Inhaltsverzeichnis

OPC Bridge **4**
Technisches Betriebshandbuch 5
Inhaltsverzeichnis 6

Teil I – Grundlagen **8**
1. Einführung und Überblick 9
2. Systemarchitektur 13
3. Grundbegriffe 15
4. Protokoll-Grundlagen im Detail 17

Teil II – Installation und Inbetriebnahme **31**
5. Systemvoraussetzungen 32
6. Installation unter Windows 33
7. Installation unter Debian/Ubuntu/Raspberry Pi 34
8. Installation als Docker-Container 35
9. Erste Inbetriebnahme Schritt für Schritt 36

Teil III – Bedienung **37**
10. Aufbau der Weboberfläche 38
11. Reiter Übersicht 39
12. Reiter Live-Werte 40
13. Reiter Trends 41
14. Reiter Log 42

Teil IV – Konfiguration **43**
15. Verbindungen – allgemein 44
16. Verbindungstyp OPC-DA 46
17. Verbindungstyp OPC-UA-Client 47
18. Verbindungstyp OPC-UA-Server 48

19. Verbindungstyp Modbus 49
20. Verbindungstyp MQTT 51
21. Verbindungstyp Sparkplug B 52
22. Verbindungstyp CAN 53
23. Verbindungstyp SQL 54
24. Verbindungstyp Webhook 55
25. Verbindungstyp Bridge-Link 56
26. Mappings und Datenfluss 58
27. Umrechnung und Formeln 60
28. Virtuelle Quellen und Redundanz 62
29. Store-and-Forward 64
30. Historie und Trends konfigurieren 65
31. Alarme, E-Mail und Webhook 66
32. CAN-Analyser und Gateway 68
33. Modbus-Register-Inspektor 70
34. WebView 71
35. Benutzer, Rollen und API-Authentifizierung 72
36. Echtzeit-Aktualisierung (WebSocket) 75
36a. Neuerungen in Version 1.1 76

Teil V – Schnittstellen und Integration **81**
37. REST-API und eingebaute Dokumentation 82
38. Konfigurationsdatei (config.json) 84

Teil VI – Betrieb **85**
39. Lizenzierung 86
40. Sicherheit und Härtung 88
41. Datensicherung und Wiederherstellung 89
42. Update und Migration 90
43. Leistung und Dimensionierung 91
44. Fehlersuche 92

Teil VII – Anwendungsbeispiele **94**
45. Beispiel: OPC-DA → OPC-UA 95
46. Beispiel: Modbus → MQTT-Cloud 96
47. Beispiel: CAN → OPC-UA 97
48. Beispiel: OT→IT über die DMZ 98

Teil VIII – Anhang **99**

A. Adressformate (vollständig) 100
B. Formel- und Funktionsreferenz 101
C. REST-Endpunkt-Referenz 102
D. Datentyp-Referenz 104
E. Standard-Ports 105
F. Datei- und Verzeichnisorte 106
G. Inbetriebnahme-Checkliste 107
H. Glossar 108
I. Häufige Fragen (FAQ) 109



OPC Bridge

Technisches Betriebshandbuch

Industrieller Protokollkonverter für OPC-DA, OPC-UA, Modbus, MQTT, Sparkplug B, CAN/J1939, SQL und REST

Produktversion **1.0.0** · Handbuchstand **November 2026**

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

Zweck und Zielgruppe. Dieses Betriebshandbuch beschreibt die OPC Bridge vollständig – von den physikalischen Grundlagen der unterstützten Feldbus- und Netzwerkprotokolle über die Installation und Konfiguration bis zur Bedienung, Wartung und Fehlersuche. Es richtet sich gleichermaßen an **Inbetriebnehmer**, **Automatisierungstechniker**, **IT-/OT-Administratoren** und **Instandhalter**. Jeder Fachbegriff wird beim ersten Auftreten erklärt; die Grundlagenkapitel setzen **kein** Vorwissen voraus. Fortgeschrittene Leser können die Grundlagen überspringen und direkt zu den Konfigurationskapiteln gehen.

Sprachfassungen. Dieses Dokument liegt in Deutsch und Englisch vor. Die englische Fassung finden Sie in `Operations-Manual-EN.md`.

Konventionen. `Monospace` kennzeichnet Eingaben, Adressen, Dateinamen und Code. *Kursiv* kennzeichnet Bedienelemente und Fachbegriffe. Hinweiskästen mit **»Hinweis«**, **»Achtung«** und **»Tipp«** heben besonders wichtige Punkte hervor.

Inhaltsverzeichnis

Teil I – Grundlagen 1. Einführung und Überblick 2. Systemarchitektur 3. Grundbegriffe 4. Protokoll-Grundlagen im Detail - 4.1 Allgemeines zu industrieller Kommunikation - 4.2 OPC-DA (Classic) - 4.3 OPC-UA - 4.4 Modbus TCP und 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 Datentypen und Byte-Reihenfolge

Teil II – Installation und Inbetriebnahme 5. Systemvoraussetzungen 6. Installation unter Windows 7. Installation unter Debian/Ubuntu/Raspberry Pi 8. Installation als Docker-Container 9. Erste Inbetriebnahme Schritt für Schritt

Teil III – Bedienung 10. Aufbau der Weboberfläche 11. Reiter Übersicht 12. Reiter Live-Werte 13. Reiter Trends 14. Reiter Log

Teil IV – Konfiguration 15. Verbindungen – allgemein 16. Verbindungstyp OPC-DA 17. Verbindungstyp OPC-UA-Client 18. Verbindungstyp OPC-UA-Server 19. Verbindungstyp Modbus 20. Verbindungstyp MQTT 21. Verbindungstyp Sparkplug B 22. Verbindungstyp CAN 23. Verbindungstyp SQL 24. Verbindungstyp Webhook 25. Verbindungstyp Bridge-Link 26. Mappings und Datenfluss 27. Umrechnung und Formeln 28. Virtuelle Quellen und Redundanz 29. Store-and-Forward 30. Historie und Trends konfigurieren 31. Alarme, E-Mail und Webhook 32. CAN-Analyser und Gateway 33. Modbus-Register-Inspektor 34. WebView 35. Benutzer, Rollen und API-Authentifizierung 36. Echtzeit-Aktualisierung (WebSocket) 36a. Neuerungen in Version 1.1 — S7, InfluxDB, Batch, Data-Diode, LDAP/OIDC/2FA, Mehrsprachigkeit

Teil V – Schnittstellen und Integration 37. REST-API und eingebaute Dokumentation 38. Konfigurationsdatei (config.json)

Teil VI – Betrieb 39. Lizenzierung 40. Sicherheit und Härtung 41. Datensicherung und Wiederherstellung 42. Update und Migration 43. Leistung und Dimensionierung 44. Fehlersuche

Teil VII – Anwendungsbeispiele 45. Beispiel: OPC-DA → OPC-UA 46. Beispiel: Modbus → MQTT-Cloud 47. Beispiel: CAN → OPC-UA 48. Beispiel: OT→IT über die DMZ

Teil VIII – Anhang - A. Adressformate (vollständig) - B. Formel- und Funktionsreferenz - C. REST-Endpunkt-Referenz - D. Datentyp-Referenz - E. Standard-Ports - F.

Datei- und Verzeichnisorte - G. Inbetriebnahme-Checkliste - H. Glossar - I. Häufige Fragen (FAQ)

Teil I – Grundlagen

1. Einführung und Überblick

1.1 Was ist die OPC Bridge?

Die **OPC Bridge** ist ein **universeller Protokollkonverter** für die industrielle Kommunikation. In einer typischen Anlage existieren viele Geräte und Systeme, die unterschiedliche „Sprachen" – sogenannte **Protokolle** – sprechen:

- eine ältere Maschinensteuerung stellt Werte per **OPC-DA** bereit,
- ein Energie- oder Durchflusszähler kommuniziert über **Modbus**,
- ein moderner Sensor sendet über **MQTT** in eine Cloud,
- ein Dieselaggregat oder Fahrzeug liefert Daten über den **CAN-Bus**,
- das übergeordnete Leitsystem erwartet **OPC-UA**.

Damit diese Geräte zusammenarbeiten, müssen ihre Werte **von einem Protokoll in ein anderes übersetzt** werden. Genau das leistet die OPC Bridge: Sie verbindet sich gleichzeitig mit mehreren Quellen und Zielen und überträgt Werte zwischen ihnen – **in beide Richtungen**.

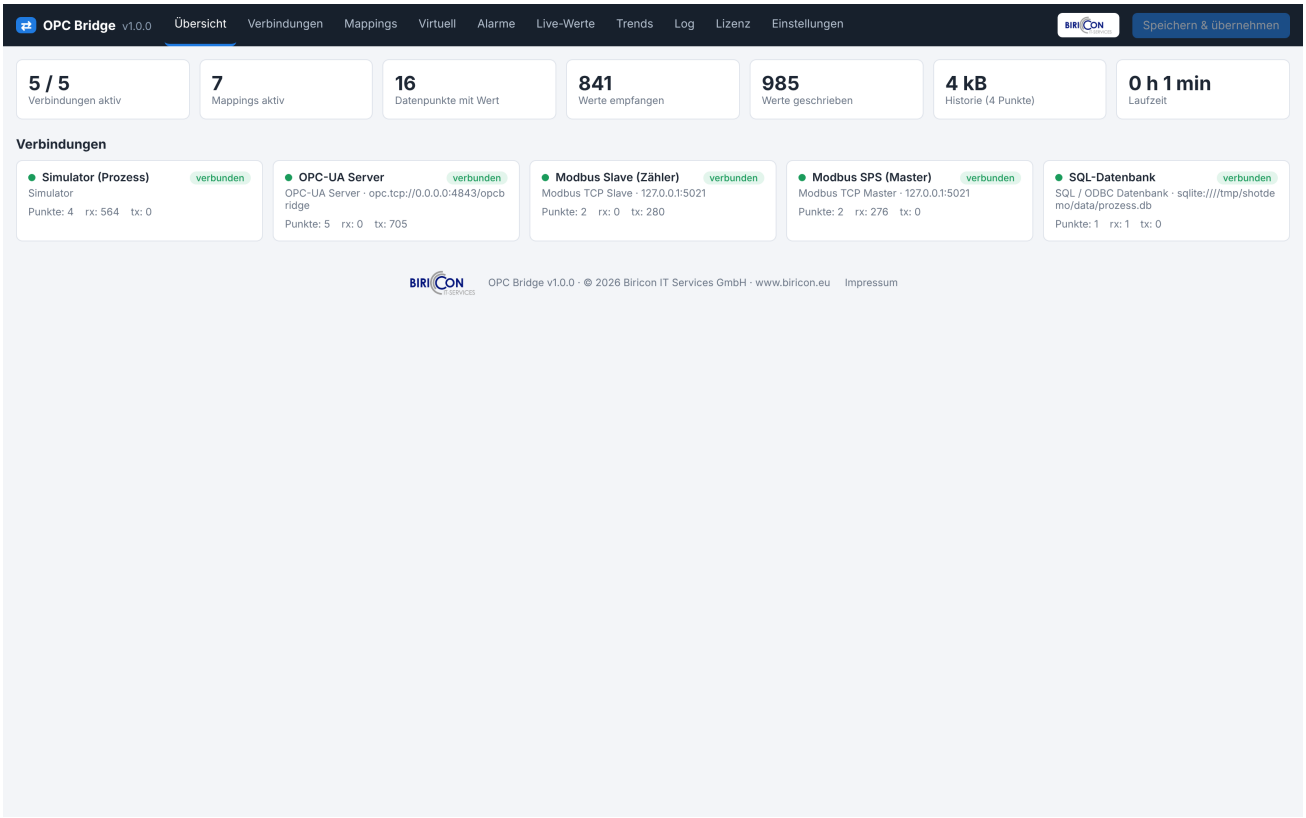


Figure 1: Übersicht

1.2 Das Bild vom Dolmetscher

Stellen Sie sich einen **runden Tisch** vor, an dem Menschen aus verschiedenen Ländern sitzen. Jeder spricht nur seine eigene Sprache. In der Mitte sitzt ein **Dolmetscher**, der jede Aussage entgegennimmt und an die anderen Teilnehmer in deren jeweiliger Sprache weitergibt.

Die OPC Bridge **ist** dieser Dolmetscher. Sie als Anwender müssen nur zwei Dinge festlegen:

1. **Wer sitzt am Tisch?** – das sind die **Verbindungen** (z. B. „die Modbus-SPS", „das OPC-UA-Leitsystem").
2. **Wer soll wem was sagen?** – das sind die **Mappings** (z. B. „Temperatur der SPS → Temperatur im Leitsystem").

Alles Weitere – das ständige Abfragen, Umrechnen, Puffern bei Störungen und Weiterleiten – erledigt die Bridge selbsttätig.

1.3 Funktionen im Überblick

Bereich	Funktionen
Protokolle	OPC-DA (Client), OPC-UA (Client und Server), **Siemens S7 (direkt)**, Modbus TCP/RTU (Master und Slave), MQTT, MQTT Sparkplug B, SQL/ODBC, **InfluxDB**, CAN (inkl. J1939), REST/Webhook, Simulator
Datenfluss	Beidseitiges Routing (A→B, B→A, ↔), Zusammenführung mehrerer Quellen, Multithreading
Echtzeit	Live-Werte per WebSocket mit Polling-Fallback
Durchsatz	**Batch-/Bulk-Schreiben** (SQL-Multi-Row, JSON-Array-Webhook, InfluxDB-Line-Protocol)
Umrechnung	Faktor/Offset, freie Formeln, Typ- und Byteorder-Konvertierung
Virtuelle Quellen	Berechnete Punkte, Redundanz/Hot-Standby

Bereich	Funktionen
Zuverlässigkeit	Persistenter Store-and-Forward (SQLite), automatische Wiederverbindung
OT→IT-Kopplung	Bridge-Link: ausgehender TLS-Tunnel ohne eingehende Firewall-Ports, optional **strikter Einweg-/Data-Diode-Modus**
Überwachung	Historie, Trends, Alarmregeln (E-Mail/Webhook), OPC-UA A&E
Teilen	WebView – token-geschützte Nur-Lese-Live-Ansicht
REST-API	Vollständige API mit eingebauter OpenAPI-Dokumentation (offline)
Sicherheit	Konfigurierbare API-Authentifizierung, Rollen, HTTPS, OPC-UA-Security, **LDAP/Active Directory, OIDC/SSO, TOTP-2FA**
Oberfläche	**Mehrsprachig** (Deutsch, Englisch, Französisch, Spanisch, Portugiesisch)
Kopierschutz	Rechnergebundene, signierte Seriennummern (Ed25519), Demo-Modus
Auslieferung	Windows-Setup (.exe), Debian-Paket (.deb), Docker-Image

1.4 Typische Einsatzszenarien

- **Modernisierung:** Eine alte OPC-DA-Anlage soll an ein neues OPC-UA-Leitsystem angebunden werden, ohne die Steuerung zu tauschen.
- **Digitalisierung/IIoT:** Modbus-Zähler sollen ihre Werte in eine MQTT-Cloud oder ein Unified Namespace (Sparkplug B) liefern.
- **Fahrzeug-/Aggregatanbindung:** CAN-Signale eines Generators werden als OPC-UA- oder Modbus-Werte bereitgestellt.
- **Sichere OT/IT-Kopplung:** Werte werden aus der Produktion über eine DMZ in die IT transportiert, ohne die Produktions-Firewall zu öffnen.

- **Protokollbrücke Modbus↔OPC:** Ein OPC-UA-Client soll auf Modbus-Register zugreifen.

---

2. Systemarchitektur

2.1 Komponentenmodell

Die OPC Bridge besteht intern aus mehreren Schichten, die zusammenarbeiten:

- **Treiber (Driver):** Für jedes Protokoll gibt es einen Treiber. Er kümmert sich um das Verbinden, Abfragen, Schreiben und Melden von Werten. Jede *Verbindung* in der Oberfläche ist eine Instanz eines Treibers.
- **Engine:** Das Herzstück. Sie verwaltet alle Treiber, führt die Mappings aus (leitet Werte weiter), wendet Formeln an, puffert bei Störungen (Store-and-Forward) und schreibt die Historie.
- **Virtuelle Quellen:** Berechnete Punkte und Redundanzgruppen, die aus vorhandenen Werten neue Werte bilden.
- **Benachrichtigung (Notifier):** Prüft Alarmregeln und versendet E-Mails/Webhooks.
- **Weboberfläche und REST-API:** Bedienung und Integration über den Browser bzw. über Programme.

2.2 Datenfluss

Ein Wert durchläuft typischerweise diesen Weg:

1. Der **Quell-Treiber** liest den Wert vom Gerät (oder empfängt ihn) und meldet ihn mit **Adresse**, **Wert**, **Qualität** und **Zeitstempel** an die Engine.
2. Die **Engine** sucht alle Mappings, in denen dieser Punkt die Quelle ist.
3. Für jedes Mapping wird – falls konfiguriert – die **Formel** angewendet und der Datentyp konvertiert.
4. Der Wert wird an den **Ziel-Treiber** übergeben. Ist das Ziel nicht erreichbar und Store-and-Forward aktiv, wird der Wert **gepuffert**.
5. Der **Ziel-Treiber** schreibt den Wert zum Gerät (oder veröffentlicht ihn).

Gleichzeitig kann der Wert in die **Historie** geschrieben, gegen **Alarmregeln** geprüft und an die **Live-Oberfläche** (per WebSocket) gesendet werden.

2.3 Betriebsmodell

Die Bridge läuft als **Dienst** (Windows-Dienst, systemd-Dienst oder Docker-Container) im Hintergrund – unabhängig davon, ob ein Browser geöffnet ist. Die

Weboberfläche ist nur das Bedien- und Beobachtungswerkzeug; der eigentliche Datenaustausch läuft im Dienst.

> **»Hinweis«** Die Bridge arbeitet weiter, auch wenn kein Browser verbunden ist. Schließen des Browsers stoppt die Datenübertragung **nicht**.

---

3. Grundbegriffe

Die folgenden Begriffe werden im gesamten Handbuch verwendet.

Begriff	Bedeutung
Verbindung (Connection)	Eine Anbindung an ein Gerät oder System. Hat einen **Typ** (= Protokoll) und eigene Einstellungen.
Treiber (Driver)	Die Software hinter einem Verbindungstyp (z. B. der Modbus-Treiber).
Punkt / Tag	Ein einzelner Wert innerhalb einer Verbindung (z. B. eine Temperatur).
Adresse	Bezeichnet den Ort/Namen eines Punkts im Gerät (z. B. `HR100` oder `ns=2;s=Temp`).
Mapping	Regel „Quelle A → Ziel B". Bestimmt, welcher Punkt zu welchem übertragen wird.
Richtung	A→B, B→A oder A↔B (beidseitig).
Qualität (Quality)	Güte eines Werts: *good*, *uncertain*, *bad*.
Zeitstempel	Zeitpunkt, zu dem der Wert gültig war.
Master / Slave	Modbus-Rollen: Master fragt ab, Slave antwortet/hält Werte bereit.
Client / Server	Der Client verbindet sich zum Server. Die Bridge kann je nach Protokoll beides sein.
Publish / Subscribe	MQTT-Prinzip: veröffentlichen bzw. abonnieren von Topics.
OT / IT / DMZ	Produktionsnetz / Büronetz / abgeschottete Zwischenzone.
Engine	Der interne Kern, der Mappings ausführt.
Store-and-Forward	Puffern von Werten, wenn ein Ziel offline ist.

Begriff	Bedeutung
Virtuelle Quelle	Berechneter Punkt oder Redundanzgruppe.

4. Protokoll-Grundlagen im Detail

Dieses Kapitel erklärt **jedes unterstützte Protokoll** von Grund auf: wozu es dient, wie es aufgebaut ist, wie eine Nachricht konkret aussieht, und worauf bei der Konfiguration zu achten ist. Sie müssen nur die Abschnitte lesen, die Ihre Anlage betreffen.

4.1 Allgemeines zu industrieller Kommunikation

In der Automatisierung unterscheidet man grob zwei Welten:

- **Feldebene (OT, *Operational Technology*):** Sensoren, Aktoren, Steuerungen (SPS). Hier zählen Echtzeit, Robustheit und einfache, schlanke Protokolle (Modbus, CAN).
- **Leit-/Unternehmensebene (IT):** SCADA, MES, ERP, Cloud. Hier zählen Sicherheit, Struktur und Interoperabilität (OPC-UA, MQTT, REST).

Protokolle lassen sich nach ihrem **Kommunikationsmuster** einteilen:

Muster	Erklärung	Beispiele
Polling (Abfrage)	Ein Teilnehmer fragt aktiv Werte ab, der andere antwortet.	Modbus-Master, OPC-UA-Client (Read)
Subscription (Abo)	Der Server schickt nur noch bei Änderungen.	OPC-UA-Subscription
Publish/Subscribe	Sender veröffentlichen, Empfänger abonnieren – entkoppelt über einen Broker.	MQTT, Sparkplug B
Broadcast/Bus	Alle Teilnehmer hören mit, Nachrichten tragen eine ID.	CAN

Die OPC Bridge beherrscht alle diese Muster gleichzeitig und übersetzt zwischen ihnen.

Ein wiederkehrendes Thema ist die **Darstellung von Zahlen**: Je nach Protokoll werden Werte als einzelne Bits, als 16-Bit-Register, als 32-Bit-Gleitkommazahlen oder als Bitfelder in Botschaften übertragen. Abschnitt 4.12 behandelt Datentypen

und Byte-Reihenfolge zentral, weil dies die häufigste Fehlerquelle bei der Inbetriebnahme ist.

4.2 OPC-DA (Classic)

Herkunft und Zweck. OPC (ursprünglich *OLE for Process Control*) entstand ab 1996, um den Datenaustausch zwischen Windows-Automatisierungssoftware und Steuerungen zu vereinheitlichen. **OPC-DA** (*Data Access*) ist der am weitesten verbreitete Teil davon – praktisch jedes ältere SCADA-System und jede Windows-Treibersoftware bietet einen OPC-DA-Server. Auch heute sind in Bestandsanlagen Millionen OPC-DA-Schnittstellen im Einsatz.

Technischer Aufbau. OPC-DA basiert auf Microsofts Komponentenmodell **COM** bzw. – über Rechnergrenzen hinweg – **DCOM**. Die Daten eines Servers sind hierarchisch organisiert:

```
OPC-DA-Server
 └─ Gruppe (z. B. "Linie1")
    └─ Item  (z. B. "Channel1.Device1.Tag1")  → Wert + Qualität + Zeitstempel
```

Ein **Item** ist ein einzelner Datenpunkt. Jedes Item hat:

- einen **ItemID** (die Adresse, z. B. `Channel1.Device1.Temperatur`),
- einen **Wert** (Zahl, Text, Wahrheitswert …),
- eine **Qualität** (good/uncertain/bad) und
- einen **Zeitstempel**.

Die OPC Bridge tritt als **OPC-DA-Client** auf: Sie verbindet sich mit einem lokalen oder entfernten DA-Server (über dessen *ProgID* wie `Matrikon.OPC.Simulation`), legt eine Gruppe an und liest bzw. schreibt Items.

Adressformat. `Channel1.Device1.Tag1` – die genaue Schreibweise gibt der jeweilige Server vor. Über die **Browse**-Funktion kann die Bridge den Adressbaum des Servers durchsuchen.

Besonderheiten und Stolpersteine.

- OPC-DA läuft **ausschließlich unter Windows** (COM/DCOM-Abhängigkeit).
- Der Zugriff über Rechnergrenzen (**DCOM**) ist notorisch kompliziert zu konfigurieren (Benutzerrechte, Firewall, DCOM-Berechtigungen). **Empfehlung:**

Bridge und DA-Server auf **demselben Rechner** betreiben – dann entfällt DCOM vollständig.

