Valtoris.

Integration Guide: Polling Eastron SDM Meters via Modbus RTU

Decoding IEEE 754 Float32 Metrics for Industrial SCADA and IoT Gateways

Document No:	AN-2026-04
Product Series:	Valtoris VT-DTU500 / Edge Gateways
Target Devices:	Eastron SDM120, SDM630, SDM230
Date:	June 2026 (Rev. 1.0)

1. Executive Summary

Eastron SDM-series smart meters are widely deployed globally for EV charger sub-metering, solar export limiting, and energy management. While they offer excellent accuracy and value, integrating them into modern SCADA or IT infrastructure often presents a significant protocol barrier.

Unlike standard industrial devices that utilize 16-bit integers, Eastron strictly encodes all telemetry data as **IEEE 754 32-bit Floating-Point Numbers**. This Application Note outlines the correct physical wiring, register mapping, and the exact byte-swapping logic required to reconstruct these values accurately.

2. Physical Layer: RS485 Terminal Pinouts

To establish a stable physical layer, use a shielded twisted-pair (STP) cable. Eastron terminal blocks vary by phase type. Connect the RS485 port to your Master Gateway as defined below.

© 2026 Valtoris Industrial Networks | www.valtoris.com

Meter Model	Eastron Terminal	Gateway Terminal	Signal Description
SDM120 (1-Phase)	**Pin 10** (A)	**A+** (Data+)	Non-inverting differential signal
	Pin 9 (B)	**B-** (Data-)	Inverting differential signal
SDM630 (3-Phase)	**Pin 23** (A)	**A+** (Data+)	Non-inverting differential signal
	Pin 24 (B)	**B-** (Data-)	Inverting differential signal

> **Engineering Note: Termination & Grounding**
>
> If daisy-chaining multiple Eastron meters on a bus exceeding 50 meters, a 120Ω termination resistor must be installed across A and B at the furthest meter. Although Eastron lacks a dedicated RS485 Ground pin, ensure the cable shield is grounded at the Gateway side to drain common-mode transient noise.

3. The Modbus Register Map

The factory default communication settings for Eastron are typically **2400 or 9600 Baud, 8 Data Bits, 1 Stop Bit, NONE or EVEN Parity** (verify via the LCD). The default Slave ID is 1.

Real-time data must be polled using Modbus **Input Registers (Function Code 0x04)**. Since all values are 32-bit floats, each metric requires requesting 2 consecutive registers.

Register (Dec)	Hex Address	Data Format	Parameter Description	Unit
30001	0x0000	Float32 (2 Regs)	Phase 1 Line Voltage	Volts (V)
30007	0x0006	Float32 (2 Regs)	Phase 1 Line Current	Amps (A)
30013	0x000C	Float32 (2 Regs)	Total Active Power (System)	Watts (W)
30073	0x0048	Float32 (2 Regs)	Total Import Energy	kWh
30343	0x0156	Float32 (2 Regs)	Total Active Energy (Import + Export)	kWh

© 2026 Valtoris Industrial Networks | www.valtoris.com

4. Deep Dive: Decoding the IEEE 754 Payload

The most common failure point during integration is receiving "garbage" values (e.g., `1.43e-42`) instead of a readable voltage. This occurs because Modbus RTU transports 16-bit words, and different manufacturers transmit the high and low words of a 32-bit float in different orders.

Let us analyze a real serial packet when polling **Phase 1 Line Voltage (Address 0x0000)**.

```
Master Request (Tx): 01 04 00 00 00 02 71 CB

Breakdown:
01 : Slave ID
04 : Function Code (Read Input Registers)
00 00 : Starting Address (30001)
00 02 : Quantity to read (2 registers = 4 bytes)
71 CB : CRC Checksum

Meter Response (Rx): 01 04 04 43 66 33 33 XX XX

Data Payload: 43 66 33 33
```

Byte Swapping (Endianness) Logic

The hex payload `43 66 33 33` represents the floating-point number. Eastron transmits this in standard **Big-Endian (ABCD)** format.

- If converted directly from IEEE 754 Hex `0x43663333` to decimal, the result is exactly **230.20 Volts**.
- If your polling script or hardware assumes a CDAB (Little-Endian byte swap) format, it will interpret the hex as `33 33 43 66`, resulting in the absurd output of `4.17e-08`.

5. Recommended System Architecture

Depending on the scale of your deployment, parsing these Float32 frames can be handled via software scripting or dedicated hardware.

A. Software Scripting (DIY / Linux)

B. Protocol Normalization (Edge Gateway)

© 2026 Valtoris Industrial Networks | www.valtoris.com

Using a USB-to-RS485 adapter connected to a Raspberry Pi or IPC running Python.

- **Implementation:** Requires utilizing libraries like `pymodbus` and the built-in `struct.unpack('>f', payload)` function to explicitly define the Big-Endian float structure.
- **Limitations:** Highly susceptible to OS-level USB driver crashes and USB sleep states, leading to communication timeouts in production environments.

Deploying a DIN-rail industrial Edge Gateway (such as the Valtoris VT-DTU500) adjacent to the meters.

- **Implementation:** The gateway's silicon natively handles IEEE 754 decoding. You simply define the register as "Float32 ABCD" in the web UI.
- **Advantage:** It completely eliminates the need for decoding scripts. The hardware autonomously polls the meter and pushes standard JSON payloads to your MQTT broker via Ethernet or Cellular.

Additional Resources

To eliminate the complexity of Modbus RTU float decoding and scale your metering deployments with industrial reliability, review the technical specifications of the Valtoris Edge Gateway Series.

© 2026 Valtoris Industrial Networks | www.valtoris.com