OPC Bridge – Technisches Betriebshandbuch

- Inhaltsverzeichnis
- Teil I – Grundlagen
  - 1. Einführung und Überblick
  - 2. Systemarchitektur
  - 3. Grundbegriffe
  - 4. Protokoll-Grundlagen im Detail
- Teil II – Installation und Inbetriebnahme
  - 5. Systemvoraussetzungen
  - 6. Installation unter Windows
  - 7. Installation unter Debian/Ubuntu (x86-64)
  - 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 Siemens S7
  - 21. Verbindungstyp MQTT
  - 22. Verbindungstyp Sparkplug B
  - 23. Verbindungstyp CAN
  - 24. Verbindungstyp SQL
  - 25. Verbindungstyp InfluxDB
  - 26. Verbindungstyp Webhook
  - 27. Verbindungstyp Bridge-Link
  - 28. Mappings und Datenfluss
  - 29. Umrechnung und Formeln
  - 30. Virtuelle Quellen und Redundanz
  - 31. Store-and-Forward
  - 32. Historie und Trends konfigurieren
  - 33. Alarme, E-Mail und Webhook
  - 34. CAN-Analyser und Gateway
  - 35. Modbus-Register-Inspektor
  - 36. WebView

        - 37. Benutzer, Rollen und API-Authentifizierung
        - 38. Echtzeit-Aktualisierung (WebSocket)
    - Teil V – Schnittstellen und Integration
        - 39. REST-API und eingebaute Dokumentation
        - 40. Konfigurationsdatei (config.json)
    - Teil VI – Betrieb
        - 41. Lizenzierung
        - 42. Sicherheit und Härtung
        - 43. Datensicherung und Wiederherstellung
        - 44. Update und Migration
        - 45. Leistung und Dimensionierung
        - 46. Fehlersuche
    - Teil VII – Anwendungsbeispiele
        - 47. Beispiel: OPC-DA → OPC-UA
        - 48. Beispiel: Modbus → MQTT-Cloud
        - 49. Beispiel: CAN → OPC-UA
        - 50. 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)

Biricon

OPC Bridge

Technisches Betriebshandbuch

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

Produktversion **1.1.0** · 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 (x86-64) 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 Siemens S7 21. Verbindungstyp MQTT 22. Verbindungstyp Sparkplug B 23. Verbindungstyp CAN 24. Verbindungstyp SQL 25. Verbindungstyp InfluxDB 26. Verbindungstyp Webhook 27. Verbindungstyp Bridge-Link 28. Mappings und Datenfluss 29. Umrechnung und Formeln 30. Virtuelle Quellen und Redundanz 31. Store-and-Forward 32. Historie und Trends konfigurieren 33. Alarme, E-Mail und Webhook 34. CAN-Analyser und Gateway 35. Modbus-Register-Inspektor 36. WebView 37. Benutzer, Rollen und API-Authentifizierung 38. Echtzeit-Aktualisierung (WebSocket)

Teil V – Schnittstellen und Integration 39. REST-API und eingebaute Dokumentation 40. Konfigurationsdatei (config.json)

Teil VI – Betrieb 41. Lizenzierung 42. Sicherheit und Härtung 43. Datensicherung und Wiederherstellung 44. Update und Migration 45. Leistung und Dimensionierung 46. Fehlersuche

Teil VII – Anwendungsbeispiele 47. Beispiel: OPC-DA → OPC-UA 48. Beispiel: Modbus → MQTT-Cloud 49. Beispiel: CAN → OPC-UA 50. 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