- Die **Bitbreite** (32/64-Bit) der Bridge muss zum installierten *OPC Core Components*-Wrapper passen.

»Tipp« Wenn Sie eine OPC-DA-Anlage modernisieren wollen, ist die typische Aufgabe: *OPC-DA lesen → als OPC-UA bereitstellen*. Dafür legen Sie eine OPC-DA-Verbindung (Quelle) und eine OPC-UA-Server-Verbindung (Ziel) an und verbinden die Punkte per Mapping.

4.3 OPC-UA

Herkunft und Zweck. OPC-UA (*Unified Architecture*, ab 2008) ist der moderne Nachfolger von OPC-DA. Er löst dessen Hauptprobleme: Er ist **plattformunabhängig** (läuft auf Windows, Linux, in Steuerungen, in der Cloud), **sicher** (Verschlüsselung, Signatur, Zertifikate, Benutzerauthentifizierung) und **selbstbeschreibend** (ein Client kann den Adressraum erkunden und Datentypen auslesen). OPC-UA ist heute der De-facto-Standard für die Anbindung von Leitsystemen (IEC 62541).

Technischer Aufbau. OPC-UA-Daten liegen in einem **Adressraum** aus **Knoten (Nodes)**. Jeder Knoten hat:

- eine **NodeId** – die eindeutige Adresse, bestehend aus einem **Namespace-Index** (`ns=`) und einem **Identifier**. Der Identifier ist entweder
  - ein **String** (`s=`), z. B. `ns=2;s=Linie1.Temperatur`, oder
  - eine **Zahl** (`i=`), z. B. `ns=3;i=1001`,
- einen **BrowseName** und **DisplayName** (lesbare Bezeichnungen),
- eine **Knotenklasse** (Variable, Objekt, Methode …),
- bei Variablen einen **Wert** mit **Datentyp**, **Qualität** (StatusCode) und **Zeitstempel**.

Der Namespace-Index trennt verschiedene „Namensräume": `ns=0` ist der Standard-Raum von OPC-UA selbst, `ns=2`, `ns=3` … sind herstellerspezifische Räume.

Zwei Betriebsarten in der Bridge.

- **OPC-UA-Client:** Die Bridge verbindet sich zu einem fremden Server (per

Endpoint-URL wie `opc.tcp://192.168.0.10:4840`) und liest/schreibt dort Knoten. Zusätzlich kann sie **Subscriptions** anlegen (Server meldet nur noch Änderungen) und **Alarms & Events (A&E)** abonnieren.

- **OPC-UA-Server:** Die Bridge stellt selbst einen Server bereit (Standard-Port 4840). Andere Systeme (Leitsysteme) verbinden sich und lesen die bereitgestellten Werte. Der von Ihnen vergebene Pfad `Anlage/Linie1/Temperatur` wird dabei automatisch in eine NodeId umgesetzt.

Sicherheit (Security). OPC-UA kennt **Security-Policies**, die Signatur und Verschlüsselung festlegen:

Policy	Bedeutung
`None`	keine Absicherung (nur in geschützten
`Basic256Sha256`	Signatur/Verschlüsselung (gängiger Sta
`Aes128_Sha256_RsaOaep`, `Aes256_Sha256_RsaPss`	moderne Policies

Dazu kommt der **MessageSecurityMode**: `None`, `Sign` (nur signiert) oder `SignAndEncrypt` (signiert und verschlüsselt). Die Partner tauschen **Zertifikate** aus und müssen sich gegenseitig vertrauen.

Alarms & Events (A&E). Ein UA-Server kann nicht nur Werte, sondern auch **Ereignisse** (Alarme, Zustandswechsel) liefern. Der UA-Client der Bridge kann diese abonnieren – gefiltert nach Quelle, Ereignistyp und Mindest-Schweregrad.

> **»Hinweis«** Weil OPC-UA selbstbeschreibend ist, ist **Browsen** hier besonders mächtig: Die Bridge kann den kompletten Adressraum des Servers durchsuchen und Punkte per Assistent massenhaft anlegen.

4.4 Modbus TCP und RTU

Herkunft und Zweck. Modbus (1979, Modicon) ist eines der ältesten und einfachsten Industrieprotokolle – und gerade deshalb enorm verbreitet: SPS, Energie- und Wärmezähler, Frequenzumrichter, Wechselrichter, Sensoren und Aktoren bieten fast immer Modbus. Seine Einfachheit macht es robust und leicht zu diagnostizieren.

Varianten.

- **Modbus RTU:** über eine serielle Leitung (RS-485 oder RS-232). Mehrere Geräte an einem Bus, jedes mit einer **Geräteadresse (Unit-ID)** 1…247. Übertragung binär mit CRC-Prüfung.
- **Modbus TCP:** über Ethernet/IP (Standard-Port **502**). Gleiche Logik, aber als TCP-Paket; die Unit-ID wird mitgeführt (bei Gateways zu seriellen Geräten relevant).

Datenmodell. Modbus kennt genau **vier Datenbereiche**:

Bereich	Kürzel	Datenbreite	Inhalt	Zugriff
Coils	`CO`	1 Bit	Schaltausgänge	lesen + schreiben
Discrete Inputs	`DI`	1 Bit	digitale Eingänge	nur lesen
Input Registers	`IR`	16 Bit	Messwerte	nur lesen
Holding Registers	`HR`	16 Bit	Soll-/Parameterwerte	lesen + schreiben

Jeder Bereich ist durchnummeriert (Adresse 0, 1, 2 …). In älterer **Modicon-Notation** werden stattdessen Bereiche durch den Zahlenbereich codiert: `0xxxx` = Coils, `1xxxx` = Discrete Inputs, `3xxxx` = Input Registers, `4xxxx` = Holding Registers (z. B. `40001` = erstes Holding Register). Die Bridge versteht beide Schreibweisen.

Nachrichtenaufbau (vereinfacht). Eine Modbus-Anfrage „Lies 2 Holding Register ab Adresse 100 von Gerät 1” besteht aus:

```
[Unit-ID=01] [Funktion=03 (Read Holding Registers)] [Startadresse=00 64] [Anzah
```

Die Antwort enthält die Anzahl Bytes und die Registerinhalte:

```
[Unit-ID=01] [Funktion=03] [Byteanzahl=04] [Reg100=41 20] [Reg101=00 00]
```

Zahlen über mehrere Register. Ein Register hat nur **16 Bit** (0…65535). Für größere Werte werden mehrere Register kombiniert:

- **32-Bit-Integer** (`int32`/`uint32`) oder **32-Bit-Gleitkomma** (`float32`): 2 Register.
- **64-Bit** (`int64`/`double`): 4 Register.
- **Text** (`string`): mehrere Register, je 2 Zeichen.

Dabei ist die **Byte-Reihenfolge (Byteorder)** entscheidend – siehe Abschnitt 4.12 und den **Register-Inspektor** (Kapitel 33), der alle Interpretationen gleichzeitig zeigt.

Rollen in der Bridge. Die Bridge kann für TCP **und** RTU jeweils sein:

- **Master** (Client): fragt aktiv Register von einem Gerät ab und schreibt sie.
- **Slave** (Server): stellt selbst Register bereit, die andere Master abfragen können. So kann die Bridge z. B. Werte aus OPC-UA als Modbus-Register „nach außen" anbieten.

Adressformat in der Bridge. `[Unit:]Bereich Offset[.Bit][:Typ[:Byteorder]]`

Beispiele: `HR100:float32`, `2:IR10:int32:CDAB`, `CO5`, `HR7.3` (Bit 3 eines Registers), `40001:uint16`, `HR200:string20`.

4.5 MQTT

Herkunft und Zweck. MQTT (*Message Queuing Telemetry Transport*, 1999, ursprünglich für Pipeline-Telemetrie) ist das Standardprotokoll des **Internet of Things**. Es ist extrem schlank und für **unzuverlässige, bandbreitenschwache Netze** (Mobilfunk, Satellit) optimiert. Ideal, um viele verteilte Geräte an eine zentrale Stelle anzubinden.

Kommunikationsmuster: Publish/Subscribe. Anders als bei Modbus fragt niemand direkt ab. Stattdessen gibt es einen zentralen **Broker** (Server). Geräte **veröffentlichen** (publish) Werte unter einem **Topic**, und Interessenten **abonnieren** (subscribe) dieses Topic:

```
Sensor  --publish "halle1/temp" = 21.4-->  [ Broker ]  --deliver-->  Abonnent(en)
```

Vorteil: Sender und Empfänger kennen sich nicht und müssen nicht gleichzeitig online sein – sie sind über den Broker **entkoppelt**.

Topics. Ein Topic ist ein hierarchischer Pfad mit / als Trenner, z. B. `werk/halle1/linie3/tem`
Beim Abonnieren sind Platzhalter möglich (+ für eine Ebene, # für alle restlichen).

Payload. Der Inhalt einer Nachricht ist frei – oft eine Zahl, häufig aber **JSON**, z. B. `{"temp": 21.4, "unit": "C"}`. Die Bridge kann gezielt einen Wert aus dem

JSON herausgreifen, indem im Adressfeld ein **JSON-Pfad** mit @ angehängt wird: `sensor/abc@data.temperature`.

Quality of Service (QoS). MQTT kennt drei Zustellgarantien: QoS 0 (höchstens einmal), QoS 1 (mindestens einmal), QoS 2 (genau einmal). Höheres QoS = sicherer, aber mehr Overhead.

Retained Messages & Last Will. Eine *Retained Message* speichert der Broker, sodass neue Abonnenten sofort den letzten Wert erhalten. Das *Last Will* ist eine Nachricht, die der Broker automatisch sendet, wenn ein Gerät unerwartet die Verbindung verliert.

Adressformat in der Bridge. `halle1/temp` oder mit JSON-Pfad `sensor/abc@data.temperature`.

4.6 MQTT Sparkplug B

Das Problem, das Sparkplug löst. „Nacktes" MQTT legt **keine Struktur** fest: Jeder Hersteller benennt Topics anders und codiert Payloads unterschiedlich. In großen Anlagen führt das zu Wildwuchs. **Sparkplug B** ist eine **offene Spezifikation oberhalb von MQTT**, die Topic-Struktur, Payload-Format (Google Protocol Buffers) und einen festen **Lebenszyklus** vorschreibt. So entsteht ein **Unified Namespace (UNS)** – eine einheitliche, durchsuchbare Echtzeit-Abbildung der gesamten Anlage. Sparkplug B wird u. a. von Ignition, HiveMQ, AVEVA und vielen IIoT-Plattformen unterstützt.

Rollen. Sparkplug kennt **Edge-Nodes** (Datenquellen am Rand der Anlage) und **Host- Applications** (z. B. SCADA). Die OPC Bridge tritt als **Edge-Node** auf.

Topic-Struktur. `spBv1.0/<GroupID>/<Nachrichtentyp>/<EdgeNodeID>`, z. B. `spBv1.0/Werk1/NDATA/Linie3`.

Lebenszyklus (das Herzstück).

Nachricht	Bedeutung
NBIRTH (Node Birth)	Beim Verbinden meldet der Node **alle** seine Metriken (Messgrößen) mit Name, Datentyp und einer kurzen **Alias-Nummer** an. Enthält bdSeq (Birth/Death-Zähler).
NDATA (Node Data)	Danach werden Änderungen nur noch über die **Alias-Nummer** gesendet (ohne den langen Namen) – das spart Bandbreite erheblich.
NDEATH (Node Death)	Das als *Last Will* hinterlegte „Sterbezeugnis". Fällt der Node aus, sendet es der Broker automatisch – alle Abonnenten wissen sofort, dass die Daten veraltet sind.
NCMD (Node Command)	Befehle an den Node, u. a. *Rebirth* (erneute Geburtsmeldung anfordern) und Schreibwerte.

Jede Nachricht trägt eine fortlaufende **Sequenznummer (seq)**; Lücken zeigen Verlust an. Die Kombination aus bdSeq und seq macht den Datenstrom nachvollziehbar.

Was die Bridge tut.

- Sendet beim Verbinden **NBIRTH** mit allen Schreibpunkten als Metriken (inkl. Aliasen und `Node Control/Rebirth`).
- Sendet Änderungen als **NDATA** (nur Alias + Wert + Sequenz).
- Hinterlegt **NDEATH** als Last Will.
- Beantwortet **Rebirth**-Anforderungen und führt **NCMD**-Schreibwerte zurück in die Anlage (bidirektional).

Datentyp-Abbildung. Generische Typen werden automatisch auf Sparkplug-Typen abgebildet (Int8/16/32/64, UInt…, Float, Double, Boolean, String).

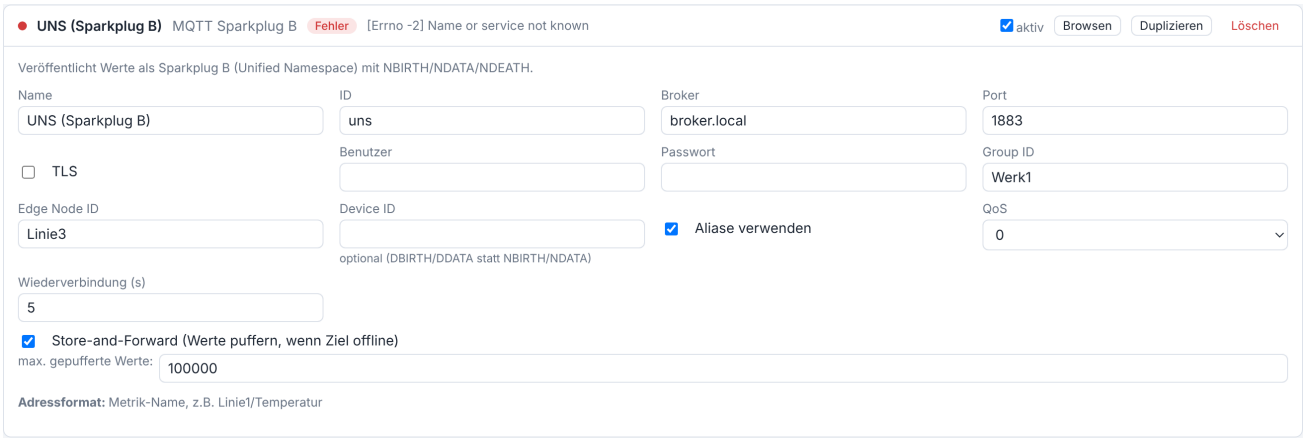


Figure 2: Sparkplug B

4.7 CAN-Bus

Herkunft und Zweck. CAN (*Controller Area Network*, Bosch, 1986) ist der dominierende Bus in Fahrzeugen, Baumaschinen, Aggregaten, Medizin- und Industriegeräten. Er ist robust, echtzeitfähig und benötigt nur zwei Drähte. Auf dem Bus laufen viele kleine Botschaften (**Frames**), die alle Teilnehmer mithören.

Frame-Aufbau. Ein CAN-Daten-Frame besteht im Kern aus:

Feld	Größe	Bedeutung
Identifier (ID)	11 Bit (Standard) oder 29 Bit (Extended)	legt die Botschaft und ihre Priorität fest (kleinere ID = höhere Priorität)
DLC	4 Bit	Anzahl der Datenbytes (0…8)
Daten	0…8 Byte	die eigentliche Nutzlast
CRC	15 Bit	Prüfsumme

Wichtig: CAN überträgt **keine Signalnamen**. In den bis zu 8 Datenbytes stecken die eigentlichen Messwerte (**Signale**) als **Bitfelder**. Welches Bit welches Signal ist, mit welcher Länge, welchem Faktor und Offset, steht entweder

- in einer **DBC-Datei** (dem Standard-Beschreibungsformat; die Bridge kann DBC importieren), oder
- muss mit dem **CAN-Analyser** der Bridge ermittelt werden (mithören, Muster erkennen).

Signaldefinition. Ein Signal wird beschrieben durch: Start-Bit, Länge (Bits), Byte-Reihenfolge (*Intel/little* oder *Motorola/big*), Vorzeichen, **Faktor** und **Offset**. Der physikalische Wert ergibt sich als `roh * Faktor + Offset`. Beispiel: Ein

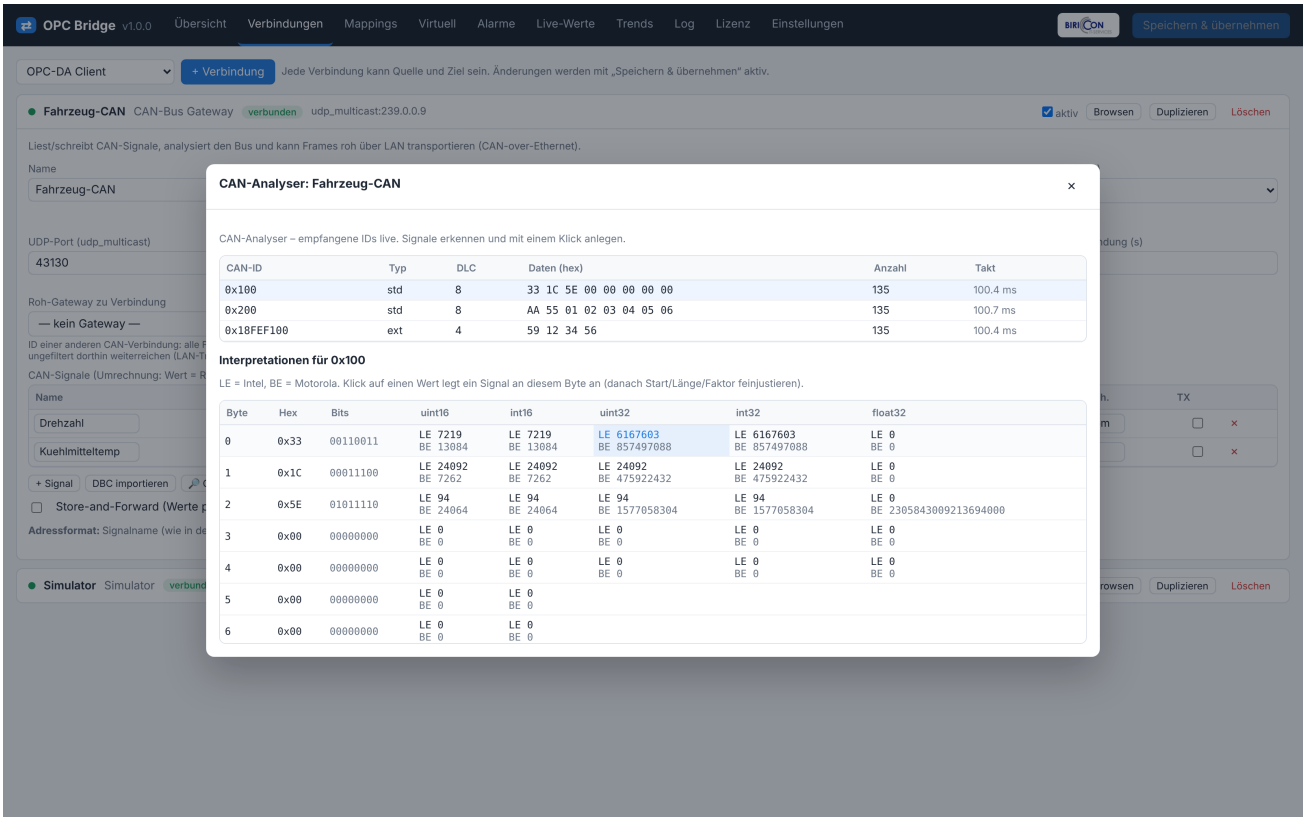


Figure 3: CAN-Analyser

- **Signal-Extraktion:** Signale aus Frames herauslösen, per Faktor/Offset umrechnen und nach OPC/Modbus/MQTT exportieren.
- **Roh-Gateway:** den kompletten Bus **ungefiltert über LAN** transportieren (z. B. zwei entfernte CAN-Segmente über Ethernet koppeln).

4.8 J1939

Zweck. SAE J1939 ist ein auf CAN aufbauender Standard für Nutzfahrzeuge, Motoren und Aggregate (Diesel-Generatoren, Baumaschinen, Landtechnik). Er legt fest, **wie** die 29-Bit-ID und die Datenbytes strukturiert sind, sodass Geräte verschiedener Hersteller zusammenarbeiten.

Aufbau der 29-Bit-ID. Die Extended-ID ist strukturiert in:

Teil	Bedeutung
Priorität	3 Bit – Buspriorität

Teil	Bedeutung
PGN (Parameter Group Number)	identifiziert die *Art* der Botschaft (z. B. „Motordrehzahl”)
SA (Source Address)	Adresse des sendenden Geräts

Die **PGN** verweist in der J1939-Spezifikation auf eine feste Liste von **SPNs** (Suspect Parameter Numbers) – den einzelnen Messgrößen in den Datenbytes. Beispiel: PGN 61444 (EEC1) enthält u. a. die Motordrehzahl.

Was die Bridge tut. Der CAN-Analyser erkennt Extended-Frames und **dekodiert PGN, SA, Priorität und Zieladresse automatisch** und zeigt sie an. So lassen sich J1939-Botschaften ohne DBC schnell identifizieren.

4.9 SQL / ODBC

Zweck. Häufig sollen Prozesswerte in eine **relationale Datenbank** geschrieben (Logging, Reporting) oder Sollwerte aus einer Datenbank gelesen werden. Unterstützt werden u. a. Microsoft SQL Server, PostgreSQL, MySQL/MariaDB und SQLite.

Aufbau. Die Bridge nutzt eine Datenbank-Abstraktionsschicht (SQLAlchemy, optional ODBC). Sie hinterlegen:

- eine **Lese-Abfrage** (SELECT), die Adresse und Wert liefert, und/oder
- eine **Schreibanweisung** (INSERT/UPDATE) mit **Platzhaltern** `:addr`, `:value`, `:ts`.

So wird jeder weitergeleitete Wert als Datensatz abgelegt bzw. gelesen. Zeitintervalle und Verbindungsparameter (Connection-String) sind konfigurierbar.

4.10 REST / Webhook

Zweck. Um Werte an beliebige Web-Dienste, Low-Code-Plattformen, Cloud-Endpunkte oder eigene Microservices zu senden.

Aufbau. Für jeden weiterzuleitenden Wert sendet die Bridge eine **HTTP-POST-Anfrage** an eine konfigurierte URL. Der Nachrichtenkörper ist ein frei definierbares

JSON-Template mit Platzhaltern:

Platzhalter	Inhalt
`{addr}`	Adresse des Punkts
`{value}`	Wert
`{quality}`	Qualität
`{ts}`	Zeitstempel (ISO-8601)
`{conn}`	Verbindungs-ID
`{name}`	Verbindungsname

Beispiel-Template: `{"tag":"{addr}","v":{value},"q":"{quality}","t":"{ts}"}`.

In Kombination mit **Store-and-Forward** gehen bei vorübergehend nicht erreichbarem Endpunkt keine Werte verloren.

