OpenADR 3, SunSpec Modbus and IEC 61850

Version OpenADR 3.1.0 · verified 2026-11-02

Which energy protocol lives on which layer — SunSpec Modbus registers, OpenADR 3 REST and MQTT, IEC 61850 logical nodes, OPC UA and Matter.

Contents

Mental model 3

The layer map 3

Modbus TCP essentials 5

SunSpec model chain 6

SunSpec scale factors and missing values 8

OpenADR 3 resources 9

OpenADR 3.1 notifications 11

IEC 61850 orientation 12

Bridges: OPC UA and Matter 13
Time and units traps 14
Further reading 15

Mental model

Energy integrations are layered. Devices such as inverters and batteries speak **SunSpec Modbus** at the edge, plants and substations speak **IEC 61850**, utilities send prices and limits to sites over **OpenADR**, and **OPC UA** or a gateway bridges between them. Before reading any spec, decide which layer your integration lives on — most confusion comes from expecting one protocol to do another layer's job.

The layer map

Layer	Protocol	You typically
Utility to site	OpenADR 3	Consume events, post reports
Substation, plant	IEC 61850	Read via a gateway
Plant SCADA, bridge	OPC UA	Browse a typed model

Layer	Protocol	You typically
Device edge	SunSpec Modbus	Poll registers
Home appliances	Matter	Read energy clusters

These are complementary, not competing. The OpenADR Alliance and the Connectivity Standards Alliance announced a liaison in May 2026 on exactly this split: Matter between appliance and home energy gateway, OpenADR 3 between that gateway and the utility. EV chargers add OCPP as their own device-to-backend link.

```
Utility BL -> VTN  ==OpenADR 3==>  site VEN
site gateway  ==SunSpec Modbus==>  inverter
site gateway  ==Matter==>          heat pump
```

Modbus TCP essentials

Function code	Operation
`0x03`	Read holding registers
`0x04`	Read input registers
`0x06`	Write single register
`0x10`	Write multiple registers

Modbus TCP listens on port 502 by default. Each request carries a 7-byte MBAP header — transaction id, protocol id `0`, length, unit id — followed by the function code and data. A single `0x03` read returns 1 to 125 registers of 16 bits each, so large models need several reads.

```
import struct


def read_holding(tid, unit, addr, count):
    pdu = struct.pack(">BHH", 0x03, addr, count)
    mbap = struct.pack(">HHHB", tid, 0,
                       len(pdu) + 1, unit)
    return mbap + pdu
```

```
print(read_holding(1, 1, 40000, 2).hex())
# 00010000000601039c400002
```

> **Note:** The unit id replaces the serial slave address. A gateway uses it to route to the device behind it; for a device directly on TCP the Modbus guide recommends `0xFF`, and `0` is also accepted.

SunSpec model chain

Element	Value
Base address	`40000`, else `0` or `50000`
Marker	Two registers spelling `SunS`
Model header	`ID`, then length `L`
Model 1	Common: `Mn`, `Md`, `SN`
Model 103	Three-phase inverter (legacy)
Model 701	DER AC measurement (IEEE 1547)

SunSpec turns raw Modbus into a self-describing map. After the `SunS` marker, models follow back to back: each

starts with its model `ID` and its length `L` in registers, so a client walks the chain by jumping `L + 2` registers until it reads the end marker `0xFFFF`. The 701–712 series replaces the 100 series for IEEE 1547-2018 devices.

```
regs = [0x5375, 0x6E53,      # "SunS"
        1, 2, 0, 0,          # model 1, L=2
        103, 1, 0,           # model 103, L=1
        0xFFFF, 0]           # end
i, models = 2, []
while regs[i] != 0xFFFF:
    models.append(regs[i])
    i += regs[i + 1] + 2
print(models)  # [1, 103]
```

> **Gotcha:** Device manuals often print 1-based register numbers (`40001`) while clients send 0-based protocol addresses (`40000`). Off by one, you read the marker's second half and find no `SunS` .

SunSpec scale factors and missing values

Type	Not implemented
`int16`	`-32768` (`0x8000`)
`uint16`, `enum16`	`0xFFFF`
`sunssf`	`-32768`
`acc32`	`0`

Integer models store a value and a separate scale factor point, such as `W` and `W_SF`: the real value is `raw × 10^sf`. Each type also has a sentinel meaning “not implemented”, which a decoder must check before scaling. Read the value and its scale factor in the same request, so the pair always comes from one snapshot.

```
def scaled(raw, sf):
    if raw == -32768 or sf == -32768:
        return None  # not implemented
    return raw * 10 ** sf
```

```
print(scaled(4521, -1))    # 452.1
print(scaled(-32768, 0))   # None
```

> **Gotcha:** The accumulator types use `0` as “not implemented”, so a zero lifetime energy counter is ambiguous. Treat a counter that is zero on a producing device as missing, not as a reset.

