OPC Bridge – Operations Manual

- Table of Contents
- Part I – Fundamentals
  - 1. Introduction and overview
  - 2. System architecture
  - 3. Basic terms
  - 4. Protocol fundamentals in detail
- Part II – Installation and commissioning
  - 5. System requirements
  - 6. Installation on Windows
  - 7. Installation on Debian/Ubuntu
  - 8. Installation as a Docker container
  - 9. First commissioning step by step
- Part III – Monitoring and daily use
  - 10. Web interface layout
  - 11. Overview tab
  - 12. Live Values tab
  - 13. Trends tab
  - 14. Log tab
- Part IV – Configuration
  - 15. Connections – general
  - 16. Connection type OPC-DA
  - 17. Connection type OPC-UA Client
  - 18. Connection type OPC-UA Server
  - 19. Connection type Siemens S7
  - 20. Connection type Modbus
  - 21. Connection type MQTT
  - 22. Connection type Sparkplug B
  - 23. Connection type CAN
  - 24. Connection type SQL
  - 25. Connection type InfluxDB
  - 26. Connection type Webhook
  - 27. Connection type Bridge-Link
  - 28. Mappings and data flow
  - 29. Conversion and formulas
  - 30. Virtual sources and redundancy
  - 31. Store-and-forward
  - 32. Configuring history and trends
  - 33. Alarms, e-mail and webhook
  - 34. CAN analyzer and gateway
  - 35. Modbus register inspector
  - 36. WebView
  - 37. Users, roles and API authentication
  - 38. Real-time updates (WebSocket)

- Part V – Interfaces and integration
  - 39. REST API and built-in documentation
  - 40. Configuration file (config.json)
- Part VI – Administration and maintenance
  - 41. Licensing
  - 42. Security and hardening
  - 43. Backup and restore
  - 44. Update and migration
  - 45. Performance and sizing
  - 46. Troubleshooting
- Part VII – Application examples
  - 47. Example: OPC-DA → OPC-UA
  - 48. Example: Modbus → MQTT cloud
  - 49. Example: CAN → OPC-UA
  - 50. Example: OT→IT across the DMZ
- Part VIII – Appendix
  - A. Address formats (complete)
  - B. Formula and function reference
  - C. REST endpoint reference
  - D. Data type reference
  - E. Default ports
  - F. File and directory locations
  - G. Commissioning checklist
  - H. Glossary
  - I. Frequently asked questions (FAQ)

Biricon

OPC Bridge

Technical Operations Manual

Industrial protocol converter for OPC-DA, OPC-UA, Modbus, MQTT, Sparkplug B, CAN/J1939, SQL and REST

Product version **1.1.0** · Manual revision **November 2026**

© 2026 **Biricon IT Services GmbH** · www.biricon.eu

> **Purpose and audience.** This operations manual describes the OPC Bridge completely – from the fundamentals of the supported fieldbus and network protocols through installation and configuration to operation, maintenance and troubleshooting. It is written for **commissioning engineers**, **automation**

> **technicians**, **IT/OT administrators** and **maintenance staff** alike. Every technical term is explained on first use; the fundamentals chapters assume **no** prior knowledge. Advanced readers may skip the fundamentals and go straight to the configuration chapters.
>
> **Language versions.** This document is available in English and German. The German version is `Betriebshandbuch-DE.md`.
>
> **Conventions.** `Monospace` denotes input, addresses, file names and code. *Italics* denote controls and technical terms. Boxes marked **»Note«**, **»Caution«** and **»Tip«** highlight important points.
>
> **About the screenshots.** German is the interface's source language, and the screenshots in this English manual were captured from that German-language build – every screenshot in this document therefore shows **German UI labels**, even though the English interface (selectable via the language switch, section 10) has the same layout with English labels. Read screenshots for their **layout and position of controls**, not for the exact wording. A number of screenshots also predate recent feature additions (online licensing, the S7 and InfluxDB connection types, LDAP/OIDC/2FA, the batch-write fields) and may not show every field described in the surrounding text.

Table of Contents

Part I – Fundamentals 1. Introduction and overview 2. System architecture 3. Basic terms 4. Protocol fundamentals in detail - 4.1 About industrial communication - 4.2 OPC-DA (Classic) - 4.3 OPC-UA - 4.4 Modbus TCP and RTU - 4.5 MQTT - 4.6 MQTT Sparkplug B - 4.7 CAN bus - 4.8 J1939 - 4.9 SQL / ODBC - 4.10 REST / Webhook - 4.11 Bridge-Link (OT→IT tunnel) - 4.12 Data types and byte order

Part II – Installation and commissioning 5. System requirements 6. Installation on Windows 7. Installation on Debian/Ubuntu 8. Installation as a Docker container 9. First commissioning step by step

Part III – Monitoring and daily use 10. Web interface layout 11. Overview tab 12. Live Values tab 13. Trends tab 14. Log tab

Part IV – Configuration 15. Connections – general 16. Connection type OPC-DA 17. Connection type OPC-UA Client 18. Connection type OPC-UA Server 19. Connection type Siemens S7 20. Connection type Modbus 21. Connection type MQTT 22. Connection type Sparkplug B 23. Connection type CAN 24. Connection type SQL 25. Connection type InfluxDB 26. Connection type Webhook 27. Connection type Bridge-Link 28. Mappings and data flow 29. Conversion and formulas 30. Virtual sources and redundancy 31. Store-and-forward 32. Configuring history and trends 33. Alarms, e-mail and webhook 34. CAN analyzer and gateway 35. Modbus register inspector 36. WebView 37. Users, roles and API authentication 38. Real-time updates (WebSocket)

Part V – Interfaces and integration 39. REST API and built-in documentation 40. Configuration file (config.json)

Part VI – Administration and maintenance 41. Licensing 42. Security and hardening 43. Backup and restore 44. Update and migration 45. Performance and sizing 46. Troubleshooting

Part VII – Application examples 47. Example: OPC-DA → OPC-UA 48. Example: Modbus → MQTT cloud 49. Example: CAN → OPC-UA 50. Example: OT→IT across the DMZ

Part VIII – Appendix - A. Address formats (complete) - B. Formula and function reference - C. REST endpoint reference - D. Data type reference - E. Default ports - F. File and directory locations - G. Commissioning checklist - H. Glossary - I. Frequently asked questions (FAQ)

Part I – Fundamentals

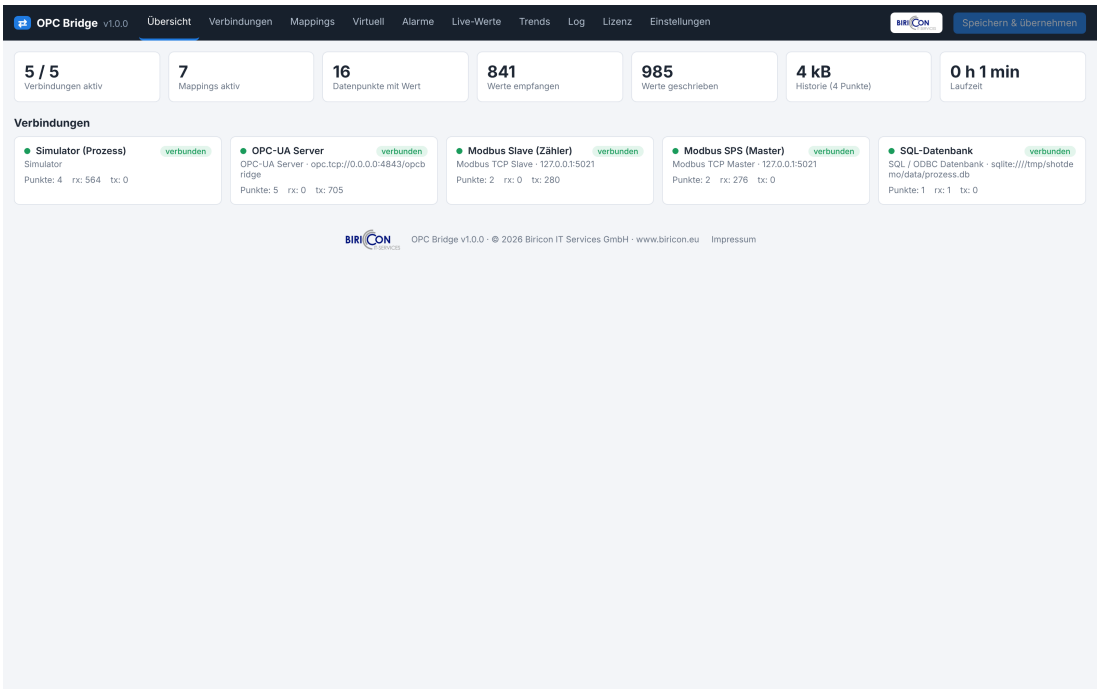
1. Introduction and overview

1.1 What is the OPC Bridge?

The **OPC Bridge** is a **universal protocol converter** for industrial communication. In a typical plant, many devices and systems speak different “languages” – so-called **protocols**:

- an older machine controller provides values via **OPC-DA**,
- an energy or flow meter communicates over **Modbus**,
- a modern sensor sends via **MQTT** to a cloud,
- a diesel generator or vehicle delivers data over the **CAN bus**,
- the supervisory control system expects **OPC-UA**.

For these devices to work together, their values must be **translated from one protocol into another**. That is exactly what the OPC Bridge does: it connects to several sources and targets at the same time and transfers values between them – **in both directions**.



Overview

1.2 The interpreter analogy

Picture a **round table** with people from different countries. Each speaks only their own language. In the middle sits an **interpreter** who takes each statement and passes it on to the others in their respective language.

The OPC Bridge **is** that interpreter. As the user you only need to define two things:

1. **Who is at the table?** – these are the **connections** (e.g. “the Modbus PLC”, “the OPC-UA control system”).
2. **Who should tell what to whom?** – these are the **mappings** (e.g. “the PLC’s temperature → the control system’s temperature”).

Everything else – constant polling, conversion, buffering on faults and forwarding – the Bridge handles automatically.

1.3 Feature overview

Area	Features
Protocols	OPC-DA (client), OPC-UA (client and server), **Siemens S7 (direct)**, Modbus TCP/RTU (master and slave), MQTT, MQTT Sparkplug B, SQL/ODBC, **InfluxDB**, CAN (incl. J1939), REST/Webhook, Simulator
Data flow	Bidirectional routing (A→B, B→A, ↔), merging of multiple sources, multithreading
Real-time	Live values over WebSocket with polling fallback

Area	Features
Throughput	**Batch/bulk writing** (SQL multi-row, JSON-array webhook, InfluxDB line protocol)
Conversion	Factor/offset, free formulas, type and byte-order conversion
Virtual sources	Computed points, redundancy/hot-standby
Reliability	Persistent store-and-forward (SQLite), automatic reconnection
OT→IT coupling	Bridge-Link: outbound TLS tunnel without inbound firewall ports, optional **strict one-way / data-diode mode**
Monitoring	History, trends, alarm rules (e-mail/webhook), OPC-UA A&E
Sharing	WebView – token-protected read-only live view
REST API	Complete API with built-in OpenAPI documentation (offline)
Security	Configurable API authentication, roles, HTTPS, OPC-UA security, **LDAP/Active Directory, OIDC/SSO, TOTP 2FA**
Interface	**Multilingual** (German, English, French, Spanish, Portuguese)
Copy protection	Machine-bound signed serial numbers (Ed25519, offline) **and online license** (ES256) with automatic activation, device limit and **clone detection**; demo mode
Updates	**Update notification** and **one-click installation** via a signed version file
Delivery	Windows setup (.exe), Debian package (.deb), Docker image

1.4 Typical use cases

- **Modernization:** connect a legacy OPC-DA plant to a new OPC-UA control system without replacing the controller.
- **Digitalization/IIoT:** deliver values from Modbus meters to an MQTT cloud or a Unified Namespace (Sparkplug B).
- **Vehicle/generator integration:** provide CAN signals of a generator set as OPC-UA or Modbus values.
- **Secure OT/IT coupling:** transport values from production through a DMZ into IT without opening the production firewall.
- **Protocol bridge Modbus↔OPC:** let an OPC-UA client access Modbus registers.

---

2. System architecture

2.1 Component model

Internally the OPC Bridge consists of several cooperating layers:

- **Drivers:** there is one driver per protocol. It handles connecting, polling, writing and reporting values. Each *connection* in the UI is an instance of a driver.
- **Engine:** the core. It manages all drivers, executes the mappings (forwards values), applies formulas, buffers on faults (store-and-forward) and writes the history.
- **Virtual sources:** computed points and redundancy groups that derive new values from existing ones.
- **Notifier:** evaluates alarm rules and sends e-mails/webhooks.
- **Web interface and REST API:** operation and integration via browser or programs.

2.2 Data flow

A value typically travels this path:

1. The **source driver** reads the value from the device (or receives it) and reports it to the engine with **address**, **value**, **quality** and **timestamp**.
2. The **engine** finds all mappings in which this point is the source.
3. For each mapping the **formula** is applied (if configured) and the data type converted.
4. The value is handed to the **target driver**. If the target is unreachable and store-and-forward is active, the value is **buffered**.
5. The **target driver** writes the value to the device (or publishes it).

At the same time the value may be written to the **history**, checked against **alarm rules**, and pushed to the **live view** (via WebSocket).

2.3 Operating model

The Bridge runs as a **service** (Windows service, systemd service or Docker container) in the background – regardless of whether a browser is open. The web interface is only the operation and monitoring tool; the actual data exchange runs inside the service.

> **»Note«** The Bridge keeps working even when no browser is connected. Closing the browser does **not** stop data transfer.

---

3. Basic terms

The following terms are used throughout this manual.

Term	Meaning
Connection	A link to a device or system. Has a **type** (= protocol) and its own settings.
Driver	The software behind a connection type (e.g. the Modbus driver).
Point / Tag	A single value within a connection (e.g. a temperature).
Address	Identifies the location/name of a point in the device (e.g. `HR100` or `ns=2;s=Temp`).
Mapping	Rule “source A → target B”. Defines which point is transferred to which.
Direction	A→B, B→A or A↔B (bidirectional).
Quality	Goodness of a value: *good*, *uncertain*, *bad*.
Timestamp	The time at which the value was valid.
Master / Slave	Modbus roles: master polls, slave answers/ provides values.
Client / Server	The client connects to the server. The Bridge can be either, depending on the protocol.
Publish / Subscribe	MQTT principle: publishing or subscribing to topics.
OT / IT / DMZ	Production network / office network / isolated intermediate zone.
Engine	The internal core that executes mappings.
Store-and-forward	Buffering values when a target is offline.
Virtual source	Computed point or redundancy group.

---

4. Protocol fundamentals in detail

This chapter explains **every supported protocol** from the ground up: what it is for, how it is built, what a message actually looks like, and what to watch out for during configuration. Read only the sections relevant to your plant.

4.1 About industrial communication

Automation is roughly divided into two worlds:

- **Field level (OT, *Operational Technology*):** sensors, actuators, controllers (PLC). Here real-time behavior, robustness and simple, lean protocols matter (Modbus, CAN).
- **Supervisory/enterprise level (IT):** SCADA, MES, ERP, cloud. Here security, structure and interoperability matter (OPC-UA, MQTT, REST).

Protocols can be grouped by their **communication pattern**:

Pattern	Explanation	Examples
Polling	One party actively requests values, the other answers.	Modbus master, OPC-UA client (read)
Subscription	The server only sends on change.	OPC-UA subscription
Publish/Subscribe	Senders publish, receivers subscribe – decoupled via a broker.	MQTT, Sparkplug B
Broadcast/bus	All participants listen; messages carry an ID.	CAN

The OPC Bridge handles all of these patterns at once and translates between them.

A recurring theme is the **representation of numbers**: depending on the protocol, values are transferred as single bits, as 16-bit registers, as 32-bit floating-point numbers or as bit fields in messages. Section 4.12 covers data types and byte order centrally, because this is the most common source of error during commissioning.

4.2 OPC-DA (Classic)

Origin and purpose. OPC (originally *OLE for Process Control*) emerged from 1996 to standardize data exchange between Windows automation software and controllers. **OPC-DA** (*Data Access*) is its most widely used part – practically every older SCADA system and Windows driver software offers an OPC-DA server. Millions of OPC-DA interfaces are still in use in legacy plants.

Technical structure. OPC-DA is based on Microsoft's component model **COM** and – across machine boundaries – **DCOM**. A server's data is organized hierarchically:

```
OPC-DA server
 └─ Group (e.g. "Line1")
     └─ Item  (e.g. "Channel1.Device1.Tag1")  → value + quality + timestamp
```

An **item** is a single data point. Each item has an **ItemID** (the address, e.g. `Channel1.Device1.Temperature`), a **value**, a **quality** (good/uncertain/bad) and a **timestamp**.

The OPC Bridge acts as an **OPC-DA client**: it connects to a local or remote DA server (via its *ProgID* such as `Matrikon.OPC.Simulation`), creates a group and reads or writes items.

Address format. `Channel1.Device1.Tag1` – the exact notation is defined by the server. The Bridge can **browse** the server's address tree.

Pitfalls.

- OPC-DA runs **only on Windows** (COM/DCOM dependency).
- Remote access (**DCOM**) is notoriously hard to configure. **Recommendation:** run the Bridge and DA server on the **same machine** – then DCOM is not needed at all.
- The **bitness** (32/64-bit) of the Bridge must match the installed *OPC Core Components* wrapper.

> **»Tip«** Modernizing an OPC-DA plant usually means: *read OPC-DA → provide as OPC-UA*. Create an OPC-DA connection (source) and an OPC-UA server connection (target), then link the points via mappings.

4.3 OPC-UA

Origin and purpose. OPC-UA (*Unified Architecture*, from 2008) is the modern successor to OPC-DA. It solves its main problems: it is **platform-independent** (Windows, Linux, controllers, cloud), **secure** (encryption, signing, certificates, user authentication) and **self-describing** (a client can explore the address space and read data types). OPC-UA is today's de-facto standard for connecting control systems (IEC 62541).

Technical structure. OPC-UA data lives in an **address space** of **nodes**. Each node has:

- a **NodeId** – the unique address, consisting of a **namespace index** (`ns=`) and an **identifier**, which is either
  - a **string** (`s=`), e.g. `ns=2;s=Line1.Temperature` , or
  - a **number** (`i=`), e.g. `ns=3;i=1001` ,
- a **BrowseName** and **DisplayName** (readable labels),
- a **node class** (variable, object, method …),
- for variables: a **value** with **data type**, **quality** (StatusCode) and **timestamp**.

The namespace index separates different "namespaces": `ns=0` is OPC-UA's own standard space; `ns=2` , `ns=3` … are vendor-specific spaces.

Two operating modes in the Bridge.

- **OPC-UA client:** the Bridge connects to a foreign server (endpoint URL such as `opc.tcp://192.168.0.10:4840`) and reads/writes nodes. It can also create **subscriptions** (server reports only changes) and subscribe to **Alarms & Events (A&E)**.
- **OPC-UA server:** the Bridge provides its own server (default port 4840). Other systems connect and read the provided values. The path you assign, `Plant/Line1/Temperature` , is automatically turned into a NodeId.

Security. OPC-UA has **security policies** defining signing and encryption:

Policy	Meaning
`None`	no protection (protected networks only)
`Basic256Sha256`	signing/encryption (common standard)
`Aes128_Sha256_RsaOaep`, `Aes256_Sha256_RsaPss`	modern policies

Plus the **MessageSecurityMode**: `None`, `Sign` (signed only) or `SignAndEncrypt` (signed and encrypted). Partners exchange **certificates** and must trust each other.

Alarms & Events (A&E). A UA server can deliver not only values but also **events** (alarms, state changes). The Bridge's UA client can subscribe to these – filtered by source, event type and minimum severity.

> »**Note**« Because OPC-UA is self-describing, **browsing** is especially powerful here: the Bridge can traverse the server's entire address space and bulk-create points via the wizard.

4.4 Modbus TCP and RTU

Origin and purpose. Modbus (1979, Modicon) is one of the oldest and simplest industrial protocols – and precisely for that reason extremely widespread: PLCs, energy and heat meters, drives, inverters, sensors and actuators almost always offer Modbus. Its simplicity makes it robust and easy to diagnose.

Variants.

- **Modbus RTU:** over a serial line (RS-485 or RS-232). Several devices on one bus, each with a **device address (unit ID)** 1…247. Binary transmission with CRC check.
- **Modbus TCP:** over Ethernet/IP (default port **502**). Same logic but as a TCP packet; the unit ID is carried along (relevant with gateways to serial devices).

Data model. Modbus has exactly **four data areas**:

Area	Code	Width	Content	Access
Coils	`CO`	1 bit	switch outputs	read + write
Discrete Inputs	`DI`	1 bit	digital inputs	read only
Input Registers	`IR`	16 bit	measured values	read only
Holding Registers	`HR`	16 bit	setpoint/parameter values	read + write

Each area is numbered (address 0, 1, 2 …). In older **Modicon notation** the area is encoded by the number range: `0xxxx` = coils, `1xxxx` = discrete inputs, `3xxxx` = input registers, `4xxxx` = holding registers (e.g. `40001` = first holding register). The Bridge understands both notations.