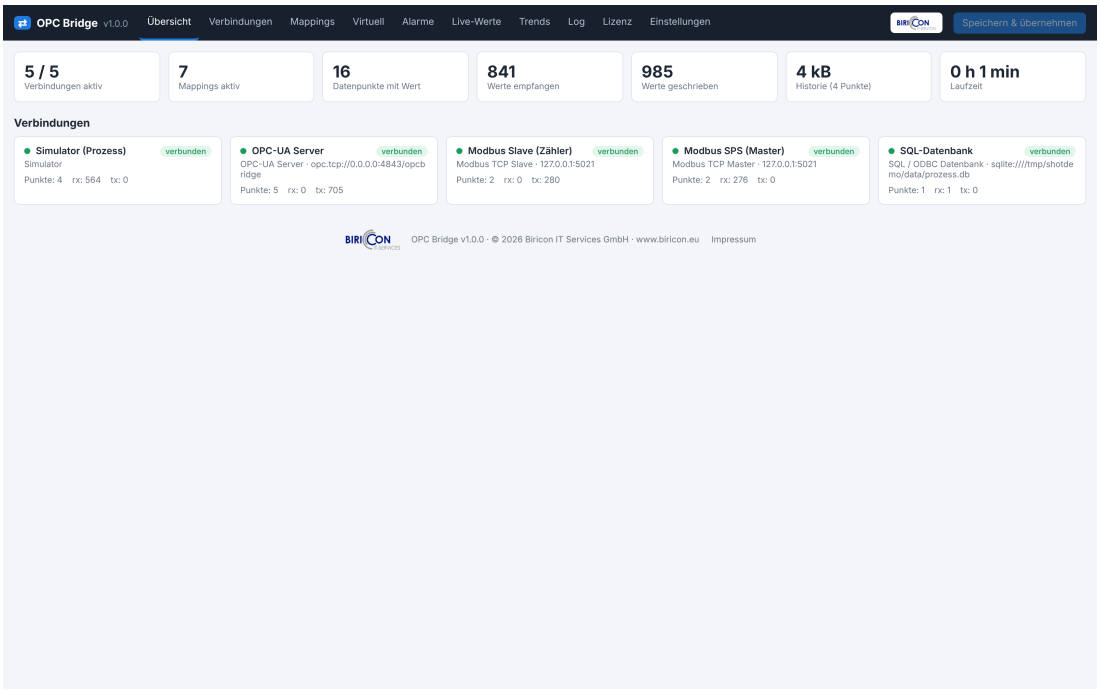
1. Einführung und Überblick

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**.



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
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	

Bereich	Funktionen
	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, offline) **und Online-Lizenz** (ES256) mit automatischer Aktivierung, Gerätegrenze und **Klon-Erkennung**; Demo-Modus
Updates	**Update-Anzeige** und **Ein-Klick-Installation** über signierte Versionsdatei
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	

