero if hasPhase=false.
`curL1/L2/L3`	float	Per-phase current (A). Zero if hasPhase=false.
`impT1, impT2`	string	Cumulative import tariff 1 (low) and tariff 2 (high) in kWh
`expT1, expT2`	string	Cumulative export tariff 1 and 2 in kWh
`tariff`	string	Current tariff: "0001" (low/T1) or "0002" (high/T2)
`failures`	string	Power failure count from meter
`sags`	string	Voltage sag / short interrupt count
`timestamp`	string	Meter timestamp: "2026-03-18 12:25:46"
`gas`	string	Gas meter reading in m³
`host`	string	mDNS hostname (e.g., "SolarSync_E77308")
`ip`	string	Current IP address
`mac`	string	WiFi MAC address
`ssid`	string	Connected WiFi network name
`version`	string	Firmware version
`freeHeap`	int	Free heap memory in bytes
`uptime`	int	Device uptime in seconds
`rssi`	int	WiFi signal strength in dBm
`rssiQual`	string	Human-readable: Excellent/Good/Fair/Weak

5. ESP32 / Arduino Integration

This is the most common integration scenario: an ESP32 device (like the SolarSync EV Charger) polls the P1 meter over WiFi to get real-time energy data.

5.1 The Correct Pattern

Critical: Use a persistent HTTPClient with setReuse(true). Do NOT call http.end() between polls. This keeps the TCP connection alive and avoids mDNS re-resolution on every request.

```
// GLOBAL — create once, reuse forever
HTTPClient p1Http;
bool p1Started = false;

void pollP1() {
  if (WiFi.status() != WL_CONNECTED) return;

  if (!p1Started) {
    p1Http.setReuse(true);   // KEEP TCP CONNECTION ALIVE
    p1Http.setTimeout(3000);
    p1Http.begin("http://solarsync_e77308.local/power");
    p1Started = true;
  }

  int code = p1Http.GET();
  if (code == 200) {
    String payload = p1Http.getString();
    StaticJsonDocument<200> doc;
    if (!deserializeJson(doc, payload)) {
      float solar = doc["Solar2Grid"];
      float grid  = doc["Grid2Home"];
      // Use solar and grid values...
    }
  } else if (code == -1) {
    // Connection lost — reset for next attempt
    p1Http.end();
    p1Started = false;
  }
  // Do NOT call p1Http.end() on success!
}
```

5.2 Common Mistakes

> **Mistake 1: Creating HTTPClient every call.** Each new HTTPClient triggers mDNS resolution. ESP32 mDNS is unreliable under load — you get "Connection Refused" (-1) every few seconds.
>
> **Mistake 2: Calling http.end() after every GET.** This closes the TCP socket, forcing a new connection (and mDNS lookup) on the next request. Only call end() on failure.
>
> **Mistake 3: Polling too fast.** Polling every 300ms gives the same data as 1000ms (the meter updates once per second) but uses 3x the TCP connections.
>
> **Mistake 4: Overwriting settings on connection failure.** If the P1 meter is briefly unreachable, don't reset your charging duty or other parameters. Only modify settings based on successful readings.

5.3 Recommended Polling Interval

```
unsigned long timerDelay = 1000;  // 1 second
```

The DSMR 5.0 meter sends one telegram per second. Polling at 1000ms gives you every update. Polling at 500ms is acceptable if you want slightly faster response.

6. Python Integration

For Raspberry Pi, home servers, or data logging:

6.1 Simple Polling

```
import requests, time, json

P1_URL = "http://192.168.68.106/power"

while True:
    try:
        r = requests.get(P1_URL, timeout=3)
        data = r.json()
        solar = data["Solar2Grid"]
        grid  = data["Grid2Home"]
        print(f"Solar: {solar:.3f} kW  Grid: {grid:.3f} kW")
    except Exception as e:
        print(f"Error: {e}")
    time.sleep(1)
```

6.2 Full Data with Logging

```
import requests, json, csv, datetime

r = requests.get("http://192.168.68.106/api/data", timeout=3)
d = r.json()

# Log to CSV
with open("energy_log.csv", "a") as f:
    w = csv.writer(f)
    w.writerow([
        datetime.datetime.now().isoformat(),
        d["pwrImp"], d["pwrExp"],
        d["impT1"], d["impT2"],
        d["expT1"], d["expT2"],
        d["gas"], d["tariff"]
    ])
```