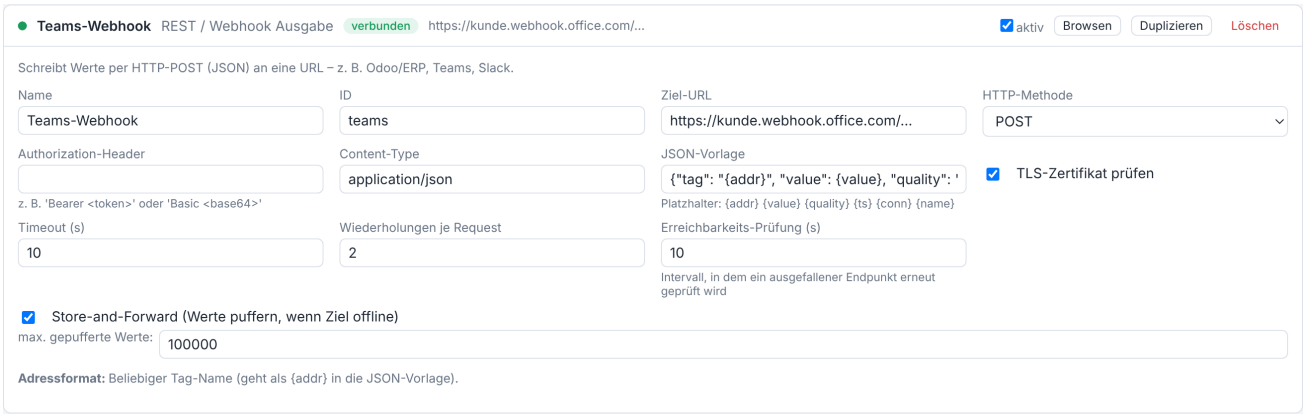


Figure 4: Webhook-Ausgang mit Store-and-Forward

4.11 Bridge-Link (OT→IT-Tunnel)

Das Sicherheitsproblem. In einer sauber segmentierten Anlage (Purdue-Modell) darf aus dem **Produktionsnetz (OT)** keine Verbindung von außen **hereinkommen**. Firewalls lassen nur **ausgehende** Verbindungen zu. Trotzdem sollen Prozesswerte in die **IT-/DMZ-Zone** gelangen.

Die Lösung. Zwei OPC-Bridge-Instanzen werden über den **Bridge-Link** gekoppelt:

- Die **OT-Seite** läuft als **Client** und baut die Verbindung **nur nach außen** auf – eine einzige, **TLS-verschlüsselte** Verbindung zur DMZ.
- Die **IT-/DMZ-Seite** läuft als **Server** und nimmt diese Verbindung an.
- Werte mit **gleichem Tag-Namen** werden **in beide Richtungen** gespiegelt.

In der OT-Firewall muss **kein eingehender Port** geöffnet werden. Zusätzlich sichert ein **gemeinsames Token** die Kopplung ab; **Store-and-Forward** auf der OT-Seite puffert bei Verbindungsabriss.

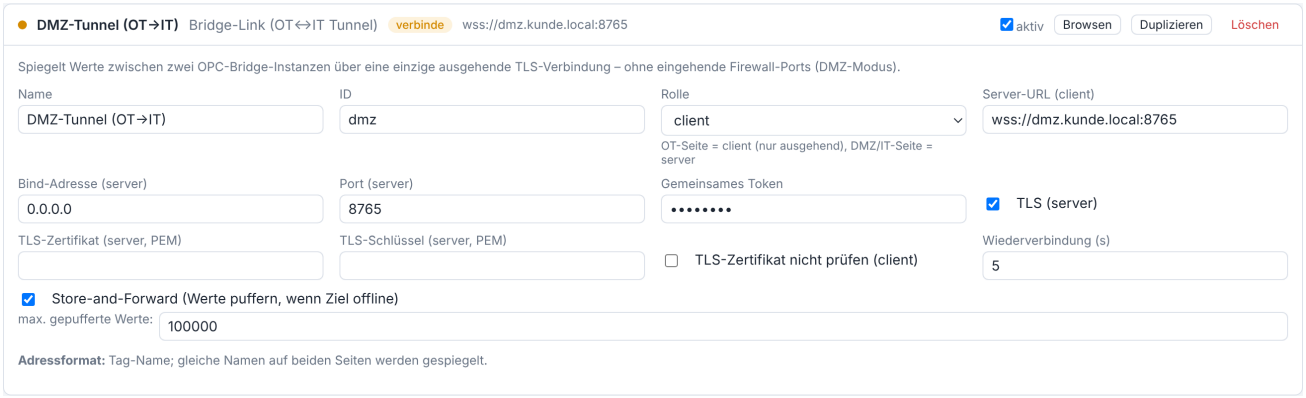


Figure 5: Bridge-Link / DMZ-Modus

4.12 Datentypen und Byte-Reihenfolge

Dies ist das **wichtigste Grundlagenthema** für die Praxis, weil falsche Datentypen oder Byte-Reihenfolgen die häufigste Ursache für „unsinnige Werte" sind.

Datentypen. Ein Wert kann verschiedene Typen haben:

Typ	Beschreibung	Wertebereich
`bool`	Wahrheitswert	true/false
`int16` / `uint16`	16-Bit ganzzahlig (mit/ohne Vorzeichen)	−32768…32767 / 0…65535
`int32` / `uint32`	32-Bit ganzzahlig	±2 Mrd. / 0…4 Mrd.
`int64`	64-Bit ganzzahlig	sehr groß
`float32`	32-Bit Gleitkomma (IEEE 754)	ca. $\pm 3{,}4 \cdot 10^{38}$
`double` (float64)	64-Bit Gleitkomma	sehr hohe Präzision
`string`	Text	je nach Länge

Warum Byte-Reihenfolge wichtig ist. Eine 32-Bit-Zahl besteht aus 4 Bytes (A, B, C, D). Verschiedene Geräte legen diese Bytes in unterschiedlicher Reihenfolge ab. Bei Modbus (zwei 16-Bit-Register) sind vier Anordnungen üblich:

Byteorder	Reihenfolge	Auch genannt
`ABCD`	Big-Endian	„Motorola”
`DCBA`	Little-Endian	„Intel”
`BADC`	Big-Endian, byte-vertauscht	Word-Swap
`CDAB`	Little-Endian, word-vertauscht	Byte-Swap (sehr häufig!)

Worked Example. Der Gleitkommawert **45,0** ist in IEEE-754 die vier Bytes `42 34 00 00`. Je nach Geräte-Byteorder erscheint er in den zwei Modbus-Registern als:

- `ABCD`: `Reg0=4234`, `Reg1=0000`
- `CDAB`: `Reg0=0000`, `Reg1=4234`
- `BADC`: `Reg0=3442`, `Reg1=0000`
- `DCBA`: `Reg0=0000`, `Reg1=3442`

Liest man mit der falschen Reihenfolge, kommt eine völlig andere Zahl heraus. Genau dafür gibt es den **Register-Inspektor** (Kapitel 33): Er zeigt alle Interpretationen gleichzeitig, sodass Sie sofort die richtige erkennen.

> **»Achtung«** Wenn ein Modbus-Wert „verrückt spielt” (z. B. extrem große Zahlen oder Null), ist fast immer die Byteorder oder der Datentyp falsch. Prüfen Sie es mit dem Register-Inspektor.

Teil II – Installation und Inbetriebnahme

5. Systemvoraussetzungen

Komponente	Minimum	Empfehlung
CPU	1 Kern, x86-64 oder ARM (z. B. Raspberry Pi 4)	2+ Kerne
RAM	512 MB	1–2 GB
Speicher	500 MB + Platz für Historie	SSD, je nach Historienumfang
Betriebssystem	Windows 10/11, Windows Server 2016+, Debian 11+, Ubuntu 20.04+, Raspberry Pi OS	aktuelle LTS-Version
Netzwerk	Zugang zu den anzubindenden Geräten	getrennte OT-/IT-Segmente

Für **OPC-DA** ist zwingend **Windows** erforderlich (COM/DCOM). Für **CAN über SocketCAN** ist **Linux** erforderlich. Alle übrigen Protokolle laufen plattformübergreifend.

Die ausgelieferten Pakete sind **kompiliert und obfuskiert** und benötigen **kein installiertes Python** auf dem Zielsystem.

---

6. Installation unter Windows

Schritt für Schritt:

1. Melden Sie sich als **Administrator** an.
2. Führen Sie **`OPCBridge-Setup-<version>.exe`** aus.
3. Folgen Sie dem Assistenten. Das Setup
   - installiert das Programm (Standard: `C:\Program Files\OPCBridge\`),
   - richtet den **Windows-Dienst „OPCBridge”** ein und startet ihn,
   - legt das Datenverzeichnis unter `%ProgramData%\OPCBridge\` an,
   - zeigt am Ende das **Erststart-Passwort** an.
4. Öffnen Sie den Browser: `http://localhost:8080/` (oder vom Nachbarrechner `http://<rechnername>:8080/`).
5. Melden Sie sich mit Benutzer **`admin`** und dem angezeigten Passwort an. Das Passwort steht auch in `%ProgramData%\OPCBridge\initial_admin_password.txt`.
6. Ändern Sie das Passwort unter *Einstellungen → Benutzer*.

Dienst verwalten. Über die Windows-Dienste-Verwaltung (`services.msc`) können Sie den Dienst „OPCBridge” starten, stoppen und den Autostart einstellen.

> **»Hinweis zu OPC-DA«** Installieren Sie – falls benötigt – die passenden *OPC Core Components*. Die Bitbreite muss übereinstimmen. Betreiben Sie Bridge und DA-Server möglichst auf demselben Rechner, um DCOM zu vermeiden.

Firewall. Geben Sie den Web-Port (Standard 8080) frei, wenn Sie von anderen Rechnern zugreifen. Für OPC-UA-Server (4840), Modbus-Slave (502) oder Bridge-Link (8765) die jeweiligen Ports freigeben, falls aktiviert.

---

7. Installation unter Debian/Ubuntu/Raspberry Pi

Installation:

```
sudo apt install ./opcbridge_1.0.0_amd64.deb
```

Das Paket

- installiert das Programm,
- legt den Dienstbenutzer `opcbridge` an,
- richtet den **systemd-Dienst `opcbridge`** ein und startet ihn,
- legt das Datenverzeichnis `/var/lib/opcbridge/` an.

Zugriff. Browser öffnen: `http://<rechner>:8080/`. Das Erststart-Passwort steht in `/var/lib/opcbridge/initial_admin_password.txt`.

Dienst verwalten:

```
systemctl status opcbridge      # Status
systemctl restart opcbridge     # Neustart
systemctl stop opcbridge        # Stoppen
journalctl -u opcbridge -f      # Log live mitlesen
```

Für CAN (SocketCAN): Das CAN-Interface muss auf dem Host eingerichtet sein, z. B.:

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

Deinstallation: `sudo apt remove opcbridge` (Daten unter `/var/lib/opcbridge/` bleiben erhalten; mit `purge` werden sie entfernt).

---

8. Installation als Docker-Container

Voraussetzung: Docker und Docker Compose sind installiert.

Start:

```
docker compose up -d
```

Die mitgelieferte `docker-compose.yml` gibt die Ports frei (Web 8080, OPC-UA 4840, Modbus-Slave 502, Bridge-Link 8765) und legt das persistente Volume `./data` an. Dort liegen `config.json`, `license.key` und die Historie.

Nützliche Befehle:

```
docker compose logs -f          # Log mitlesen
docker compose restart          # Neustart
docker compose down             # Stoppen und entfernen (Daten im
↪  Volume bleiben)
```

CAN im Container. Für SocketCAN den Container mit Host-Netz und Capability starten (in der Compose-Datei vorbereitet): `network_mode: host` und `cap_add: [NET_ADMIN]`.

Eigenes Image bauen (optional): `docker compose build` erzeugt das Image aus dem mitgelieferten `Dockerfile`.

---

9. Erste Inbetriebnahme Schritt für Schritt

Diese Kurzanleitung bringt Sie in wenigen Minuten zum ersten funktionierenden Datenfluss.

1. **Anmelden.** Weboberfläche öffnen, mit `admin` und Erststart-Passwort einloggen.
2. **Passwort ändern.** *Einstellungen* → *Benutzer*, neues Passwort eintragen, **Speichern & übernehmen**.
3. **Erste Quelle anlegen.** Reiter *Verbindungen* → Typ wählen (z. B. *Simulator* zum Testen oder *Modbus TCP*) → **Hinzufügen** → Felder ausfüllen.
4. **Erstes Ziel anlegen.** Zweite Verbindung anlegen (z. B. *OPC-UA-Server*).
5. **Mapping anlegen.** Reiter *Mappings* → Quelle (A) und Ziel (B) samt Adresse wählen → Richtung *A* → *B*.
6. **Speichern & übernehmen.** Oben rechts. Die Engine startet kurz neu.
7. **Kontrolle.** Reiter *Live-Werte*: Kommt der Wert an? Reiter *Übersicht*: Sind die Verbindungen grün?

   »Merksatz« *Nichts ist aktiv, bis Sie oben rechts „Speichern & übernehmen" geklickt haben.* Alle Änderungen in Verbindungen, Mappings und Einstellungen werden erst mit diesem Klick wirksam (die Engine wird dabei kurz neu gestartet).

Teil III – Bedienung

10. Aufbau der Weboberfläche

Die Weboberfläche gliedert sich in **Reiter** am oberen Rand. Oben rechts befindet sich der Knopf **„Speichern & übernehmen”** sowie – bei ungespeicherten Änderungen – ein Hinweis und ein Knopf zum Verwerfen.

Reiter	Zweck	Kapitel
Übersicht	Statusüberblick aller Verbindungen und Mappings	11
Verbindungen	Geräte/Systeme anlegen und konfigurieren	15–25
Mappings	Werte einander zuordnen	26
Virtuell	Berechnete Punkte und Redundanz	28
Alarme	Benachrichtigungsregeln, OPC-UA-Events	31
Live-Werte	Aktuelle Werte, manuelles Schreiben	12
Trends	Historische Werte grafisch	13
Log	Betriebsmeldungen	14
Lizenz	Lizenzstatus und Aktivierung	39
Einstellungen	Webinterface, Benutzer, Historie, WebView, SMTP	30, 35

Das **Firmenlogo** oben rechts verweist auf www.biricon.eu; über das Impressum sind die Kontaktdaten erreichbar.

11. Reiter Übersicht

Die Übersicht zeigt auf einen Blick den Gesundheitszustand der Bridge:

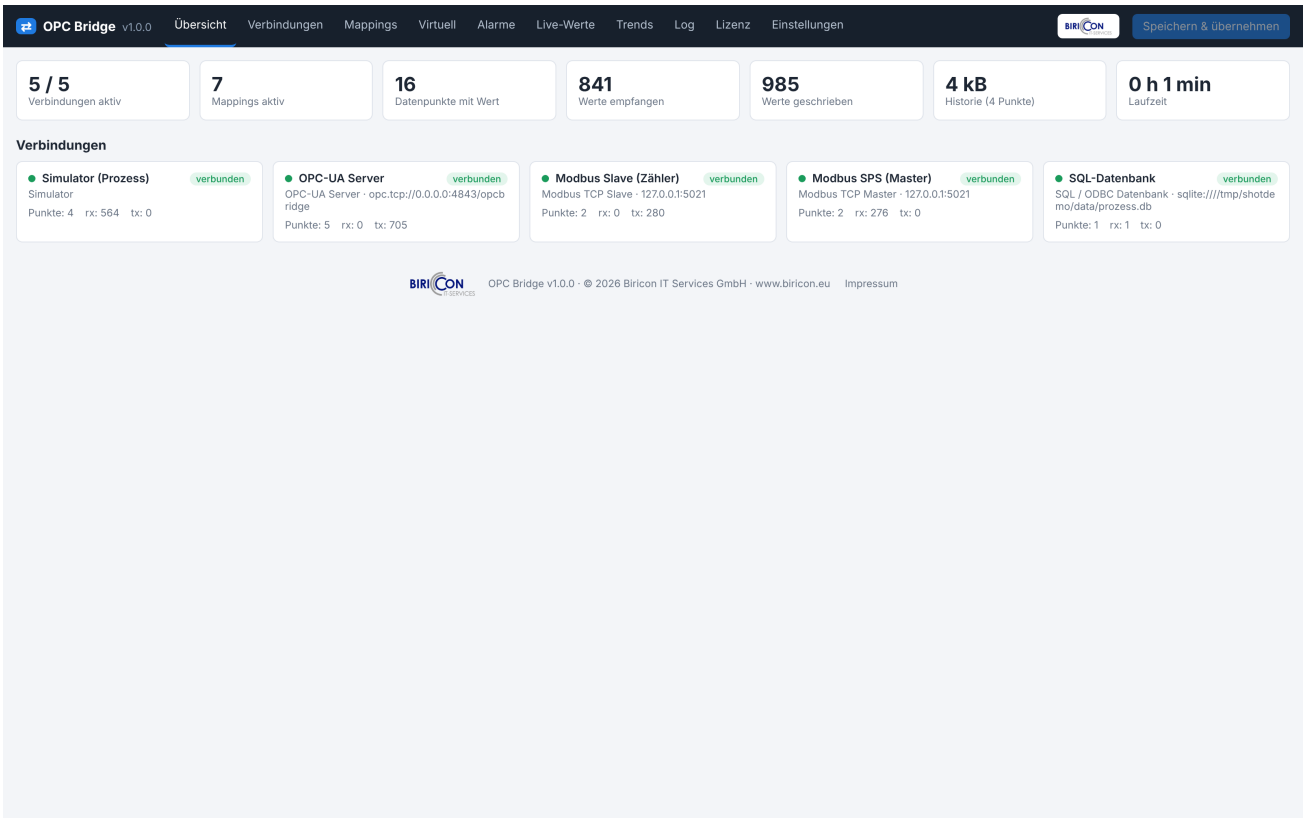


Figure 6: Übersicht

- **Verbindungen:** Anzahl und Zustand (verbunden/verbindet/Fehler/gestoppt). Jede Verbindung wird mit Farbe und Statusmeldung dargestellt.
- **Mappings:** Anzahl aktiver Zuordnungen.
- **Store-and-Forward:** gepufferte Werte je Zielverbindung (falls aktiv).
- **Lizenz:** aktueller Lizenzstatus (lizenziert/Demo/Restzeit).

Die Werte werden **in Echtzeit** aktualisiert (siehe Kapitel 36).

---

12. Reiter Live-Werte

Hier sehen Sie die **aktuellen Werte** aller überwachten Punkte in Echtzeit – mit Wert, Qualität und Zeitstempel.

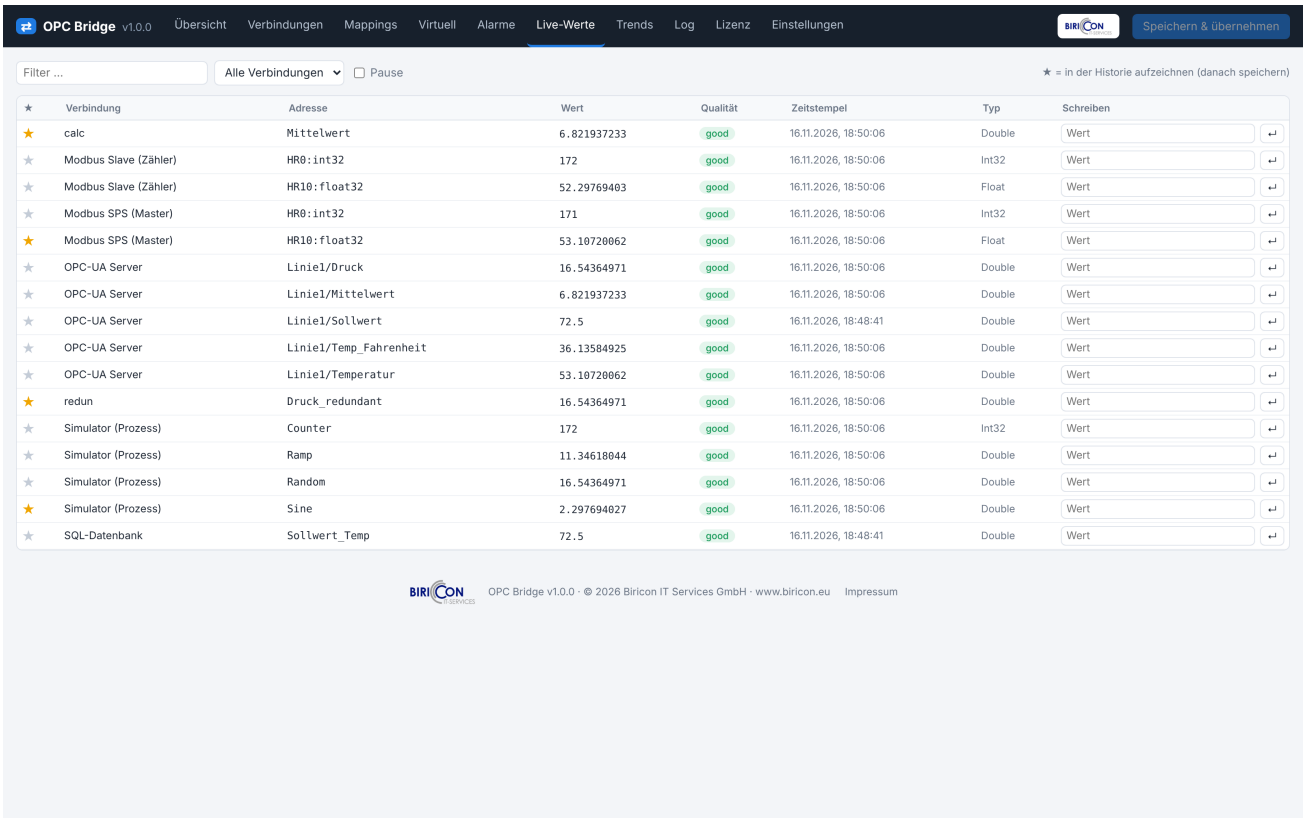


Figure 7: Live-Werte

Funktionen:

- **Pause:** friert die Anzeige ein (die Datenübertragung läuft im Hintergrund weiter).
- **Manuelles Schreiben:** Bei schreibbaren Zielpunkten können Sie zu Testzwecken einen Wert direkt eingeben und senden. Das ist bei der Inbetriebnahme sehr nützlich, um Aktoren oder Register gezielt zu prüfen.
- **Filter/Suche:** Punkte nach Name oder Verbindung eingrenzen.

  »Tipp« Nutzen Sie die Live-Werte zusammen mit dem manuellen Schreiben, um eine neue Mapping-Kette zu testen, bevor Sie die echte Quelle anschließen.

13. Reiter Trends

Der Trend zeigt **historische Werte grafisch** über die Zeit.