Begriff	Bedeutung
	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.
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 Netzen)
`Basic256Sha256`	Signatur/Verschlüsselung (gängiger Standard)
`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]
[Anzahl=00 02]
```

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-Integer** (`int64` / `uint64`) oder **64-Bit-Gleitkomma** (`float64`): 4 Register.
- **Text** (`stringN`): mehrere Register, je 2 Zeichen.

Dabei ist die **Byte-Reihenfolge (Byteorder)** entscheidend – siehe Abschnitt 4.12 und den **Register-Inspektor** (Kapitel 35), 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/temperatur`. 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`.

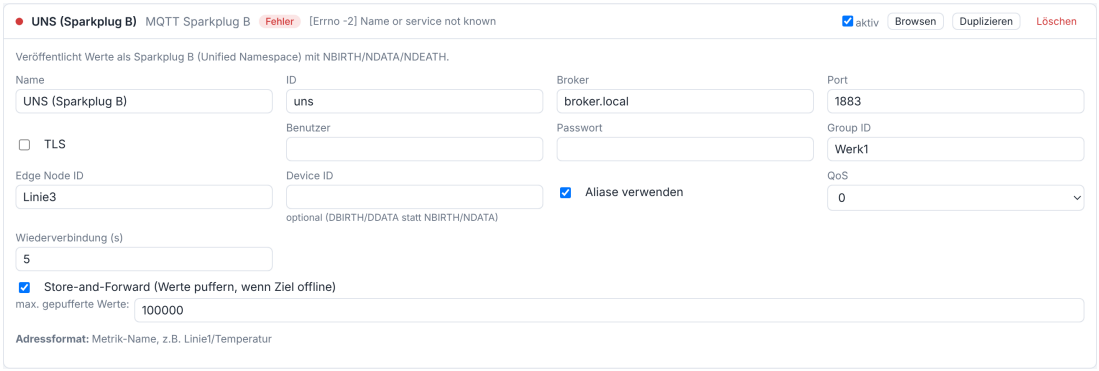
Lebenszyklus (das Herzstück).

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).



Sparkplug B

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

4.7 CAN-Bus

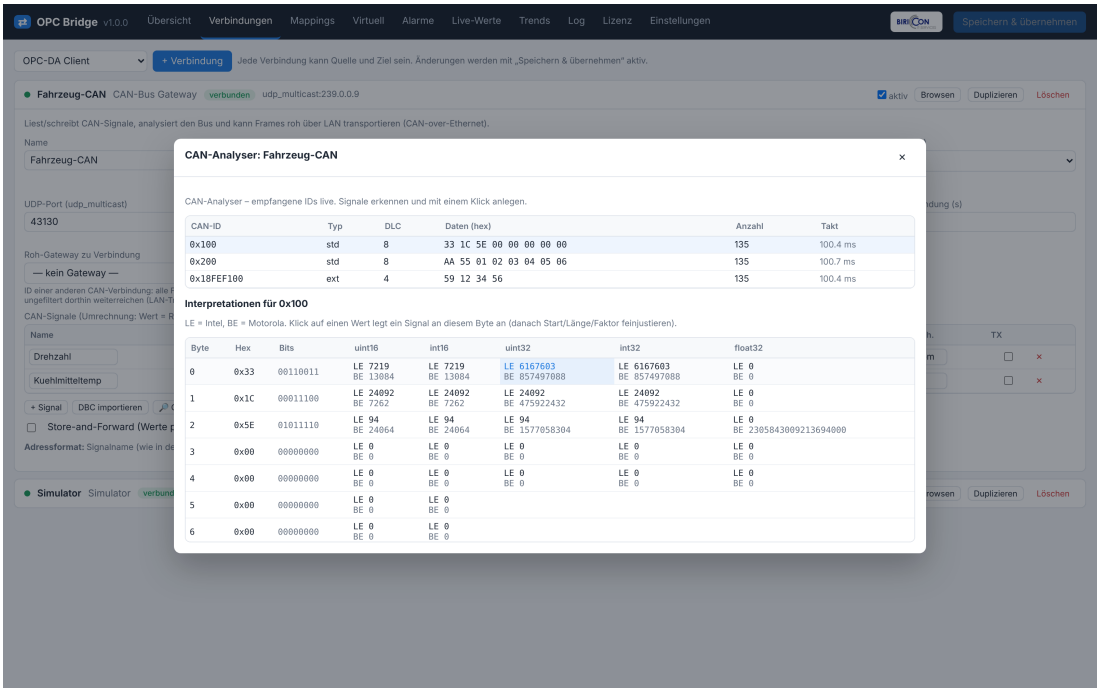
Herkunft und Zweck. CAN (*Controller Area Network*, Bosch, 1986) ist der dominierende Bus in Fahrzeugen, Baumaschinen, Aggregaten, Medizin- und Industriegeräten. Er ist robust, echtzeitfähig und benötigt nur zwei Drähte. Auf dem Bus laufen viele kleine Botschaften (**Frames**), die alle Teilnehmer mithören.

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).



CAN-Analyser

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 16-Bit-Rohwert mit Faktor 0,1 und Offset −40 ergibt bei Rohwert 850 den Wert `850*0,1−40 = 45,0 °C`.

Schnittstellen. Die Bridge spricht CAN über verschiedene Hardware/Transporte: **SocketCAN** (Linux), **UDP-Multicast** (CAN-over-Ethernet), **slcan** (serielle CAN-Adapter), **PCAN** (Peak), **Kvaser**, **Vector** sowie einen **virtuellen** Bus zum Testen.

Zwei Betriebsarten.

- **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
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.

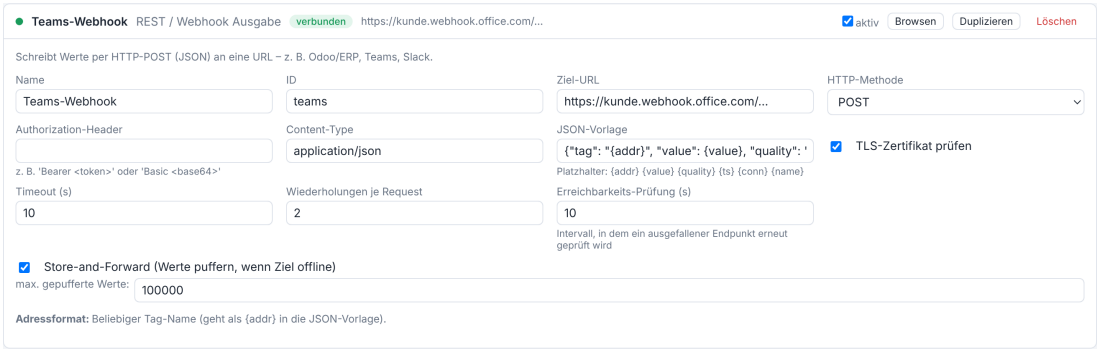
Aufbau. Für jeden weiterzuleitenden Wert sendet die Bridge eine **HTTP-POST-Anfrage** an eine konfigurierte URL. Der Nachrichtenkörper ist ein frei definierbares **JSON-Template** mit Platzhaltern:

Platzhalter	Inhalt
`{addr}`	Adresse des Punkts
`{value}`	Wert (als JSON eingesetzt – Zahl bleibt Zahl)
`{quality}`	Qualität
`{ts}`	Zeitstempel (ISO-8601)
`{conn}`	**Name der Bridge-Verbindung** (nicht die Adresse!)
`{name}`	Adresse des Punkts (identisch mit `{addr}`)

> »**Achtung**« `{conn}` liefert den **Verbindungsnamen**, `{name}` dagegen – trotz des Namens – dieselbe **Adresse** wie `{addr}`. Das ist auf den ersten Blick nicht intuitiv, entspricht aber dem tatsächlichen Verhalten des Treibers.

Standard-Vorlage (`DEFAULT_TEMPLATE`): `{"tag": "{addr}", "value": {value}, "quality": "{quality}", "timestamp": "{ts}"}`.

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



Webhook-Ausgang mit Store-and-Forward

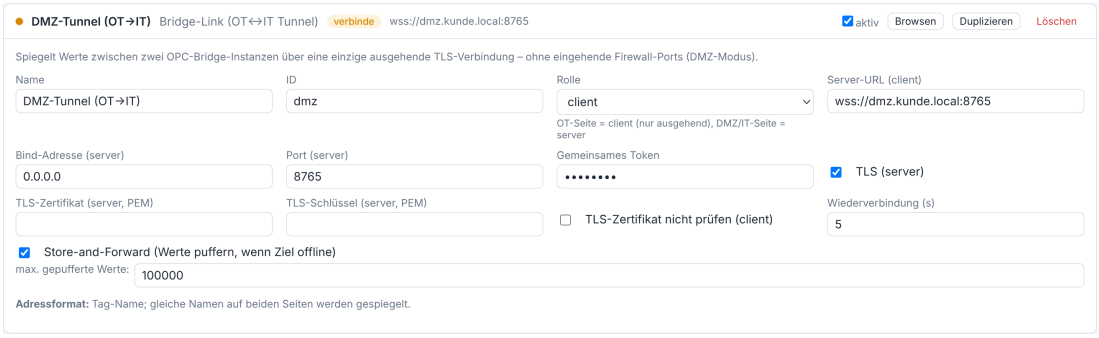
4.11 Bridge-Link (OT→IT-Tunnel)

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.



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` / `uint64`	64-Bit ganzzahlig (mit/ohne Vorzeichen)	sehr groß

Typ	Beschreibung	Wertebereich
`float32`	32-Bit Gleitkomma (IEEE 754)	ca. ±3,4·10³⁸
`float64`	64-Bit Gleitkomma (IEEE 754)	sehr hohe Präzision
`stringN`	Text (N Zeichen)	je nach Länge

> »**Hinweis**« `double` ist **kein** gültiges Kürzel in der OPC Bridge – das 64-Bit-Gleitkomma- Format heißt in Adressen und Mappings durchgängig `float64` .

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`	Bytes innerhalb jedes Registers vertauscht	**Byte-Tausch** (Byte-Swap)
`CDAB`	die beiden Register vertauscht	**Wort-Tausch** (Word-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 35): 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 (amd64)	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+ (jeweils x86-64)	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.

> »**Hinweis**« Die ausgelieferten Pakete (Windows-Setup, Debian-Paket, Docker-Image) werden derzeit **ausschließlich für x86-64 (amd64)** gebaut. Es gibt **noch keinen** fertigen Raspberry-Pi-/ARM-Build (weder `.deb` noch Docker-Image) als Teil der Auslieferung. Die OPC Bridge ist aus dem Quelltext heraus auch unter Debian/Ubuntu auf ARM (z. B. Raspberry Pi) lauffähig, wenn sie selbst für die passende Architektur gebaut wird – das ist aber Eigenbau und nicht Teil des offiziellen Pakets.

---

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 (x86-64)

> »**Hinweis**« Das Debian-Paket wird derzeit nur für **x86-64 (amd64)** gebaut. Ein Raspberry-Pi-/ARM-Paket ist **nicht** Teil der Auslieferung (siehe Kapitel 5).

Installation:

```
sudo apt install ./opcbridge_1.1.0_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 und legt das persistente, **benannte** Docker-Volume `opcbridge-data` an (auf `/data` im Container). Dort liegen `config.json`, `license.key` und die Historie.

Container-Port	Host-Port	Dienst
8080	**8080**	Webinterface / REST-API
4840	**4840**	OPC-UA-Server (optional)
502	**1502**	Modbus-Slave (optional)
8765	**8765**	Bridge-Link (optional, DMZ-Server)

> »**Achtung**« Der Modbus-Slave ist im Container intern auf Port 502, wird aber standardmäßig auf den **Host-Port 1502** veröffentlicht (`"1502:502"` in `docker-compose.yml`), nicht auf 502. Ein Master, der sich von außen auf den Docker-Host verbindet, muss daher Port **1502** verwenden (oder die Portzuordnung in `docker-compose.yml` anpassen).
>
> »**Hinweis**« Ein Bind-Mount (`./data:/data`) legt das Verzeichnis beim ersten Start als `root` an; der Container läuft aber unter einer eigenen UID (10001) und kann dann nicht schreiben. Das mitgelieferte benannte Volume vermeidet dieses Problem, da es Inhalt und Berechtigungen aus dem Image übernimmt. Wer dennoch einen Bind-Mount möchte, muss das Verzeichnis vorher selbst anlegen und chownen: `mkdir data && sudo chown 10001:10001 data` .