OpenADR 3 resources

Path	Written by	Holds
`/programs`	BL	Business context, e.g. a tariff
`/events`	BL	Prices or limits over intervals
`/reports`	VEN	Usage or status for an event
`/subscriptions`	VEN or BL	Webhook callbacks

Path	Written by	Holds
`/vens`, `/resources`	VEN or BL	Clients and their devices

OpenADR 3 is a REST API. The **VTN** is the server; **business logic** (BL, the utility side) and **VENs** (the customer side) are both its clients, authenticated with OAuth 2.0 client credentials via `/auth/token`. An event holds `intervals`, each with `payloads` whose `type` (such as `PRICE`, `SIMPLE` or `IMPORT_CAPACITY_LIMIT`) is explained by a matching `payloadDescriptors` entry.

```
{"programID": "1", "eventName": "tou-peak",
  "payloadDescriptors": [{
    "objectType": "EVENT_PAYLOAD_DESCRIPTOR",
    "payloadType": "PRICE",
    "units": "KWH", "currency": "USD"}],
  "intervalPeriod": {
    "start": "2026-11-02T17:00:00Z",
    "duration": "PT1H"},
```

```
  "intervals": [{"id": 0, "payloads": [
    {"type": "PRICE", "values": [0.42]}]}]}
```

OpenADR 3.1 notifications

Binding	Suits
`WEBHOOK`	Cloud VENs that can accept inbound HTTP
`MQTT`	VENs behind firewalls, e.g. in homes

OpenADR 3.0 and 3.0.1 pushed changes only by webhook. 3.1.0, published in 2025, adds MQTT so a VEN behind a home firewall can receive pushes over a connection it opens itself. `GET /notifiers` lists the bindings and the broker URIs; `GET /notifiers/mqtt/topics/...` returns the topic name per operation. Each message is a `notification` with `objectType`, `operation` and the full `object`.

```
{"topics": {
  "CREATE": "programs/create",
```

```
  "UPDATE": "programs/update",
  "DELETE": "programs/delete",
  "ALL": "programs/+"}}
```

> **Gotcha:** Topic names are chosen by each VTN and may be specific to your client. Always ask the topics endpoint; never hard-code the example strings above.

IEC 61850 orientation

Part	Defines
61850-6	SCL, the XML configuration language
61850-7-3	Common data classes, e.g. `MV`
61850-7-4	Logical node classes
61850-7-420	DER logical nodes (ed. 2, 2021)
61850-8-1	Mapping to MMS over Ethernet

IEC 61850 models a device as logical devices containing **logical nodes** — four-letter classes such as `MMXU`

(measurements), `XCBR` (circuit breaker) or `LLN0` (the logical device's own node) — which hold data objects and attributes. You rarely talk MMS yourself; a gateway or OPC UA server exposes these names, and you need to read them.

```
LD0/MMXU1.TotW.mag.f   total active power
LD0/MMXU1.Hz.mag.f     frequency
LD0/XCBR1.Pos.stVal    breaker position
```

Bridges: OPC UA and Matter

Technology	Role in energy
OPC UA (IEC 62541)	Typed, browsable plant model
OPC UA for IEC 61850	Companion spec mapping 61850
Matter 1.4	Home energy clusters and devices

OPC UA exposes an address space of typed nodes that a client browses instead of memorising register maps,

and the OPC Foundation's IEC 61850 companion specification maps logical nodes into it. In homes, Matter 1.4 defines Electrical Power Measurement, Electrical Energy Measurement, Device Energy Management and Energy EVSE clusters, which a gateway can translate to OpenADR reports and limits.

```
inverter --Modbus--> gateway --OPC UA--> SCADA
heat pump --Matter--> HEMS --OpenADR--> VTN
```

Time and units traps

Field	Rule
OpenADR `dateTime`	RFC 3339, include the offset
OpenADR `duration`	ISO 8601, e.g. `PT15M`
`PERCENT` units	-1.0 to 1.0, not 0 to 100
`P9999Y` duration	May mean infinity
`0001-01-01` start	May mean now

OpenADR timestamps carry their own offset and intervals are durations, so 15-minute price intervals are

`PT15M` from a stated `start`. Units come from the payload descriptor (`KW` is power, `KWH` is energy), and SunSpec values need their scale factor before any unit means anything.

```
from datetime import datetime, timedelta


start = datetime.fromisoformat(
    "2026-11-02T17:00:00-08:00")
end = start + timedelta(minutes=15)  # PT15M
print(end.isoformat())
# 2026-11-02T17:15:00-08:00
```

> **Warning:** OpenADR `PERCENT` 0.5 means 50%. Treating it as 0.5% sends devices a limit a hundred times too small, with no error.

Further reading

- OpenADR Alliance — specification downloads

- OpenADR 3.1.0 OpenAPI and Definitions (public mirror)
- SunSpec Modbus
- SunSpec model definitions
- Modbus Application Protocol V1.1b3
- IEC 61850-7-420:2021
- Matter 1.4.1 Application Cluster Specification