Message structure (simplified). A Modbus request “read 2 holding registers from address 100 of device 1” consists of:

```
[Unit ID=01] [Function=03 (Read Holding Registers)] [Start=00 64] [Count=00 02]
```

The response contains the byte count and the register contents:

```
[Unit ID=01] [Function=03] [Byte count=04] [Reg100=41 20] [Reg101=00 00]
```

Numbers across multiple registers. A register holds only **16 bits** (0…65535). Larger values combine several registers:

- **32-bit integer** (`int32` / `uint32`) or **32-bit float** (`float32`): 2 registers.
- **64-bit** (`int64` / `uint64` / `float64`): 4 registers. Note: in Modbus addresses the 64-bit floating-point keyword is **`float64`**, not `double` – `double` is not a recognized Modbus type abbreviation.
- **Text** (`string`): several registers, 2 characters each.

The **byte order** is crucial here – see section 4.12 and the **register inspector** (chapter 35), which shows all interpretations at once.

Roles in the Bridge. For TCP **and** RTU the Bridge can be:

- **Master** (client): actively polls registers from a device and writes them.
- **Slave** (server): provides its own registers that other masters can poll. This lets the Bridge, for example, expose OPC-UA values as Modbus registers “outward”.

Address format in the Bridge. `[Unit:]Area Offset[.Bit][:Type[:ByteOrder]]`

Examples: `HR100:float32`, `2:IR10:int32:CDAB`, `CO5`, `HR7.3` (bit 3 of a register), `40001:uint16`, `HR200:string20`.

4.5 MQTT

Origin and purpose. MQTT (*Message Queuing Telemetry Transport*, 1999, originally for pipeline telemetry) is the standard protocol of the **Internet of Things**. It is extremely lean and optimized for **unreliable, low-bandwidth networks** (cellular, satellite). Ideal for connecting many distributed devices to a central place.

Communication pattern: publish/subscribe. Unlike Modbus, nobody polls directly. Instead there is a central **broker** (server). Devices **publish** values under a **topic**, and interested parties **subscribe** to that topic:

```
Sensor  --publish "hall1/temp" = 21.4-->  [ Broker ]  --deliver-->  subscriber(s)
```

Advantage: sender and receiver do not know each other and need not be online at the same time – they are **decoupled** via the broker.

Topics. A topic is a hierarchical path with `/` as separator, e.g. `plant/hall1/line3/temperature`. Wildcards are possible when subscribing (`+` for one level, `#` for all remaining).

Payload. The message content is free – often a number, frequently **JSON**, e.g. `{"temp": 21.4, "unit": "C"}`. The Bridge can extract a specific value from the JSON by appending a **JSON path** with `@`: `sensor/abc@data.temperature`.

Quality of Service (QoS). MQTT has three delivery guarantees: QoS 0 (at most once), QoS 1 (at least once), QoS 2 (exactly once). Higher QoS = safer but more overhead.

Retained messages & last will. A *retained message* is stored by the broker so new subscribers immediately receive the last value. The *last will* is a message the broker sends automatically if a device loses its connection unexpectedly.

Address format in the Bridge. `hall1/temp` or with JSON path `sensor/abc@data.temperature`.

4.6 MQTT Sparkplug B

The problem Sparkplug solves. "Bare" MQTT defines **no structure**: every vendor names topics differently and encodes payloads differently. In large plants this leads to chaos. **Sparkplug B** is an **open specification on top of MQTT** that prescribes topic structure, payload format (Google Protocol Buffers) and a fixed **lifecycle**. This creates a **Unified Namespace (UNS)** – a consistent, searchable real-time model of the entire plant. Sparkplug B is supported by Ignition, HiveMQ, AVEVA and many IIoT platforms.

Roles. Sparkplug has **edge nodes** (data sources at the plant edge) and **host applications** (e.g. SCADA). The OPC Bridge acts as an **edge node**.

Topic structure. `spBv1.0/<GroupID>/<MessageType>/<EdgeNodeID>`, e.g. `spBv1.0/Plant1/NDATA/Line3`.

Lifecycle (the heart of it).

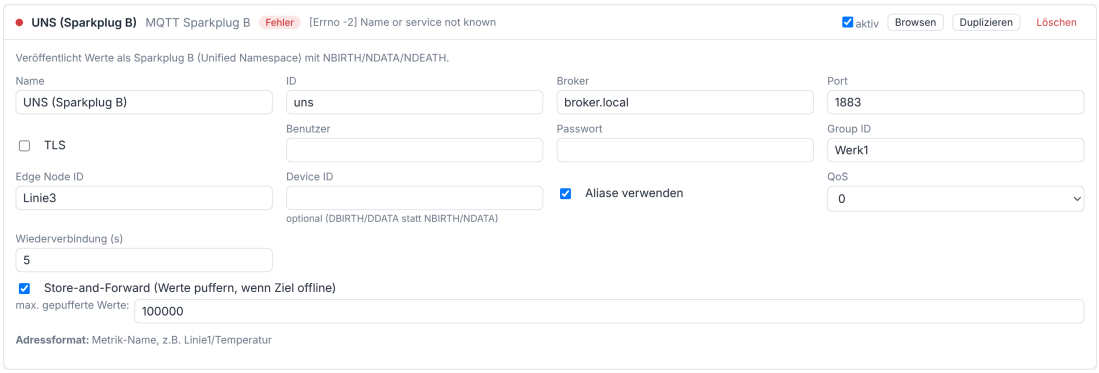
Message	Meaning
NBIRTH (Node Birth)	On connect, the node announces **all** its metrics with name, data type and a short **alias number**. Contains `bdSeq` (birth/death counter).
NDATA (Node Data)	Afterwards, changes are sent using only the **alias number** (without the long name) – saving bandwidth significantly.
NDEATH (Node Death)	The "death certificate" registered as the *last will*. If the node fails, the broker sends it automatically – all subscribers immediately know the data is stale.

Message	Meaning
NCMD (Node Command)	Commands to the node, including *Rebirth* (request a new birth message) and write values.

Each message carries an incrementing **sequence number (seq)**; gaps indicate loss. The combination of `bdSeq` and `seq` makes the data stream traceable.

What the Bridge does.

- Sends **NBIRTH** on connect with all write points as metrics (including aliases and `Node Control/Rebirth`).
- Sends changes as **NDATA** (alias + value + sequence only).
- Registers **NDEATH** as the last will.
- Answers **Rebirth** requests and writes **NCMD** values back into the plant (bidirectional).



Sparkplug B

Data type mapping. Generic types are mapped automatically to Sparkplug types (Int8/16/32/64, UInt…, Float, Double, Boolean, String).

4.7 CAN bus

Origin and purpose. CAN (*Controller Area Network*, Bosch, 1986) is the dominant bus in vehicles, construction machinery, generators, medical and industrial equipment. It is robust, real-time capable and needs only two wires. The bus carries many small messages (**frames**) that all participants listen to.

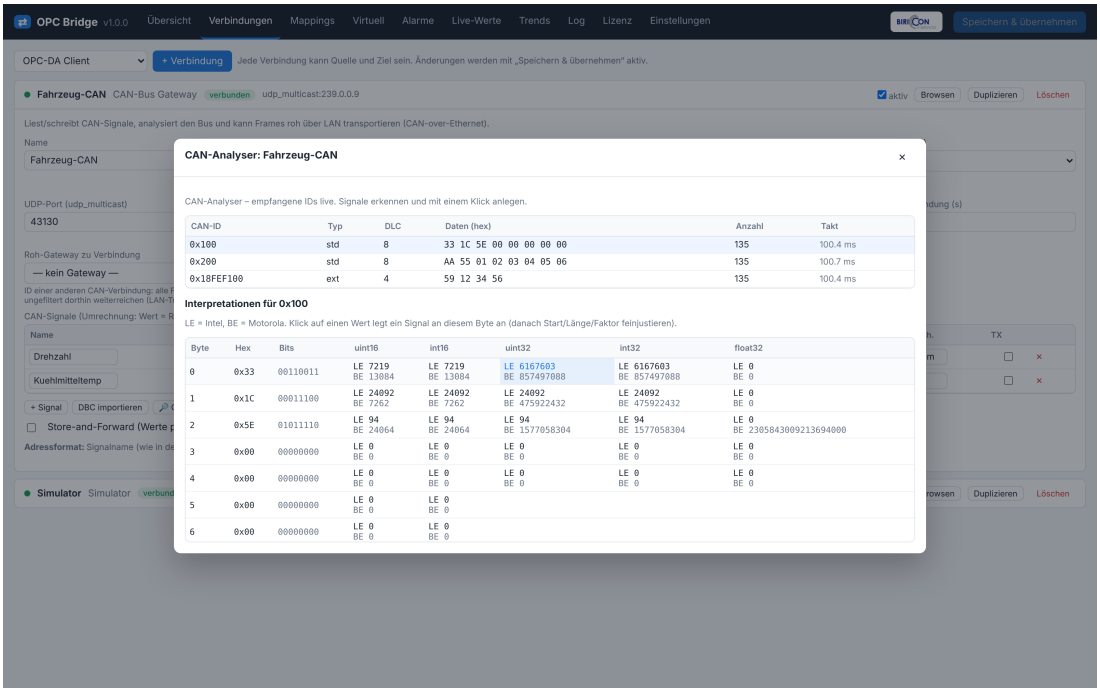
Frame structure. A CAN data frame essentially contains:

Field	Size	Meaning
Identifier (ID)	11 bit (standard) or 29 bit (extended)	identifies the message and its priority (lower ID = higher priority)
DLC	4 bit	number of data bytes (0…8)
Data	0…8 bytes	the actual payload

Field	Size	Meaning
CRC	15 bit	checksum

Importantly, CAN transmits **no signal names**. The actual values (**signals**) sit as **bit fields** in the up to 8 data bytes. Which bit is which signal, with which length, factor and offset, is defined either

- in a **DBC file** (the standard description format; the Bridge can import DBC), or
- must be determined with the Bridge's **CAN analyzer** (listen, recognize patterns).



CAN analyzer

Signal definition. A signal is described by: start bit, length (bits), byte order (*Intel/little* or *Motorola/big*), sign, **factor** and **offset**. The physical value is `raw * factor + offset`. Example: a 16-bit raw value with factor 0.1 and offset −40 gives, for raw value 850, the value `850*0.1−40 = 45.0 °C`.

Interfaces. The Bridge speaks CAN over various hardware/transports: **SocketCAN** (Linux), **UDP multicast** (CAN-over-Ethernet), **slcan** (serial CAN adapters), **PCAN** (Peak), **Kvaser**, **Vector** and a **virtual** bus for testing.

Two operating modes.

- **Signal extraction:** extract signals from frames, convert via factor/offset and export to OPC/Modbus/MQTT.
- **Raw gateway:** transport the entire bus **unfiltered over LAN** (CAN-over-Ethernet), e.g. to couple two remote CAN segments over Ethernet.

4.8 J1939

Purpose. SAE J1939 is a CAN-based standard for commercial vehicles, engines and generator sets (diesel generators, construction and agricultural machinery). It defines **how** the 29-bit ID and the data bytes are structured so that devices from different vendors interoperate.

Structure of the 29-bit ID. The extended ID is structured into:

Part	Meaning
Priority	3 bit – bus priority
PGN (Parameter Group Number)	identifies the *kind* of message (e.g. “engine speed”)
SA (Source Address)	address of the sending device

The **PGN** points, in the J1939 specification, to a fixed list of **SPNs** (Suspect Parameter Numbers) – the individual measured values in the data bytes. Example: PGN 61444 (EEC1) contains the engine speed among others.

What the Bridge does. The CAN analyzer detects extended frames and **automatically decodes PGN, SA, priority and destination address** and displays them. This makes it quick to identify J1939 messages without a DBC.

4.9 SQL / ODBC

Purpose. Process values often need to be written to a **relational database** (logging, reporting) or setpoints read from one. Supported databases include Microsoft SQL Server, PostgreSQL, MySQL/MariaDB and SQLite.

Structure. The Bridge uses a database abstraction layer (SQLAlchemy, optionally ODBC). You configure:

- a **read query** (SELECT) delivering address and value, and/or
- a **write statement** (INSERT/UPDATE) with **placeholders** `:addr`, `:value`, `:ts`.

Thus each forwarded value is stored or read as a record. Intervals and connection parameters (connection string) are configurable.

4.10 REST / Webhook

Purpose. To send values to arbitrary web services, low-code platforms, cloud endpoints or custom microservices.

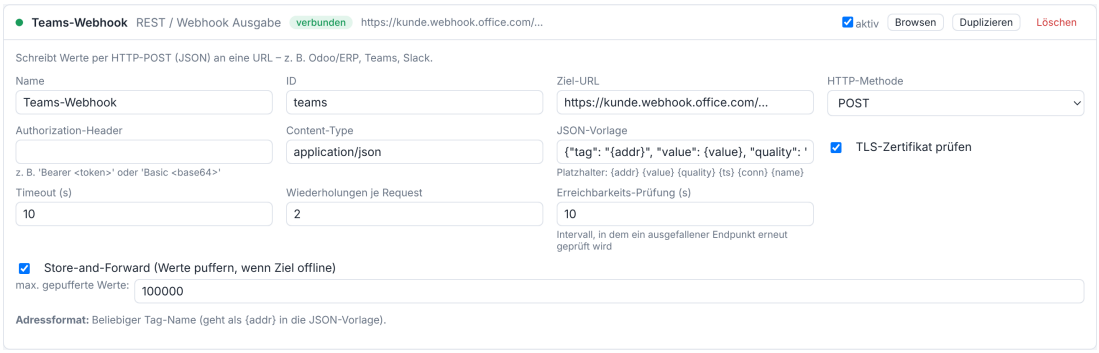
Structure. For each value to forward, the Bridge sends an **HTTP POST request** to a configured URL. The body is a freely definable **JSON template** with placeholders:

Placeholder	Content
`{addr}`	point address
`{value}`	value
`{quality}`	quality
`{ts}`	timestamp (ISO-8601)
`{conn}`	connection **name** (not the technical ID – counter-intuitive, but verified in code)
`{name}`	the point's **address** (again counter-intuitive: despite the name, this is the address, not the connection name)

> »**Note**« `{conn}` and `{name}` are swapped relative to what their names suggest: `{conn}` renders the connection's display *name*, and `{name}` renders the point's *address*. Use `{addr}` if you specifically want the address – it is equivalent to `{name}` but less confusing to read in a template.

Default template (used when the field is left at its default): `{"tag": "{addr}", "value": {value}, "quality": "{quality}", "timestamp": "{ts}"}`.

Combined with **store-and-forward**, no values are lost when the endpoint is temporarily unreachable.



Webhook output with store-and-forward

4.11 Bridge-Link (OT→IT tunnel)

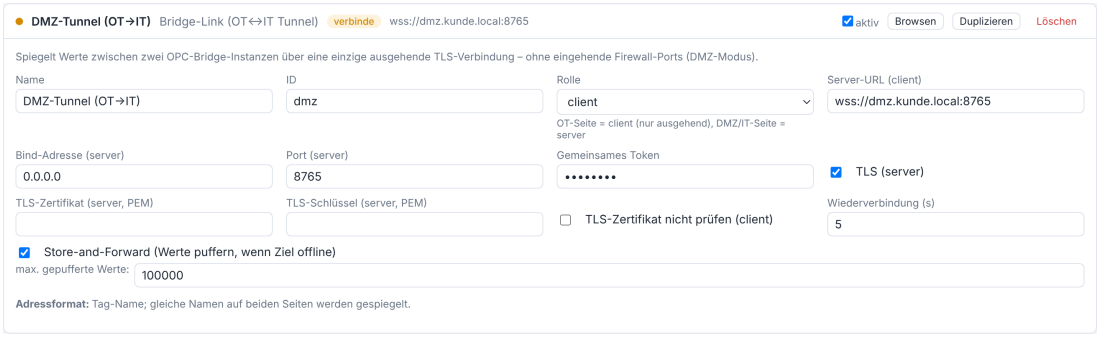
The security problem. In a properly segmented plant (Purdue model), no connection from outside may **enter** the **production network (OT)**. Firewalls allow only **outbound** connections. Yet process values must reach the **IT/DMZ zone**.

The solution. Two OPC Bridge instances are coupled via the **Bridge-Link**:

- The **OT side** runs as a **client** and establishes the connection **outbound only** – a single, **TLS-encrypted** connection to the DMZ.
- The **IT/DMZ side** runs as a **server** and accepts this connection.

- Values with the **same tag name** are mirrored **in both directions**.

No **inbound port** needs to be opened in the OT firewall. A **shared token** secures the coupling; **store-and-forward** on the OT side buffers on connection loss.



Bridge-Link / DMZ mode

4.12 Data types and byte order

This is the **most important practical fundamental**, because wrong data types or byte orders are the most common cause of “nonsense values”.

Data types. A value can have different types:

Type	Description	Range
`bool`	truth value	true/false
`int16` / `uint16`	16-bit integer (signed/ unsigned)	−32768…32767 / 0…65535
`int32` / `uint32`	32-bit integer	±2 bn / 0…4 bn
`int64` / `uint64`	64-bit integer	very large
`float32`	32-bit floating point (IEEE 754)	approx. $\pm 3.4 \cdot 10^{38}$
`float64`	64-bit floating point (also called “double precision”)	high precision
`string`	text	depending on length

> »**Note**« `double` is **not** a valid type keyword in Modbus/S7 addresses or the REST API – the correct abbreviation for a 64-bit float is always **`float64`**. “Double precision” is only the generic/Sparkplug-B type name shown in some places in the UI.

Why byte order matters. A 32-bit number consists of 4 bytes (A, B, C, D). Different devices store these bytes in different orders. With Modbus (two 16-bit registers), four arrangements are common:

Byte order	Order	Also called
`ABCD`	big-endian	“Motorola”, standard
`DCBA`	little-endian	“Intel”
`CDAB`	big-endian, word-swapped	word swap (very common!)
`BADC`	little-endian, byte-swapped	byte swap

Worked example. The floating-point value **45.0** is, in IEEE-754, the four bytes `42 34 00 00`. Depending on the device’s byte order it appears in the two Modbus registers as:

- `ABCD`: Reg0=`4234`, Reg1=`0000`
- `CDAB`: Reg0=`0000`, Reg1=`4234` (word swap)
- `BADC`: Reg0=`3442`, Reg1=`0000` (byte swap)
- `DCBA`: Reg0=`0000`, Reg1=`3442`

Reading with the wrong order yields a completely different number. That is exactly what the **register inspector** (chapter 35) is for: it shows all interpretations at once so you can spot the correct one immediately.

> »**Caution**« If a Modbus value “goes crazy” (e.g. extremely large numbers or zero), the byte order or data type is almost always wrong. Check it with the register inspector.

Part II – Installation and commissioning

5. System requirements

Component	Minimum	Recommended
CPU	1 core, x86-64 (amd64)	2+ cores
RAM	512 MB	1–2 GB
Storage	500 MB + space for history	SSD, depending on history volume
Operating system	Windows 10/11, Windows Server 2016+, Debian 11+, Ubuntu 20.04+	current LTS version
Network	access to the devices to connect	separate OT/IT segments

OPC-DA strictly requires **Windows** (COM/DCOM). **CAN over SocketCAN** requires **Linux**. All other protocols run cross-platform. The delivered packages are **compiled and obfuscated** and require **no Python** installed on the target system.

> »**Note**« The shipped Windows setup, Debian package and Docker image are currently built for **x86-64 (amd64) only**. There is **no official Raspberry Pi/ARM build** yet. The Bridge's Python source is platform-independent and will run on Debian/Ubuntu on ARM (e.g. a Raspberry Pi) if you build the package yourself for that architecture, but this is not part of the shipped releases.

---

6. Installation on Windows

Step by step:

1. Log in as **administrator**.
2. Run `OPCBridge-Setup-<version>.exe`.
3. Follow the wizard. The setup
   - installs the program (default: `C:\Program Files\OPCBridge\`),
   - creates and starts the **Windows service "OPCBridge"**,
   - creates the data directory under `%ProgramData%\OPCBridge\`,
   - shows the **initial password** at the end.
4. Open the browser: `http://localhost:8080/` (or from another machine `http://<hostname>:8080/`).
5. Log in with user `admin` and the shown password. It is also stored in `%ProgramData%\OPCBridge\initial_admin_password.txt`.

6. Change the password under *Settings* → *Users*.

Managing the service. Use the Windows service manager (`services.msc`) to start, stop and set the startup type of the “OPCBridge” service.

> **»Note on OPC-DA«** Install the matching *OPC Core Components* if needed. The bitness must match. Run the Bridge and DA server on the same machine where possible to avoid DCOM.

Firewall. Open the web port (default 8080) for access from other machines. Open the ports for OPC-UA server (4840), Modbus slave (502) or Bridge-Link (8765) if enabled.