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 – ganz ohne reale Hardware, nur mit dem eingebauten **Simulator**.

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 **Simulator** wählen → **+ Verbindung** klicken. Der Simulator erzeugt ohne jede Hardware laufend Testwerte (u. a. eine Sinuskurve `Sine`, einen Zähler `Counter` und einen Schalter `Toggle`) und eignet sich damit ideal, um die Bedienung ohne reale Anlage zu erproben.
4. **Erstes Ziel anlegen.** Zweite Verbindung anlegen, z. B. *OPC-UA-Server* (Standard-Port
    1. oder *Modbus TCP Slave*.
5. **Mapping anlegen.** Reiter *Mappings* → **+ Mapping** klicken → Quelle (A) = Simulator + Adresse `Sine`, Ziel (B) = die zweite Verbindung + Zieladresse → 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?

> »**Tipp**« Im Repository liegt `config.example.json` als fertiges, sofort lauffähiges Beispiel mit Simulator, OPC-UA-Server, Modbus-Slave/-Master und mehreren Mappings. Sie können diese Datei als Ausgangspunkt über `PUT /api/config` einspielen oder – bei gestopptem Dienst – als `config.json` ins Datenverzeichnis kopieren, um die Bedienung an einem bereits funktionierenden Beispiel nachzuvollziehen, statt bei Null zu beginnen.

> »**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.

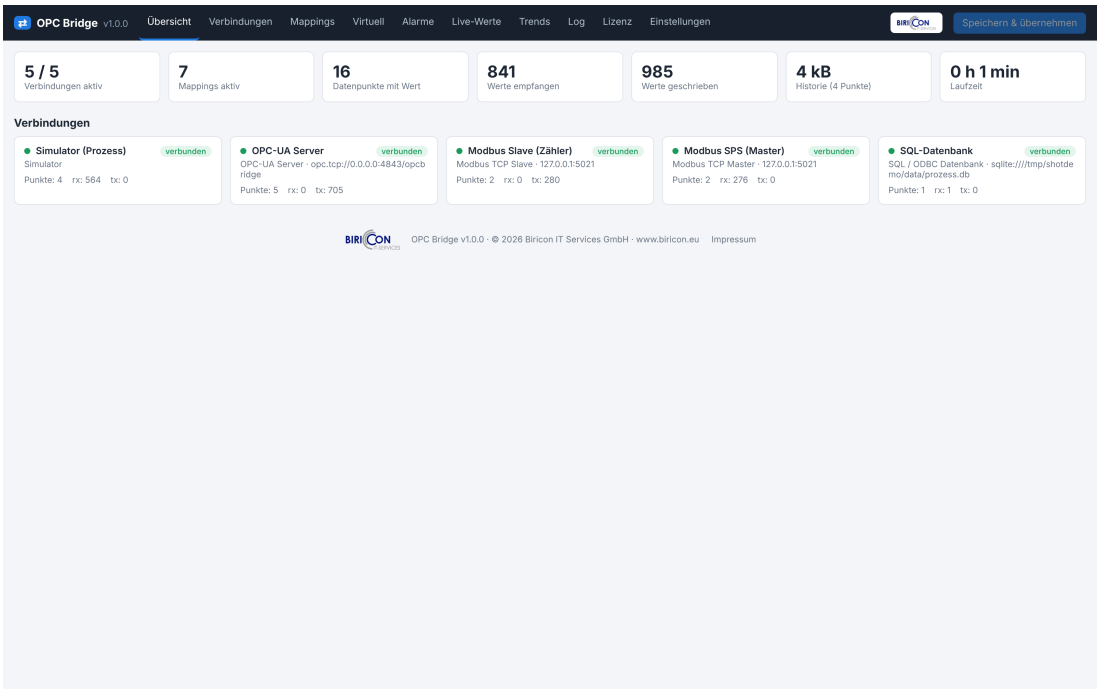
Reiter	Zweck	Kapitel
Übersicht	Statusüberblick aller Verbindungen und Mappings	11
Verbindungen	Geräte/Systeme anlegen und konfigurieren	15–27
Mappings	Werte einander zuordnen	28
Virtuell	Berechnete Punkte und Redundanz	30
Alarme	Benachrichtigungsregeln, OPC-UA-Events, **SMTP (E-Mail-Versand)**	33
Live-Werte	Aktuelle Werte, manuelles Schreiben	12
Trends	Historische Werte grafisch	13
Log	Betriebsmeldungen	14
Lizenz	Lizenzstatus und Aktivierung (Offline- und Online-Lizenz)	41
Einstellungen	Webinterface, Benutzer/LDAP/OIDC/2FA, Historie, WebView, Online-Lizenz, Updates	32, 37