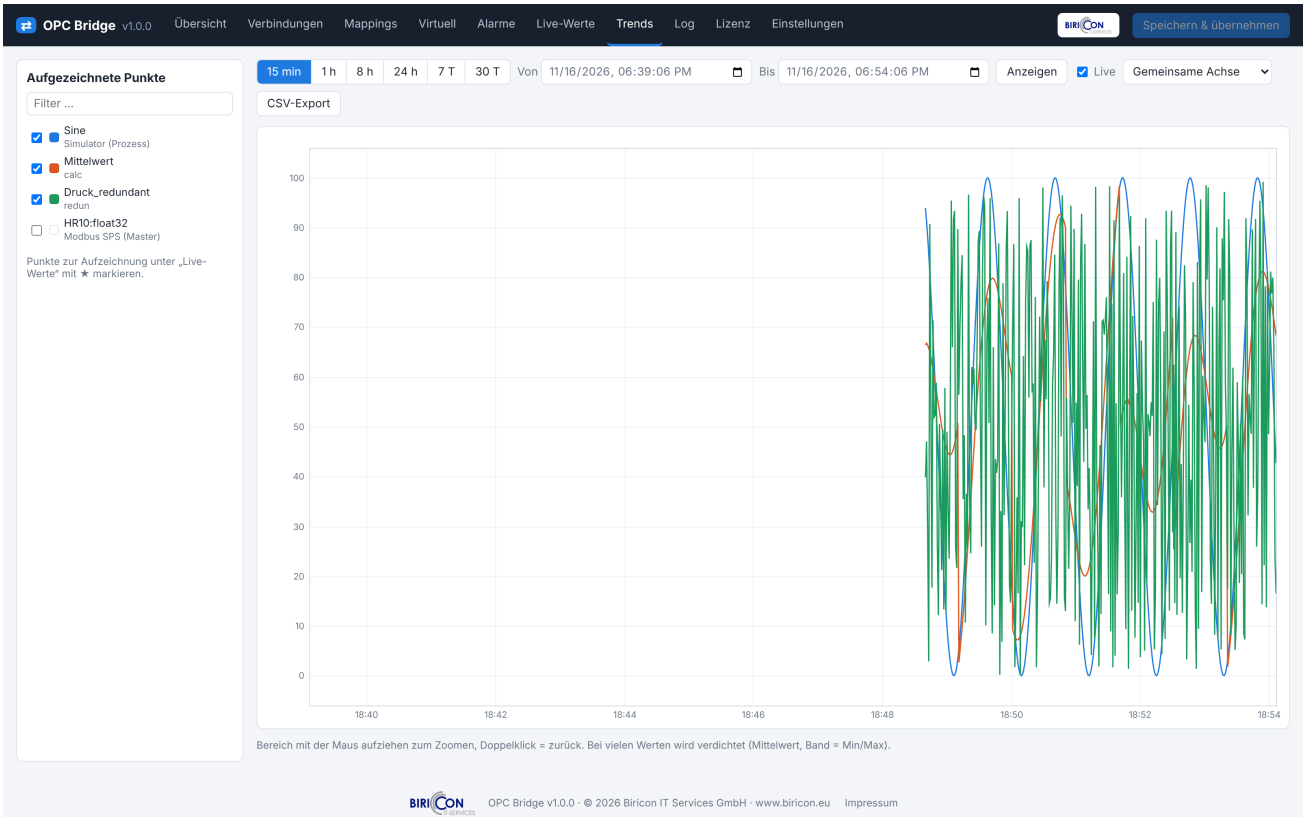


Figure 8: Trends

- **Punkte wählen:** Welche aufgezeichneten Punkte dargestellt werden.
- **Zeitbereich:** z. B. letzte Stunde, letzter Tag, frei wählbar.
- **CSV-Export:** Die dargestellten Daten als CSV-Datei herunterladen (z. B. für Excel).

Voraussetzung ist, dass die **Historie** aktiviert und die Punkte zur Aufzeichnung ausgewählt sind (Kapitel 30).

---

14. Reiter Log

Das Log zeigt **Betriebsmeldungen** der Bridge: Verbindungsauf- und -abbau, Fehler, Warnungen. Es ist die **erste Anlaufstelle bei Problemen**.

- Meldungen sind nach Zeit sortiert und nach Schweregrad gekennzeichnet.
- Bei Verbindungsfehlern steht hier die konkrete Ursache (z. B. „Zeitüberschreitung", „Verbindung abgelehnt", „Adresse unbekannt").

Auf Systemebene stehen die Meldungen zusätzlich im Windows-Ereignisprotokoll bzw. unter Linux in `journalctl -u opcbridge`.

Teil IV – Konfiguration

15. Verbindungen – allgemein

Im Reiter **Verbindungen** legen Sie jede Anbindung an ein Gerät oder System an.

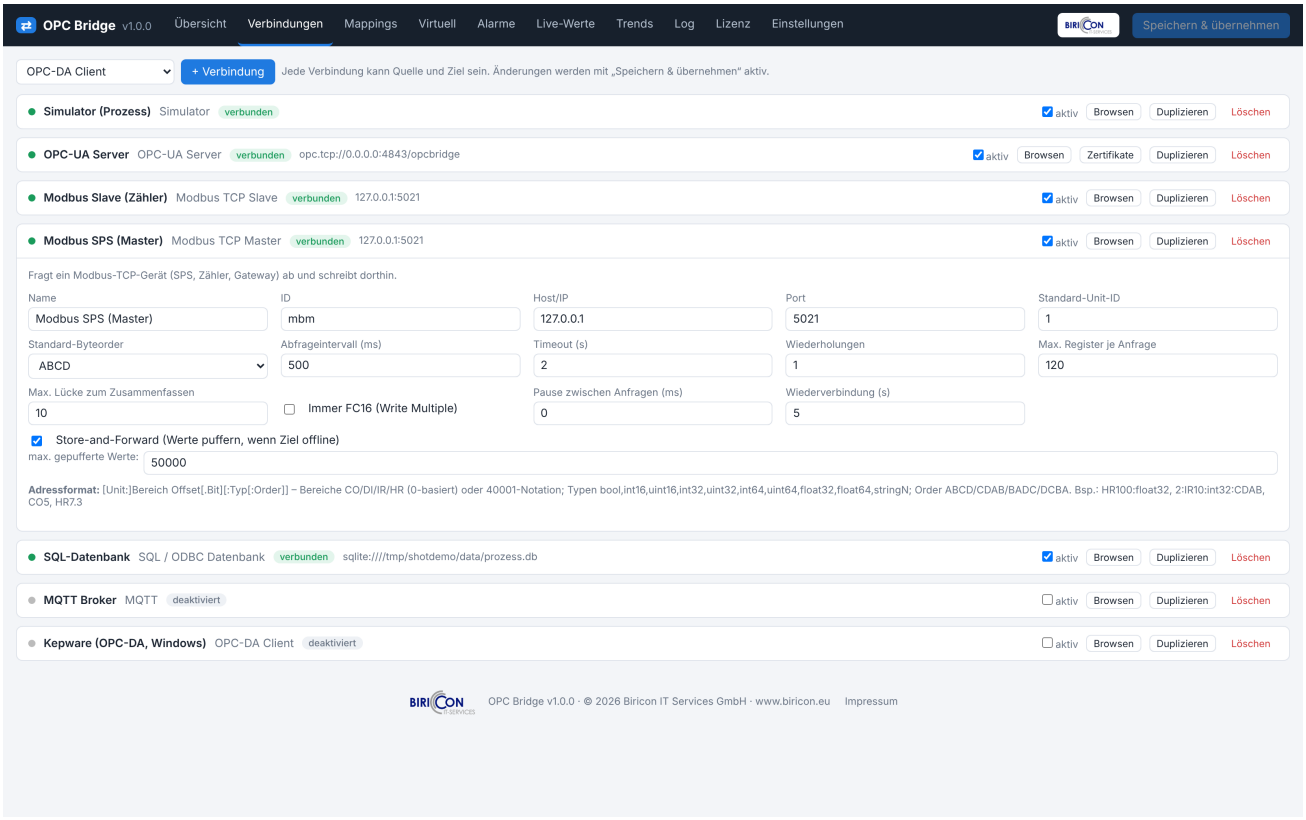


Figure 9: Verbindungen

Neue Verbindung anlegen:

1. Oben den **Typ** (das Protokoll) aus der Liste wählen.
2. Auf **Hinzufügen** klicken – eine neue Karte erscheint.
3. Die Karte durch Klick auf die Kopfzeile **aufklappen**.
4. Die Felder ausfüllen (jeder Typ zeigt genau die Felder, die er braucht – siehe folgende Kapitel).
5. **Speichern & übernehmen**.

Bedienelemente jeder Karte:

Element	Funktion
Name	frei wählbare Bezeichnung (erscheint überall in der Oberfläche)

Element	Funktion
ID	technische Kennung (automatisch, in Adressen/Mappings referenziert)
aktiv	schaltet die Verbindung ein/aus
Browsen	durchsucht das Gerät nach verfügbaren Punkten (wenn das Protokoll es erlaubt)
Duplizieren	erstellt eine Kopie der Karte
Löschen	entfernt die Verbindung
Store-and-Forward	Pufferung aktivieren (Kapitel 29)

In den folgenden Kapiteln sind die **Felder je Verbindungstyp** vollständig beschrieben. Die Spalte *Standard* nennt den Vorgabewert.

---

16. Verbindungstyp OPC-DA

Rolle: Client (liest/schreibt Items eines OPC-DA-Servers). **Nur unter Windows.**

Feld	Bedeutung	Standard
Server ProgID	Kennung des DA-Servers, z. B. `Matrikon.OPC.Simulation`	–
Rechner	Hostname/IP bei entferntem Server (leer = lokal, empfohlen)	leer
Abtastrate (ms)	wie oft Items gelesen werden	1000
Lesequelle	`cache` (Server-Zwischenspeicher, schnell) oder `device` (direkt vom Gerät)	cache
Automation-Wrapper	Pfad/ProgID des OPC-Automation-Wrappers (falls nötig)	leer
Wiederverbindung (s)	Wartezeit vor erneutem Verbindungsversuch	10
Datentyp beim Browsen abfragen	ermittelt beim Browsen auch den Typ	ein

Adressformat: `Channel1.Device1.Tag1` (vom Server vorgegeben; per Browsen ermittelbar).

> **»Achtung«** Entfernter Zugriff erfordert DCOM-Konfiguration. Empfehlung: Bridge lokal auf dem DA-Server-Rechner betreiben.

---

17. Verbindungstyp OPC-UA-Client

Rolle: Client (liest/schreibt Knoten eines fremden UA-Servers, optional A&E).

Feld	Bedeutung	Standard
Endpoint-URL	Adresse des UA-Servers, z. B. `opc.tcp://192.168.0.10:4840`	`opc.tcp://localhost:4840`
Security	Security-Policy und -Modus (siehe 4.3)	None
Benutzer / Passwort	Anmeldung am Server (falls gefordert)	leer
Publishing-Intervall (ms)	wie oft der Server Änderungen meldet	500
Sampling-Intervall (ms)	wie oft der Server intern abtastet	250
Timeout (s)	Zeitüberschreitung	10
Wiederverbindung (s)	Wartezeit vor erneutem Versuch	10
Alarme & Ereignisse (A&E) abonnieren	Ereignisse vom Server empfangen	aus
Ereignisquelle (NodeId)	Quelle der Ereignisse (Standard: Server-Objekt `i=2253`)	`i=2253`
Ereignistyp (NodeId)	zu abonnierender Ereignistyp (`i=2041` = BaseEventType)	`i=2041`
Mindest-Schweregrad (1–1000)	nur Ereignisse ab dieser Schwere	1

Adressformat: `ns=2;s=Linie1.Temp` oder `ns=3;i=1001`. Per Browsen durchsuchbar.

18. Verbindungstyp OPC-UA-Server

Rolle: Server (stellt eigene Werte bereit, die andere Systeme lesen/schreiben).

Feld	Bedeutung	Standard
Bind-Adresse	Netzwerkschnittstelle (`0.0.0.0` = alle)	0.0.0.0
Port	UA-Port	4840
Endpoint-Pfad	Pfadanteil der Endpoint-URL	`/opcbridge`
Servername	angezeigter Name	OPC Bridge
Namespace-URI	Kennung des eigenen Namensraums	`urn:opcbridge`
Wurzelordner	optionaler Oberordner im Adressraum	leer
Security-Policies	erlaubte Policies (Mehrfachauswahl)	None, Basic256Sha256 SignAndEncrypt
Client-Zertifikate	`trust_all` (alle akzeptieren) oder `truststore` (nur hinterlegte)	trust_all
Anonymer Zugriff	`off` / `read` / `readwrite`	read
Benutzer	Benutzerliste mit Rechten	–

Adressierung: Der Pfad `Anlage/Linie1/Temperatur`, den Sie am Zielpunkt angeben, wird automatisch zu einer NodeId `ns=<idx>;s=Anlage/Linie1/Temperatur`.

> **»Achtung«** `trust_all` und `anonymous=readwrite` sind nur in geschützten Netzen vertretbar. Für Produktivbetrieb Zertifikats-Truststore und Benutzer verwenden.

19. Verbindungstyp Modbus

Es gibt vier Modbus-Verbindungstypen: **TCP Master**, **RTU Master**, **TCP Slave**, **RTU Slave**.

19.1 Modbus TCP Master

Feld	Bedeutung	Standard
Host/IP	Adresse des Modbus-Geräts	192.168.0.10
Port	TCP-Port	502
Standard-Unit-ID	Geräteadresse (bei Gateways relevant)	1
Standard-Byteorder	Vorgabe für mehrteilige Werte	ABCD
Abfrageintervall (ms)	wie oft gelesen wird	1000
Timeout (s)	Zeitüberschreitung	2
Wiederholungen	erneute Versuche bei Fehler	1
Max. Register je Anfrage	begrenzt die Blockgröße	120
Max. Lücke zum Zusammenfassen	fasst nahe Register zu einer Anfrage zusammen	10
Immer FC16 (Write Multiple)	auch Einzelwerte mit Funktion 16 schreiben	aus
Pause zwischen Anfragen (ms)	entlastet langsame Geräte	0
Wiederverbindung (s)	Wartezeit vor erneutem Versuch	5

19.2 Modbus RTU Master

Wie TCP Master, aber über die serielle Schnittstelle. Zusätzliche/abweichende Felder:

Feld	Bedeutung	Standard
Serielle Schnittstelle	z. B. `COM1` (Windows) oder `/dev/ttyUSB0` (Linux)	COM1
Baudrate	1200…115200	19200
Parität	N/E/O (keine/gerade/ungerade)	E
Datenbits	7 oder 8	8
Stopbits	1 oder 2	1

19.3 Modbus TCP Slave

Feld	Bedeutung	Standard
Bind-Adresse	Netzwerkschnittstelle	0.0.0.0
Port	TCP-Port	502
Standard-Unit-ID	eigene Geräteadresse	1
Erlaubte Unit-IDs	zusätzliche akzeptierte IDs (z. B. `1,2,5`)	leer
Standard-Byteorder	Vorgabe für mehrteilige Werte	ABCD

19.4 Modbus RTU Slave

Wie TCP Slave, aber seriell (Felder *Serielle Schnittstelle*, *Baudrate*, *Parität*, *Datenbits*, *Stopbits* wie beim RTU Master).

Adressformat (alle Modbus-Typen): `[Unit:]Bereich Offset[.Bit][:Typ[:Byteorder]]` z. B. `HR100:float32:CDAB`, `2:IR10:int32`, `CO5`, `HR7.3`, `40001:uint16`, `HR200:string20`.

---

20. Verbindungstyp MQTT

Rolle: Publish/Subscribe an einem MQTT-Broker.

Feld	Bedeutung	Standard
Broker	Hostname/IP des Brokers	localhost
Port	1883 (unverschlüsselt) / 8883 (TLS)	1883
TLS	Verschlüsselung aktivieren	aus
TLS-Zertifikat nicht prüfen	für selbstsignierte Zertifikate	aus
CA-Datei	Pfad zur CA für Zertifikatsprüfung	leer
Benutzer / Passwort	Broker-Anmeldung	leer
Client-ID	eindeutige Kennung (leer = automatisch)	leer
QoS	Zustellgarantie 0/1/2	0
Retain beim Publizieren	letzten Wert im Broker halten	ein
Payload-Format (Publish)	`raw` (nur Wert) oder `json`	raw
Topic-Präfix	wird allen Topics vorangestellt	leer
Filter zum Browsen	Topic-Filter beim Durchsuchen (# = alles)	#
Wiederverbindung (s)	Wartezeit vor erneutem Versuch	5

Adressformat: `halle1/temp`; JSON-Wert herausgreifen mit `sensor/abc@data.temperature`

21. Verbindungstyp Sparkplug B

Rolle: Sparkplug-B-Edge-Node (UNS) an einem MQTT-Broker.

Feld	Bedeutung	Standard
Broker / Port	MQTT-Broker (oft TLS auf 8883)	localhost / 1883
TLS / TLS-Zertifikat nicht prüfen	Verschlüsselung	aus
Benutzer / Passwort	Broker-Anmeldung	leer
Group ID	oberste Ebene im Sparkplug-Topic	Biricon
Edge Node ID	Kennung dieses Edge-Nodes	Bridge1
Device ID	optionale Geräteebene unter dem Node	leer
Aliase verwenden	Alias-Komprimierung (empfohlen)	ein
QoS	0 oder 1	0
Wiederverbindung (s)	Wartezeit vor erneutem Versuch	5

Adressformat: Metrikname, z. B. `Temperatur` oder `Linie3/Stueckzahl`.

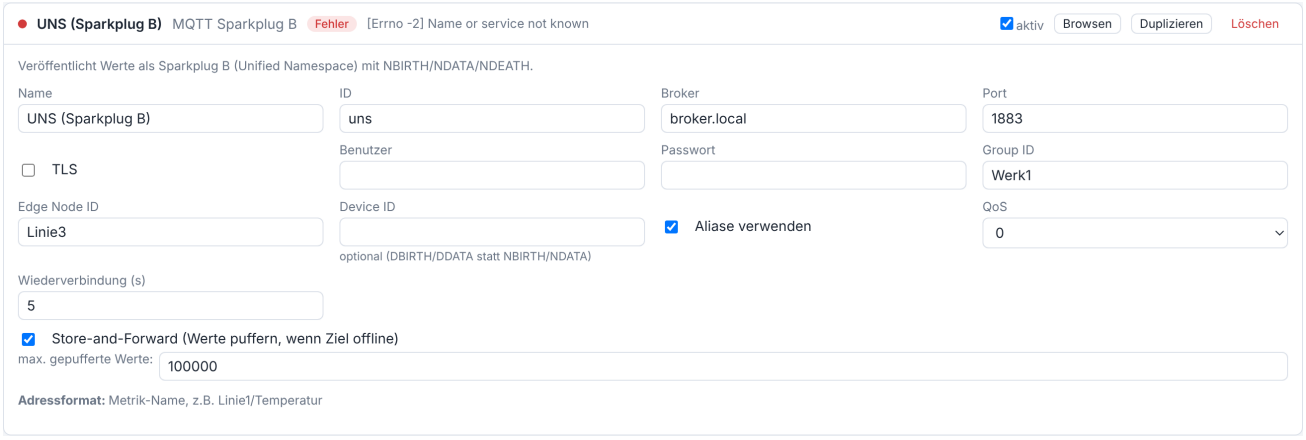


Figure 10: Sparkplug B

22. Verbindungstyp CAN

Rolle: CAN-Gateway (Signal-Extraktion und/oder Roh-Transport).

Feld	Bedeutung	Standard
Schnittstelle	socketcan / udp_multicast / slcan / pcan / kvaser / vector / seeedstudio / usb2can / virtual	socketcan
Kanal	z. B. `can0` (SocketCAN) oder Adapterkanal	can0
Bitrate (bit/s)	Busgeschwindigkeit (muss zum Bus passen!)	500000
UDP-Port (udp_multicast)	Port bei CAN-over-Ethernet	43113
CAN-FD	erweiterte Frames mit mehr Daten	aus
Eigene Sendungen empfangen	gesendete Frames mitlesen	aus
Zyklisches Senden (ms)	periodisch senden (0 = aus)	0
Wiederverbindung (s)	Wartezeit vor erneutem Versuch	5
Roh-Gateway zu Verbindung	ID einer zweiten CAN-Verbindung für Roh-Transport	leer
Gateway beidseitig	Frames in beide Richtungen weiterleiten	ein
Signale	Liste der extrahierten Signale (Bitlage, Faktor, Offset …)	–

Adressformat: Signalname aus der Signalliste (per Analyser/DBC angelegt). Siehe Kapitel 32.

23. Verbindungstyp SQL

Rolle: Werte in eine Datenbank schreiben und/oder daraus lesen.

Feld	Bedeutung	Standard
Verbindungs-URL (SQLAlchemy)	z. B. `postgresql://user:pw@host/db`, `mssql+pyodbc://…`, `sqlite:///pfad.db`	`sqlite:///opcbridge_data.db`
Abfrageintervall (ms)	wie oft gelesen wird	2000
Lesemodus	`keyvalue` (Zeilen mit addr/value), `columns` (Spalten als Punkte), `off`	keyvalue
Lese-Abfrage (SELECT)	SQL zum Lesen	`SELECT addr, value FROM tags`
Schreib-Anweisung	SQL mit `:addr`, `:value`, `:ts`	`INSERT INTO tags(addr, value, ts) VALUES (:addr, :value, :ts)`
Nur bei Wertänderung schreiben	spart Schreibzugriffe	ein
Verbindung erneuern nach (s)	Pool-Recycling	1800
Wiederverbindung (s)	Wartezeit vor erneutem Versuch	5

24. Verbindungstyp Webhook

Rolle: Werte per HTTP an eine URL senden.

Feld	Bedeutung	Standard
Ziel-URL	Endpunkt	`https://erp.example.com/api/op`
HTTP-Methode	POST / PUT / PATCH	POST
Authorization-Header	z. B. `Bearer <token>`	leer
Content-Type	MIME-Typ des Körpers	application/json
JSON-Vorlage	Template mit `{addr}` `{value}` `{quality}` `{ts}` `{conn}` `{name}`	siehe 4.10
TLS-Zertifikat prüfen	Zertifikatsprüfung	ein
Timeout (s)	Zeitüberschreitung	10
Wiederholungen je Request	erneute Versuche	2
Erreichbarkeits-Prüfung (s)	Intervall des Health-Checks	10

Adressformat: frei gewählter Tag-Name (erscheint als `{addr}`).

---

25. Verbindungstyp Bridge-Link

Rolle: Werte-Spiegelung zwischen zwei Bridge-Instanzen über eine ausgehende TLS-Verbindung.

Feld	Bedeutung	Standard
Rolle	`client` (OT-Seite, nur ausgehend) oder `server` (DMZ/IT-Seite)	client
Server-URL (client)	Ziel des Clients, z. B. `wss://dmz-host:8765`	`wss://dmz-host:8765`
Bind-Adresse (server)	Netzwerkschnittstelle des Servers	0.0.0.0
Port (server)	Server-Port	8765
Gemeinsames Token	Geheimnis für beide Seiten (muss übereinstimmen)	leer
TLS (server)	Verschlüsselung am Server	ein
TLS-Zertifikat/-Schlüssel (server, PEM)	Zertifikat des Servers	leer
TLS-Zertifikat nicht prüfen (client)	für selbstsignierte Serverzertifikate	aus
Wiederverbindung (s)	Wartezeit vor erneutem Versuch	5

Adressformat: frei gewählter Tag-Name; **gleiche Namen** auf beiden Seiten werden gespiegelt. Anwendungsbeispiel siehe Kapitel 48.



Figure 11: Bridge-Link / DMZ-Modus

26. Mappings und Datenfluss

Ein **Mapping** ist die Regel, die festlegt, **welcher Punkt zu welchem** übertragen wird.