---

7. Installation on Debian/Ubuntu

Installation:

```
sudo apt install ./opcbridge_1.1.0_amd64.deb
```

The package installs the program, creates the service user `opcbridge`, creates and starts the **systemd service** `opcbridge`, and creates the data directory `/var/lib/opcbridge/`.

Access. Open `http://<host>:8080/`. The initial password is in `/var/lib/opcbridge/initial_admin_password.txt`.

Managing the service:

```
systemctl status opcbridge      # status
systemctl restart opcbridge     # restart
systemctl stop opcbridge        # stop
journalctl -u opcbridge -f      # follow the log live
```

For CAN (SocketCAN): the CAN interface must be configured on the host, e.g.:

```
sudo ip link set can0 up type can bitrate 250000
```

Uninstall: `sudo apt remove opcbridge` (data under `/var/lib/opcbridge/` is kept; `purge` removes it).

---

8. Installation as a Docker container

Prerequisite: Docker and Docker Compose are installed.

Start:

```
docker compose up -d
```

The supplied `docker-compose.yml` publishes the ports **web 8080**, **OPC-UA 4840**, **Modbus slave** `1502:502` (host port 1502 → container port 502 – the *internal* Modbus slave still listens on the standard port 502, but on the host it is reachable at **1502**, not 502, because 502 is a privileged port the container should not need host-level privileges for) and **Bridge-Link 8765**. It mounts a **named Docker volume** `opcbridge-data` at `/data` inside the container, holding `config.json`, `license.key`, the history and the other state files (Appendix F) – **not** a `./data` bind mount. A named volume lets the image's own `chown` to the unprivileged container user take effect; a plain bind mount such as `./data:/data` would be created as `root` by Docker and the container (which runs as a non-root UID) could fail to write to it on first start.

Useful commands:

```
docker compose logs -f          # follow the log
docker compose restart          # restart
docker compose down             # stop and remove (volume opcbridge-data kept)
```

CAN in the container. For SocketCAN start the container with host networking and capability (prepared in the compose file): `network_mode: host` and `cap_add: [NET_ADMIN]`.

Build your own image (optional): `docker compose build` builds the image from the supplied `Dockerfile`.

---

9. First commissioning step by step

This quick guide gets you to a first working data flow in minutes – **without any real hardware**, using the built-in **Simulator** connection type, before you connect a real device.

1. **Log in.** Open the web interface, log in with `admin` and the initial password.
2. **Change the password.** *Settings* → *Users*, enter a new password, **Save & apply**.
3. **Create the first source: a Simulator connection.** *Connections* tab → choose type *Simulator* → **+ Connection** → expand the card and set a poll interval (e.g. 1000 ms, default). The Simulator needs no device at all – it generates values you can map and watch immediately. It exposes ready-made points `Sine`, `Ramp`, `Random` (all numeric/`Double`), `Counter` (`Int32`), `Toggle` (`Boolean`, flips every 5 s) and `Clock` (text), plus any freeform point name you type (e.g. `Memory/Setpoint`) for manual write testing.
4. **Create the first target.** Add a second connection, e.g. another *Simulator* connection (to see values written back without touching real equipment) or, once you are ready, a real *OPC-UA Server*, *Modbus TCP Master*, etc.
5. **Create a mapping.** *Mappings* tab → **+ Mapping** → choose source (A) connection/address (e.g. Simulator / `Sine`) and target (B) connection/address → direction *A* → *B*.

6. **Save & apply.** Top right. The engine restarts briefly.
7. **Check.** *Live Values* tab: does the value arrive and change every second? *Overview* tab: are the connections green?

> »**Tip**« `config.example.json` at the root of the installation is a ready-to-use starting configuration that already wires up a Simulator connection (`sim`), a Modbus TCP master (`mbm`) and a few history points – copy it over your `config.json` in the data directory (service stopped) if you want a working example to explore or adapt, instead of starting from an empty configuration.

> »**Remember**« *Nothing is active until you click “Save & apply” at the top right.* All changes to connections, mappings and settings only take effect with this click (the engine restarts briefly).

Part III – Monitoring and daily use

10. Web interface layout

The web interface is organized into **tabs** at the top. Top right is the **“Save & apply”** button and – when there are unsaved changes – a notice and a button to discard.

Tab	Purpose	Chapter
Overview	status of all connections and mappings	11
Connections	create and configure devices/ systems	15–27
Mappings	map values to each other	28
Virtual	computed points and redundancy	30
Alarms	notification rules, OPC-UA events, SMTP settings	33
Live Values	current values, manual writing	12
Trends	historical values as charts	13
Log	operating messages	14
License	license status and activation	41
Settings	web interface, users, history, WebView, online license, updates	32, 37

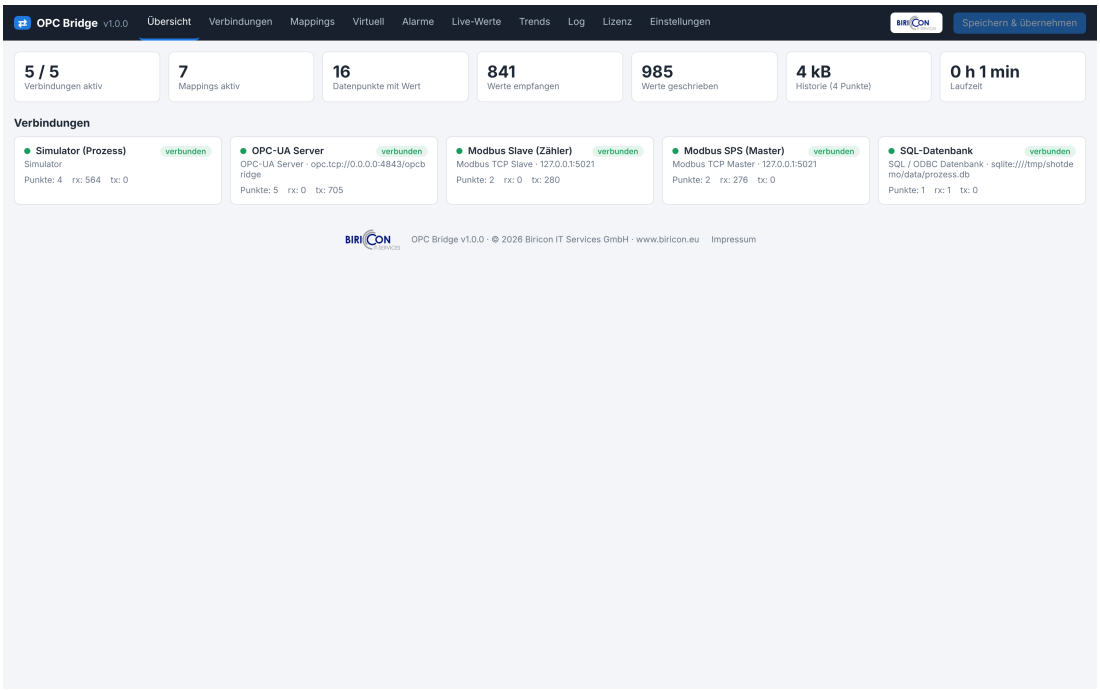
The **company logo** at the top right links to www.biricon.eu; the imprint provides contact details.

Language selection. The web interface is **multilingual**. A language selector at the top right switches between **German, English, French, Spanish and Portuguese**; the choice is stored in the browser (per browser/device, not in `config.json`). German is the source language for the interface text: if a translation is missing for a given label, the German term is shown as a fallback, so further translations can be completed incrementally without breaking the UI.

---

11. Overview tab

The overview shows the Bridge’s health at a glance:



Overview

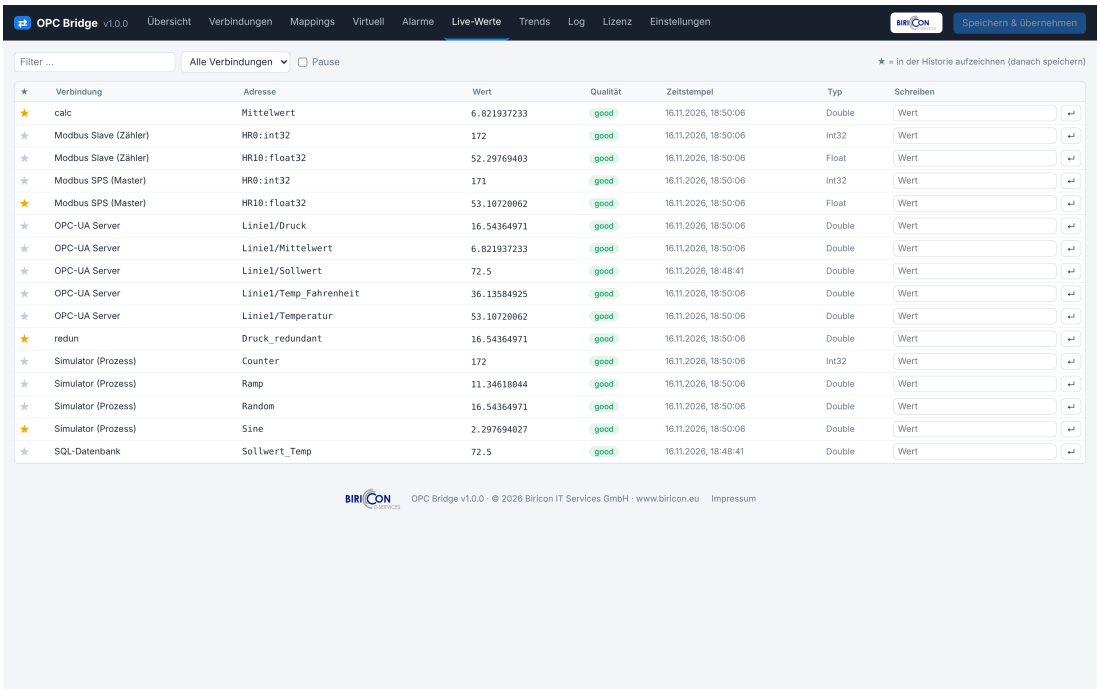
- **Connections:** number and state (connected/connecting/error/stopped), each shown with color and status message.
- **Mappings:** number of active mappings.
- **Store-and-forward:** buffered values per target connection (if active).
- **License:** current status (licensed/demo/remaining time).

Values update **in real time** (see chapter 38).

---

12. Live Values tab

Here you see the **current values** of all monitored points in real time – with value, quality and timestamp.



Live Values

Functions:

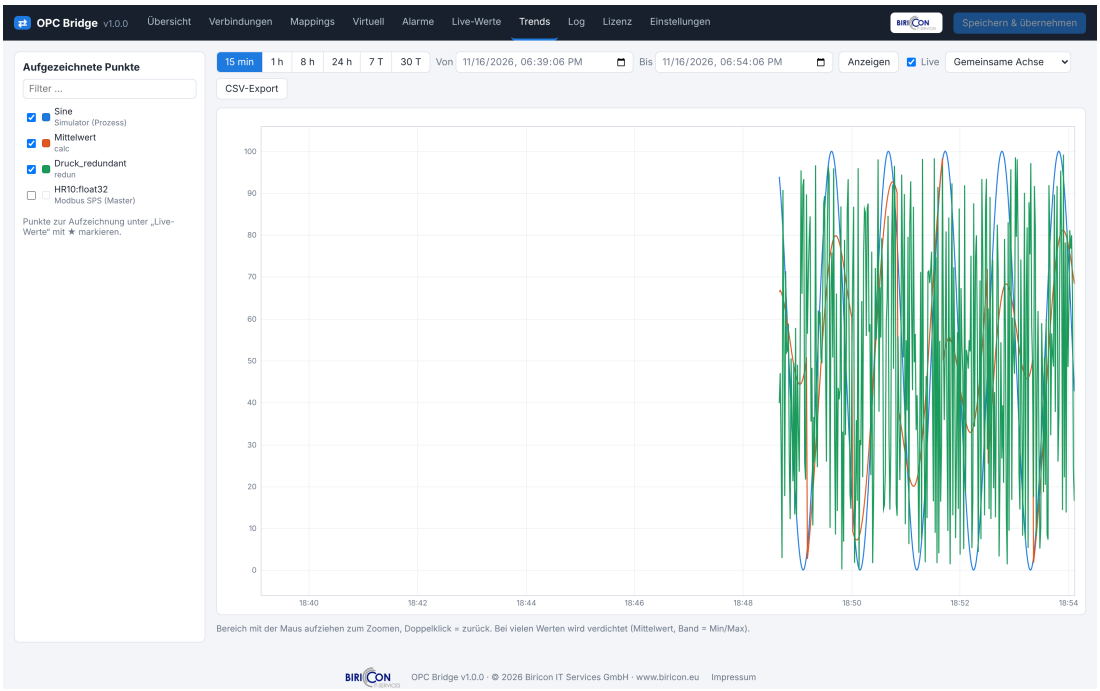
- **Pause:** freezes the display (data transfer continues in the background).
- **Manual writing:** for writable target points you can enter a value directly for testing and send it. Very useful during commissioning to exercise actuators or registers specifically.
- **Filter/search:** narrow points by name or connection.

> »**Tip**« Use live values together with manual writing to test a new mapping chain before connecting the real source.

---

13. Trends tab

The trend shows **historical values as a chart** over time.



Trends

- **Choose points:** which recorded points are shown.
- **Time range:** e.g. last hour, last day, freely selectable.
- **CSV export:** download the displayed data as a CSV file (e.g. for Excel).

This requires the **history** to be enabled and the points selected for recording (chapter 32).

---

14. Log tab

The log shows **operating messages** of the Bridge: connections up and down, errors, warnings. It is the **first place to look when problems occur**.

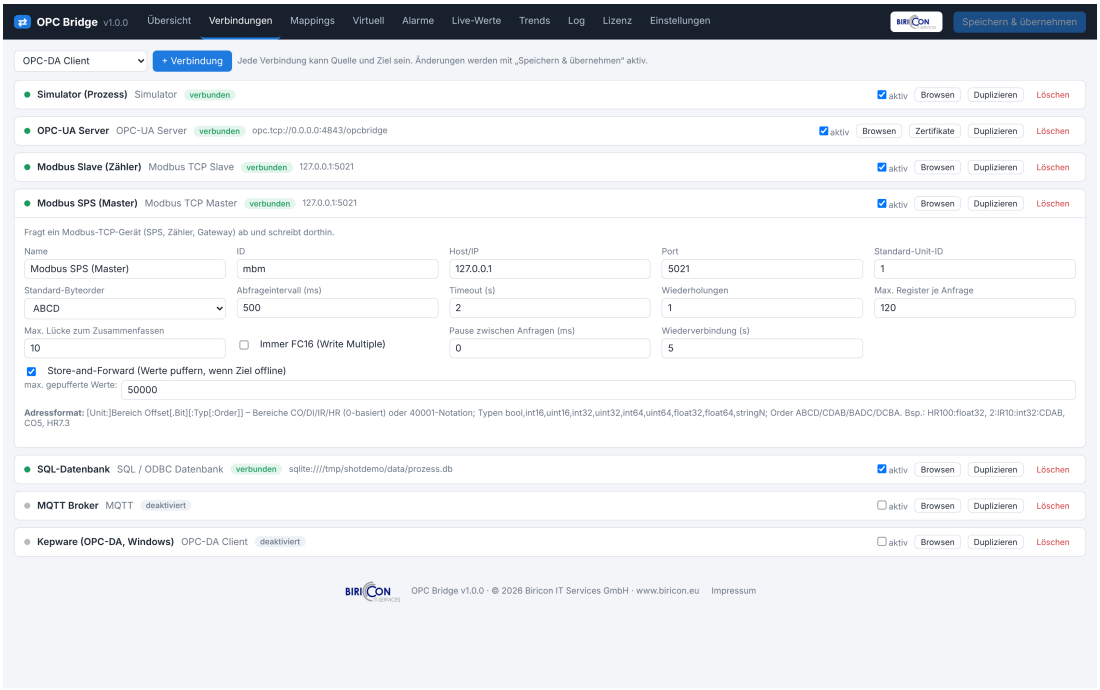
- Messages are sorted by time and marked by severity.
- On connection errors the concrete cause is shown here (e.g. “timeout”, “connection refused”, “unknown address”).

At system level the same messages are additionally written to log files in the data directory (Appendix F): `opcbridge.log` (the application log, equivalent to what the Log tab shows) and `service.log` (output of the service/process wrapper itself, e.g. start/stop and crashes before the application log is available). On Linux these are also visible via `journalctl -u opcbridge`. There is no Windows Event Log integration.

Part IV – Configuration

15. Connections – general

In the **Connections** tab you create each link to a device or system.



Connections

Creating a new connection:

1. Choose the **type** (the protocol) from the list at the top.
2. Click **+ Connection** – a new card appears.
3. **Expand** the card by clicking its header.
4. Fill in the fields (each type shows exactly the fields it needs – see the following chapters).
5. **Save & apply**.

Controls on each card:

Element	Function
Name	free label (shown everywhere in the UI)
ID	technical identifier (automatic, referenced in addresses/mappings)
active	enables/disables the connection
Browse	search the device for available points (if the protocol allows)
Duplicate	create a copy of the card
Delete	remove the connection

Element	Function
Store-and-forward	enable buffering (chapter 31)

The following chapters describe the **fields per connection type** in full. The *Default* column gives the default value.

---

16. Connection type OPC-DA

Role: client (reads/writes items of an OPC-DA server). **Windows only.**

Field	Meaning	Default
Server ProgID	identifier of the DA server, e.g. `Matrikon.OPC.Simulation`	–
Host	hostname/IP for a remote server (empty = local, recommended)	empty
Update rate (ms)	how often items are read	1000
Read source	`cache` (server buffer, fast) or `device` (direct from device)	cache
Automation wrapper	path/ProgID of the OPC automation wrapper (if needed)	empty
Reconnect (s)	wait time before reconnecting	10
Query data type on browse	also determine the type when browsing	on

Address format: `Channel1.Device1.Tag1` (defined by the server; browsable).

> »**Caution**« Remote access requires DCOM configuration. Recommendation: run the Bridge locally on the DA server machine.

---

17. Connection type OPC-UA Client

Role: client (reads/writes nodes of a foreign UA server, optional A&E).

Field	Meaning	Default
Endpoint URL	address of the UA server, e.g. `opc.tcp://192.168.0.10:4840`	`opc.tcp://localhost:4840`
Security		None

Field	Meaning	Default
	security policy and mode (see 4.3)	
User / Password	log in to the server (if required)	empty
Publishing interval (ms)	how often the server reports changes	500
Sampling interval (ms)	how often the server samples internally	250
Timeout (s)	timeout	10
Reconnect (s)	wait before retry	10
Subscribe Alarms & Events (A&E)	receive events from the server	off
Event source (NodeId)	source of events (default: server object `i=2253`)	`i=2253`
Event type (NodeId)	event type to subscribe (`i=2041` = BaseEventType)	`i=2041`
Minimum severity (1–1000)	only events from this severity	1

Address format: `ns=2;s=Line1.Temp` or `ns=3;i=1001`. Browsable.

---

18. Connection type OPC-UA Server

Role: server (provides its own values that others read/write).

Field	Meaning	Default
Bind address	network interface (`0.0.0.0` = all)	0.0.0.0
Port	UA port	4840
Endpoint path	path part of the endpoint URL	`/opcbridge`
Server name	displayed name	OPC Bridge
Namespace URI	identifier of the own namespace	`urn:opcbridge`
Root folder	optional parent folder in the address space	empty
Security policies	allowed policies (multiple)	None, Basic256Sha256 SignAndEncrypt
Client certificates	`trust_all` or `truststore` (only stored ones)	trust_all

Field	Meaning	Default
Anonymous access	`off` / `read` / `readwrite`	read
Users	user list with permissions	–

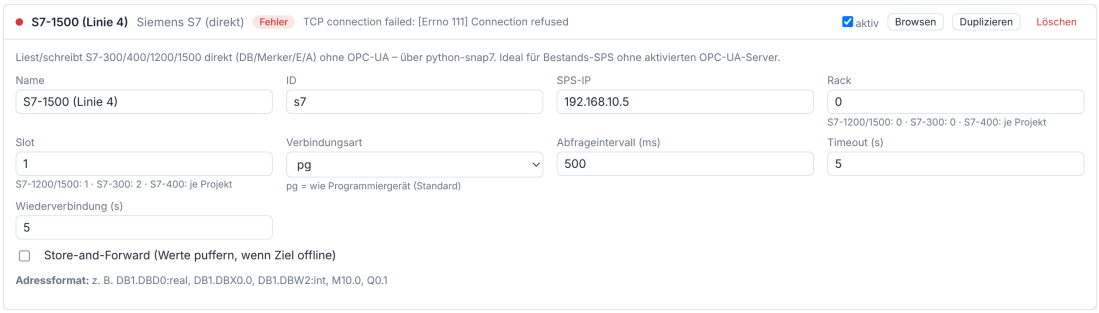
Addressing: the path `Plant/Line1/Temperature` you set on the target point is automatically turned into a NodeId `ns=<idx>;s=Plant/Line1/Temperature`.