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:



Übersicht

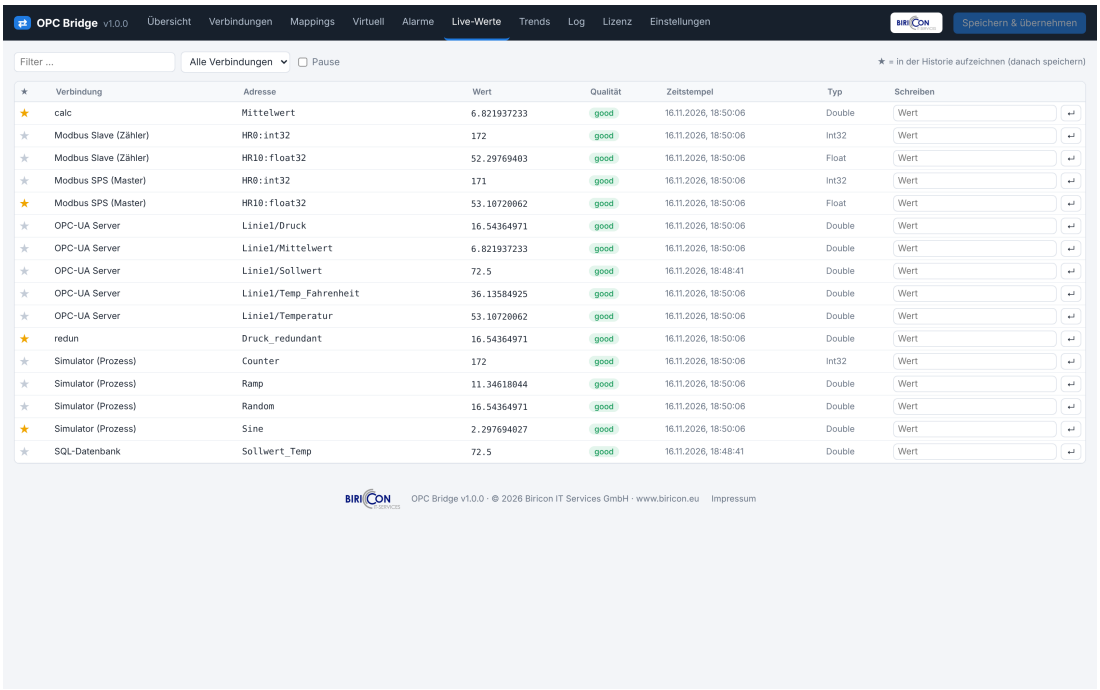
- **Verbindungen:** Anzahl und Zustand (verbunden/verbindet/Fehler/gestoppt). Jede Verbindung wird mit Farbe und Statusmeldung dargestellt.
- **Mappings:** Anzahl aktiver Zuordnungen.
- **Store-and-Forward:** gepufferte Werte je Zielverbindung (falls aktiv).
- **Lizenz:** aktueller Lizenzstatus (lizenziert/Demo/Restzeit).

Die Werte werden **in Echtzeit** aktualisiert (siehe Kapitel 38).

---

12. Reiter Live-Werte

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



Live-Werte

Funktionen:

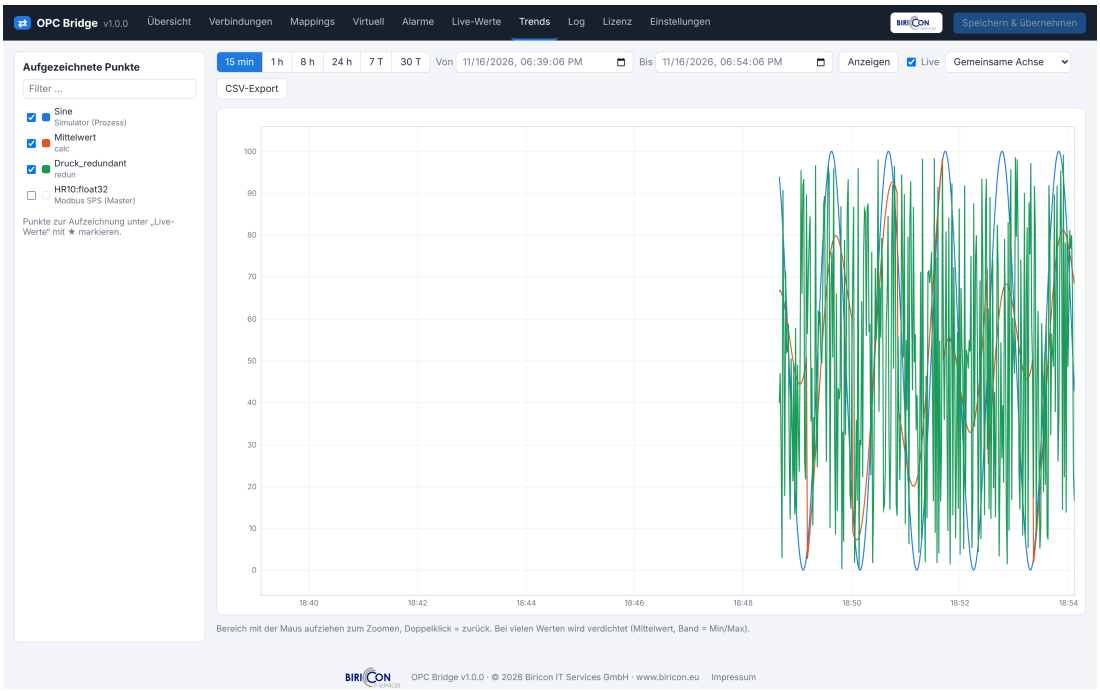
- **Pause:** friert die Anzeige ein (die Datenübertragung läuft im Hintergrund weiter).
- **Manuelles Schreiben:** Bei schreibbaren Zielpunkten können Sie zu Testzwecken einen Wert direkt eingeben und senden. Das ist bei der Inbetriebnahme sehr nützlich, um Aktoren oder Register gezielt zu prüfen.
-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



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 32).

---

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").

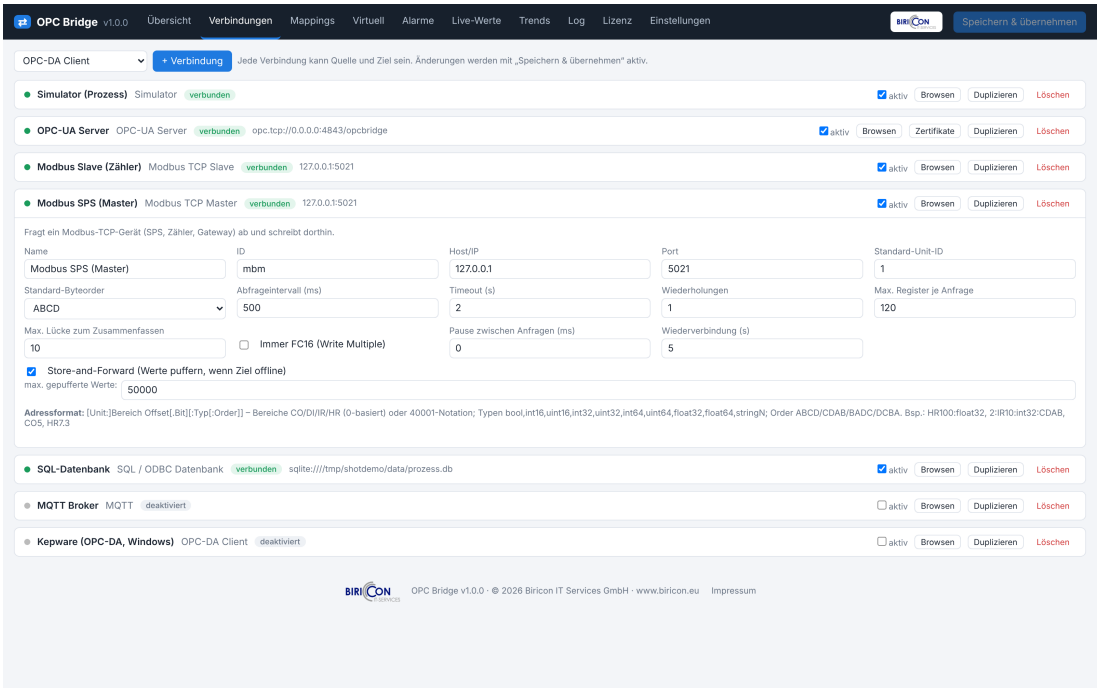
Auf Systemebene stehen die Meldungen zusätzlich in Protokolldateien im **Datenverzeichnis** (Anhang F) – es gibt **kein** Windows-Ereignisprotokoll:

- `opcbridge.log` – das eigentliche Anwendungslog der Bridge (rotierend, max. 5 MB je Datei, 5 Backups), auf allen Plattformen.
- `service.log` – unter Windows zusätzlich die Standardausgabe/-fehlerausgabe des Windows-Dienstes (umgeleitet über NSSM); nützlich, wenn der Dienst gar nicht erst startet.
- Unter Linux steht außerdem `journalctl -u opcbridge -f` zur Verfügung (systemd-Journal).