7. Home Assistant Integration

Add the SolarSync P1 Meter as a REST sensor in your configuration.yaml:

```
sensor:
  - platform: rest
    name: "SolarSync Solar Export"
    resource: "http://192.168.68.106/power"
    value_template: "{{ value_json.Solar2Grid }}"
    unit_of_measurement: "kW"
    scan_interval: 2

  - platform: rest
    name: "SolarSync Grid Import"
    resource: "http://192.168.68.106/power"
    value_template: "{{ value_json.Grid2Home }}"
    unit_of_measurement: "kW"
    scan_interval: 2

  - platform: rest
    name: "SolarSync Gas"
    resource: "http://192.168.68.106/api/data"
    value_template: "{{ value_json.gas }}"
    unit_of_measurement: "m³"
    scan_interval: 60
```

For the full /api/data endpoint with multiple fields, use a REST sensor with json_attributes:

```
  - platform: rest
    name: "SolarSync Full"
    resource: "http://192.168.68.106/api/data"
    value_template: "{{ value_json.netSolar }}"
    json_attributes:
      - netGrid
      - pwrImp
      - pwrExp
      - impT1
      - impT2
      - expT1
      - expT2
      - gas
      - tariff
      - timestamp
    scan_interval: 5
```

8. Quick Testing

8.1 curl (terminal)

```
# Quick power check
curl http://192.168.68.106/power

# Full data (formatted)
curl -s http://192.168.68.106/api/data | python -m json.tool

# Device info
curl http://192.168.68.106/info
```

8.2 JavaScript (browser console)

```
fetch("http://192.168.68.106/power")
  .then(r => r.json())
  .then(d => console.log(
    `Solar: ${d.Solar2Grid} kW, Grid: ${d.Grid2Home} kW`
  ));
```

8.3 Node.js

```
const http = require("http");

setInterval(() => {
  http.get("http://192.168.68.106/power", (res) => {
    let data = "";
    res.on("data", chunk => data += chunk);
    res.on("end", () => {
      const d = JSON.parse(data);
      console.log(`Solar: ${d.Solar2Grid} Grid: ${d.Grid2Home}`);
    });
  });
}, 1000);
```

9. Example: Solar-Adaptive EV Charging

This is the real-world integration used by the SolarSync EV Charger. The charger polls the P1 meter every second and adjusts its charging power based on available solar energy.

9.1 Algorithm

1. Poll /power every 1 second
2. If Solar2Grid > 0.5 kW: increase charging duty by 5% (more solar available)
3. If Solar2Grid > 0.04 kW: increase by 1%
4. If Grid2Home > 0.5 kW: decrease charging duty by 5% (using too much grid)
5. If Grid2Home > 0.04 kW: decrease by 1%
6. Clamp duty between 10% (~6A) and 85% (~51A)

9.2 Key Design Rules

- Separate manual duty from solar duty — never let P1 connection failures affect manual mode settings.
- On P1 connection failure: pause solar adjustments, do NOT reset the duty to a default value.
- Use HTTPClient.setReuse(true) for reliable mDNS polling.
- Poll at 1000ms, not faster.

9.3 Pseudocode

```
// Every 1 second:
poll_p1_meter();

if (mode == SOLAR_ADAPTIVE && charging_on) {
  if (solar2grid > 0.5)      duty += 5;
  else if (solar2grid > 0.04) duty += 1;
  else if (grid2home > 0.5)   duty -= 5;
  else if (grid2home > 0.04)  duty -= 1;

  duty = clamp(duty, 10, 85);
  set_pilot_pwm(duty);
}
```

10. Troubleshooting

Issue	Solution
Connection Refused (-1)	Use setReuse(true) on HTTPClient. Or use IP address instead of mDNS hostname. Also check: are you polling too fast? Slow down to 1000ms.
mDNS works in browser but not from ESP32	ESP32 HTTPClient re-resolves mDNS on every new connection. Use persistent connection with setReuse(true) and never call http.end() on success.
Values are all zero	The P1 meter may still be starting up. Wait 5 seconds after boot. Check the serial debug output on the P1 meter for "P1: OK" messages.
JSON parse error	Check the raw response with curl. The /power endpoint returns text/plain. Use StaticJsonDocument<200> for /power and <1024> for /api/data.
Device IP changed	Set a static IP in your router's DHCP settings using the MAC address shown on the settings page. Or use mDNS (reliable from browsers and Python).
Data lags behind real meter	The P1 meter reads and parses every ~400ms. Dashboard refreshes every 1 second. There is an inherent 0.5-1.5 second delay from physical meter to your client.

11. Support

Manufacturer: BattPulse

Product page: battpulse.com/product/solarsync-p1-meter

Website: battpulse.com

EV Charger integration: SolarSync EV Charger