> »**Caution**« `trust_all` and `anonymous=readwrite` are only acceptable in protected networks. For production use a certificate truststore and users.

---

19. Connection type Siemens S7

Role: direct client access to Siemens S7-300/400/1200/1500 controllers over **RFC1006/ISO-on-TCP** – reads/writes **data blocks (DB)**, **flags/merkers (M)**, **inputs (I/E)** and **outputs (Q/A)** directly, **without** OPC-UA having to be enabled or licensed on the PLC. In existing plants this is often the fastest way to connect an S7. Requires the **python-snap7** library (bundled in the delivered package/image).



Siemens S7 connection

Field	Meaning	Default
PLC IP	address of the controller	192.168.0.1
Rack	S7 rack number	0 (S7-1200/1500: 0 · S7-300: 0 · S7-400: per project)
Slot	S7 slot number	1 (S7-1200/1500: 1 · S7-300: 2 · S7-400: per project)
Connection type	`pg` (programming device, default), `op` (operator panel) or `basic`	pg
Poll interval (ms)	how often to read	1000
Timeout (s)	timeout	5
Reconnect (s)	wait before retry	5

Address format:

```
 DB<n>.DBX<byte>.<bit>   bool    e.g. DB1.DBX0.0
DB<n>.DBB<byte>         byte    e.g. DB1.DBB2
DB<n>.DBW<byte>         int     e.g. DB1.DBW4
DB<n>.DBD<byte>         real (default) — or :dint/:udint/:dword   e.g. DB10.DBD20
M<byte>.<bit> / MB / MW / MD        flags/merkers
I<byte>.<bit> (also written E) · Q<byte>.<bit> (also written A)   inputs/outputs
```

Optional type suffix:
`:bool :byte :int :uint :word :dint :udint :dword :real :string<n>`. Without a suffix, `DBX` / `.bit` defaults to `bool`, `DBB` /byte addresses default to `byte`, `DBW` /word addresses default to `int`, and **`DBD` /double-word addresses default to `real`** – the same default applies to the bare `MD` / `QD` / `ID` double-word forms. Examples: `DB10.DBD20:real`, `DB10.DBW0:int`, `M10.0`, `DB1.DBD0:dint`, `DB5.DBB0:string16`.

---

20. Connection type Modbus

There are four Modbus connection types: **TCP Master**, **RTU Master**, **TCP Slave**, **RTU Slave**.

20.1 Modbus TCP Master

Field	Meaning	Default
Host/IP	address of the Modbus device	192.168.0.10
Port	TCP port	502
Default unit ID	device address (relevant with gateways)	1
Default byte order	default for multi-register values	ABCD
Poll interval (ms)	how often to read	1000
Timeout (s)	timeout	2
Retries	retries on error	1
Max registers per request	limits the block size	120
Max gap to combine	combines nearby registers into one request	10
Always FC16 (Write Multiple)	also write single values with function 16	off
Delay between requests (ms)	relieves slow devices	0
Reconnect (s)	wait before retry	5

20.2 Modbus RTU Master

Like TCP Master but over the serial line. Additional/differing fields:

Field	Meaning	Default
Serial port	e.g. `COM1` (Windows) or `/dev/ttyUSB0` (Linux)	COM1

Field	Meaning	Default
Baud rate	1200…115200	19200
Parity	N/E/O (none/even/odd)	E
Data bits	7 or 8	8
Stop bits	1 or 2	1

20.3 Modbus TCP Slave

Field	Meaning	Default
Bind address	network interface	0.0.0.0
Port	TCP port	502
Default unit ID	own device address	1
Allowed unit IDs	additional accepted IDs (e.g. `1,2,5`)	empty
Default byte order	default for multi-register values	ABCD

20.4 Modbus RTU Slave

Like TCP Slave but serial (fields *Serial port*, *Baud rate*, *Parity*, *Data bits*, *Stop bits* as for the RTU master).

Address format (all Modbus types): `[Unit:]Area Offset[.Bit][:Type[:ByteOrder]]`, e.g. `HR100:float32:CDAB`, `2:IR10:int32`, `CO5`, `HR7.3`, `40001:uint16`, `HR200:string20`.

---

21. Connection type MQTT

Role: publish/subscribe at an MQTT broker.

Field	Meaning	Default
Broker	hostname/IP of the broker	localhost
Port	1883 (plain) / 8883 (TLS)	1883
TLS	enable encryption	off
Do not verify TLS certificate	for self-signed certificates	off
CA file	path to the CA for certificate verification	empty
User / Password	broker login	empty
Client ID	unique identifier (empty = automatic)	empty
QoS	delivery guarantee 0/1/2	0
Retain on publish	keep last value in the broker	on
Payload format (publish)	`raw` (value only) or `json`	raw

Field	Meaning	Default
Topic prefix	prepended to all topics	empty
Browse filter	topic filter when browsing (`#` = all)	`#`
Reconnect (s)	wait before retry	5

Address format: `hall1/temp` ; extract a JSON value with `sensor/abc@data.temperature` .

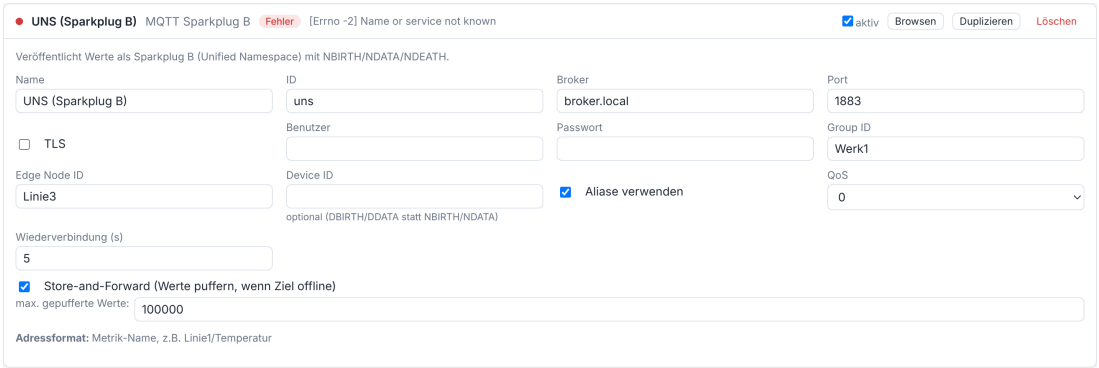
---

22. Connection type Sparkplug B

Role: Sparkplug B edge node (UNS) at an MQTT broker.

Field	Meaning	Default
Broker / Port	MQTT broker (often TLS on 8883)	localhost / 1883
TLS / Do not verify TLS certificate	encryption	off
User / Password	broker login	empty
Group ID	top level in the Sparkplug topic	Biricon
Edge Node ID	identifier of this edge node	Bridge1
Device ID	optional device level under the node	empty
Use aliases	alias compression (recommended)	on
QoS	0 or 1	0
Reconnect (s)	wait before retry	5

Address format: metric name, e.g. `Temperature` or `Line3/Count` .



Sparkplug B

---

23. Connection type CAN

Role: CAN gateway (signal extraction and/or raw transport).

Field	Meaning	Default
Interface	socketcan / udp_multicast / slcan / pcan / kvaser / vector / seeedstudio / usb2can / virtual	socketcan
Channel	e.g. `can0` (SocketCAN) or adapter channel	can0
Bitrate (bit/s)	bus speed (must match the bus!)	500000
UDP port (udp_multicast)	port for CAN-over-Ethernet	43113
CAN-FD	extended frames with more data	off
Receive own transmissions	read back own frames	off
Cyclic transmit (ms)	send periodically (0 = off)	0
Reconnect (s)	wait before retry	5
Raw gateway to connection	ID of a second CAN connection for raw transport	empty
Gateway bidirectional	forward frames both ways	on
Signals	list of extracted signals (bit position, factor, offset …)	–

Address format: signal name from the signal list (created via analyzer/DBC). See chapter 34.

---

24. Connection type SQL

Role: write values to a database and/or read from one.

Field	Meaning	Default
Connection URL (SQLAlchemy)	e.g. `postgresql://user:pw@host/db`, `mssql+pyodbc://…`, `sqlite:///path.db`	`sqlite:///opcbridge_data.db`
Poll interval (ms)	how often to read	2000
Read mode	`keyvalue` (rows with addr/value), `columns` (columns as points), `off`	keyvalue
Read query (SELECT)	SQL for reading	`SELECT addr, value FROM tags`
Write statement		

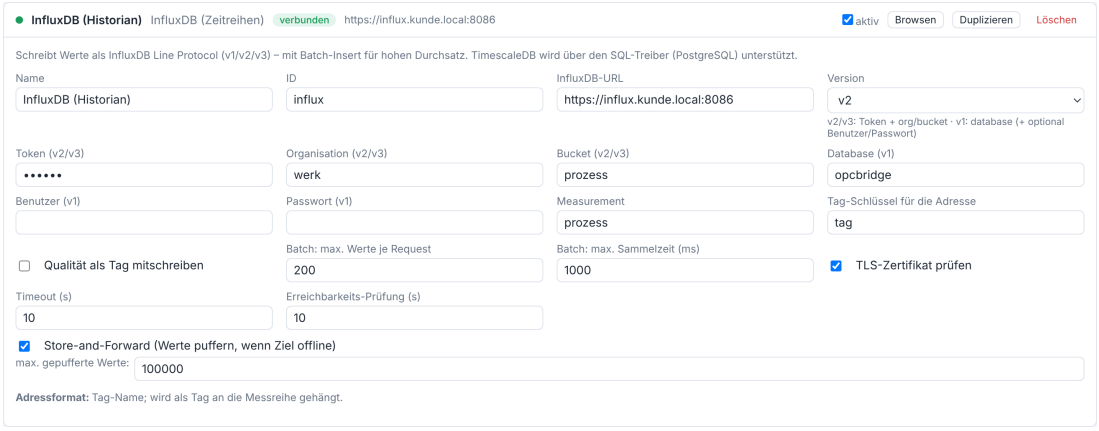
Field	Meaning	Default
	SQL with `:addr`, `:value`, `:ts`	`INSERT INTO tags(addr, value, ts) VALUES (:addr, :value, :ts)`
Write only on change	saves write accesses	on
Batch: max. rows per insert	0 = write individually; >1 bundles several values into one multi-row `INSERT`	0
Batch: max. collect time (ms)	send at the latest after this time even if the row count is not reached	500
Recycle connection after (s)	pool recycling	1800
Reconnect (s)	wait before retry	5

> »**Tip**« Batching reduces overhead with many fast-changing tags: the SQL driver writes a single **multi-row** `INSERT` per batch in one transaction instead of one statement per value. It combines well with **store-and-forward** – buffered values are flushed in batches, too.

---

25. Connection type InfluxDB

Role: write values as **InfluxDB line protocol** over HTTP, with batch writing for high throughput. Supports InfluxDB **v1** and **v2/v3**. (TimescaleDB is supported through the existing **SQL** connection type with a PostgreSQL URL instead – there is no separate TimescaleDB driver.)



InfluxDB connection

Field	Meaning	Default
InfluxDB URL	server address	`http://localhost:8086`

Field	Meaning	Default
Version	`v2` (token + org + bucket) or `v1` (database, optional user/password)	v2
Token (v2/v3)	API token	empty
Organization (v2/v3)	org name	empty
Bucket (v2/v3)	target bucket	`opcbridge`
Database (v1)	target database	`opcbridge`
User / Password (v1)	optional v1 authentication	empty
Measurement	measurement name	`opcbridge`
Tag key for the address	which tag key the point address is written under	`tag`
Include quality as a tag	writes the sample quality as an extra tag	off
Batch: max. values per request	0 = individually; higher values bundle a multi-line write	200
Batch: max. collect time (ms)	send at the latest after this time	1000
Verify TLS certificate	certificate verification	on
Timeout (s)	timeout	10
Reachability check (s)	health-check interval	10

Address format: free tag name; it is written as the value of the configured **tag key** (default `tag`) alongside the measurement.

---

26. Connection type Webhook

Role: send values via HTTP to a URL.

Field	Meaning	Default
Target URL	endpoint	`https://erp.example.com/api/opcbridge`
HTTP method	POST / PUT / PATCH	POST
Authorization header	e.g. `Bearer <token>`	empty
Content-Type	MIME type of the body	application/json
JSON template	template with `{addr}` `{value}` `{quality}` `{ts}` `{conn}` `{name}` (`{conn}`	`{"tag": "{addr}", "value": {value},`

Field	Meaning	Default
	=connection name, `{name}` =address – see 4.10)	`"quality": "{quality}", "timestamp": "{ts}"}`
Verify TLS certificate	certificate verification	on
Batch: max. values per request	0 = individually; higher values bundle values into one JSON array	0
Batch: max. collect time (ms)	send at the latest after this time	500
Timeout (s)	timeout	10
Retries per request	retries	2
Reachability check (s)	health-check interval	10

Address format: free tag name (appears as `{addr}`).

---

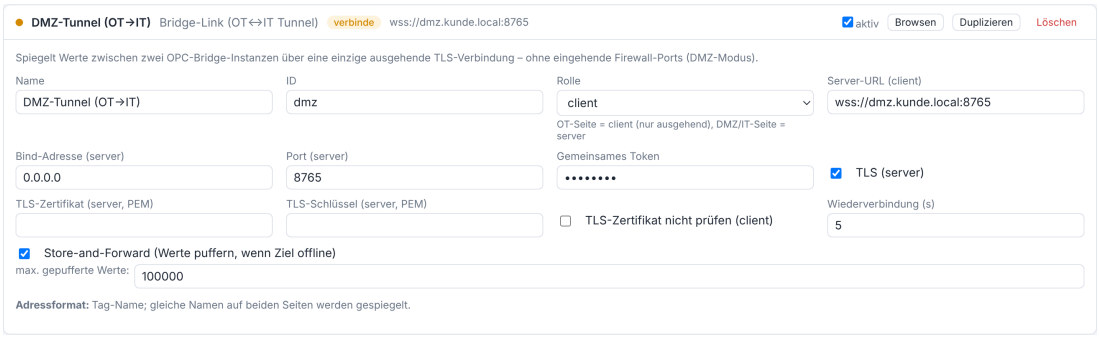
27. Connection type Bridge-Link

Role: mirror values between two Bridge instances over an outbound TLS connection.

Field	Meaning	Default
Role	`client` (OT side, outbound only) or `server` (DMZ/IT side)	client
Server URL (client)	client target, e.g. `wss://dmz-host:8765`	`wss://dmz-host:8765`
Bind address (server)	network interface of the server	0.0.0.0
Port (server)	server port	8765
Shared token	secret for both sides (must match)	empty
TLS (server)	encryption at the server	on
TLS certificate/key (server, PEM)	the server’s certificate	empty
Do not verify TLS certificate (client)	for self-signed server certificates	off
Strict one-way mode OT→IT (data diode)	blocks the IT→OT return channel completely (field `oneway`)	off
Reconnect (s)	wait before retry	5

Address format: free tag name; **identical names** on both sides are mirrored. See chapter 50 for an example.

Strict one-way mode (data diode). For critical-infrastructure and high-security environments, enable **“Strict one-way OT→IT (data diode)”** on the client and/or server side: the **client (OT)** then discards **all** data it receives from the server, and the **server (IT) never** sends data to clients at all – so the return channel IT→OT is provably blocked on both ends. Enable it on both sides for a true data diode; values then flow exclusively OT→IT, which is ideal for security audits.

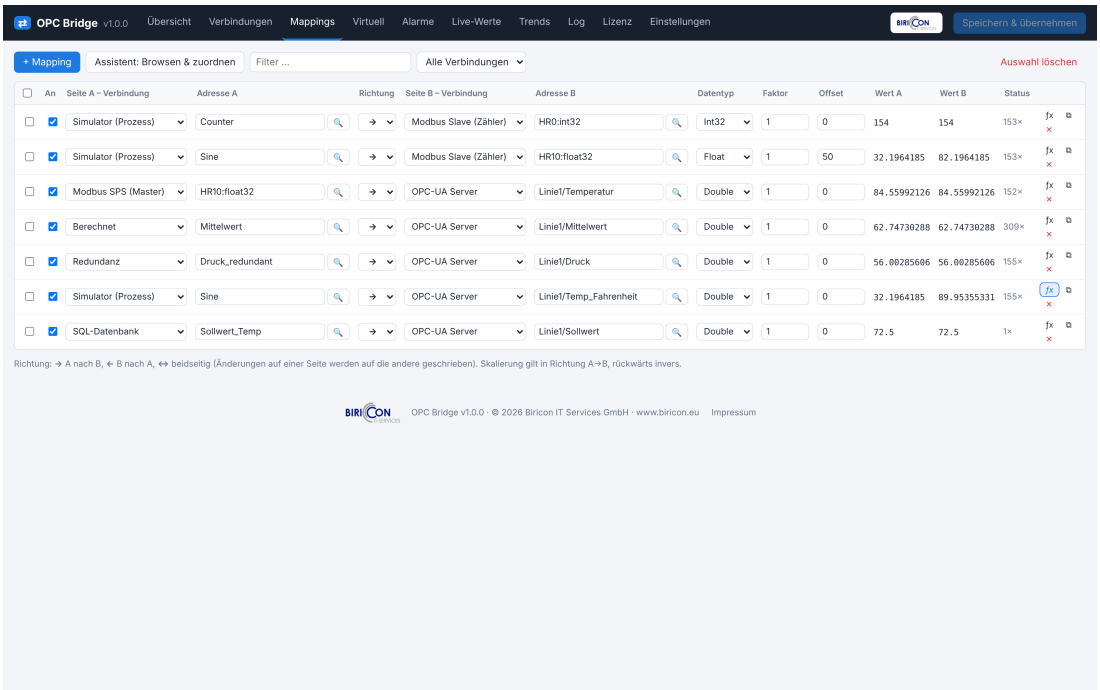


Bridge-Link / DMZ mode

---

28. Mappings and data flow

A **mapping** is the rule that defines **which point is transferred to which**.



Mappings

Creating a mapping:

1. *Mappings* tab → **+ Mapping**.

2. Choose **source A**: connection + address (browsable via the button).
3. Choose **target B**: connection + address.
4. Set the **direction**:
    - **A → B:** A is read and written to B.
    - **B → A:** the reverse.
    - **A ↔ B:** bidirectional; the most recently changed value is transferred.
5. Optionally set **data type** and **formula** (chapter 29).
6. **Save & apply**.

Field overview per mapping:

Field	Meaning
A / B (connection + address)	the two endpoints
Direction	A→B, B→A, A↔B
Data type	target type (e.g. Double, Int32, Bool)
Factor	linear scaling factor applied **before** the formula: `y = x · factor + offset` (default `1`)
Offset	linear scaling offset, see above (default `0`)
Formula	optional additional conversion (variable `x`), applied after factor/offset
active	enable/disable the mapping

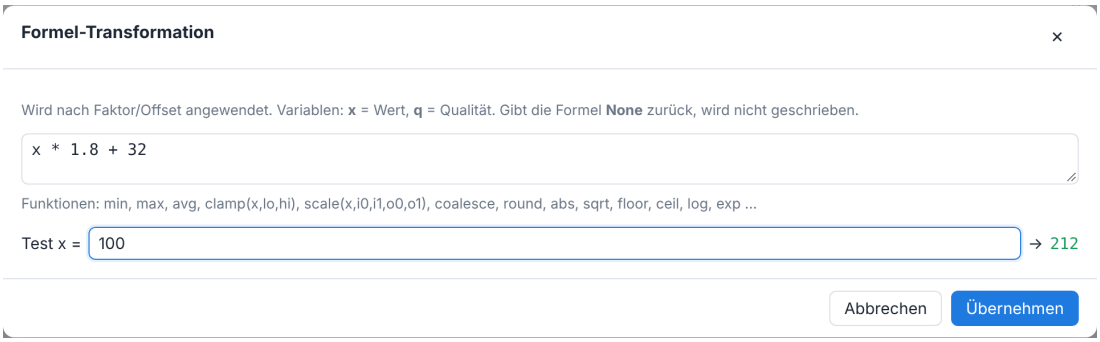
Factor and Offset. For simple linear scaling you do not need a formula at all: the two dedicated **Factor** and **Offset** fields compute `y = x · factor + offset` directly (e.g. factor `0.1`, offset `-40` for a raw value that needs no `scale()`/formula expression). They are defined for the **A→B** direction; for a bidirectional mapping (**A↔B**) the Bridge automatically applies the **inverse** calculation (`x = (y − offset) / factor`) when a value flows back from B to A, so you only ever enter factor/offset once. The optional **Formula** field is evaluated *after* factor/offset and is for anything a simple linear scaling cannot express (thresholds, rounding, `scale()` / `clamp()`, text, …).

Multiple sources to one target / one point to many targets. You can use the same source point in several mappings (distributing to several targets) and route several sources to different targets. To merge *several sources into one value*, use virtual/computed points (chapter 30).