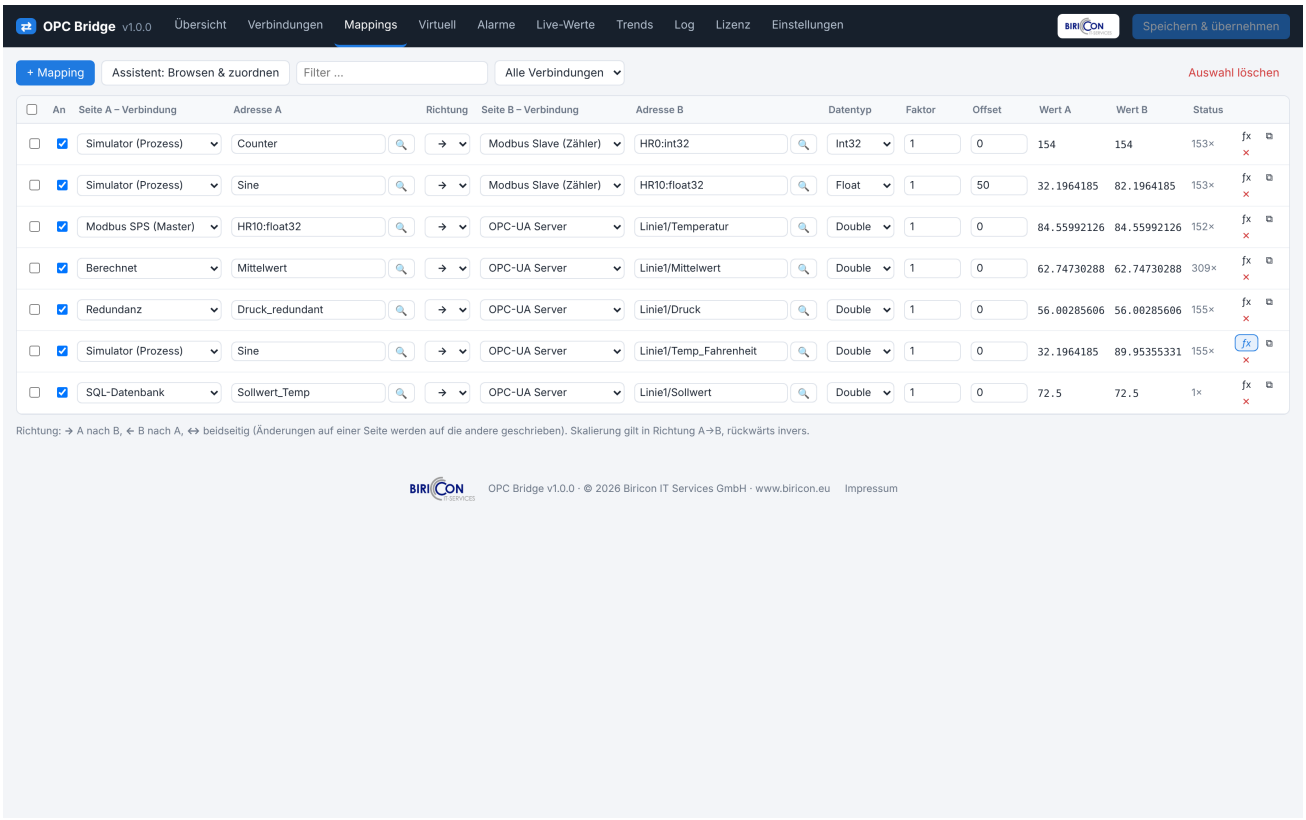


Figure 12: Mappings

Ein Mapping anlegen:

1. Reiter *Mappings* → **Hinzufügen**.
2. **Quelle A** wählen: Verbindung + Adresse (per □-Knopf durchsuchbar).
3. **Ziel B** wählen: Verbindung + Adresse.
4. **Richtung** festlegen:
   - **A → B:** A wird gelesen und nach B geschrieben.
   - **B → A:** umgekehrt.
   - **A ↔ B:** beidseitig; der zuletzt geänderte Wert wird übertragen.
5. Optional **Datentyp** und **Formel** (Kapitel 27) angeben.
6. **Speichern & übernehmen**.

Feldübersicht je Mapping:

Feld	Bedeutung
A / B (Verbindung + Adresse)	die beiden Endpunkte
Richtung	A→B, B→A, A↔B
Datentyp	Zieltyp (z. B. Double, Int32, Bool)
Formel	optionale Umrechnung (Variable x)
aktiv	Mapping ein-/ausschalten

Mehrere Quellen auf ein Ziel / ein Punkt an viele Ziele. Sie können denselben Quellpunkt in mehreren Mappings verwenden (an mehrere Ziele verteilen) und mehrere Quellen auf unterschiedliche Ziele führen. Für das Zusammenführen *mehrerer Quellen zu einem Wert* nutzen Sie virtuelle/berechnete Punkte (Kapitel 28).

27. Umrechnung und Formeln

Oft müssen Werte umgerechnet werden: Einheiten (°C↔°F), Skalierung (Rohwert→Prozent), Begrenzung. Pro Mapping lässt sich eine **Formel** hinterlegen. Die Variable **x** steht für den eingehenden Wert.



Figure 13: Formel

Beispiele:

Zweck	Formel
°C → °F	`x*1.8+32`
Skalierung /10	`x/10`
Rohwert 0–27648 → 0–100 %	`scale(x, 0, 27648, 0, 100)`
Begrenzen auf 0–100	`clamp(x, 0, 100)`
Runden	`round(x, 1)`
Schwellwert → Bool	`x > 50`

Eine vollständige Liste der verfügbaren Funktionen steht in Anhang B.

Assistent (Massen-Mapping). Über den Browse-Assistenten lassen sich viele Punkte auf einmal durchsuchen, auswählen und zuordnen – ideal bei großen Adressräumen.

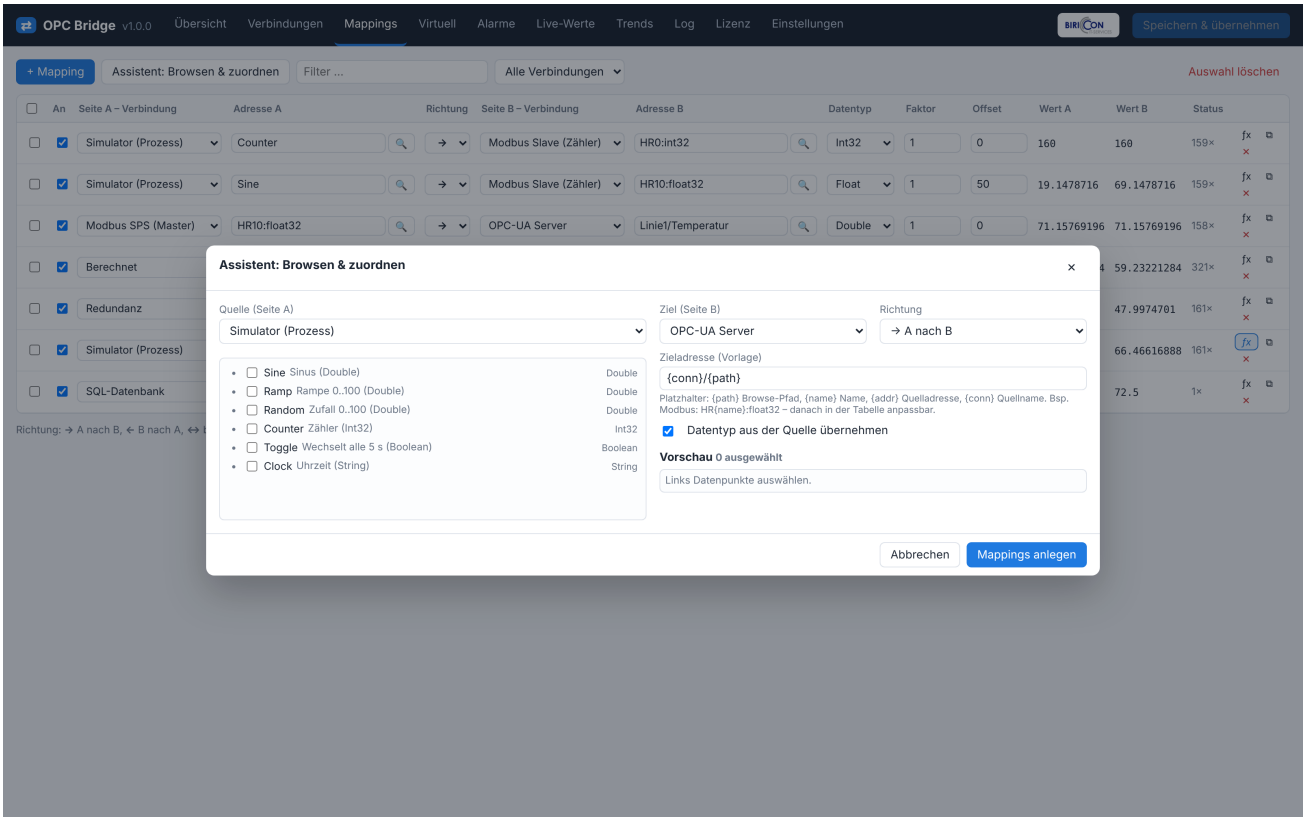


Figure 14: Assistent

28. Virtuelle Quellen und Redundanz

Im Reiter **Virtuell** erzeugen Sie Werte, die nicht direkt von einem Gerät kommen.

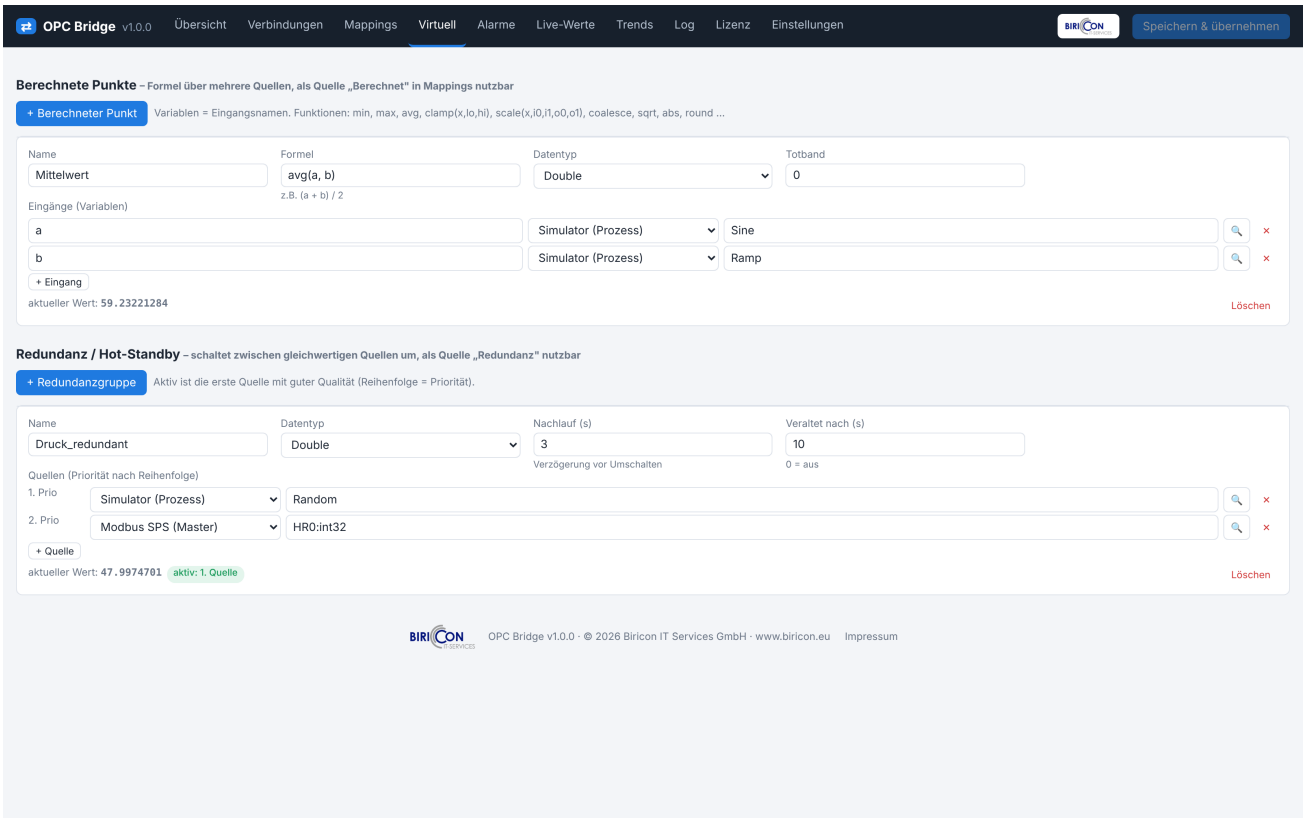


Figure 15: Virtuelle Quellen

28.1 Berechnete Punkte

Ein berechneter Punkt entsteht aus **mehreren Eingangspunkten** über eine Formel. Beispiele: Mittelwert dreier Temperaturen, Summe zweier Zähler, Differenz, Minimum/Maximum. Der berechnete Punkt ist danach wie jeder andere Punkt in Mappings, Trends und Alarmen nutzbar.

Anlegen: Name vergeben, Eingänge (Verbindung+Adresse) hinzufügen, Formel schreiben (die Eingänge sind als Variablen verfügbar), Datentyp wählen.

28.2 Redundanz / Hot-Standby

Eine Redundanzgruppe hat **mehrere Quellen für denselben Wert**. Die Bridge liefert automatisch die **beste verfügbare** Quelle und schaltet bei Ausfall oder

schlechter Qualität auf die nächste um (Failover). Nutzbar z. B. bei redundanten Sensoren oder doppelt angebundenen Steuerungen.

Anlegen: Name vergeben, Quellen in Prioritätsreihenfolge hinzufügen, Kriterium (z. B. Qualität, Veraltungszeit) festlegen.

---

29. Store-and-Forward

Zweck. Ist ein **Ziel** vorübergehend offline, gehen ohne Puffer Werte verloren. Mit **Store-and-Forward** werden sie in einer **SQLite-Warteschlange auf der Platte** persistent gepuffert – das übersteht sogar einen **Neustart** der Bridge – und nach der Wiederverbindung **in korrekter Reihenfolge** nachgeliefert.



Figure 16: Store-and-Forward

Aktivieren: In der **Zielverbindung** das Häkchen *Store-and-Forward* setzen und die maximale Puffergröße (Anzahl Werte) angeben. Der Übersicht und dem Status ist die Anzahl gepufferter Werte je Verbindung zu entnehmen.

Funktionsweise:

1. Ziel wird als „error" erkannt (nicht erreichbar).
2. Weiterzuleitende Werte wandern in die Warteschlange (bis zur Maximalgröße; danach werden die ältesten verworfen).
3. Sobald das Ziel wieder „connected" ist, wird die Warteschlange der Reihe nach abgearbeitet.

   »**Tipp**« Store-and-Forward ist besonders für Cloud-/Webhook-Ziele und für die OT-Seite des Bridge-Link wichtig, wo Verbindungen zeitweise abreißen können.

30. Historie und Trends konfigurieren

Im Reiter **Einstellungen → Historie**:

Feld	Bedeutung
Historie aktiv	Aufzeichnung ein/aus
Aufbewahrung (Tage)	wie lange Daten behalten werden
Min. Abstand je Punkt (s)	Drosselung (verhindert zu dichte Aufzeichnung)
Datenbankdatei	SQLite-Datei der Historie
Aufgezeichnete Punkte	Liste der Punkte mit optionalem **Totband**

Das **Totband** unterdrückt Aufzeichnungen bei kleinen Schwankungen (Rauschen): Erst eine Änderung größer als das Totband wird gespeichert. Die Darstellung erfolgt im Reiter *Trends* (Kapitel 13) mit CSV-Export.

31. Alarme, E-Mail und Webhook

Im Reiter **Alarme** definieren Sie Benachrichtigungsregeln.

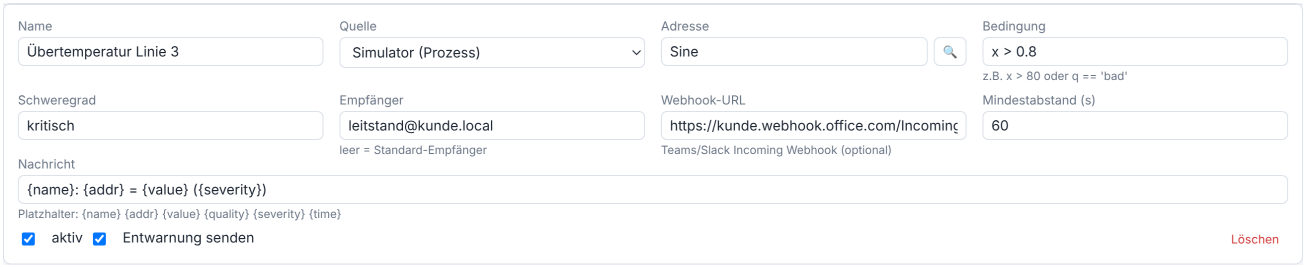


Figure 17: Alarmregel mit Webhook

Felder einer Regel:

Feld	Bedeutung
Name	Bezeichnung der Regel
Quelle / Adresse	der überwachte Punkt
Bedingung	Ausdruck über x (Wert) und q (Qualität), z. B. `x > 90`, `q == 'bad'`
Schweregrad	z. B. Info/Warnung/Kritisch
Empfänger	E-Mail-Adresse(n); leer = Standard-Empfänger
Webhook-URL	Microsoft-Teams-/Slack-/generischer Incoming-Webhook (optional)
Mindestabstand (s)	verhindert Alarmfluten
Nachricht	Text mit Platzhaltern `{name}` `{addr}` `{value}` `{quality}` `{severity}` `{time}`
aktiv	Regel ein/aus
Entwarnung senden	Meldung, wenn die Bedingung nicht mehr erfüllt ist

E-Mail (SMTP) einrichten: *Einstellungen* → *SMTP* – Host, Port, TLS, Benutzer, Absender, Standard-Empfänger. SMS ist über E-Mail-zu-SMS-Gateways möglich.

Webhook: Hinterlegen Sie je Regel eine Incoming-Webhook-URL (Teams/Slack). Der Alarmtext wird als JSON (`{"text": …}`) gesendet.

OPC-UA Alarms & Events (A&E): Vom UA-Client empfangene Ereignisse eines Servers erscheinen ebenfalls im Alarm-Bereich.

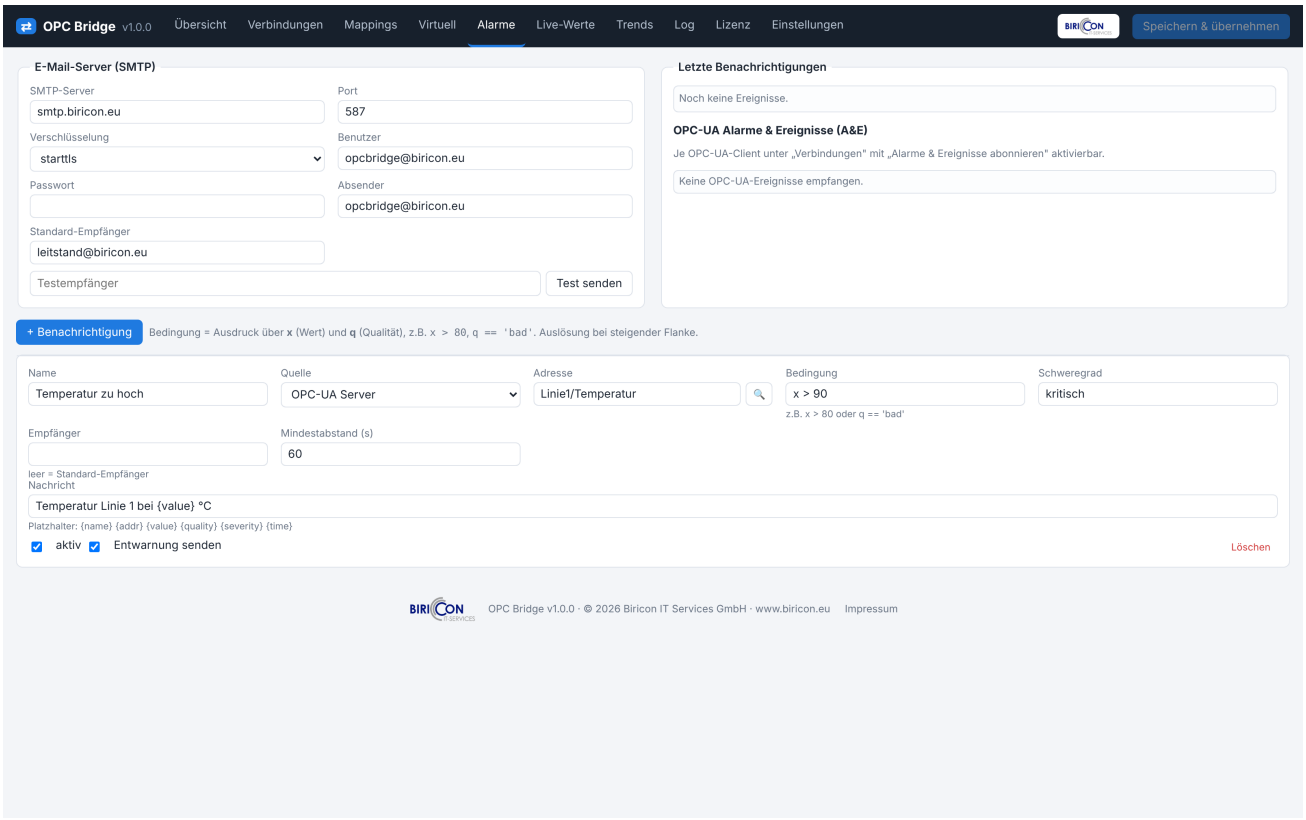


Figure 18: Alarme

32. CAN-Analyser und Gateway

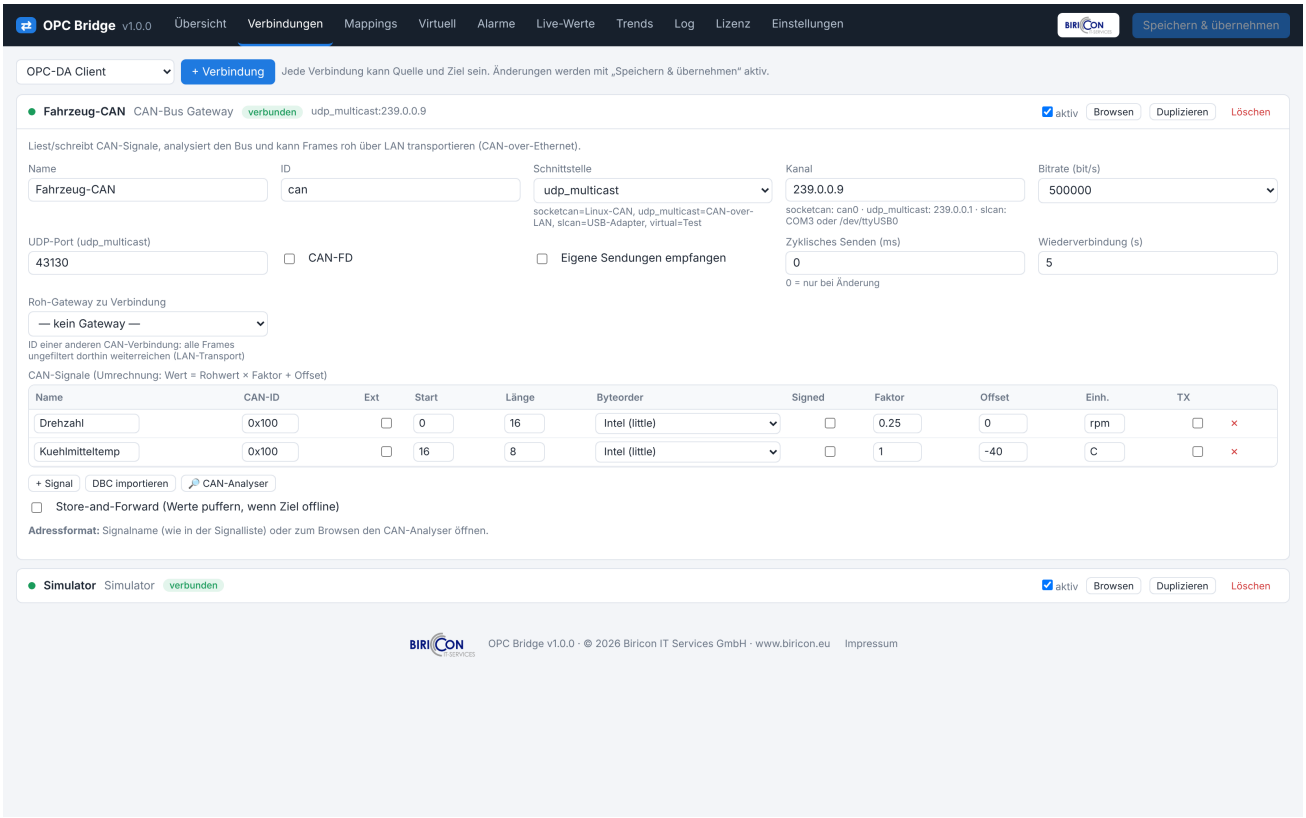


Figure 19: CAN-Gateway

Signale finden (Analyser). Der **CAN-Analyser** zeigt die mitlaufenden Frames in Echtzeit: ID, Länge, Datenbytes, Häufigkeit. Bei Extended-Frames werden **J1939**-Informationen (PGN, SA, Priorität) automatisch dekodiert.

