Interfaces

The adapter serves the same readings over JSON, MQTT and Modbus TCP. This is how you fetch them.

JSON API

The adapter exposes two REST endpoints. Both expect an authorization token in the header, which you generate and copy in the web interface under *API → JSON*.

Readings

```
GET https://<IP>/api/data/measurement.json
Authorization: <token from the web interface>
```

The response is keyed by OBIS code. Each value carries a Unix timestamp; `sma_time` is the device uptime in seconds.

```
{
  "1-0:1.8.0": { "value": 105119, "time": 1104625548 },
  "1-0:2.8.0": { "value": 0,      "time": 1104625548 },
  "1-0:1.7.0": { "value": 33,     "time": 1104625548 },
  "1-0:2.7.0": { "value": 0,      "time": 1104625548 },
  "api_version": "v1",
  "sma_time": 14011.8
}
```

OBIS	MEANING
`1-0:1.8.0`	Active energy import, meter reading
`1-0:2.8.0`	Active energy export, meter reading
`1-0:1.7.0`	Instantaneous active power, import
`1-0:2.7.0`	Instantaneous active power, export

> Which figures actually arrive depends on the meter type and the grid operator. Only the values the meter emits are populated.

Status and diagnostics

```
GET https://<IP>/api/sma/status.json
```

```
{
  "sma_status_id": 2,
  "api_version": "1.0",
  "sma_time": 14285.1,
  "meter": {
    "supplier": "Netz Oberösterreich",
    "manufacturer": "Siemens",
    "interface": "IR-Lesekopf",
    "name": "TD351x"
  },
  "sma_module_type": "UART",
  "meter_valid_package_cnt": 142829,
  "meter_invalid_package_cnt": 0,
  "sma_status": "running",
  "wifi": { "ip": "10.2.2.211", "rssi": -34, "ssid": "NetworkSSID" },
  "fw_version": "c59cf4b510",
  "partition": "ota_1"
}
```


`meter_valid_package_cnt` and `meter_invalid_package_cnt` are the most useful fields for troubleshooting: if the first does not climb, no telegrams are arriving at all; if the second climbs, the meter key is usually wrong.

MQTT

The adapter is an MQTT *client* and does not include a broker. You specify the broker and the topic; meter data is published in the same JSON format as the REST interface serves.

Configure this in the web interface under *API → MQTT*. Because the values go to the broker rather than being fetched from the adapter, MQTT is also the right route when more than three clients need the data — the adapter itself allows only three concurrent connections.

Modbus TCP

Must be enabled in the web interface under *API → Modbus*. The register layout follows the SunSpec standard for meter models.

> **Mind the offset of 69**
> Registers 0 to 69 serve as a header. “Position 1” from the SunSpec specification therefore lives at register 70. Add 69 to every address from the specification.

MODBUS	SUNSPEC	CONTENT
73	4	Current, phase L1
74	5	Current, phase L2
75	6	Current, phase L3
76	7	Current scale factor (default -3, i.e. 10^{-3})
109–110	39–40	Active energy export
116–117	47–48	Active energy import

Values are scaled, and the scale factor sits in its own register (for example “AC Voltage Scale Factor”). Read it rather than assuming it — it can differ by meter. Registers the meter does not supply stay empty.

For checking from the command line, `modbus-cli` works well. Append `\I` for 32-bit registers:

```
modbus $IP 73       # current L1
modbus $IP 76       # scale factor
modbus $IP 116\I    # active energy import (32 bit)
```

SunSpec specification: Meter Models A12023-1.2 (https://sunspec.org/wp-content/uploads/2019/08/SunSpec-Specification-Meter-Models-A12023-1.2.pdf)

Dashboard

The built-in web interface is reachable at `https://<IP>` and shows the measured active power for import, export and total as a chart, plus the current values in a table. Whether access is allowed over HTTPS only or over HTTP as well is set under *Security*.