---

29. Conversion and formulas

Values often need conversion: units (°C↔°F), scaling (raw→percent), limiting. Each mapping can have a **formula**. The variable `x` represents the incoming value.



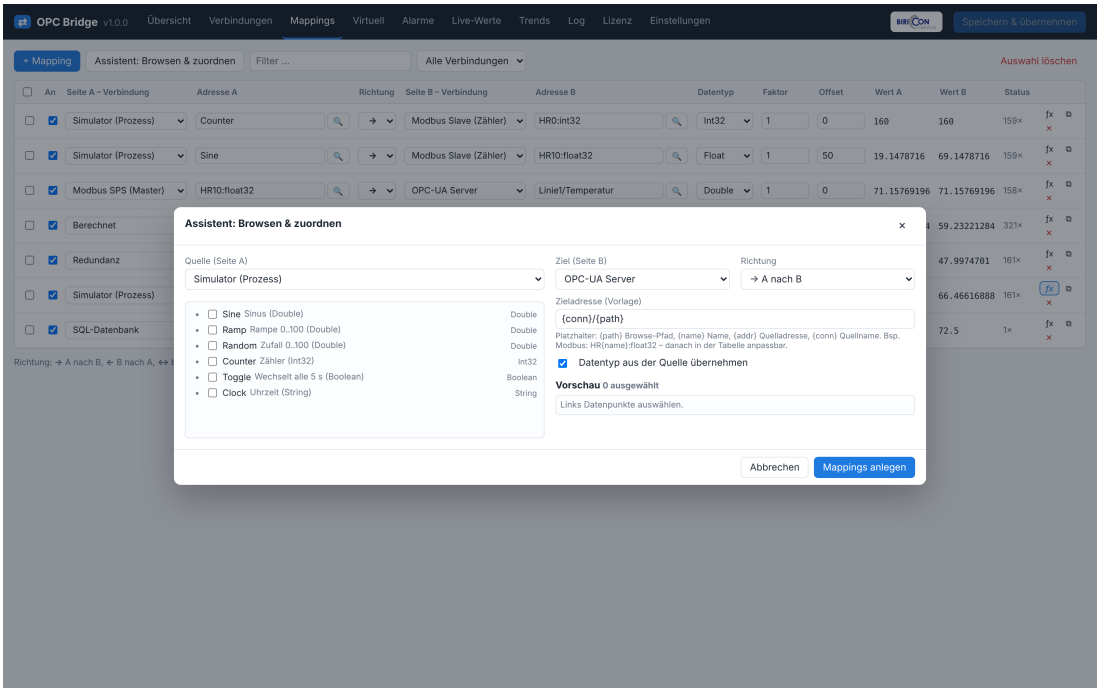
Formula

Examples:

Purpose	Formula
°C → °F	`x*1.8+32`
Scale /10	`x/10`
Raw 0–27648 → 0–100 %	`scale(x, 0, 27648, 0, 100)`
Limit to 0–100	`clamp(x, 0, 100)`
Round	`round(x, 1)`
Threshold → bool	`x > 50`

A complete list of functions is in Appendix B.

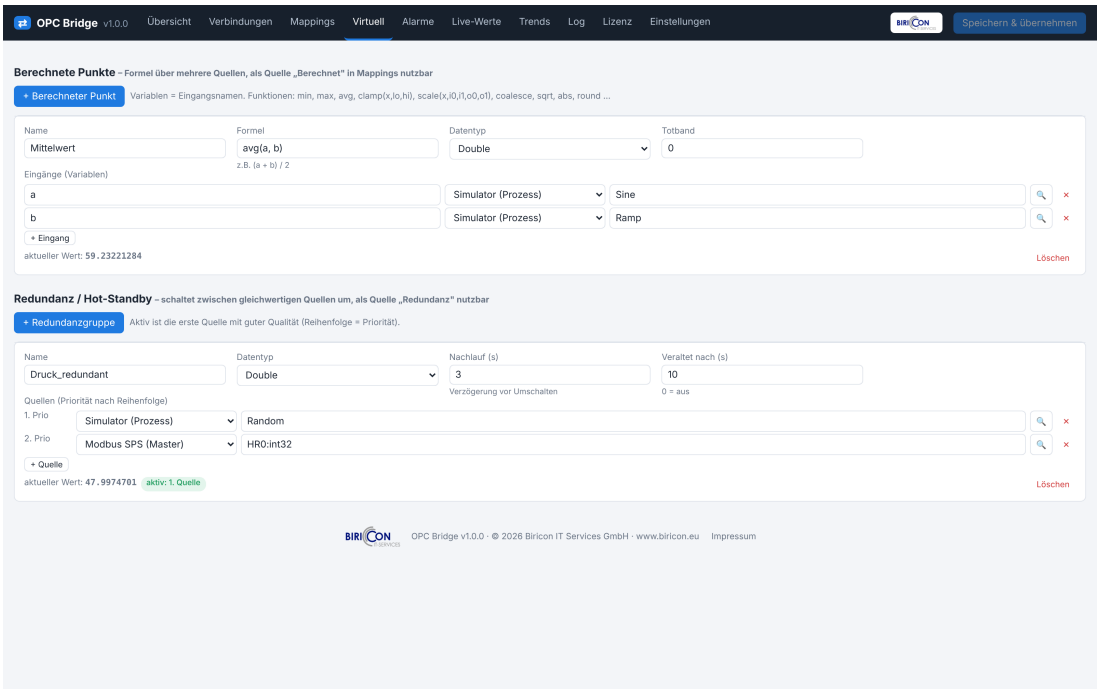
Wizard (bulk mapping). The browse wizard lets you search, select and map many points at once – ideal for large address spaces.



Wizard

30. Virtual sources and redundancy

In the **Virtual** tab you create values that do not come directly from a device.



Virtual sources

30.1 Computed points

A computed point is derived from **several input points** via a formula. Examples: average of three temperatures, sum of two counters, difference, minimum/maximum. The computed point is then usable like any other point in mappings, trends and alarms.

Create: assign a name, add inputs (connection+address), write the formula (inputs are available as variables), choose the data type.

30.2 Redundancy / hot-standby

A redundancy group has **several sources for the same value**. The Bridge automatically delivers the **best available** source and switches to the next on failure or bad quality (failover). Usable e.g. with redundant sensors or dually connected controllers.

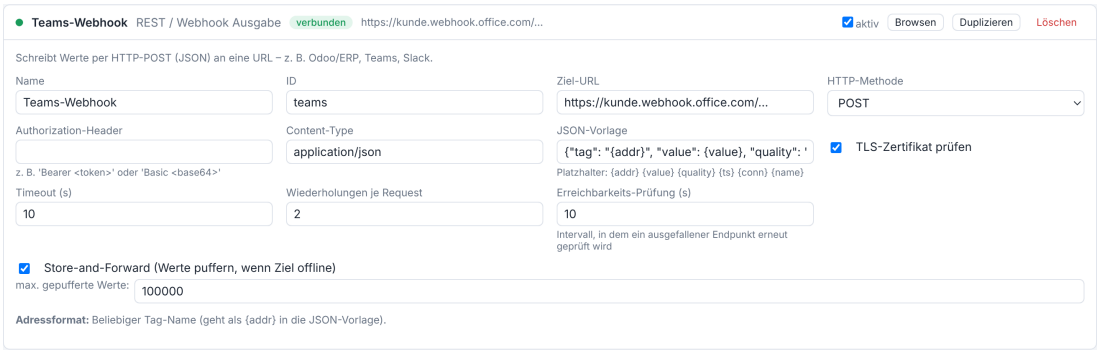
Create: assign a name, add sources in priority order, set the criterion (e.g. quality, staleness).

---

31. Store-and-forward

Purpose. If a **target** is temporarily offline, values would be lost without buffering. With **store-and-forward** they are buffered persistently in an **SQLite queue on disk** –

surviving even a **restart** of the Bridge – and delivered **in correct order** after reconnection.



Store-and-forward

Enable: on the **target connection** tick *Store-and-forward* and set the maximum buffer size (number of values). The overview and status show the number of buffered values per connection.

How it works:

1. The target is detected as “error” (unreachable).
2. Values to forward go into the queue (up to the maximum; then the oldest are dropped).
3. As soon as the target is “connected” again, the queue is processed in order.

> »**Tip**« Store-and-forward is especially important for cloud/webhook targets and the OT side of the Bridge-Link, where connections may drop temporarily.

---

32. Configuring history and trends

In **Settings → History**:

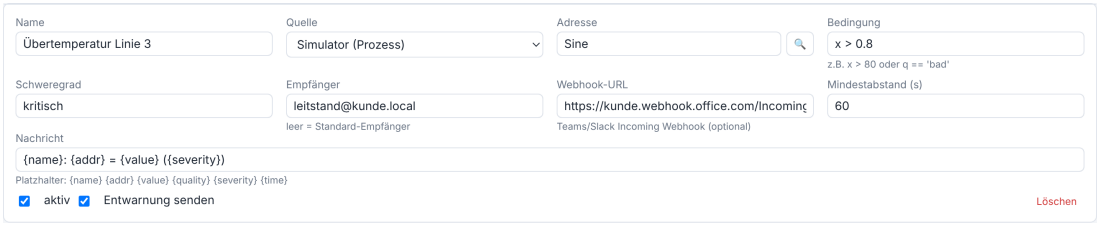
Field	Meaning
History enabled	recording on/off
Retention (days)	how long data is kept
Min. interval per point (s)	throttling (prevents overly dense recording)
Database file	SQLite file of the history
Recorded points	list of points with optional **deadband**

The **deadband** suppresses recording of small fluctuations (noise): only a change greater than the deadband is stored. Display is in the *Trends* tab (chapter 13) with CSV export.

---

33. Alarms, e-mail and webhook

In the **Alarms** tab you define notification rules.



Alarm rule with webhook

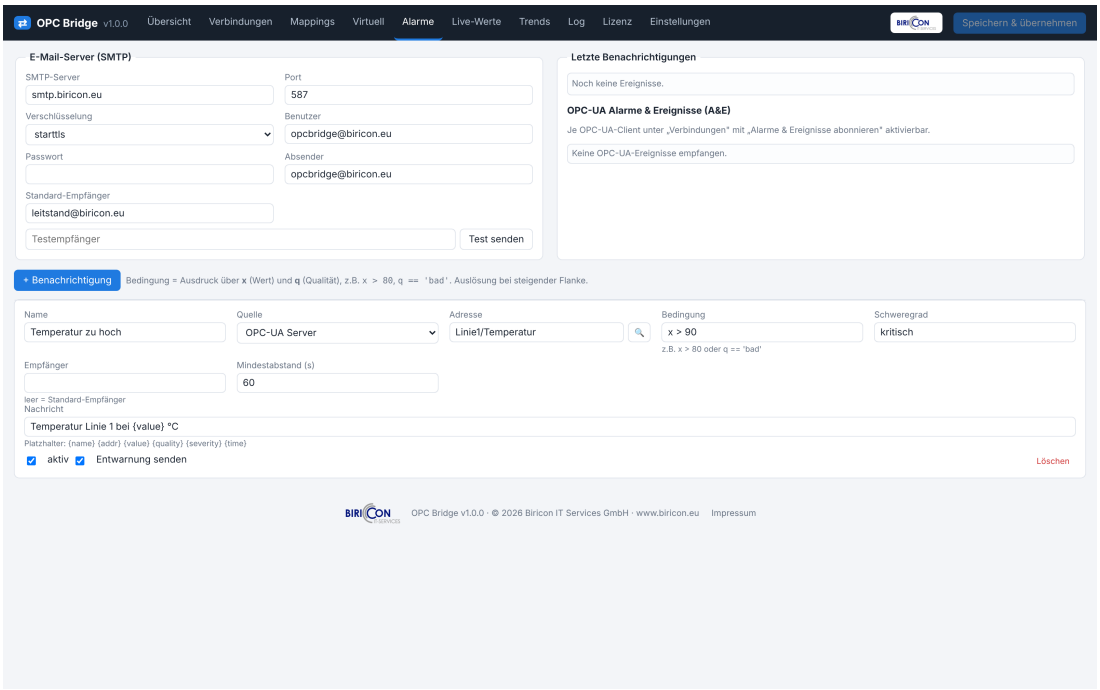
Fields of a rule:

Field	Meaning
Name	label of the rule
Source / Address	the monitored point
Condition	expression over `x` (value) and `q` (quality), e.g. `x > 90`, `q == 'bad'`
Severity	e.g. Info/Warning/Critical
Recipient	e-mail address(es); empty = default recipient
Webhook URL	Microsoft Teams/Slack/generic incoming webhook (optional)
Minimum interval (s)	prevents alarm floods
Message	text with placeholders `{name} {addr} {value} {quality} {severity} {time}`
active	enable/disable
Send clear	message when the condition is no longer met

Set up e-mail (SMTP): the SMTP settings (host, port, TLS, user, sender, default recipient) live **on the Alarms tab itself**, not under Settings – look for the SMTP fields section there (the UI code renders them alongside the alarm rules, via `SMTP_FIELDS` in `app.js`). SMS is possible via e-mail-to-SMS gateways.

Webhook: set an incoming webhook URL per rule (Teams/Slack). The alarm text is sent as JSON (`{"text": …}`).

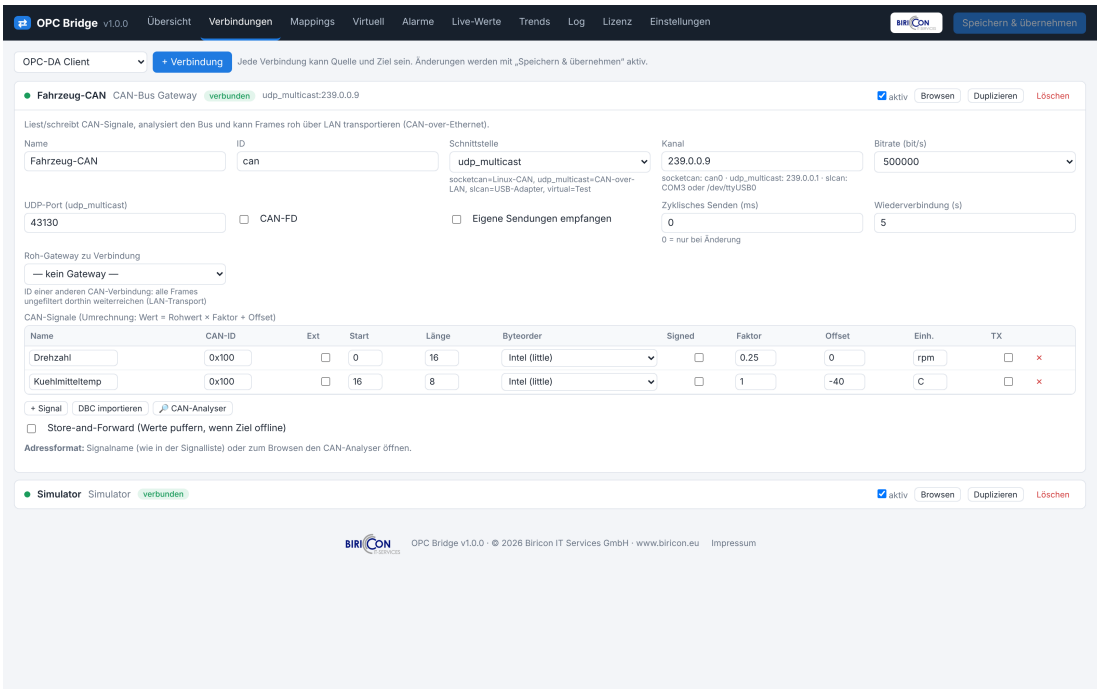
OPC-UA Alarms & Events (A&E): events received from a server by the UA client also appear in the alarm area.



Alarms

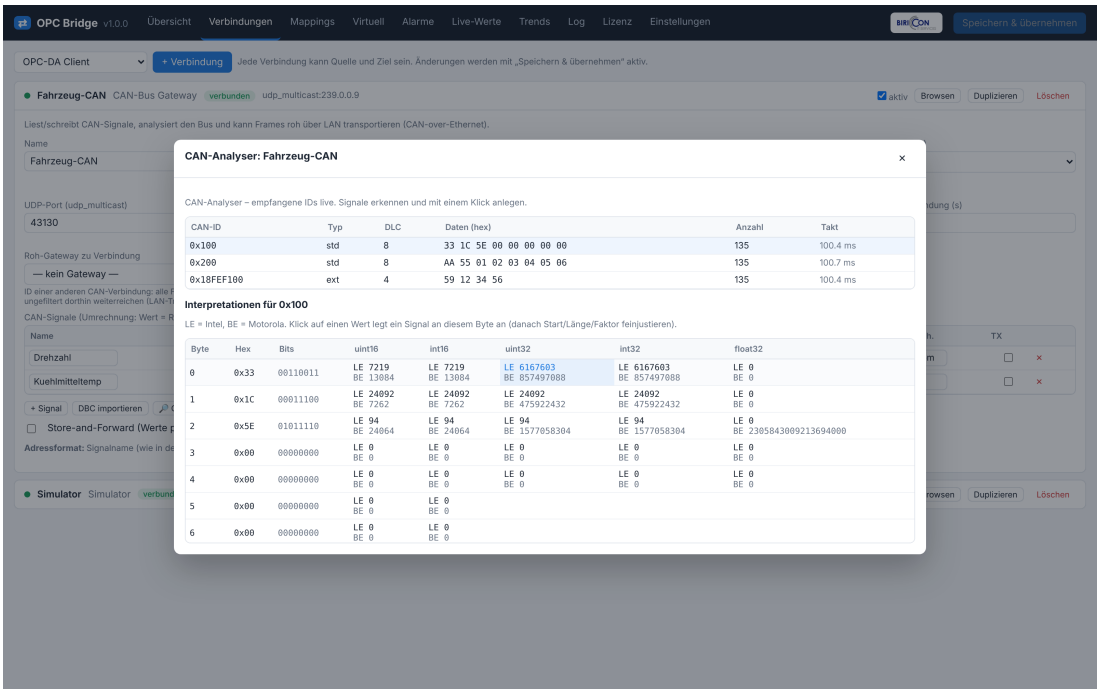
---

34. CAN analyzer and gateway



CAN gateway

Finding signals (analyzer). The **CAN analyzer** shows the running frames in real time: ID, length, data bytes, frequency. For extended frames, **J1939** information (PGN, SA, priority) is decoded automatically.



CAN analyzer

Creating signals. For each required signal you define:

Parameter	Meaning
Name	label (= address in mappings)
CAN ID	message containing the signal
Start bit / length	bit position in the data field
Byte order	Intel (little) / Motorola (big)
Sign	signed/unsigned
Factor / Offset	physical value = raw · factor + offset

Alternatively import the signal definitions via **DBC import**.

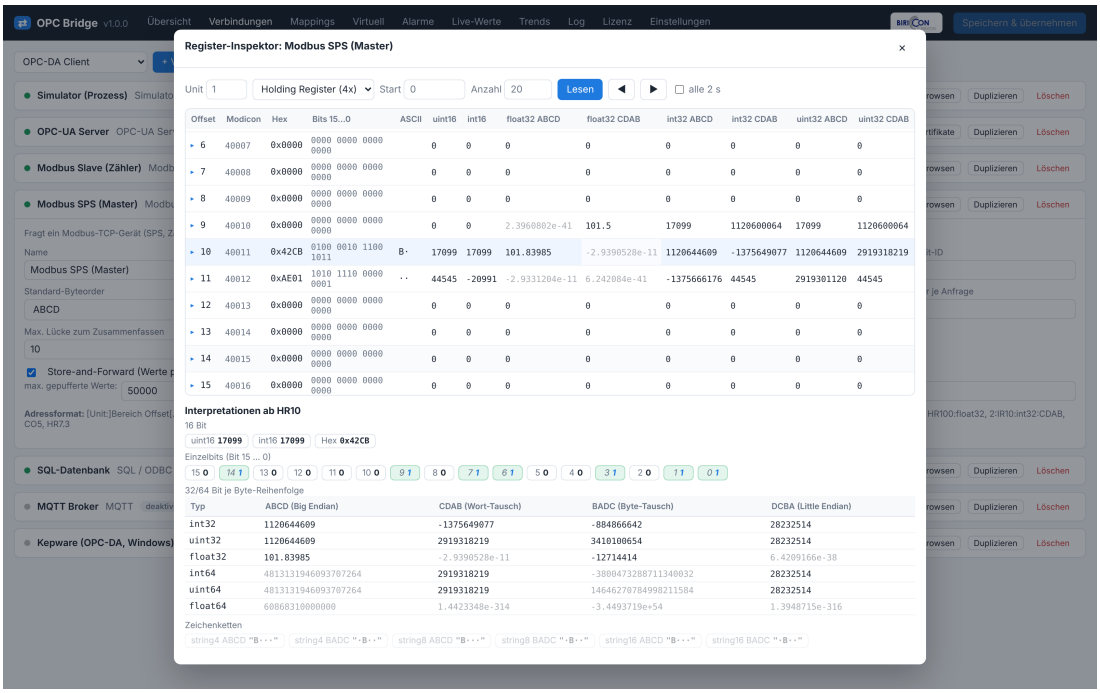
Raw gateway. Instead of extracting signals, the entire bus can be transported **unfiltered over LAN** (field *Raw gateway to connection*), coupling two remote CAN segments over Ethernet (optionally bidirectional).