Signale anlegen. Für jedes benötigte Signal definieren Sie:

Parameter	Bedeutung
Name	Bezeichnung (= Adresse in Mappings)
CAN-ID	Botschaft, in der das Signal steckt
Start-Bit / Länge	Bitlage im Datenfeld
Byte-Reihenfolge	Intel (little) / Motorola (big)
Vorzeichen	mit/ohne
Faktor / Offset	physikalischer Wert = roh · Faktor + Offset

Alternativ per **DBC-Import** die Signaldefinitionen einlesen.

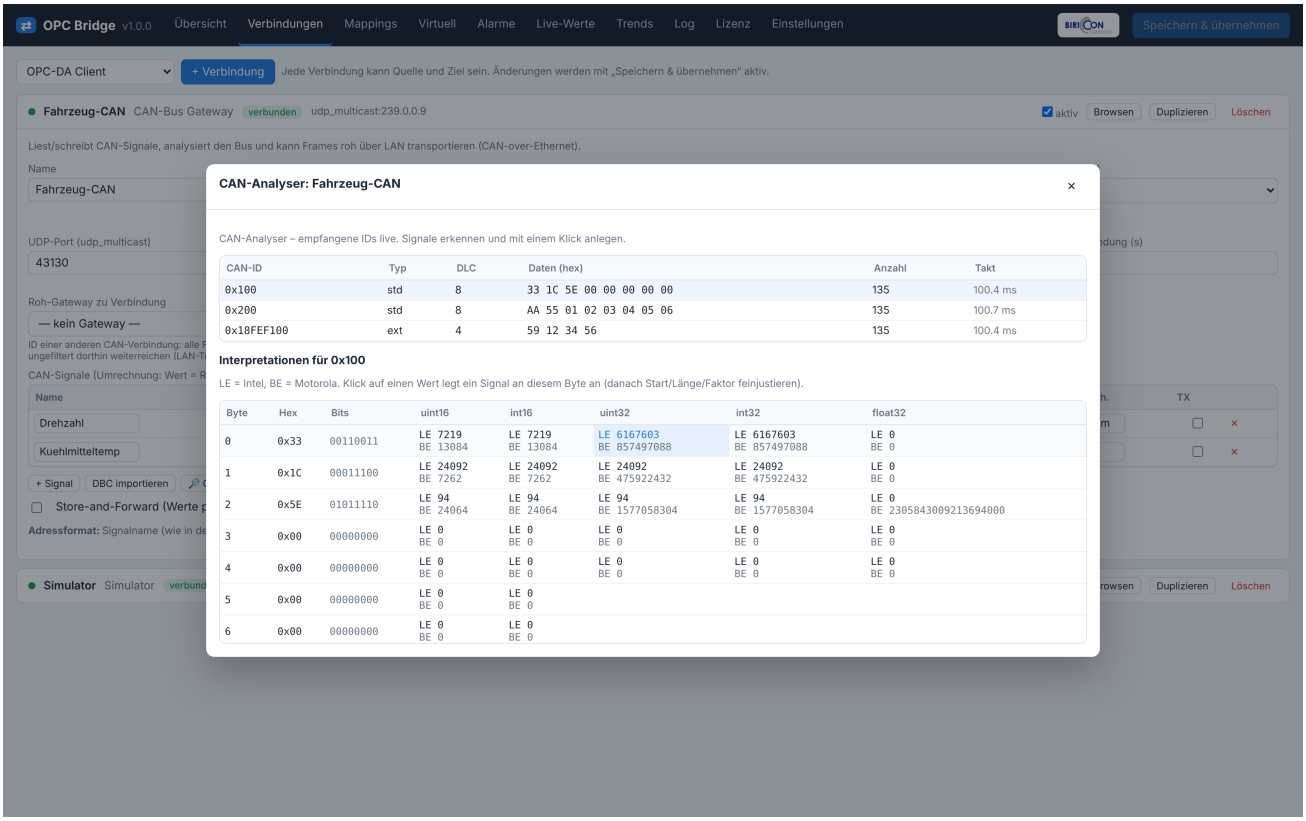


Figure 20: CAN-Analyser

Roh-Gateway. Statt Signale zu extrahieren, kann der komplette Bus **ungefiltert über LAN** transportiert werden (Feld *Roh-Gateway zu Verbindung*). So lassen sich zwei entfernte CAN-Segmente über Ethernet koppeln (optional beidseitig).

Werte exportieren. Extrahierte Signale werden wie jeder Punkt per Mapping nach OPC, Modbus oder MQTT geführt – inklusive Umrechnungsformeln.

33. Modbus-Register-Inspektor

Modbus-Werte sind oft kryptisch: Welches Register, welcher Datentyp, welche Byte-Reihenfolge? Der **Register-Inspektor** liest einen Registerbereich und zeigt **alle Interpretationen gleichzeitig**.

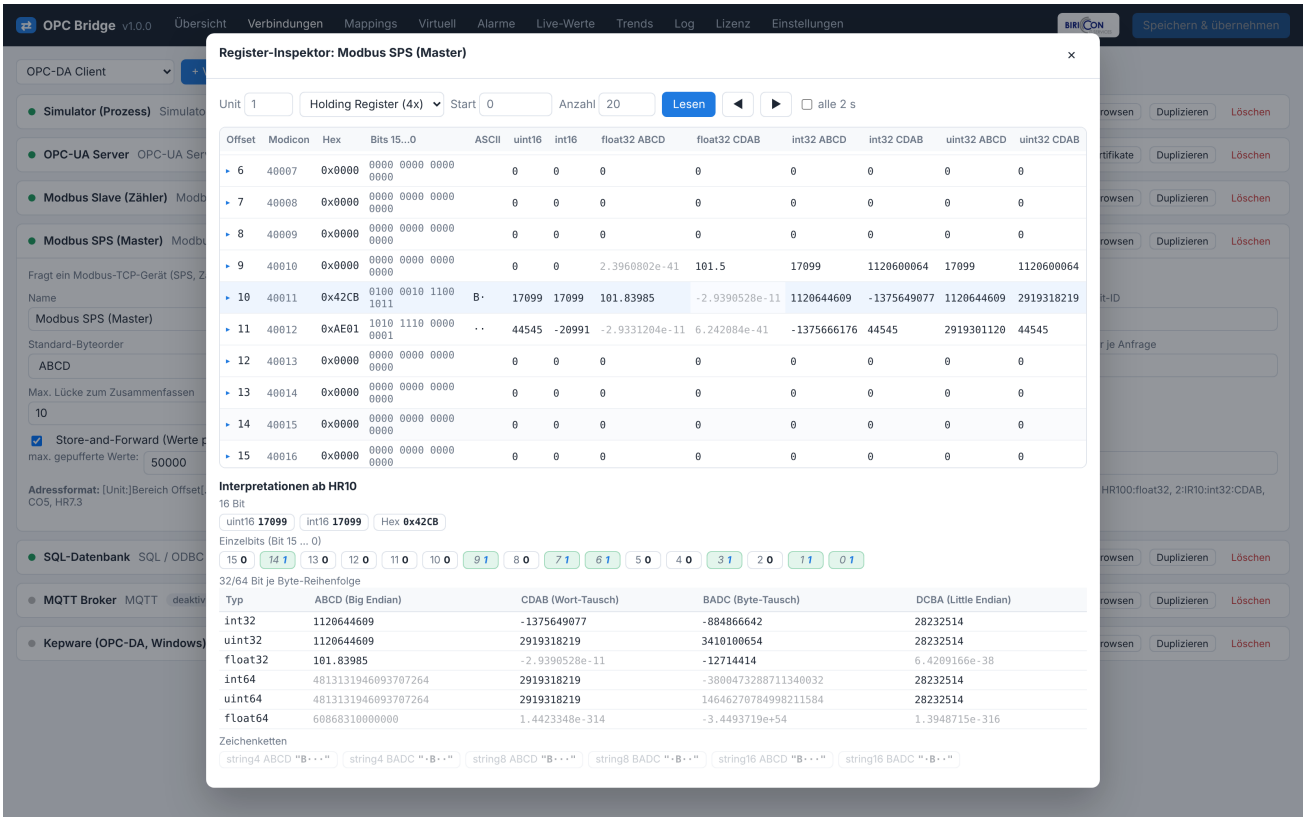


Figure 21: Register-Inspektor

Vorgehen:

1. Modbus-Verbindung, Bereich (HR/IR) und Startadresse sowie Anzahl wählen.
2. Der Inspektor liest und zeigt je Registerpaar die Deutung als `int16`, `uint16`, `int32`, `uint32`, `float32` – in **allen** Byteorders (`ABCD`, `CDAB`, `BADC`, `DCBA`).
3. Die Zeile mit dem **plausiblen Wert** verrät Datentyp und Byteorder. Diese tragen Sie dann in die Adresse ein, z. B. `HR100:float32:CDAB`.

   »Tipp« Der Register-Inspektor spart bei der Inbetriebnahme viel Zeit und Rätselraten. Nutzen Sie ihn immer, wenn ein Modbus-Wert nicht stimmt.

34. WebView

Die **WebView** ist eine öffentliche, **token-geschützte** Live-Seite (`/view?token=…`), die ausgewählte Werte als Kacheln zeigt – **ohne Login**.

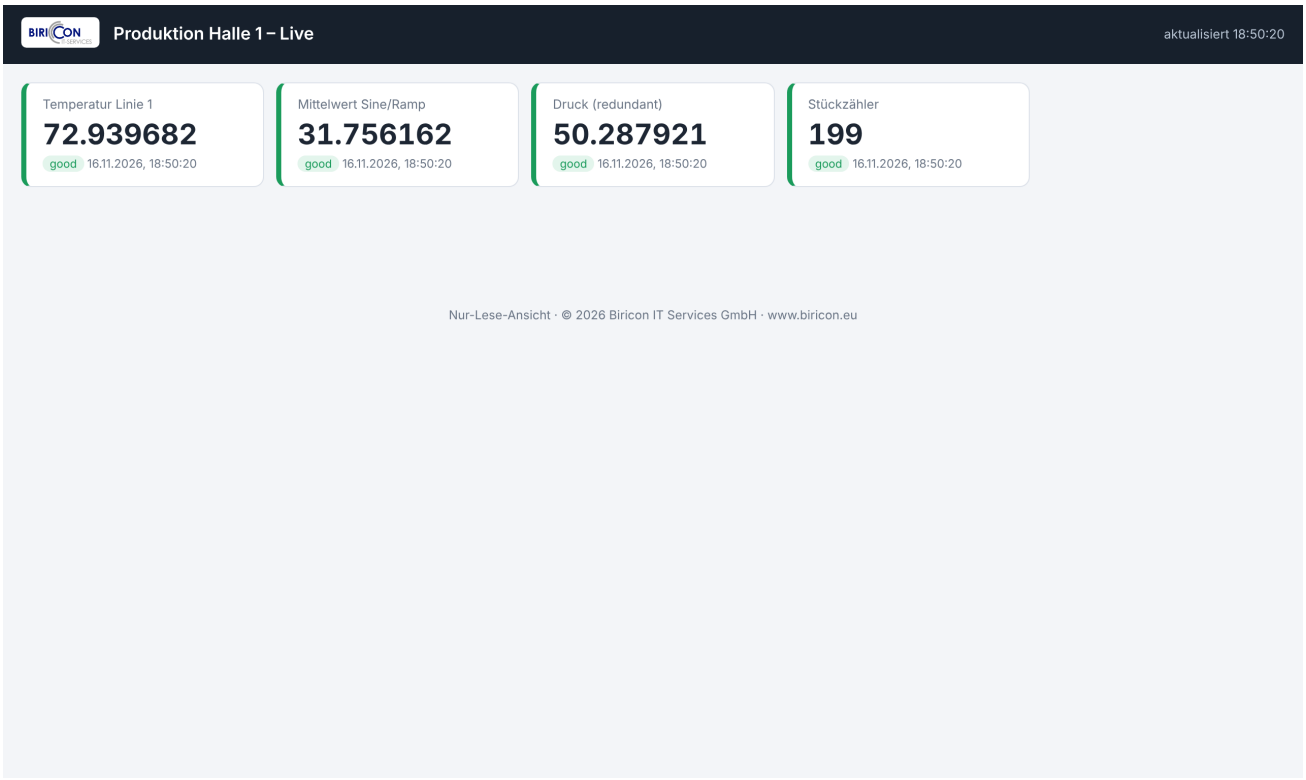


Figure 22: WebView

Einrichten: *Einstellungen* → *WebView* – aktivieren, Titel vergeben, Punkte auswählen, Token festlegen. Der Link kann an Leitstand oder Kunde weitergegeben werden.

Feld	Bedeutung
WebView aktiv	Funktion ein/aus
Titel	Überschrift der Seite
Token	Geheimnis im Link (`/view?token=…`)
Punkte	welche Werte angezeigt werden

»Achtung« Jeder mit Link+Token sieht die freigegebenen Werte. Nur benötigte Punkte freigeben und im Internetbetrieb HTTPS davorsetzen.

35. Benutzer, Rollen und API-Authentifizierung

Im Reiter **Einstellungen**.

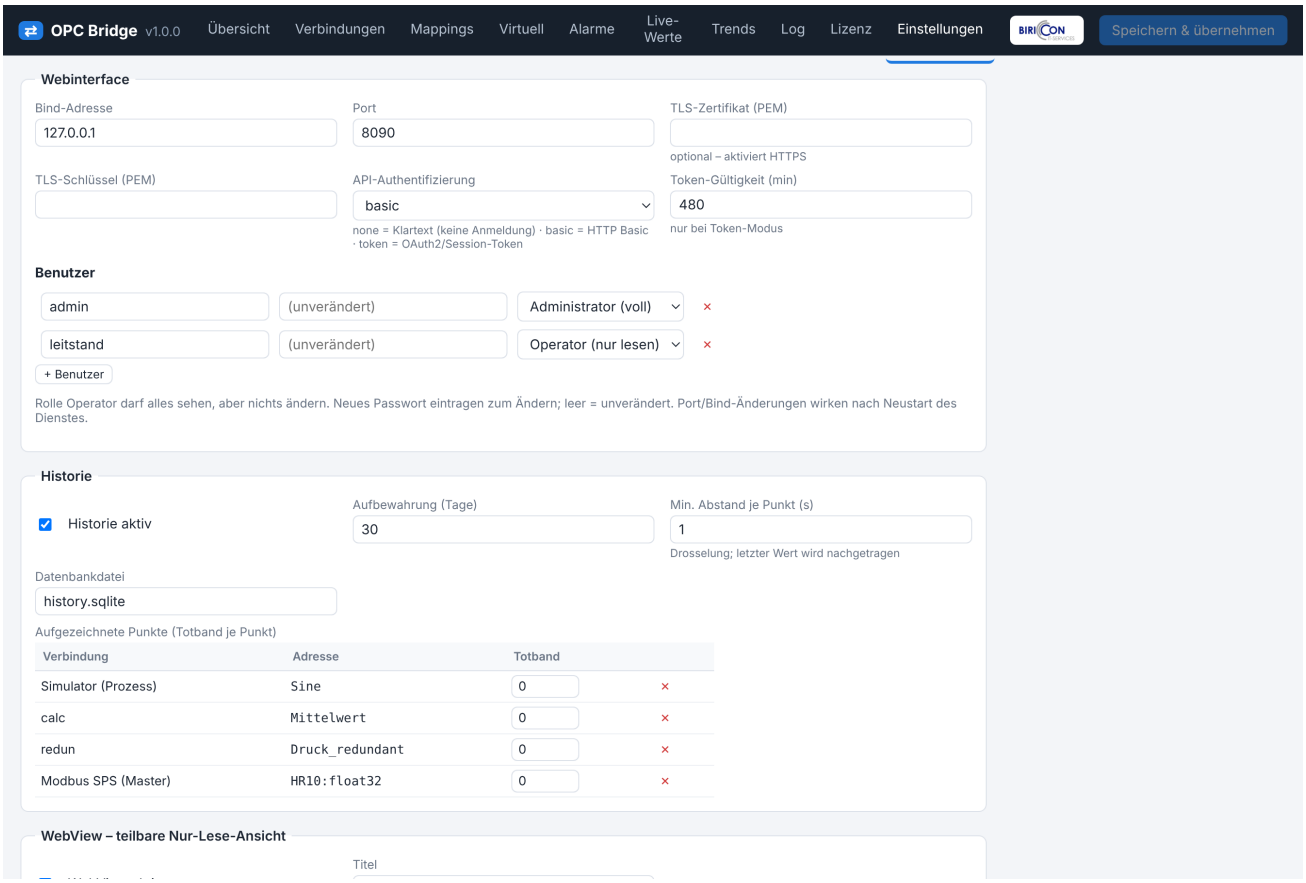


Figure 23: API-Authentifizierung

35.1 Benutzer und Rollen

Rolle	Rechte
Administrator	voller Zugriff (lesen, konfigurieren, schreiben)
Operator	nur lesen – sieht alles, ändert nichts

Passwörter werden als **PBKDF2-Hash** gespeichert. Neues Passwort eintragen zum Ändern; leer = unverändert. Schreibende Zugriffe sind serverseitig auf Administratoren beschränkt.

35.2 API-Authentifizierung (konfigurierbar)

Modus	Bedeutung
Klartext (`none`)	**keine Anmeldung** – offener Zugriff. Nur im abgeschotteten Maschinennetz!
HTTP-Basic (`basic`)	Login wie in der Weboberfläche.
OAuth2-/Session-Token (`token`)	Anmeldung über `/api/login` bzw. `/api/token`, danach `Bearer`-Token mit einstellbarer **Gültigkeitsdauer** (Token-TTL).

35.3 Webinterface-Einstellungen

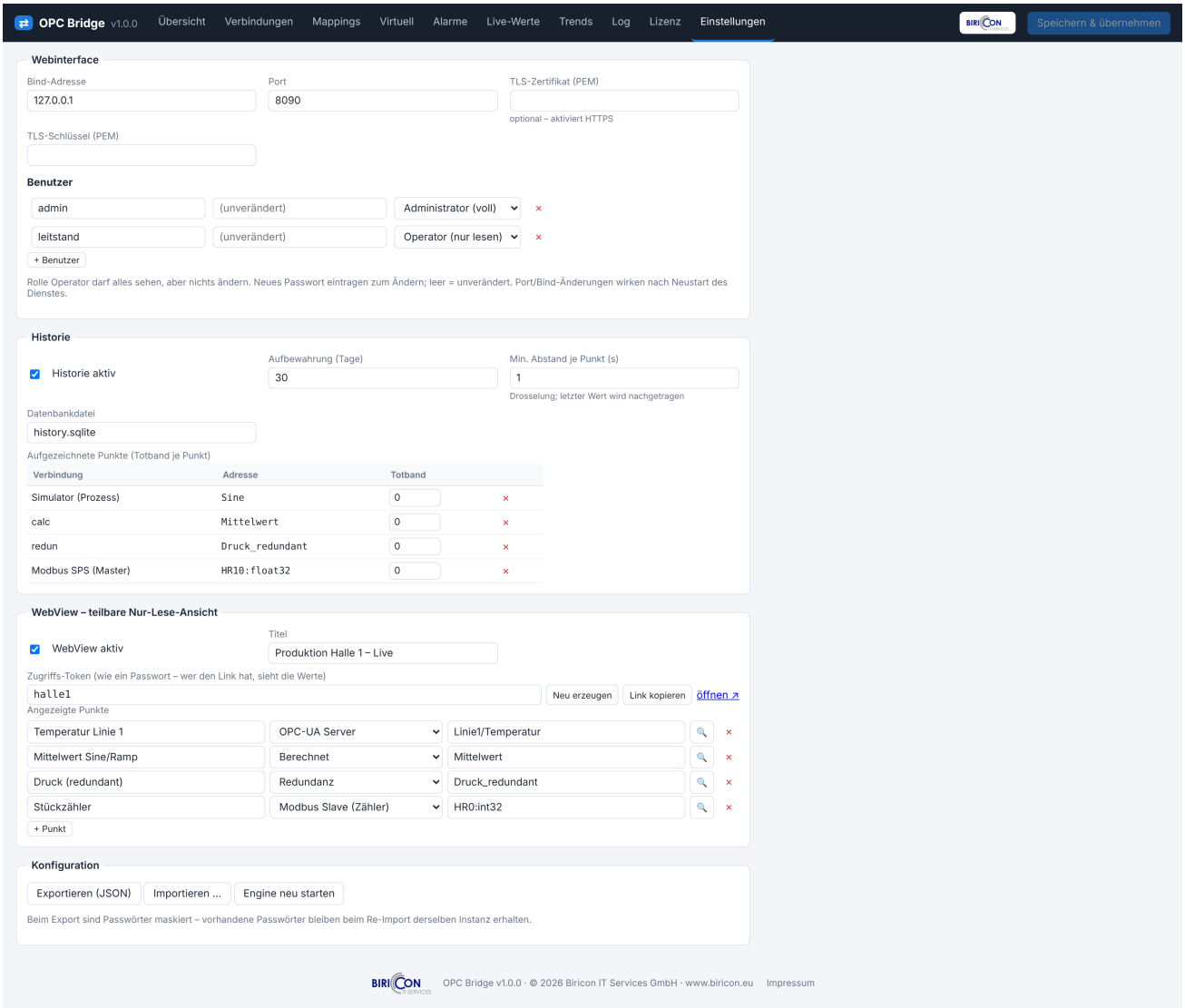


Figure 24: Einstellungen

Feld	Bedeutung
Bind-Adresse / Port	Netzwerkschnittstelle und Port der Oberfläche (wirkt nach Dienst-Neustart)
TLS-Zertifikat / -Schlüssel (PEM)	aktiviert HTTPS
API-Authentifizierung	siehe 35.2
Token-Gültigkeit (min)	Lebensdauer der Tokens im Token-Modus

36. Echtzeit-Aktualisierung (WebSocket)

Die Live-Werte und Status werden **in Echtzeit per WebSocket** an die Oberfläche **gepusht** – ohne Neuladen, mit minimaler Verzögerung. Fällt die WebSocket-Verbindung aus (z. B. durch einen Proxy), schaltet die Oberfläche **automatisch auf regelmäßiges Abfragen (Polling)** um und versucht später erneut, die WebSocket aufzubauen.

Wichtig für die Bedienung: Die Live-Aktualisierung berührt **nur Anzeige-Zellen**. Eingabe- und Konfigurationsfelder bleiben **jederzeit bedienbar** – eine eintreffende Aktualisierung überschreibt **niemals** Ihre laufende Eingabe. Sie können also in Ruhe konfigurieren, während im Hintergrund Live-Werte eintreffen.

Technisch nutzt die WebSocket ein **kurzlebiges Ticket** (über `/api/wsticket`), das im gewählten Authentifizierungsmodus abgesichert ist.