Exporting values. Extracted signals are routed like any point via mapping to OPC, Modbus or MQTT – including conversion formulas.

---

35. Modbus register inspector

Modbus values are often cryptic: which register, which data type, which byte order? The **register inspector** reads a register range and shows **all interpretations at once**.



Register inspector

Where to open it. The register inspector has no separate tab – open it from **Connections**: click **Browse** on a Modbus connection's card. For a Modbus connection, Browse opens the inspector dialog directly (instead of a generic address-tree browser, since Modbus has none).

Procedure:

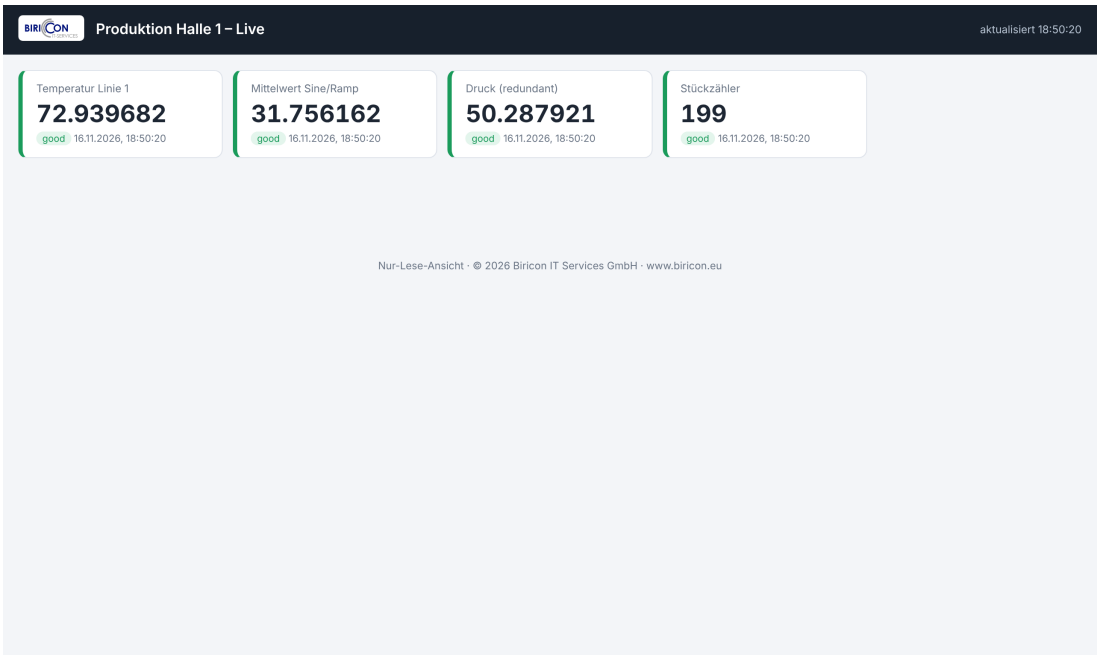
1. Choose the **area** (`HR` holding, `IR` input, `CO` coils, `DI` discrete inputs), the start address and the count, then read.
2. For register areas (HR/IR) the inspector shows, per selected start address, the interpretation as `int16`, `uint16`, `int32`, `uint32`, `int64`, `uint64`, `float32` and `float64` – in **all four** byte orders (`ABCD`, `CDAB`, `BADC`, `DCBA`) – plus a raw-text (string) interpretation. Implausible values (out of range, non-finite, garbled text) are greyed out to help you spot the right line faster.
3. Click a value to adopt that data type/byte order directly into the address field, e.g. `HR100:float32:CDAB`. For CO/DI the inspector simply shows the raw bit value per address (no type/byte-order ambiguity there).

> »**Tip**« The register inspector saves a lot of time and guesswork during commissioning. Use it whenever a Modbus value is wrong.

---

36. WebView

The **WebView** is a public, **token-protected** live page (`/view?token=…`) that shows selected values as tiles – **without login**.



WebView

Set up: *Settings* → *WebView* – enable, assign a title, select points, set a token. The link can be shared with the control room or a customer.

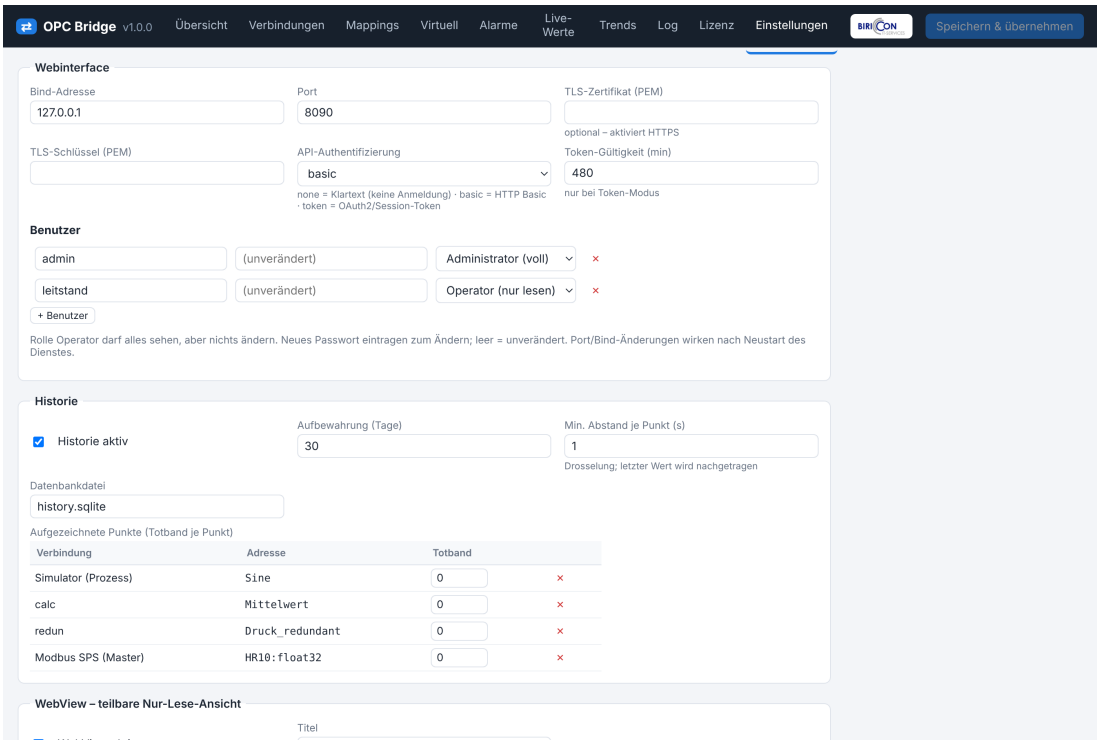
Field	Meaning
WebView enabled	function on/off
Title	page heading
Token	secret in the link (`/view?token=…`)
Points	which values are shown

> »**Caution**« Anyone with link+token sees the shared values. Share only the needed points and put HTTPS in front for internet operation.

---

37. Users, roles and API authentication

In the **Settings** tab.



API authentication

37.1 Users and roles

Role	Permissions
Administrator	full access (read, configure, write)
Operator	read only – sees everything, changes nothing

Passwords are stored as a **PBKDF2-SHA256 hash** (200,000 iterations). Enter a new password to change it; empty = unchanged. Write accesses are restricted server-side to administrators.

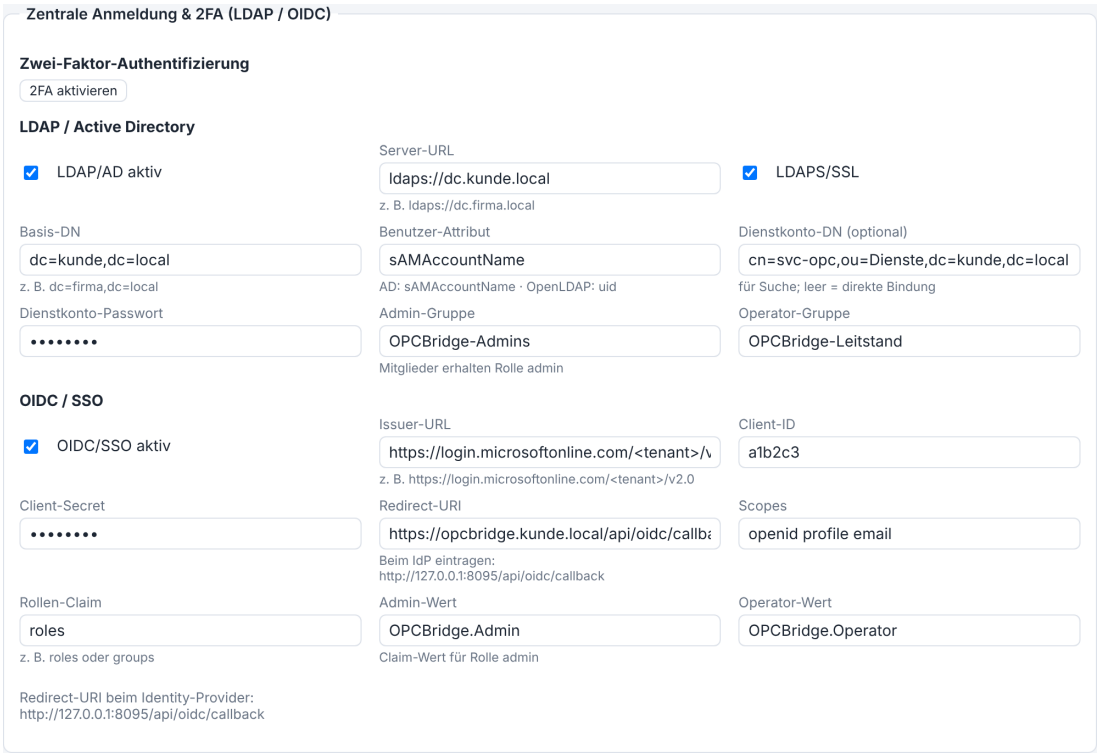
37.2 API authentication (configurable)

Mode	Meaning
Plaintext (`none`)	**no login** – open access. **Anyone reaching the port is an
administrator**, without entering any credentials. Isolated machine network only!	
HTTP Basic (`basic`)	login like in the web interface (`Authorization: Basic …` on every

request). If the user has 2FA enabled, Basic credentials are rejected for the API and `/api/login` with the TOTP code must be used instead to obtain a Bearer token (the *page* still loads with Basic credentials in this mode – only the API enforces the stronger check). | | **OAuth2/Session token** (`token`) | login via `/api/login` or `/api/token`, then a `Bearer` token with a configurable **lifetime** (token TTL). |

37.3 Central sign-in: LDAP/Active Directory, OIDC/SSO and TOTP two-factor

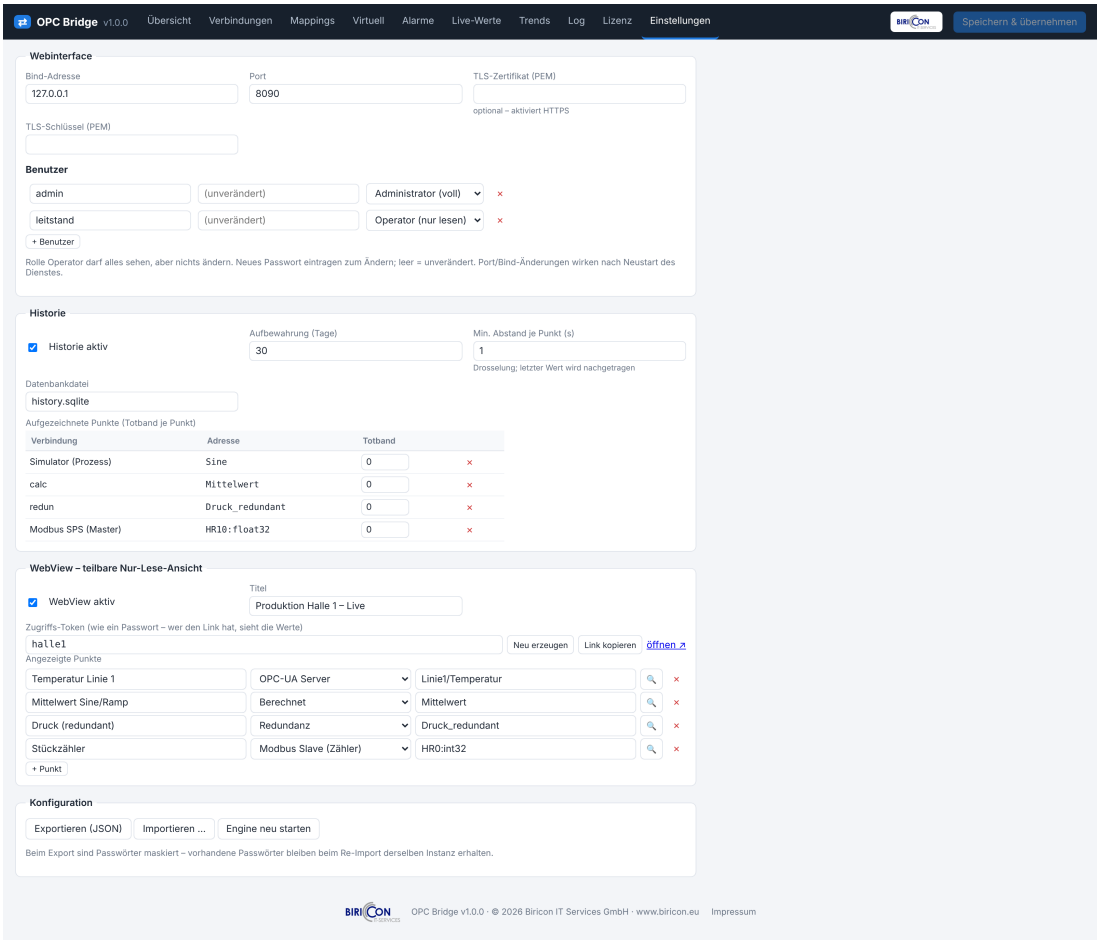
Sign-in can optionally be connected to central user management and secured with two-factor authentication, for IT audits in larger operations. All three are **optional and disabled by default** – existing local users keep working unchanged when none of this is configured.



LDAP/OIDC/2FA settings

- **LDAP / Active Directory:** sign-in via a directory service, with group-to-role mapping (an admin group and an operator group), either a service-account search or a direct bind, and optional LDAPS/SSL.
- **OIDC / SSO:** authorization-code flow against an external identity provider (e.g. Microsoft Entra ID, Keycloak, Google). A **“Sign in with SSO”** button then appears on the login page; roles are derived from a configured claim. Register the **redirect URI** (`/api/oidc/callback`) at the identity provider.
- **TOTP 2FA:** each local user can enable 2FA in their own settings (authenticator app via a secret/QR code). With 2FA active, login additionally requires the current 6-digit code, and that user’s sign-in always ends up using **token mode** for the API (see 37.2) even if the overall API authentication mode is Basic or plaintext.

37.4 Web interface settings



Settings

Field	Meaning
Bind address / Port	network interface and port of the UI (takes effect after service restart)
TLS certificate / key (PEM)	enables HTTPS
API authentication	see 37.2
Token lifetime (min)	token lifetime in token mode

---

38. Real-time updates (WebSocket)

Live values and status are **pushed in real time over WebSocket** to the interface – without reloading, with minimal delay. If the WebSocket connection drops (e.g. due to a proxy), the interface **automatically falls back to periodic polling** and later retries the WebSocket.

Important for operation: the live update touches **display cells only**. Input and configuration fields remain **operable at all times** – an incoming update **never**

overwrites your current input. You can configure in peace while live values arrive in the background.

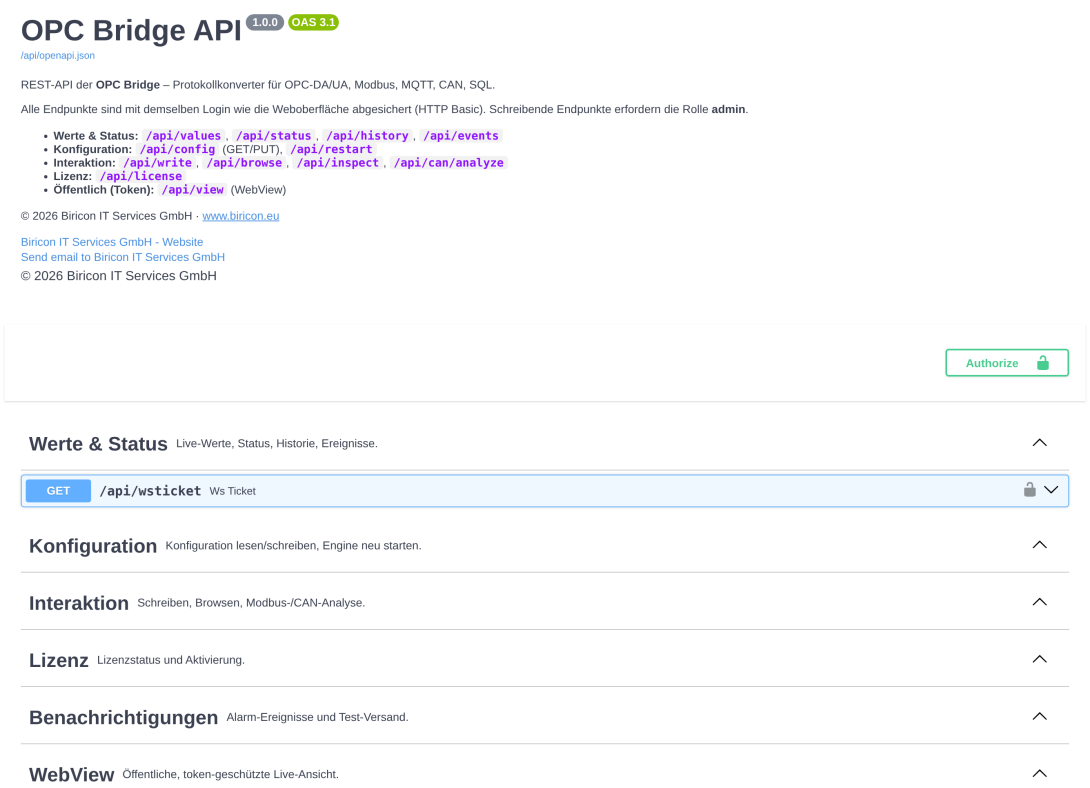
Technically the WebSocket uses a **short-lived ticket** (via `/api/wsticket`) secured according to the chosen authentication mode.

Part V – Interfaces and integration

39. REST API and built-in documentation

All functions of the Bridge are also available via a **REST API** – ideal for automation, custom dashboards or integration with supervisory systems.

Built-in interactive documentation. The API documentation is a **fixed part of the product**. You reach it at `/api/docs`. The complete interface is **bundled** and loads **nothing from the internet** (no CDN dependency) – so it works even in an **air-gapped/OT network**. It is based on the open **OpenAPI 3.1 specification** at `/api/openapi.json`. A ReDoc view is available at `/api/redoc`.



Built-in REST API documentation

Every endpoint is testable directly in the interface (“Try it out”). A complete endpoint list is in Appendix C.

Authentication. Depending on the mode (chapter 37.2): none (plaintext), HTTP Basic or Bearer token. Example (obtain and use a token):

```
 # Obtain a token (token mode) - /api/login takes a JSON body, not HTTP Basic
   credentials.
# "code" is only required if the user has 2FA (TOTP) enabled.
curl -X POST http://host:8080/api/login \
     -H "Content-Type: application/json" \
```

```
     -d '{"username":"admin","password":"secret","code":""}'
# -> {"token":"...","token_type":"bearer","expires_in":...,"role":"admin"}

# Read values
curl -H "Authorization: Bearer <token>" http://host:8080/api/values
```

> »**Note**« In **Basic** auth mode, the browser/API client instead sends `Authorization: Basic <base64 user:pass>` directly on every request (no `/api/login` call needed) – unless the user has 2FA enabled, in which case Basic credentials are rejected for the API and `/api/login` with the TOTP `code` must be used to obtain a Bearer token.

Writing endpoints (`PUT /api/config`, `POST /api/write`, `POST /api/restart` ...) require the **Administrator** role.

Secret masking on `GET /api/config`. Because `config.json` otherwise holds most credentials in plain text (chapter 40), `GET /api/config` returns a **redacted** copy: every connection setting named `password`, `token` or `auth_header` is replaced with `********`, as are the **WebView token**, the **online-license enrollment token**, the **SMTP password**, and the **password portion of SQL connection URLs** (e.g. `postgresql://user:********@host/db` – host, user and database name stay visible). This makes `GET /api/config` safe to export, back up or hand to support. When you `PUT` a configuration that still contains the `********` placeholder for a given field, the Bridge keeps the **existing** stored value for that field instead of overwriting it with the literal mask – so round-tripping an exported config (export, edit something unrelated, import) does not wipe out credentials.