36a. Neuerungen in Version 1.1

Dieses Kapitel fasst die Erweiterungen der Version 1.1 zusammen. Die Funktionen fügen sich nahtlos in die bisherige Bedienung ein (schema-getriebene Verbindungsformulare, Mappings, Store-and-Forward usw.).

36a.1 Siemens-S7-Treiber (direkt, ohne OPC-UA)

Der neue Verbindungstyp **Siemens S7 (direkt)** spricht S7-300/400/1200/1500 direkt über RFC1006/ISO-on-TCP an und liest/schreibt **Datenbausteine (DB)**, **Merker (M)**, **Eingänge (E/I)** und **Ausgänge (A/Q)** – ohne dass an der SPS OPC-UA aktiviert oder lizenziert sein muss. Das ist im Bestand oft der schnellste Weg, eine S7 anzubinden.

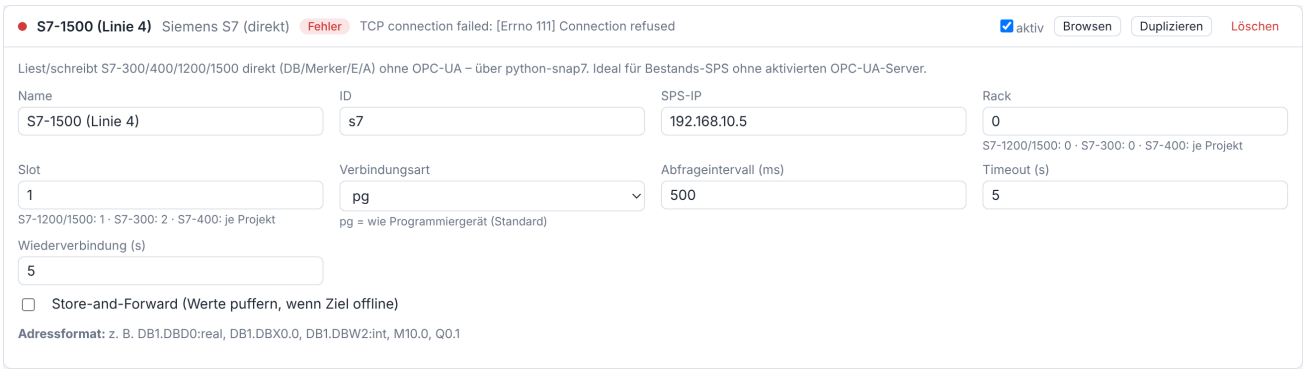


Figure 25: Siemens-S7-Verbindung

- **Verbindung:** SPS-IP, **Rack** und **Slot** (S7-1200/1500: 0/1 · S7-300: 0/2).
- **Adressformat:** `DB1.DBX0.0` (Bool), `DB1.DBW4` (Int), `DB10.DBD20:real` (Real), `DB1.DBD0:dint`, `M10.0`, `MW20`, `Q0.1`, optionaler Typ-Suffix (`:bool :int :dint :real :word :dword :string<n>`).
- **Technik:** benötigt die Bibliothek **python-snap7** (im Paket/Image enthalten).

36a.2 InfluxDB-Zeitreihentreiber

Der neue Ausgabetreiber **InfluxDB (Zeitreihen)** schreibt Werte im **InfluxDB Line Protocol** per HTTP – für InfluxDB v1 und v2/v3 – mit **Batch-Schreiben** für hohen Durchsatz.

- **v2/v3:** Token + Organisation + Bucket · **v1:** Database (+ optional Benutzer/Passwort).

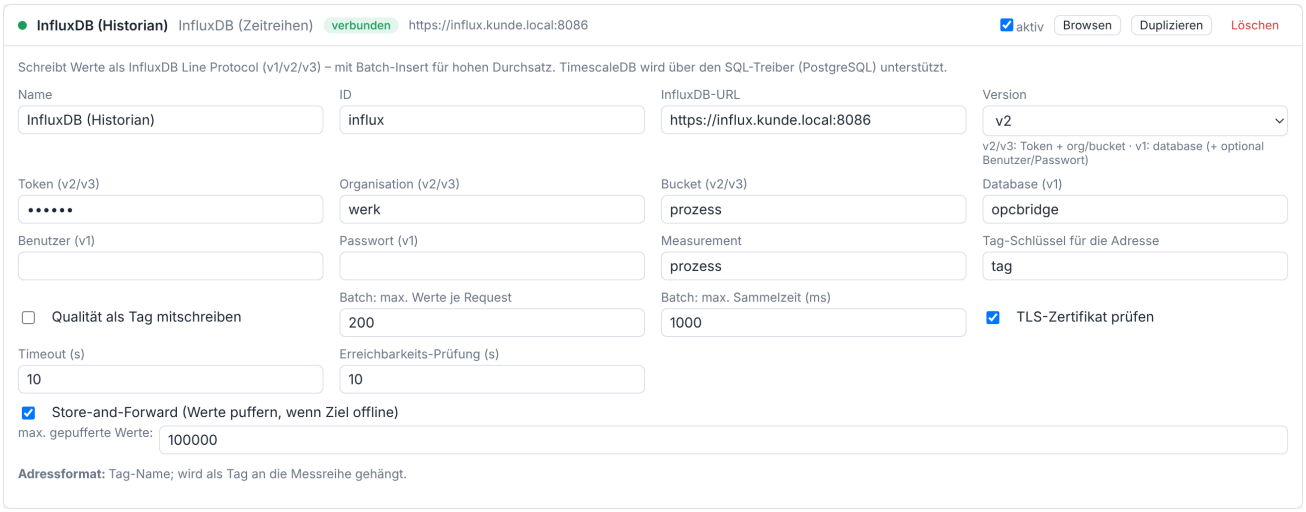


Figure 26: InfluxDB-Verbindung

- **Measurement** und **Tag-Schlüssel** frei wählbar; Qualität optional als Tag.
- **TimescaleDB** wird über den vorhandenen **SQL-Treiber** (PostgreSQL-URL) unterstützt.

36a.3 Batch-/Bulk-Schreiben (SQL, Webhook, InfluxDB)

Bei vielen schnell ändernden Tags erzeugt ein einzelner Request pro Wert hohen Overhead. Die Ausgabetreiber **SQL**, **Webhook** und **InfluxDB** bieten daher einen **Batch-Modus**:

- Felder **Batch: max. Werte je Request** (0 = einzeln) und **Batch: max. Sammelzeit (ms)**.
- Es wird gesendet, sobald die Anzahl erreicht ist **oder** die Sammelzeit abläuft.
- SQL schreibt als **Multi-Row-Insert** in einer Transaktion, Webhook als **JSON-Array**, InfluxDB als **mehrzeiliges Line Protocol**.
- Spielt optimal mit **Store-and-Forward** zusammen (gepufferte Werte werden gebündelt nachgeliefert).

36a.4 Strikter Einweg-Modus (Data-Diode) im Bridge-Link

Für KRITIS- und Hochsicherheitsumgebungen bietet der **Bridge-Link** jetzt einen **strikten Einweg-Modus (Data-Diode)**: Der Rückkanal **IT→OT** wird vollständig blockiert.

- Schalter **„Strikter Einweg-Modus OT→IT (Data-Diode)“** auf Client-

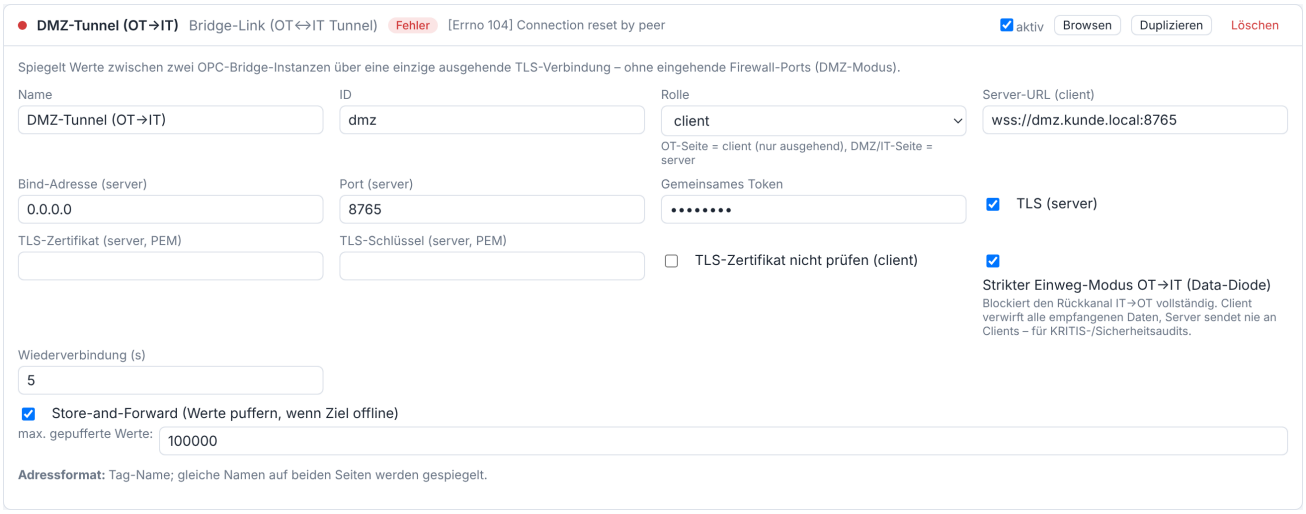


Figure 27: Bridge-Link mit Data-Diode

und/oder Serverseite.

- Der **Client (OT)** verwirft dann **alle empfangenen** Daten, der **Server (IT)** sendet **nie** an Clients – so ist der Rückkanal auf beiden Seiten nachweisbar unterbunden.
- Ideal für Sicherheitsaudits: Werte fließen ausschließlich OT→IT.

36a.5 Zentrale Anmeldung: LDAP/AD, OIDC/SSO und 2FA

Für IT-Audits in größeren Betrieben lässt sich die Anmeldung nun an die zentrale Benutzerverwaltung anbinden und mit Zwei-Faktor absichern.

- **LDAP / Active Directory:** Anmeldung per Verzeichnisdienst, Gruppen-zu-Rollen-Abbildung (Admin-/Operator-Gruppe), Dienstkonto-Suche oder direkte Bindung, LDAPS/SSL.
- **OIDC / SSO:** Authorization-Code-Flow (z. B. Microsoft Entra ID, Keycloak, Google). Der Button **„Mit SSO anmelden"** erscheint im Login; Rollen werden aus einem Claim abgeleitet. Die **Redirect-URI** (`/api/oidc/callback`) wird beim Identity-Provider hinterlegt.
- **TOTP-2FA:** Jeder lokale Benutzer kann in den Einstellungen 2FA aktivieren (Authenticator- App per Secret/QR). Bei aktiver 2FA verlangt der Login zusätzlich einen 6-stelligen Code; die Anmeldung erfolgt dann über den Token-Modus.

  »**Hinweis**« LDAP/OIDC/2FA sind optional und standardmäßig deaktiviert. Die bestehenden lokalen Benutzer bleiben unverändert nutzbar.

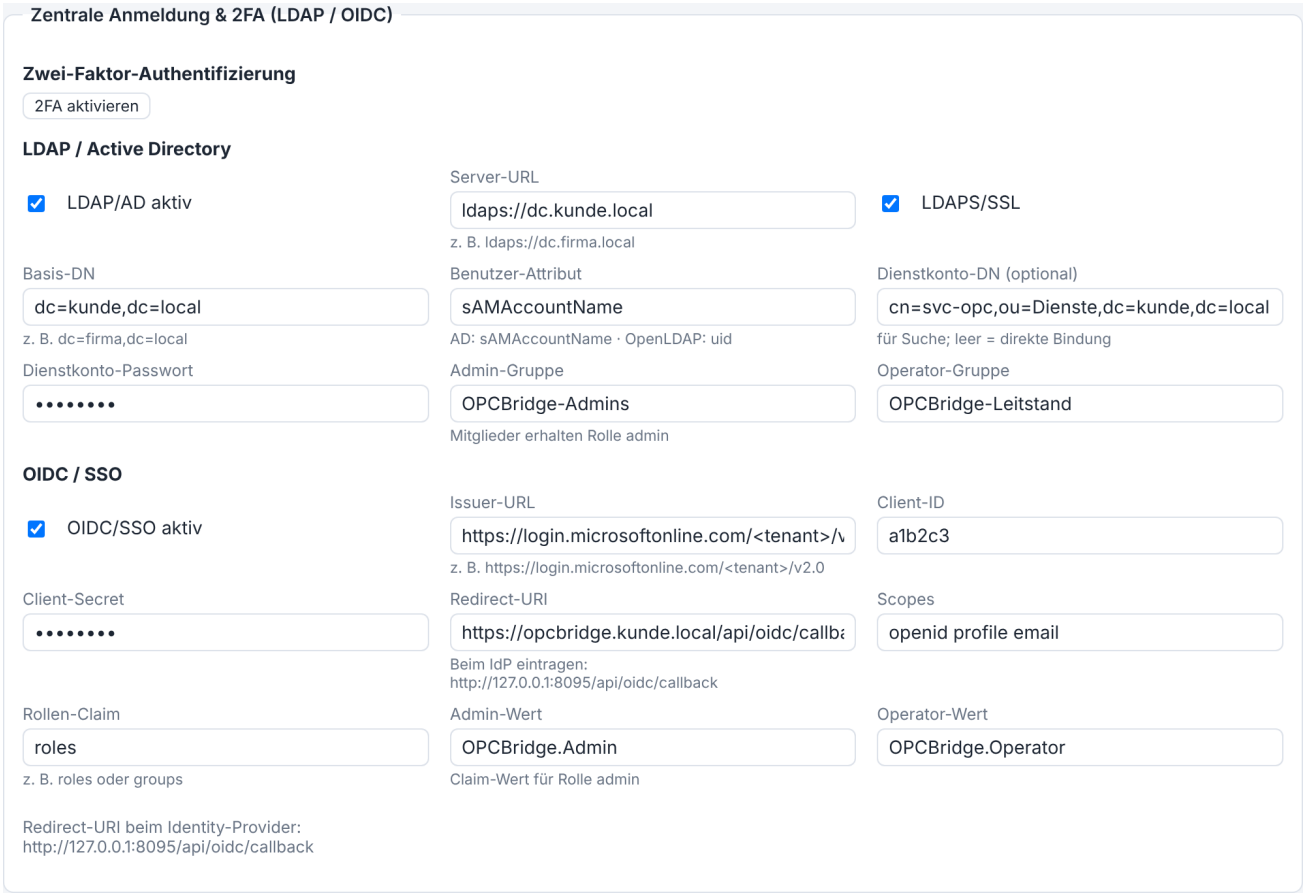


Figure 28: LDAP/OIDC/2FA-Einstellungen

36a.6 Mehrsprachige Oberfläche

Die Weboberfläche ist **mehrsprachig**. Über den Sprachwähler oben rechts lässt sich zwischen **Deutsch, Englisch, Französisch, Spanisch und Portugiesisch** umschalten; die Wahl wird im Browser gespeichert.

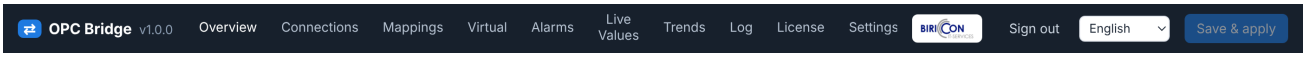


Figure 29: Mehrsprachige Oberfläche (Englisch)

Deutsch ist die Quellsprache. Fehlt für einen Text eine Übersetzung, wird der deutsche Begriff angezeigt – so lassen sich weitere Übersetzungen schrittweise ergänzen.

Teil V – Schnittstellen und Integration

37. REST-API und eingebaute Dokumentation

Alle Funktionen der Bridge sind auch über eine **REST-API** erreichbar – ideal für Automatisierung, eigene Dashboards oder die Anbindung an übergeordnete Systeme.

Eingebaute, interaktive Dokumentation. Die API-Dokumentation ist **fester Bestandteil des Produkts**. Sie erreichen sie unter **`/api/docs`**. Die komplette Oberfläche ist **mitgeliefert** und lädt **nichts aus dem Internet** (keine CDN-Abhängigkeit) – sie funktioniert daher auch im **Air-Gap-/OT-Netz**. Grundlage ist die offene **OpenAPI-3.1-Spezifikation** unter `/api/openapi.json`. Alternativ steht eine ReDoc-Ansicht unter `/api/redoc` bereit.

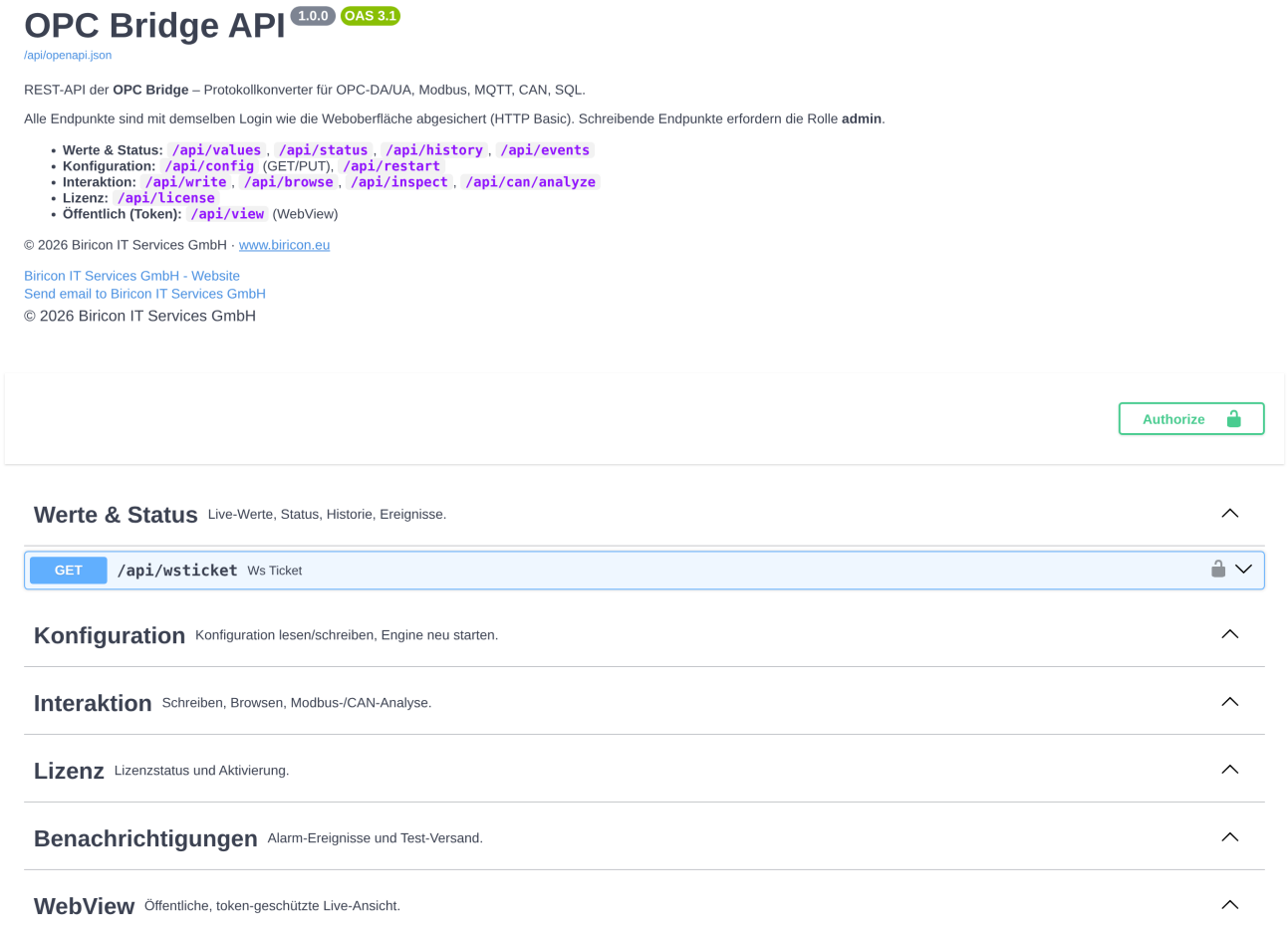


Figure 30: Eingebaute REST-API-Dokumentation

Jeder Endpunkt ist in der Oberfläche direkt testbar („Try it out"). Eine vollständige Endpunktliste finden Sie in Anhang C.