---

40. Configuration file (config.json)

The entire configuration is stored in **`config.json`** in the data directory (see Appendix F). Normally you edit it via the interface; for backup, mass rollout or version control, direct access is useful.

Structure (simplified):

```
{
  "web":         { "host": "0.0.0.0", "port": 8080, "api_auth": "basic", "users": [
 … ] },
  "history":     { "enabled": true, "retention_days": 30, "points": [ … ] },
  "connections":[ { "id": "sim", "type": "sim", "settings": { … } }, … ],
  "mappings":    [ { "a": {"conn":"…","addr":"…"}, "b": {"conn":"…","addr":"…"},
 "dir":"a>b" }, … ],
  "computed":    [ … ], "redundancy": [ … ], "notifications": [ … ],
  "smtp":        { … }, "webview": { … }
}
```

»**Caution – plain-text secrets.** Only **web-interface user passwords** (`web.users[].password_hash`) are hashed (PBKDF2-SHA256, 200,000 iterations) and cannot be recovered from the file. **Every other credential lives in `config.json` as plain text**: database passwords in SQL connection URLs, MQTT/broker and OPC-UA passwords, webhook `Authorization` headers, Bridge-Link and InfluxDB tokens, LDAP bind passwords, OIDC client secrets, API keys embedded in any connection's settings – none of these are encrypted or hashed on disk. This is a deliberate design choice (the Bridge needs the plain value to connect to each target), **not a documentation gap** – but it means `config.json` must be treated as a secret itself: restrict filesystem permissions, never commit it to a public repository unredacted, and remember this when sharing a backup or a support bundle. The masked copy returned by `GET /api/config` (chapter 39) is safe to share; the file on disk is not.

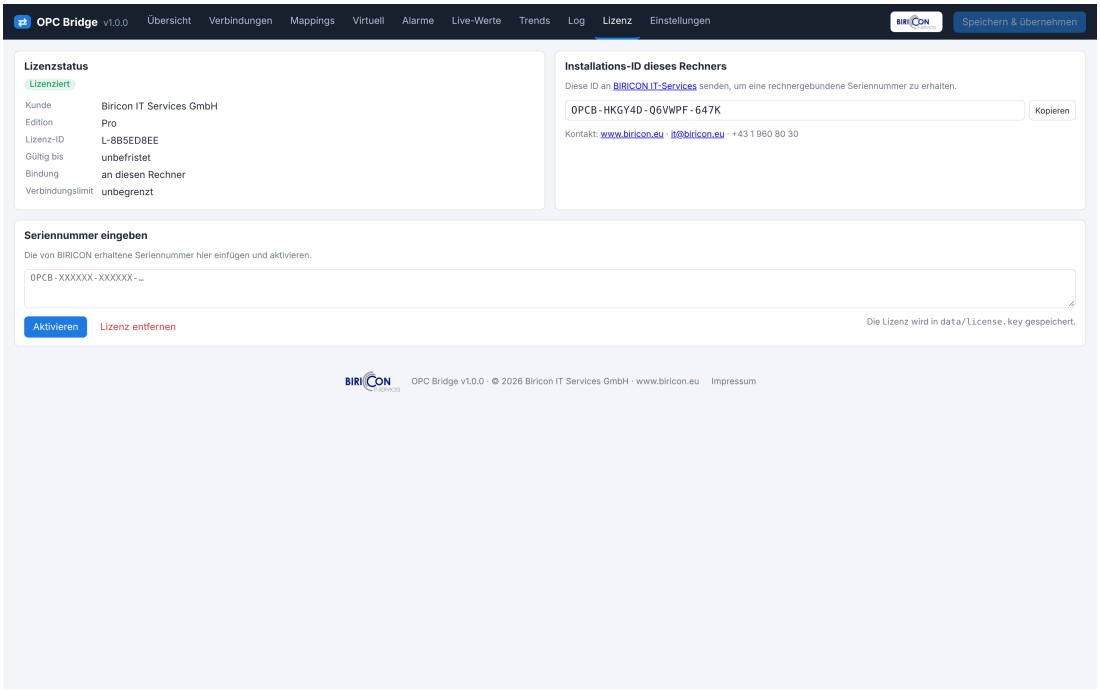
Edit the file only with the service stopped, or load it via `PUT /api/config` . Invalid JSON prevents startup – always keep a backup (chapter 43).

Config as code. Because the entire configuration is in one file, it can be maintained in version control (e.g. Git) and rolled out to several plants.

Part VI – Administration and maintenance

41. Licensing

The OPC Bridge is protected two ways, which can be used independently or together: a **machine-bound offline serial number** (Ed25519-signed, verifiable without internet access) and an optional **online license** verified against the Biricon license server. Either one is enough to license the installation.



License

41.1 Demo mode

Without any valid license the Bridge runs in **demo mode**:

- **Full function for one hour, in total – not per start.** The first time the Bridge is ever started, it records that moment permanently (`demo_started_at.txt` in the data directory, Appendix F). The one-hour window runs from that single, one-time timestamp and is **not** refreshed by restarting the service or the process – restarting no longer buys another hour.
- Permanently **at most 2 connections** (`DEMO_MAX_CONNECTIONS`), even while still inside the one-hour window.
- After the hour elapses, **routing/forwarding stops** (manual writes are blocked too); the web interface and API stay reachable so you can activate a license.

41.2 Offline activation (serial number)

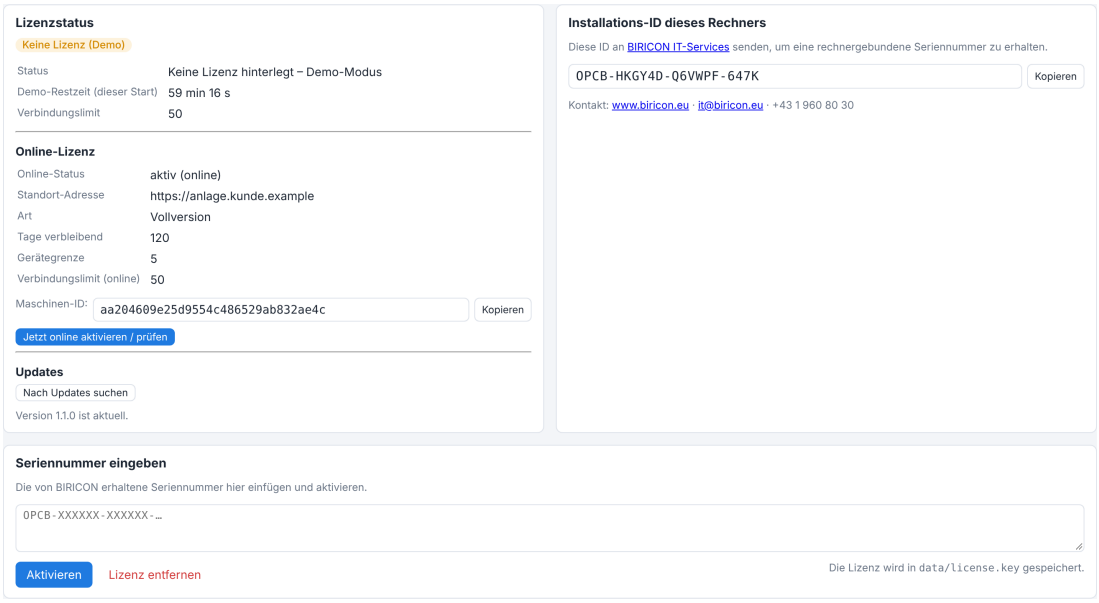
1. Open the *License* tab and read the **installation ID** (derived from a machine fingerprint that is persisted in the data directory – see the Docker note below).
2. Send the installation ID to **Biricon**.
3. Enter the received **serial number** in the field and click **Activate**.

The serial number is valid for this exact machine; hardware replacement or a non-persisted reinstall requires re-issuing. It encodes customer, **edition**, issue/expiry date, an optional machine binding and limits (e.g. the **connection limit**).

> »**Note – Docker machine binding.** The machine fingerprint used for binding is **generated once and persisted** as a `machine_id` file in the data directory (the `/data` volume in Docker), not re-derived from the host each time. This makes licensing **stable across container re-creation** – a container recreated from the same volume keeps the same installation ID. Only a fresh, empty volume (or a reinstall without preserving the data directory) produces a new installation ID and requires re-activation.

41.3 Online license (automatic activation, device limit, clone detection)

In addition to the offline serial number, the Bridge can optionally verify and activate its license **online** against the Biricon license server, automatically whenever internet access is available. The two schemes can coexist; the offline serial keeps working unchanged in isolated/air-gapped networks.



License page with online license and updates

Configuration is under **Settings → Online license & updates**:

Field	Meaning
Online license active	master switch (default off); the offline serial stays valid either way
License server URL	default `https://www.biricon.eu`
Site address	a fixed `https://host[:port]` identifying *this installation's site* to the license server – the same value must be on file there for the license
License ID (auto-activation)	lets the Bridge self-activate without entering a serial number, format `XXXXX-XXXXX-XXXXX-XXXXX`
Rollout token (optional)	an alternative to the license ID for mass/bulk activation across many installations
Strict	if set, the license **ends** once the grace period ("Karenz") has passed without a successful online check – see below



Settings: online license and updates

How it works.

- **Automatic activation.** If a **license ID** or a **rollout token** is configured, the Bridge calls `POST /licence/v1/aktivieren` once to self-activate – no manual serial number needed. From then on (and also when an offline serial's own license ID is being checked instead) it re-verifies with `POST /licence/v1/pruefen`, at every start and roughly once a day (the server's response tells the Bridge when to check again).
- **Machine ID & clone detection.** The Bridge derives a **machine ID** from the same persisted machine fingerprint used for offline binding (hashed again specifically for the online protocol); the raw value never leaves the computer. Each seat of the license's **device limit** is one distinct machine ID. Reinstalling without preserving the data directory, or cloning a VM in a way that produces a new machine ID, counts as a new device and needs a new activation/seat; a bit-identical clone that keeps the same machine ID occupies the same seat (clone detection). The machine ID is shown on the License page.

- **Forgery/replay protection.** Every check sends a fresh random nonce; the server's signed (ES256) response must echo it back, so an old captured response cannot be replayed.
- **Grace period ("Karenz") – with a hard edge, not a soft one.** After a successful check, the server grants a signed validity window (`gueltig_bis`) and a grace deadline (`karenz_bis`): if the network is briefly unreachable, the **last successful, signed check** keeps the license working until `karenz_bis` , never beyond the actual expiry date. With **Strict** enabled, once `karenz_bis` passes **without** a new successful check, the license ends there and then – there is no further leeway.
- **Immediate stop on expiry or revocation – no grace period at all.** This is different from running out of grace: a **valid refusal** from the server (e.g. the license was revoked, or is reported as expired) blocks the license **immediately**, whether or not any grace time would otherwise remain. The same applies to an offline serial past its expiry date. Internally, the engine checks this on every single routed value (not only at startup or at the next scheduled check): as soon as the license is in this "expired" state, **all routing and manual writes stop at once**, without restarting the service and without a grace window – see 41.4.
- **Edition and connection limit.** The granted license carries an **edition** name and, optionally, a **connection limit** (`geraete_grenze` /the connection limit under `grenzen`). Connections beyond the limit are simply never started (see 41.4); there is no demo-style soft warning period.

41.4 Enforcement: what happens exactly, at runtime

- **At startup,** the engine counts enabled connections against the active connection limit (the licensed limit, or `2` in demo mode). Connections beyond the limit are **not started** at all and a warning is logged ("…nicht gestartet: Lizenz erlaubt max. *N* Verbindungen…" – license allows at most *N* connections). If the license is already expired/revoked at startup, **no connection is started** at all.
- **While running,** the Bridge re-evaluates the license continuously (e.g. after the hourly/ daily online re-check, or when the demo hour elapses) – not only on restart. The moment the license becomes `expired` (offline date passed, or a valid online refusal/expiry), **routing of every mapping and every manual write is blocked immediately**, in place, without restarting the engine and **without any grace period**. The Overview and Log tabs show the reason ("License expired – forwarding stopped. Please renew the license." / "Demo time expired – forwarding stopped. Please enter a serial number.").
- This immediate-stop behavior is a deliberate hardening: there used to be an implicit fallback to demo-mode behavior with a grace window on expiry – that fallback has been **removed**. Expiry and revocation now behave identically and immediately.

41.5 License types

- **Perpetual:** valid indefinitely.

- **Time-limited (annual/maintenance):** with an expiry date; renew in time to avoid the immediate stop described above.

---

42. Security and hardening

42.1 Defaults you should not leave unchanged

- **“Plaintext” (`none`) API auth mode means anyone who can reach the port is an administrator** – no login, no credentials, full read/write access (see 37.2). Use it only on a physically isolated machine network, never otherwise.
- **The default bind is `0.0.0.0:8080` over plain HTTP** unless you configure a TLS certificate/key. Until you do, both the web interface and the REST API are unencrypted on the network – credentials, tokens and values travel in clear text. Configure TLS directly (35.3) or put a TLS-terminating reverse proxy (e.g. nginx) in front before exposing the Bridge beyond a trusted local network.
- **Delete the initial admin password file after first login.** `initial_admin_password.txt` in the data directory (Appendix F) contains the one-time generated administrator password in plain text. It is not needed again once you have logged in and changed the password – delete it (or at least restrict its file permissions) so it cannot later be read by anyone with filesystem access.

42.2 General recommendations for production

1. **Change passwords.** Change the initial password immediately; use strong passwords.
2. **Use roles.** Give observers only the **Operator** role.
3. **API authentication** at least **Basic**, better **Token**. Use **Plaintext** only in a physically isolated network (see 42.1).
4. **Enable HTTPS.** Provide a TLS certificate/key or put a reverse proxy (nginx) with TLS in front.
5. **Consider central sign-in and 2FA.** LDAP/Active Directory and OIDC/SSO let you reuse existing identity management instead of local accounts; TOTP two-factor adds a second factor for local users. All three are optional and off by default (chapter 37.3).
6. **Secure OPC-UA.** Policy `Basic256Sha256 SignAndEncrypt`, truststore instead of `trust_all`, users instead of `anonymous=readwrite`. Manage the certificate truststore under each OPC-UA connection’s **PKI folder** (`pki/<connection-id>/` in the data directory: `own_cert.der`, `trusted/`, `rejected/`) – peers not yet trusted land in `rejected` and must be moved to `trusted` explicitly (also reachable via `/api/pki/{conn}`, chapter 39, and chapter 46’s troubleshooting table).
7. **Network segmentation.** Separate OT and IT; use the **Bridge-Link** (DMZ mode) for the coupling so no inbound OT ports are needed, and consider its **strict one-way (data diode)** option (chapter 27) where values must only ever flow OT→IT.
8. **WebView** only with a strong token and HTTPS; share only required points.

9. **Minimize ports.** Enable only needed services (OPC-UA server, Modbus slave) and open only those in the firewall.
10. **Updates** promptly (chapter 44).

42.3 Outbound connections the Bridge itself makes

Beyond the web interface/API it serves (inbound), the Bridge process itself initiates a small, fixed set of **outbound** connections when the corresponding feature is enabled:

- The **update check** (chapter 44.1): daily HTTPS request to the configured update URL (biricon.eu by default), enabled by default.
- The **online license check** (chapter 41.3): HTTPS requests to the configured license server (biricon.eu by default), only when explicitly enabled.
- Whatever **connection targets you configure yourself** reach out to: webhook URLs, MQTT/ Sparkplug brokers, SQL database servers, InfluxDB servers, Bridge-Link client connections, LDAP servers, and the OIDC identity provider. Each of these is only contacted because a corresponding connection/setting was configured – review your connection list if you need a complete outbound allow-list for a firewall.

---

43. Backup and restore

43.1 Config-only export/import via the web UI

For a quick, config-only backup (or to move a configuration between a test and a production instance) use the **Export**/**Import** buttons in the web interface: *Export* downloads the current configuration as a JSON file; *Import* loads a JSON file back into the editor (you must still click **“Save & apply”** to activate it). This is equivalent to what `GET` / `PUT /api/config` return and accept (chapter 39) – in particular, **Export produces the same masked copy** as `GET /api/config`: every connection `password`, `token` and `auth_header` field, the WebView token, the online-license enrollment token, the SMTP password, and the password portion of SQL connection URLs come out as `********`, not as the real secret (chapter 40). Re-importing such a file and saving it keeps the **previously stored** secrets in place (the backend recognizes the mask and does not overwrite them with the literal placeholder) – so a round-trip export/import does not destroy credentials, but a masked export is **not** sufficient on its own to recreate connections with their secrets on a brand-new installation that never had them.

43.2 What to back up for a complete, restorable backup

A config-only export is not a full backup. The complete **data directory** (Appendix F) is what actually needs to be backed up and contains:

- `config.json` – the configuration, including connection/mapping secrets **in plain text** (chapter 40) – treat this file itself as sensitive,
- `license.key` – the offline license,

- `machine_id` – the persisted machine fingerprint used for license binding (important for Docker – losing this without also losing the license means re-activation is needed even though “nothing changed”),
- `online_state.json` – the online-license check state (signed grant, if auto-activated),
- `update_state.json` – last known update-check result (not essential, but harmless to keep),
- `demo_started_at.txt` – the one-time demo-window start marker (only relevant without a license),
- `history.sqlite` – the history database (if the history should be backed up),
- `sf_queue.sqlite` – the persistent store-and-forward queue (only relevant with an active buffer – restoring an old copy can replay stale buffered values),
- the `pki/` folder (one subfolder per OPC-UA connection) – each connection’s own certificate and its trusted/rejected peer certificates; losing this means OPC-UA peers must be re-trusted after a restore,
- `opcbridge.log` / `service.log` – not required for restore, but useful to keep for incident history.

43.3 Backup commands per platform

Linux (.deb):

```
sudo systemctl stop opcbridge
sudo tar czf opcbridge-backup-$(date +%F).tgz -C /var/lib opcbridge
sudo systemctl start opcbridge
```

Windows: stop the “OPCBridge” service (`services.msc` or `net stop OPCBridge`), then copy or archive the data directory, then start the service again, e.g. in PowerShell:

```
Stop-Service OPCBridge
Compress-Archive -Path "$env:ProgramData\OPCBridge\*" -DestinationPath "C:
  \Backups\opcbridge-$(Get-Date -Format yyyy-MM-dd).zip"
Start-Service OPCBridge
```

Docker: back up the named volume `opcbridge-data` (not a host directory, chapter 8), e.g. by running a throwaway container that archives the volume to the host:

```
docker compose stop opcbridge
docker run --rm -v opcbridge-data:/data -v "$(pwd)":/backup alpine \
  tar czf /backup/opcbridge-backup-$(date +%F).tgz -C / data
docker compose start opcbridge
```

Restore (any platform): stop the service/container, replace the data directory/volume contents with the backup, start it again. Because the **license is machine-bound**, a backup restored onto different hardware (or a fresh Docker volume without the preserved `machine_id`) produces a different installation ID and needs re-activation.

> **»Tip«** For config-only backups (no history, no license state), `config.json` is enough – use the Export button (43.1) and put the result in version control to track changes. Remember that file still holds connection secrets in plain text, so treat the repository itself as confidential.

---

44. Update and migration

44.1 Update notification and one-click installation

The Bridge checks a **signed version file** (ES256, signed with the Biricon update key) for newer versions. This check is **enabled by default** and polls **biricon.eu** automatically, roughly **once a day** (`UPDATE_INTERVAL_S` = 86400 seconds) – it can be disabled, or pointed at a different URL, under *Settings*. If a newer version is available, the UI shows it with release notes and, when a signed package is provided for your platform, an **“Install now”** button (`POST /api/update/install`, Administrator only).

- **Secure by construction:** only a version file **signed** with the Biricon update key is trusted; the downloaded package is additionally verified against the **signed SHA-256** before anything is installed. A tampered or unsigned file/package is rejected outright.
- **Windows:** the signed installer is downloaded and run **silently** (no dialogs); the service is replaced as part of the install.
- **Docker:** there is no in-place one-click install – the notice points you at the image to pull (`docker compose pull && docker compose up -d`).
- **Debian/Ubuntu – new behavior:** if the installed package is run as the packaged **systemd service**, that service unit sets `NoNewPrivileges=true` (see `build/package_deb.sh`), which makes privilege escalation (including `sudo`, even with a permissive sudoers rule) impossible from that process, by design. The Bridge **detects this up front and no longer even attempts** `sudo` – instead of failing with a confusing `sudo` error, `POST /api/update/install` immediately returns a clear message asking you to run the install **manually**: `sudo apt-get install -y <package>` . (If the Bridge happens to already be running with root privileges, or outside the hardened service, it installs the `.deb` directly via `apt-get` without needing `sudo` at all.)
- Without a provided, signed package for your platform, the notice stays a plain heads-up with a download link – no installation is attempted.