Authentifizierung. Je nach Modus (Kapitel 35.2): keine (Klartext), HTTP-Basic oder Bearer-Token. Beispiel (Token holen und verwenden):

```
# Token holen (Basic-/Token-Modus)
curl -u admin:geheim -X POST http://host:8080/api/login
# -> {"token":"..."}

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

Schreibende Endpunkte (`PUT /api/config`, `POST /api/write`, `POST /api/restart` …) erfordern die Rolle **Administrator**.

---

38. Konfigurationsdatei (config.json)

Die gesamte Konfiguration liegt in **config.json** im Datenverzeichnis (siehe Anhang F). Normalerweise bearbeiten Sie sie über die Oberfläche; für Backup, Massen-Rollout oder Versionsverwaltung ist der direkte Zugriff nützlich.

Aufbau (vereinfacht):

```
{
  "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": { … }
}
```

> **»Achtung«** Passwörter werden als Hash gespeichert. Bearbeiten Sie die Datei nur bei gestopptem Dienst oder laden Sie sie über PUT `/api/config`. Ungültiges JSON verhindert den Start – bewahren Sie immer ein Backup auf (Kapitel 41).

Config-as-Code. Da die gesamte Konfiguration in einer Datei liegt, kann sie in einer Versionsverwaltung (z. B. Git) gepflegt und auf mehrere Anlagen ausgerollt werden.

Teil VI – Betrieb

39. Lizenzierung

Die OPC Bridge ist durch **rechnergebundene Seriennummern** geschützt (Ed25519-signiert, offline prüfbar, fälschungssicher).

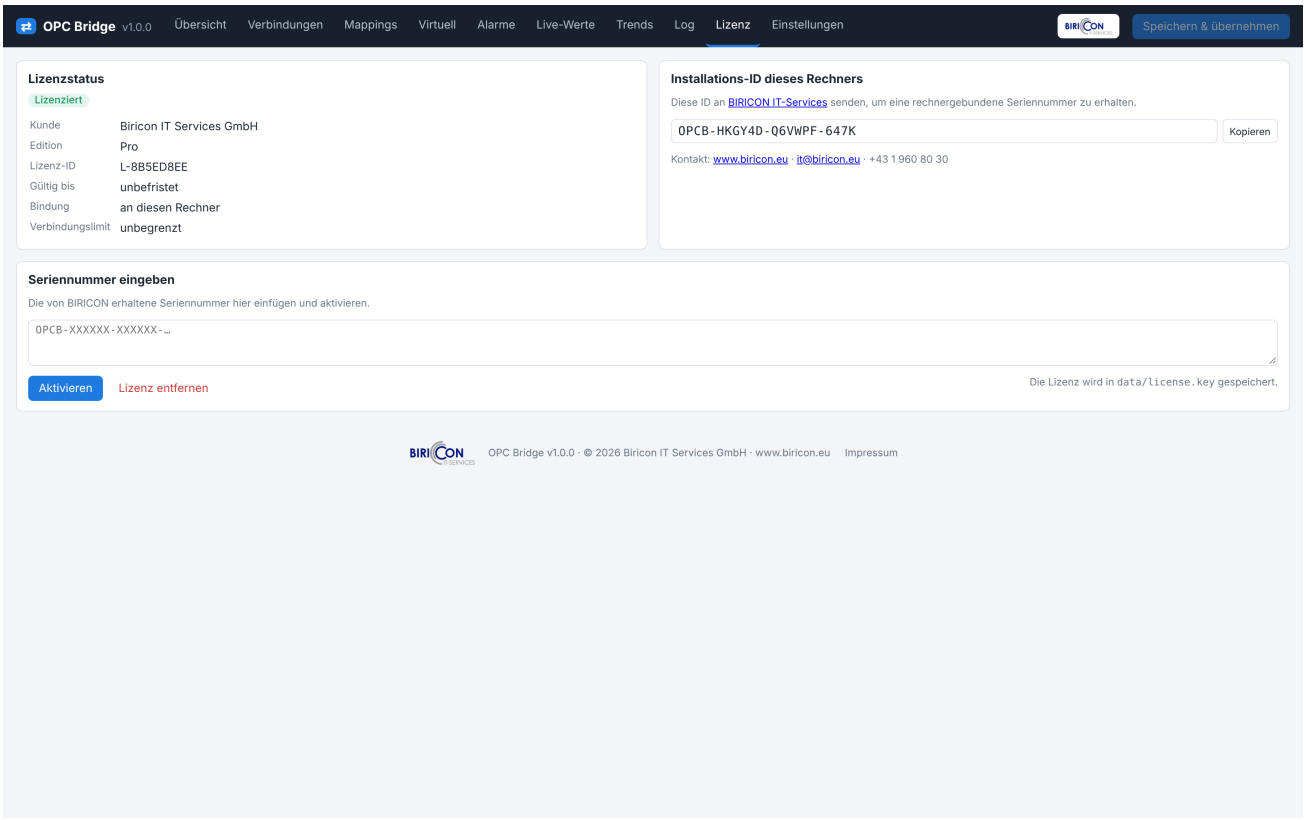


Figure 31: Lizenz

39.1 Demo-Modus

Ohne Lizenz läuft die Bridge im **Demo-Modus**:

- je Start **60 Minuten** voll funktionsfähig, danach wird die Weiterleitung gestoppt,
- dauerhaft **maximal 2 Verbindungen**.

Zum Weiterbetrieb genügt ein Neustart (erneut 60 Minuten) – für den Dauerbetrieb ist eine Lizenz erforderlich.

39.2 Aktivierung

1. Reiter *Lizenz* öffnen und die **Installations-ID** ablesen (sie ist an diesen Rechner gebunden).
2. Die Installations-ID an **Biricon** senden.
3. Die erhaltene **Seriennummer** im Feld eintragen und **Aktivieren**.

Die Lizenz gilt genau für diesen Rechner. Bei Hardwaretausch ist eine Neu-Ausstellung nötig.

39.3 Lizenzarten

- **Unbefristet:** dauerhaft gültig.
- **Befristet (Jahres-/Wartungslizenz):** mit Ablaufdatum; rechtzeitig verlängern.

---

40. Sicherheit und Härtung

Empfehlungen für den Produktivbetrieb:

1. **Passwörter ändern.** Erststart-Passwort sofort ändern; starke Passwörter verwenden.
2. **Rollen nutzen.** Für Beobachter nur die Rolle **Operator** vergeben.
3. **API-Authentifizierung** mindestens auf **Basic**, besser **Token** stellen. **Klartext** nur im physisch abgeschotteten Netz.
4. **HTTPS aktivieren.** TLS-Zertifikat/-Schlüssel hinterlegen oder einen Reverse-Proxy (nginx) mit TLS davorsetzen.
5. **OPC-UA absichern.** Security-Policy `Basic256Sha256 SignAndEncrypt`, Truststore statt `trust_all`, Benutzer statt `anonymous=readwrite`.
6. **Netzsegmentierung.** OT und IT trennen; für die Kopplung den **Bridge-Link** (DMZ-Modus) verwenden, damit keine eingehenden OT-Ports nötig sind.
7. **WebView** nur mit starkem Token und HTTPS; nur nötige Punkte freigeben.
8. **Ports minimieren.** Nur benötigte Dienste (OPC-UA-Server, Modbus-Slave) aktivieren und in der Firewall öffnen.
9. **Updates** zeitnah einspielen (Kapitel 42).

---

41. Datensicherung und Wiederherstellung

Was sichern? Das gesamte **Datenverzeichnis** (Anhang F) enthält alles Wichtige:

- `config.json` – die Konfiguration,
- `license.key` – die Lizenz,
- die Historien-SQLite-Datei (falls die Historie gesichert werden soll),
- die Store-and-Forward-Warteschlange (nur bei laufendem Puffer relevant).

Sicherung (Linux-Beispiel):

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

Wiederherstellung: Dienst stoppen, Datenverzeichnis aus dem Backup zurückspielen, Dienst starten. Da die **Lizenz rechnergebunden** ist, funktioniert sie nur auf demselben Rechner; bei neuem Rechner eine neue Lizenz anfordern.

> **»Tipp«** Für reine Konfig-Backups genügt `config.json`. Legen Sie diese Datei in eine Versionsverwaltung, um Änderungen nachvollziehen zu können.

---

42. Update und Migration

Windows: Neues Setup ausführen – es aktualisiert das Programm und behält Daten und Lizenz. Vorher ein Backup des Datenverzeichnisses anlegen.

Debian/Ubuntu:

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

Das Paket aktualisiert das Programm; Daten unter `/var/lib/opcbridge/` bleiben erhalten.

Docker: Neues Image beziehen/bauen und `docker compose up -d` – das Volume `./data` bleibt bestehen.

Migration auf anderen Rechner: Datenverzeichnis kopieren, Programm installieren – und eine **neue Lizenz** anfordern (rechnergebunden).

---

43. Leistung und Dimensionierung

- **Abfrageintervalle** bestimmen die Last: Kleine `poll_ms`/`update_rate_ms` erhöhen die Aktualität, aber auch CPU- und Netzlast. Wählen Sie so langsam wie fachlich vertretbar.
- **Modbus-Blockbildung:** `max_regs` und `max_gap` fassen Register zu wenigen Anfragen zusammen – wichtig für viele Register und langsame Geräte.
- **Historie:** Totband und „Min. Abstand je Punkt" begrenzen das Datenvolumen. Planen Sie Plattenplatz entsprechend Aufbewahrungszeit und Punktezahl.
- **Viele Verbindungen:** Jede Verbindung läuft entkoppelt (Multithreading); eine langsame Gegenstelle bremst die anderen nicht.
- **Netzwerk:** Für MQTT/Sparkplug/Webhook über WAN ist Store-and-Forward dringend empfohlen.

---

44. Fehlersuche

Die **erste Anlaufstelle** ist immer der Reiter **Log** bzw. unter Linux `journalctl -u opcbridge -f`.

Symptom	Mögliche Ursache	Lösung
Keine Anmeldung möglich	Passwort vergessen	Erststart-Passwort in `initial_admin_password.txt` (Anhang F).
Verbindung bleibt „verbinde”/rot	IP/Port/Firewall/Gerät aus	Adresse, Port, Netzwerk prüfen; Fehlermeldung im Log lesen.
Werte „bad”	Quelle liefert nicht, Adresse falsch	Adresse per Browsen prüfen; bei Modbus Register-Inspektor nutzen.
Modbus-Zahl unsinnig	falscher Typ/Byteorder	Register-Inspektor → richtige Interpretation wählen (z. B. `:float32:CDAB`).
OPC-DA nicht erreichbar	DCOM/Bitbreite/Windows	Bridge lokal auf DA-Server-Rechner; Wrapper-Bitbreite prüfen.
OPC-UA „BadSecurity-ChecksFailed”	Zertifikat nicht vertraut	Zertifikate gegenseitig vertrauen (Truststore) oder Policy anpassen.
Änderung wirkt nicht	nicht übernommen	„Speichern & übernehmen” klicken.
E-Mail/Webhook kommt nicht	SMTP/URL falsch	SMTP-Daten und Webhook-URL prüfen; Testknopf; Log beachten.
Live-Werte stehen still	WebSocket blockiert (Proxy)	Oberfläche fällt auf Polling zurück; Proxy/Firewall für WebSockets öffnen.
Sparkplug: Node fehlt	Host war vor NBIRTH offline	Im Host einen **Rebirth** anfordern (NCMD).

Symptom	Mögliche Ursache	Lösung
Bridge-Link verbindet nicht	Token/TLS/Port/Firewall	gleiches Token beidseitig; ausgehenden Port der OT-Seite freigeben; Zertifikat prüfen.
CAN keine Frames	Bitrate/Interface falsch	Bitrate an den Bus anpassen; SocketCAN-Interface up; Terminierung prüfen.
Demo stoppt nach 60 min	keine Lizenz	Lizenz aktivieren (Kapitel 39).
Dienst startet nicht (Linux)	Startfehler/ungültige config.json	`journalctl -u opcbridge -f`; config.json aus Backup.

Teil VII – Anwendungsbeispiele

Die folgenden Beispiele zeigen vollständige Konfigurationen „von A bis Z”. Sie lassen sich als Vorlage für eigene Anlagen verwenden.

45. Beispiel: OPC-DA → OPC-UA

Ziel: Eine alte OPC-DA-Steuerung soll ihre Werte einem modernen OPC-UA-Leitsystem bereitstellen.

1. **Quelle anlegen:** Verbindung *OPC-DA Client*, ProgID des DA-Servers eintragen (Bridge auf demselben Rechner betreiben).
2. **Ziel anlegen:** Verbindung *OPC-UA Server*, Port 4840, Security `Basic256Sha256` `SignAndEncrypt`.
3. **Punkte browsen:** In der OPC-DA-Verbindung die gewünschten Items durchsuchen.
4. **Mappings:** Je Item ein Mapping *DA.Item* → *UA-Pfad* (z. B. `Anlage/Linie1/Temperatur`), Richtung A→B. Mit dem Assistenten mehrere auf einmal.
5. **Speichern & übernehmen.** Das Leitsystem verbindet sich nun als UA-Client mit `opc.tcp://<bridge>:4840/opcbridge`.

---

46. Beispiel: Modbus → MQTT-Cloud

Ziel: Ein Modbus-Energiezähler soll seine Messwerte in eine MQTT-Cloud senden.

1. **Quelle:** Verbindung *Modbus TCP Master*, Host/IP des Zählers, Port 502.
2. **Werte identifizieren:** Mit dem **Register-Inspektor** Datentyp und Byte-order der Register bestimmen (z. B. `HR100:float32:CDAB` = Wirkleistung).
3. **Ziel:** Verbindung *MQTT*, Broker-Adresse, TLS ein, Benutzer/Passwort, Topic-Präfix `werk/zaehler1/`.
4. **Mappings:** `HR100:float32:CDAB` → `leistung`, `HR102:float32:CDAB` → `energie` usw.
5. **Store-and-Forward** an der MQTT-Verbindung aktivieren (WAN kann abreißen).
6. **Speichern & übernehmen.**

---

47. Beispiel: CAN → OPC-UA

Ziel: Die Drehzahl und Temperatur eines J1939-Dieselaggregats als OPC-UA bereitstellen.

1. **Quelle:** Verbindung *CAN*, Schnittstelle `socketcan`, Kanal `can0`, Bitrate 250000.
2. **Analyser:** Frames mitlesen; J1939-PGN identifizieren (z. B. PGN 61444 für Drehzahl).
3. **Signale anlegen:** `Drehzahl` (Start-Bit/Länge/Faktor 0.125) und `Kuehlmitteltemp` (Faktor 1, Offset −40) – oder per DBC-Import.
4. **Ziel:** Verbindung *OPC-UA Server*.
5. **Mappings:** `Drehzahl → Aggregat/Motor/Drehzahl`, `Kuehlmitteltemp → Aggregat/Motor/Temp`.
6. **Speichern & übernehmen.**

---

48. Beispiel: OT→IT über die DMZ

Ziel: Prozesswerte aus der Produktion (OT) in die IT-Zone bringen, ohne die OT-Firewall für eingehende Verbindungen zu öffnen.

OT-Seite (Produktionsnetz):

1. Bridge installieren; die Prozessquellen anbinden (Modbus/OPC-UA/…).
2. Verbindung *Bridge-Link*, Rolle **client**, Server-URL `wss://dmz-host:8765`, gemeinsames **Token** vergeben, Store-and-Forward aktivieren.
3. Mappings: Prozesspunkte → Bridge-Link-Tags (gleiche Namen wie auf der IT-Seite).

IT-/DMZ-Seite:

1. Bridge installieren.
2. Verbindung *Bridge-Link*, Rolle **server**, Port 8765, TLS mit Zertifikat, **gleiches Token**.
3. Mappings: Bridge-Link-Tags → Ziel (OPC-UA-Server fürs Leitsystem, Datenbank, MQTT …).

Ergebnis: Die OT-Seite baut die Verbindung ausschließlich **ausgehend** auf. In der OT-Firewall ist **kein eingehender Port** nötig. Werte fließen gespiegelt in die IT; bei Abriss puffert die OT-Seite.

Teil VIII – Anhang

A. Adressformate (vollständig)

Verbindung	Format / Beispiele
OPC-DA	`Channel1.Device1.Tag1`
OPC-UA (Client)	`ns=2;s=Linie1.Temp`, `ns=3;i=1001`
OPC-UA (Server)	Pfad `Anlage/Linie1/Temperatur` → NodeId `ns=<idx>;s=…`
Siemens S7	`DB1.DBX0.0` (Bool), `DB1.DBW4` (Int), `DB10.DBD20:real`, `DB1.DBD0:dint`, `M10.0`, `MW20`, `Q0.1`; Typ-Suffix `:bool :int :dint :real :word :dword :string<n>`
Modbus	`[Unit:]Bereich Offset[.Bit][:Typ[:Byteorder]]` — `HR100:float32`, `2:IR10:int32:CDAB`, `CO5`, `HR7.3`, `40001:uint16`, `HR200:string20`
MQTT	`halle1/temp`; JSON-Pfad `sensor/abc@data.temperature`
Sparkplug B	Metrikname, z. B. `Temperatur`, `Linie3/Stueckzahl`
SQL	Spalten-/Schlüsselname; Schreibanweisung mit `:addr`, `:value`, `:ts`
InfluxDB	Tag-Name (wird als Tag an die Messreihe gehängt)
CAN	Signalname aus der Signalliste (Analyser/DBC)
Webhook / Bridge-Link	frei gewählter Tag-Name

Modbus-Bereiche: `CO` (Coils), `DI` (Discrete Inputs), `IR` (Input Registers), `HR` (Holding Registers); 0-basiert oder Modicon-Notation (`0xxxx`/`1xxxx`/`3xxxx`/`4xxxx`). **Byteorder:** `ABCD`, `CDAB`, `BADC`, `DCBA`.

B. Formel- und Funktionsreferenz

Die Variable **x** ist der eingehende Wert; bei berechneten Punkten stehen die Eingänge als benannte Variablen zur Verfügung.

Kategorie	Funktionen / Operatoren
Grundrechnen	`+`, `-`, `*`, `/`, `%` (Modulo), `**` (Potenz)
Vergleich/Logik	`>`, `<`, `>=`, `<=`, `==`, `!=`, `and`, `or`, `not`
Runden	`round(x, n)`, `floor(x)`, `ceil(x)`, `int(x)`, `abs(x)`
Begrenzen/Skalieren	`clamp(x, min, max)`, `scale(x, in_min, in_max, out_min, out_max)`
Min/Max	`min(a, b, …)`, `max(a, b, …)`
Mathematik	`sqrt(x)`, `pow(x, y)`, `log(x)`, `exp(x)`, `sin`, `cos`, `tan`
Bedingung	`x if Bedingung else y`

Beispiele: `x*1.8+32`, `scale(x,0,27648,0,100)`, `clamp(x,0,100)`, `round(x,1)`, `x if x>0 else 0`.

> **»Hinweis«** Der Formel-Interpreter ist aus Sicherheitsgründen auf mathematische Ausdrücke beschränkt (kein Dateizugriff, keine Systemaufrufe).

C. REST-Endpunkt-Referenz

Basis-URL: `http://<host>:8080`. Interaktive Doku: `/api/docs`.

Methode	Pfad	Zweck
GET	`/api/authinfo`	Auth-Modus abfragen (öffentlich)
POST	`/api/login`	Anmelden (optional `code` für 2FA), Token erhalten
POST	`/api/token`	OAuth2-Token (Formular)
POST	`/api/logout`	Token zurückziehen
GET/POST	`/api/2fa/status`, `/api/2fa/setup`, `/api/2fa/enable`, `/api/2fa/disable`	Zwei-Faktor verwalten
GET	`/api/oidc/start`, `/api/oidc/callback`	OIDC-/SSO-Anmeldung
GET	`/api/meta`	Produkt-/Versionsinfo
GET	`/api/status`	Status aller Verbindungen/Mappings
GET	`/api/values`	aktuelle Werte
GET	`/api/history`	Historiendaten (JSON)
GET	`/api/history.csv`	Historie als CSV
GET	`/api/events`	OPC-UA-Ereignisse
GET	`/api/notifications`	Alarm-Ereignisse
POST	`/api/notifications/test`	Testbenachrichtigung senden
GET/PUT	`/api/config`	Konfiguration lesen/schreiben (PUT: Admin)
POST	`/api/restart`	Engine neu starten (Admin)
POST	`/api/browse`	Adressbaum durchsuchen
POST	`/api/write`	Wert schreiben (Admin)
POST	`/api/inspect`	Modbus-Register-Inspektor
POST	`/api/can/analyze`	CAN-Analyse

Methode	Pfad	Zweck
POST	`/api/can/dbc`	DBC-Import (Admin)
GET/POST/DELETE	`/api/license`	Lizenzstatus / aktivieren / entfernen
GET	`/api/wsticket`	Ticket für WebSocket
WS	`/ws`	Echtzeit-Datenstrom
GET	`/api/docs`, `/api/redoc`, `/api/openapi.json`	API-Dokumentation
GET	`/view`, `/api/view`	WebView (token-geschützt)

Schreibende Endpunkte erfordern die Rolle **Administrator**.

D. Datentyp-Referenz

Typ	Beschreibung	Wertebereich	Modbus-Register
`bool`	Wahrheitswert	true/false	1 Bit (CO/DI) oder Bit eines Registers
`int16`	16-Bit mit Vorzeichen	−32768…32767	1
`uint16`	16-Bit ohne Vorzeichen	0…65535	1
`int32`	32-Bit mit Vorzeichen	±2,1 Mrd.	2
`uint32`	32-Bit ohne Vorzeichen	0…4,29 Mrd.	2
`int64`	64-Bit mit Vorzeichen	sehr groß	4
`float32`	32-Bit Gleitkomma (IEEE 754)	ca. $\pm 3{,}4 \cdot 10^{38}$	2
`double`	64-Bit Gleitkomma	hohe Präzision	4
`string`	Text	je nach Länge	n (2 Zeichen/Register)

E. Standard-Ports

Dienst	Port	Hinweis
Webinterface / REST	8080	konfigurierbar
OPC-UA-Server	4840	wenn aktiviert
Modbus-Slave	502	wenn aktiviert
MQTT (Broker, Gegenstelle)	1883 / 8883	8883 = TLS
Bridge-Link	8765	DMZ-Server, konfigurierbar
InfluxDB (Gegenstelle)	8086	v1/v2/v3
Siemens S7 (Gegenstelle)	102	RFC1006/ISO-on-TCP
CAN-over-Ethernet (UDP)	43113	bei `udp_multicast`

F. Datei- und Verzeichnisorte

Plattform	Datenverzeichnis	Erststart-Passwort
Windows	`%ProgramData%\OPCBridge\`	`…\initial_admin_password.txt`
Linux (.deb)	`/var/lib/opcbridge/`	`…/initial_admin_password.txt`
Docker	Volume `./data`	`./data/initial_admin_password.t`

Im Datenverzeichnis liegen u. a.: `config.json` (Konfiguration), `license.key` (Lizenz), die Historien-SQLite-Datei und die persistente Store-and-Forward-Warteschlange.

G. Inbetriebnahme-Checkliste

- ☐ Bridge installiert, Dienst läuft, Weboberfläche erreichbar
- ☐ Erststart-Passwort geändert, ggf. Operator-Benutzer angelegt
- ☐ API-Authentifizierung passend gesetzt (Klartext/Basic/Token)
- ☐ HTTPS aktiviert (Produktivbetrieb)
- ☐ Quell-Verbindung(en) angelegt und **grün**
- ☐ Ziel-Verbindung(en) angelegt und **grün**
- ☐ Bei Modbus: Datentyp/Byteorder per Register-Inspektor geprüft
- ☐ Mappings angelegt, Richtung korrekt
- ☐ „Speichern & übernehmen” geklickt
- ☐ Live-Werte kontrolliert (kommen Werte an?)
- ☐ Store-and-Forward an WAN-/Cloud-Zielen aktiviert
- ☐ Historie konfiguriert (falls benötigt)
- ☐ Alarme eingerichtet und getestet (E-Mail/Webhook)
- ☐ Lizenz aktiviert (Dauerbetrieb)
- ☐ Backup des Datenverzeichnisses erstellt

H. Glossar

Begriff	Bedeutung
A&E	Alarms & Events – OPC-UA-Ereignismeldungen
Byteorder	Reihenfolge der Bytes bei mehrteiligen Zahlen (ABCD/CDAB/BADC/DCBA)
DBC	Beschreibungsdatei für CAN-Signale
DMZ	abgeschottete Netzzone zwischen OT und IT
Edge-Node	Datenquelle am Rand der Anlage (Sparkplug)
Engine	interner Kern, der Mappings ausführt
Mapping	Regel „Wert A → Wert B”
Master/Slave	Modbus-Rollen (abfragen / bereitstellen)
OT/IT	Produktionsnetz / Unternehmensnetz
PGN/SA	Parameter Group Number / Source Address (J1939)
Publish/Subscribe	MQTT-Prinzip (veröffentlichen/abonnieren)
Qualität	Güte eines Werts (good/uncertain/bad)
Store-and-Forward	Puffern bei Ausfall eines Ziels
Tag/Punkt	einzelner Wert
UNS	Unified Namespace (einheitliche Datenstruktur über Sparkplug)

I. Häufige Fragen (FAQ)

Muss der Browser offen bleiben, damit Daten fließen? Nein. Die Bridge läuft als Dienst im Hintergrund. Der Browser ist nur zur Bedienung.

Kann ein Wert in beide Richtungen gehen? Ja, mit Richtung A↔B. Der zuletzt geänderte Wert wird übertragen.

Was passiert bei Verbindungsabbruch zum Ziel? Ohne Store-and-Forward gehen Werte verloren; mit S&F werden sie gepuffert und nachgeliefert.

Warum zeigt mein Modbus-Wert Unsinn? Fast immer falscher Datentyp oder falsche Byteorder – mit dem Register-Inspektor prüfen.

Brauche ich Internet für die API-Dokumentation? Nein. Die Dokumentation ist vollständig mitgeliefert und funktioniert offline.

Läuft OPC-DA unter Linux? Nein, OPC-DA benötigt Windows. Alle anderen Protokolle laufen plattformübergreifend.

Wie bringe ich Werte sicher aus der Produktion in die IT? Mit dem Bridge-Link im DMZ-Modus – nur ausgehende Verbindung, kein eingehender OT-Port.

Wie sichere ich die Konfiguration? Datenverzeichnis sichern (mindestens `config.json`), siehe Kapitel 41.

OPC Bridge – Technisches Betriebshandbuch Version 1.0.0 · © 2026 **Biricon IT Services GmbH** · www.biricon.eu

Alle Marken sind Eigentum ihrer jeweiligen Inhaber. Technische Änderungen vorbehalten.