44.2 Startup self-check (tamper protection)

Release builds (packaged/compiled, not a raw development checkout) ship a **signed hash list** (`integrity.json`, Ed25519-signed) covering every delivered program file. At **startup** the Bridge re-hashes its own files and verifies them against this signed list; if a file was changed, replaced, or is missing, **the program refuses to start**. The same check also reruns in the background every 6 hours while running: a failure there does

not crash the running process, but the license is then treated as **not granted** (the installation falls back to demo-mode limits) until the files are restored from a clean install.

44.3 Checking the running version

`GET /api/meta` (or the *About*/version display in the UI) reports the running version; the same information is in `GET /api/update` alongside the update-check status. On the command line, `opcbridge --version` (or the service's startup log line) also shows it.

44.4 Platform-specific update, rollback and uninstall

Windows: run the new setup – it updates the program and keeps data and license. Back up the data directory beforehand. **To roll back/downgrade**, run an **older** setup build over the current install the same way (or uninstall first) – the data directory and license are untouched either way, since they live outside the install folder. **To uninstall**, use *Settings → Apps* (or *Control Panel → Programs and Features → Uninstall a program*) and remove "OPC Bridge"; this does not delete the data directory by itself.

Debian/Ubuntu:

```
sudo apt install ./opcbridge_<new-version>_amd64.deb
```

The package updates the program; data under `/var/lib/opcbridge/` is kept. **To roll back**, install an older `.deb` the same way (`sudo apt install ./opcbridge_<older-version>_amd64.deb`); downgrades are not blocked by the package itself. **To uninstall:** `sudo apt remove opcbridge` (keeps `/var/lib/opcbridge/`) or `sudo apt purge opcbridge` (also removes it).

Docker: `docker compose pull && docker compose up -d` (or `docker compose build` for a self-built image) – the named volume `opcbridge-data` is kept. **To roll back**, pin the image tag to an older version in `docker-compose.yml` and `docker compose up -d` again. **To uninstall:** `docker compose down` (add `-v` only if you also want to delete the `opcbridge-data` volume and its license/configuration).

Migration to another machine: copy the data directory, install the program – and request a **new license** (machine-bound; see chapter 41 for how the Docker machine ID is persisted).

---

45. Performance and sizing

- **Poll intervals** determine load: small `poll_ms` / `update_rate_ms` increase freshness but also CPU and network load. Choose as slow as technically acceptable.
- **Modbus block building:** `max_regs` and `max_gap` combine registers into few requests – important for many registers and slow devices.

- **History:** deadband and “min. interval per point” limit the data volume. Plan disk space according to retention and number of points.
- **Many connections:** each connection runs decoupled (multithreading); a slow peer does not slow the others.
- **Network:** for MQTT/Sparkplug/webhook over WAN, store-and-forward is strongly recommended.

---

46. Troubleshooting

The **first place to look** is always the **Log** tab, or on Linux `journalctl -u opcbridge -f`.

Symptom	Possible cause	Solution
Cannot log in / password forgotten	password forgotten (the `initial_admin_password.txt` file only ever held the very first password – it is useless once the password has been changed)	stop the service, edit `config.json` in t web user list to empty: `"web": { "use` Bridge re-creates the default `admin` ac shown again in `initial_admin_passwo` password immediately.
Connection stays “connecting”/red	IP/port/firewall/device off	check address, port, network; read the
Values are “bad”	source not delivering, wrong address	check address via browse; for Modbus
Modbus number is nonsense	wrong type/byte order	register inspector → pick the right inter
OPC-DA unreachable	DCOM/bitness/Windows	run the Bridge locally on the DA server
OPC-UA “BadSecurityChecksFailed”	certificate not trusted	trust certificates mutually (truststore) c
Change has no effect	not applied	click “Save & apply”.
E-mail/webhook not arriving	wrong SMTP/URL	check SMTP data and webhook URL; te
Live values frozen	WebSocket blocked (proxy)	the UI falls back to polling; open proxy/
Sparkplug: node missing	host was offline before NBIRTH	request a **Rebirth** in the host (NCMD).
Bridge-Link not connecting	token/TLS/port/firewall	same token on both sides; open the OT certificate.
CAN no frames	wrong bitrate/interface	match the bitrate to the bus; bring the termination.
Demo stops after 60 min	no license (the one-hour window is total, from the very first start, not per restart)	activate a license (chapter 41).
A connection never starts, log shows “…nicht gestartet: Lizenz erlaubt max. *N* Verbindungen…”	license (or demo) **connection limit** reached – connections beyond the limit are never started	reduce the number of enabled connect license with a higher/no connection lim

Symptom	Possible cause	Solution
Forwarding was running, then suddenly stops with “License expired” / “Demo time expired” in the Log/ Overview	the license expired, was revoked online, or the demo hour elapsed – this now takes effect **immediately, with no grace period**	renew/reactivate the license (chapter 4 a bug.
2FA user can log into the web page in Basic auth mode, but API calls (and some actions) fail with 401	in **Basic** API-auth mode, the login page still loads with username/ password, but a 2FA user’s API access always requires a Bearer token	call `/api/login` with the TOTP `code` t `Authorization: Bearer <token>` (cha
OPC-UA client/server rejects a certificate (“BadSecurityChecksFailed” or similar) even though you expected to trust it	the peer certificate landed in the connection’s `pki/<id>/rejected` folder instead of `trusted`	open the connection’s certificate mana connection’s certificate section in the U *trusted*, then reconnect.
S7 connection won’t connect / times out	wrong rack/slot for the controller family, firewall blocking TCP port 102 (RFC1006/ISO-on-TCP), or `python-snap7` not installed	check rack/slot against the PLC type (S slot 2), confirm port 102 is reachable, e in the delivered package/image).
S7 address error (“unbekannter S7-Typ-Suffix…”)	an unrecognized type suffix (e.g. a typo like `:float` instead of `:real`)	use only the valid suffixes `:bool :byte :int :uint :word :dint` (chapter 19).
Service won’t start (Linux)	startup error/invalid config.json, or the signed integrity check failed after a modified program file	`journalctl -u opcbridge -f` ; restore failure, reinstall the package rather tha 44.2).
Windows: service “OPCBridge” won’t start / stops immediately	port already in use (8080/4840/502), invalid `config.json` , or a blocked data directory	check `services.msc` for the exact erro `opcbridge.log` in `%ProgramData%\OPC` process holds the configured ports.
Windows: one-click update does nothing / setup doesn’t appear	the signed package download failed, or antivirus/SmartScreen blocked the silently-launched installer	check `GET /api/update` for an error m setup manually instead.
Windows: “Access denied” writing to the data directory	the data directory under `%ProgramData%\OPCBridge\` is no longer world-writable by design (an earlier installer bug that granted “Everyone” write access has been fixed)	run the Bridge as the service account it user trying to edit files directly), or adju operator account genuinely needs write

Part VII – Application examples

47. Example: OPC-DA → OPC-UA

Goal: make values of a legacy OPC-DA controller available to a modern OPC-UA control system.

1. **Create the source:** *OPC-DA Client* connection, enter the DA server ProgID (run the Bridge on the same machine).
2. **Create the target:** *OPC-UA Server* connection, port 4840, security `Basic256Sha256 SignAndEncrypt`.
3. **Browse points:** in the OPC-DA connection, browse the desired items.
4. **Mappings:** one mapping per item *DA.Item → UA path* (e.g. `Plant/Line1/Temperature`), direction A→B. Use the wizard for several at once.
5. **Save & apply.** The control system now connects as a UA client to `opc.tcp://<bridge>:4840/opcbridge`.

---

48. Example: Modbus → MQTT cloud

Goal: a Modbus energy meter should send its measured values to an MQTT cloud.

1. **Source:** *Modbus TCP Master* connection, meter host/IP, port 502.
2. **Identify values:** use the **register inspector** to determine type and byte order (e.g. `HR100:float32:CDAB` = active power).
3. **Target:** *MQTT* connection, broker address, TLS on, user/password, topic prefix `plant/meter1/`.
4. **Mappings:** `HR100:float32:CDAB → power`, `HR102:float32:CDAB → energy`, etc.
5. **Store-and-forward** enabled on the MQTT connection (WAN may drop).
6. **Save & apply.**

---

49. Example: CAN → OPC-UA

Goal: provide the engine speed and temperature of a J1939 diesel generator as OPC-UA.

1. **Source:** *CAN* connection, interface `socketcan`, channel `can0`, bitrate 250000.
2. **Analyzer:** read frames; identify the J1939 PGN (e.g. PGN 61444 for speed).
3. **Create signals:** `Speed` (start bit/length/factor 0.125) and `CoolantTemp` (factor 1, offset −40) – or via DBC import.
4. **Target:** *OPC-UA Server* connection.
5. **Mappings:** `Speed → Generator/Engine/Speed`, `CoolantTemp → Generator/Engine/Temp`.
6. **Save & apply.**

---

50. Example: OT→IT across the DMZ

Goal: bring process values from production (OT) into the IT zone without opening the OT firewall for inbound connections.

OT side (production network):

1. Install the Bridge; connect the process sources (Modbus/OPC-UA/…).
2. *Bridge-Link* connection, role **client**, server URL `wss://dmz-host:8765`, set a shared **token**, enable store-and-forward.
3. Mappings: process points → Bridge-Link tags (same names as on the IT side).

IT/DMZ side:

1. Install the Bridge.
2. *Bridge-Link* connection, role **server**, port 8765, TLS with certificate, **same token**.
3. Mappings: Bridge-Link tags → target (OPC-UA server for the control system, database, MQTT …).

Result: the OT side establishes the connection **outbound only**. No **inbound port** is needed in the OT firewall. Values flow mirrored into IT; on disconnection the OT side buffers.

Part VIII – Appendix

A. Address formats (complete)

Connection	Format / examples
OPC-DA	`Channel1.Device1.Tag1`
OPC-UA (client)	`ns=2;s=Line1.Temp`, `ns=3;i=1001`
OPC-UA (server)	path `Plant/Line1/Temperature` → NodeId `ns=<idx>;s=…`
Siemens S7	`DB1.DBX0.0` (bool), `DB1.DBW4` (int), `DB10.DBD20` (real, the default for DBD/MD/ID/QD), `DB1.DBD0:dint`, `M10.0`, `MW20`, `Q0.1`; type suffix `:bool :byte :int :uint :word :dint :udint :dword :real :string<n>`
Modbus	`[Unit:]Area Offset[.Bit][:Type[:ByteOrder]]` — `HR100:float32`, `2:IR10:int32:CDAB`, `CO5`, `HR7.3`, `40001:uint16`, `HR200:string20`
MQTT	`hall1/temp`; JSON path `sensor/abc@data.temperature`
Sparkplug B	metric name, e.g. `Temperature`, `Line3/Count`
SQL	column/key name; write statement with `:addr`, `:value`, `:ts`
InfluxDB	tag name (appended as a tag to the measurement)
CAN	signal name from the signal list (analyzer/DBC)
Webhook / Bridge-Link	free tag name

Modbus areas: `CO` (coils), `DI` (discrete inputs), `IR` (input registers), `HR` (holding registers); 0-based or Modicon notation (`0xxxx` / `1xxxx` / `3xxxx` / `4xxxx`). **Byte order:** `ABCD`, `CDAB`, `BADC`, `DCBA`.

B. Formula and function reference

The variable `x` is the incoming value; for computed points the inputs are available as named variables.

Category	Functions / operators
Arithmetic	`+`, `-`, `*`, `/`, `//` (floor division), `%` (modulo), `**` (power)
Comparison/logic	`>`, `<`, `>=`, `<=`, `==`, `!=`, `and`, `or`, `not`

Category	Functions / operators
Bitwise	`&` (and), `|` (or), `^` (xor), `~` (invert), `<<` (left shift), `>>` (right shift)
Rounding	`round(x, n)`, `floor(x)`, `ceil(x)`, `int(x)`, `abs(x)`
Limit/scale	`clamp(x, min, max)`, `scale(x, in_min, in_max, out_min, out_max)`
Aggregation	`min(a, b, …)`, `max(a, b, …)`, `sum(a, b, …)`, `avg(a, b, …)` (alias `mean(...)`), `len(x)`
Null handling	`coalesce(a, b, …)` – first argument that is not `None` / `null`
Conditional	`x if condition else y` (ternary), `ifelse(condition, a, b)` (function form), `and_(*a)` / `or_(*a)` (function forms of all()/any() over several arguments)
Math	`sqrt(x)`, `pow(x, y)`, `log(x)`, `log10(x)`, `exp(x)`, `sin(x)`, `cos(x)`, `tan(x)`
Conversion	`int(x)`, `float(x)`, `bool(x)`, `str(x)`
Constants	`pi`, `e`, `true` / `false` / `True` / `False`, `None` / `null`

`avg` / `sum` / `min` / `max` / `coalesce` also accept a single list/tuple argument (e.g. from several computed-point inputs) and flatten it automatically. Division by zero returns an empty/ `None` result instead of raising an error.

Examples: `x*1.8+32`, `scale(x,0,27648,0,100)`, `clamp(x,0,100)`, `round(x,1)`, `x if x>0 else 0`, `ifelse(x>50, 1, 0)`, `avg(t1, t2, t3)`, `coalesce(a, b, 0)`, `(x >> 8) & 0xFF` (high byte of a 16-bit value).

> »**Note**« For safety the formula interpreter only allows this fixed whitelist of operators, literals and functions (as an AST walk, not `eval` of arbitrary Python) – no attribute access, no loops, no imports, no file access, no system calls.

C. REST endpoint reference

Base URL: `http://<host>:8080`. Interactive docs: `/api/docs`.

Method	Path	Purpose
GET	`/api/authinfo`	query the auth mode (public)
POST	`/api/login`	log in (optional `code` for 2FA), obtain token
POST	`/api/token`	OAuth2 token (form)
POST	`/api/logout`	revoke token

Method	Path	Purpose
GET/POST	`/api/2fa/status`, `/api/2fa/setup`, `/api/2fa/enable`, `/api/2fa/disable`	manage two-factor
GET	`/api/oidc/start`, `/api/oidc/callback`	OIDC/SSO sign-in
GET	`/api/meta`	product/version info
GET	`/api/status`	status of all connections/mappings
GET	`/api/values`	current values
GET	`/api/history`	history data (JSON)
GET	`/api/history.csv`	history as CSV
GET	`/api/events`	OPC-UA events
GET	`/api/notifications`	alarm events
POST	`/api/notifications/test`	send a test notification
GET	`/api/log`	operating log (Log tab), polled with `?after=<seq>`
GET/PUT	`/api/config`	read/write configuration (PUT: admin; GET masks all secrets, see chapter 40)
POST	`/api/restart`	restart the engine (admin)
POST	`/api/browse`	browse the address tree
POST	`/api/opcda/servers`	list locally registered OPC-DA ProgIDs
POST	`/api/write`	write a value (admin)
POST	`/api/inspect`	Modbus register inspector
POST	`/api/can/analyze`	CAN analysis
POST	`/api/can/dbc`	DBC import (admin)
GET/POST/DELETE	`/api/license`	license status / activate / remove
POST	`/api/license/online/activate`	run the online license check/activation now (admin)
GET	`/api/pki/{conn}`	list the OPC-UA connection's own/trusted/rejected certificates
GET	`/api/pki/{conn}/own_cert.der`	download the connection's own certificate
POST	`/api/pki/{conn}/trust`	

Method	Path	Purpose
		move a rejected certificate into the trusted store (admin)
POST	`/api/pki/{conn}/delete`	delete a stored certificate (admin)
GET	`/api/update`	update-checker status (available version, release notes)
POST	`/api/update/install`	one-click install of the newest signed version (admin)
GET	`/api/wsticket`	ticket for WebSocket
WS	`/ws`	real-time data stream
GET	`/api/docs`, `/api/redoc`, `/api/openapi.json`	API documentation
GET	`/view`, `/api/view`	WebView (token-protected)

Writing endpoints require the **Administrator** role. This list is not exhaustive – the authoritative, always up-to-date list is the built-in documentation at `/api/docs`.

D. Data type reference

Type	Description	Range	Modbus registers
`bool`	truth value	true/false	1 bit (CO/DI) or bit of a register
`int16`	16-bit signed	−32768…32767	1
`uint16`	16-bit unsigned	0…65535	1
`int32`	32-bit signed	±2.1 bn	2
`uint32`	32-bit unsigned	0…4.29 bn	2
`int64`	64-bit signed	very large	4
`uint64`	64-bit unsigned	$0…1.8 \cdot 10^{19}$	4
`float32`	32-bit float (IEEE 754)	approx. $\pm 3.4 \cdot 10^{38}$	2
`float64`	64-bit float (also called double precision)	high precision	4
`string`	text	depending on length	n (2 chars/register)

E. Default ports

Service	Port	Note
Web interface / REST	8080	configurable
OPC-UA server	4840	if enabled
Modbus slave	502	if enabled
MQTT (broker, peer)	1883 / 8883	8883 = TLS
Bridge-Link	8765	DMZ server, configurable
InfluxDB (peer)	8086	v1/v2/v3
Siemens S7 (peer)	102	RFC1006/ISO-on-TCP
CAN-over-Ethernet (UDP)	43113	with `udp_multicast`

These are the ports the Bridge itself listens on. In the supplied Docker Compose setup the **Modbus slave is additionally remapped to host port 1502** (`1502:502` , chapter 8) to avoid needing a privileged host port – inside the container it is still port 502.

F. File and directory locations

Platform	Data directory	Initial password
Windows	`%ProgramData%\OPCBridge\`	`…\initial_admin_password.txt`
Linux (.deb)	`/var/lib/opcbridge/`	`…/initial_admin_password.txt`
Docker	named volume `opcbridge-data` , mounted at `/data`	`/data/initial_admin_password.txt`

The data directory also holds: `config.json` (configuration – connection secrets in plain text, chapter 40), `license.key` (offline license), `machine_id` (persisted license-binding fingerprint), `online_state.json` (online-license state), `update_state.json` (update-check state), `demo_started_at.txt` (one-time demo-window start marker), `history.sqlite` (history database), `sf_queue.sqlite` (persistent store-and-forward queue), the `pki/` folder (per-OPC-UA -connection certificates: own/trusted/rejected), and the log files `opcbridge.log` and `service.log` (chapter 14). See chapter 43 for the full backup procedure.

G. Commissioning checklist

- [ ] Bridge installed, service running, web interface reachable
- [ ] Initial password changed, operator user created if needed
- [ ] API authentication set appropriately (plaintext/basic/token)

- [ ] HTTPS enabled (production)
- [ ] Source connection(s) created and **green**
- [ ] Target connection(s) created and **green**
- [ ] For Modbus: type/byte order verified with the register inspector
- [ ] Mappings created, direction correct
- [ ] “Save & apply” clicked
- [ ] Live values checked (do values arrive?)
- [ ] Store-and-forward enabled on WAN/cloud targets
- [ ] History configured (if needed)
- [ ] Alarms set up and tested (e-mail/webhook)
- [ ] License activated (continuous operation)
- [ ] Backup of the data directory created

H. Glossary

Term	Meaning
A&E	Alarms & Events – OPC-UA event messages
Byte order	order of bytes in multi-part numbers (ABCD/CDAB/BADC/DCBA)
DBC	description file for CAN signals
DMZ	isolated network zone between OT and IT
Edge node	data source at the plant edge (Sparkplug)
Engine	internal core executing mappings
Mapping	rule “value A → value B”
Master/Slave	Modbus roles (poll / provide)
Online license	optional license mode verified against the Biricon license server (chapter 41.3), as opposed to the offline serial number
OT/IT	production network / enterprise network
PGN/SA	Parameter Group Number / Source Address (J1939)
PKI folder	per-connection certificate store for OPC-UA (`pki/<id>/` : own/trusted/rejected)
Publish/Subscribe	MQTT principle (publish/subscribe)
Quality	goodness of a value (good/uncertain/bad)
Store-and-forward	buffering on target failure
Tag/Point	a single value
UNS	Unified Namespace (consistent data structure over Sparkplug)

I. Frequently asked questions (FAQ)

Must the browser stay open for data to flow? No. The Bridge runs as a background service. The browser is only for operation.

Can a value go in both directions? Yes, with direction A↔B. The most recently changed value is transferred.

What happens when the target connection drops? Without store-and-forward values are lost; with S&F they are buffered and delivered later.

Why does my Modbus value show nonsense? Almost always a wrong data type or byte order – check with the register inspector.

Do I need the internet for the API documentation? No. The documentation is fully bundled and works offline.

Does OPC-DA run on Linux? No, OPC-DA requires Windows. All other protocols run cross-platform.

How do I bring values securely from production into IT? Use the Bridge-Link in DMZ mode – outbound connection only, no inbound OT port.

How do I back up the configuration? Back up the data directory (at least `config.json`), see chapter 43.

OPC Bridge – Technical Operations Manual Version 1.1.0 · © 2026 **Biricon IT Services GmbH** · www.biricon.eu

All trademarks are property of their respective owners. Subject to technical change.