Teil IV – Konfiguration

15. Verbindungen – allgemein

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



Verbindungen

Neue Verbindung anlegen:

1. Oben den **Typ** (das Protokoll) aus der Liste wählen.
2. Auf **„+ Verbindung"** 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)
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)

Element	Funktion
Duplizieren	erstellt eine Kopie der Karte
Löschen	entfernt die Verbindung
Store-and-Forward	Pufferung aktivieren (Kapitel 31)

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		leer

Feld	Bedeutung	Standard
	optionaler Oberordner im Adressraum	
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

Feld	Bedeutung	Standard
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

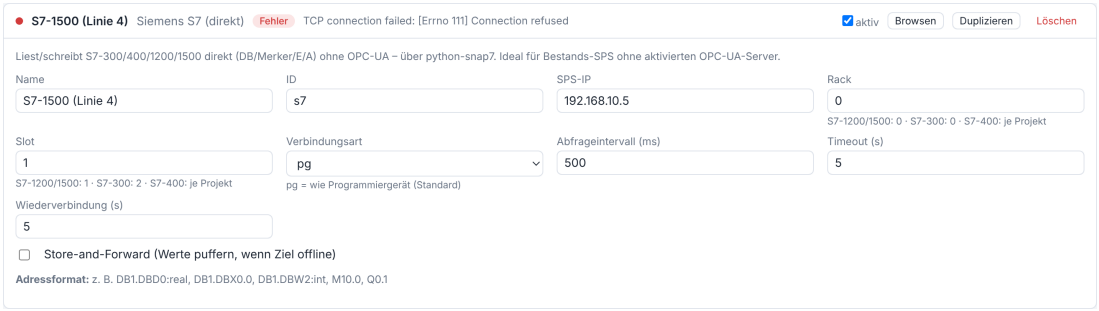
Wie TCP Slave, aber seriell (Felder *Serielle Schnittstelle*, *Baudrate*, *Parität*, *Datenbits*, *Stopbits* wie beim RTU Master).

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 Siemens S7

Rolle: Client. Spricht Siemens S7-300/400/1200/1500 **direkt** über das RFC1006/ISO-on-TCP-Protokoll an – ohne dass an der SPS OPC-UA aktiviert oder lizenziert sein muss. Das ist im Bestand oft der schnellste Weg, eine S7 anzubinden. Benötigt die Bibliothek **python-snap7** (im Paket/Image enthalten).



Siemens-S7-Verbindung

> **»Hinweis zum Bild«** Der Screenshot zeigt den Stand von Version 1.0; die Oberfläche zeigt inzwischen zusätzlich die Felder *Verbindungsart* und *Timeout* (siehe Tabelle unten).

Feld	Bedeutung	Standard
SPS-IP	Adresse der Steuerung	192.168.0.1
Rack	S7-1200/1500: 0 · S7-300: 0 · S7-400: je Projekt	0
Slot	S7-1200/1500: 1 · S7-300: 2 · S7-400: je Projekt	1
Verbindungsart	`pg` (wie Programmiergerät, Standard), `op` oder `basic`	pg
Abfrageintervall (ms)	wie oft gelesen wird	1000
Timeout (s)	Zeitüberschreitung	5
Wiederverbindung (s)	Wartezeit vor erneutem Versuch	5

Adressformat. Gelesen/geschrieben werden **Datenbausteine (DB)**, **Merker (M)**, **Eingänge (I/E)** und **Ausgänge (Q/A)**:

Adresse	Typ	Beispiel
`DB<n>.DBX<byte>.<bit>`	Bool	`DB1.DBX0.0`
`DB<n>.DBB<byte>`	Byte	`DB1.DBB2`
`DB<n>.DBW<byte>`	Int/Word	`DB1.DBW4`
`DB<n>.DBD<byte>`	**ohne Suffix: Real** (sonst `:dint` / `:udint` / `:dword`)	`DB10.DBD20` → Real; `DB1.DBD0:dint` → DInt
`M<byte>.<bit>` / `MB` / `MW` / `MD`	Merker	`M10.0`, `MW20`
`I<byte>.<bit>` (E) / `Q<byte>.<bit>` (A)	Eingänge/Ausgänge	`Q0.1`

Optionaler **Typ-Suffix**: `:bool :byte :int :uint :word :dint :udint :dword :real :string<n>`. Beispiele: `DB10.DBD20:real`, `DB10.DBW0:int`, `M10.0`, `DB1.DBD0:dint`, `DB5.DBB0:string16`.

> **»Achtung«** `DBD` ohne Typ-Suffix wird als **Real** (32-Bit-Gleitkomma) interpretiert. Für ein 32-Bit-Integer auf einer `DBD`-Adresse ist der Suffix `:dint` bzw. `:udint` zwingend anzugeben – sonst entsteht ein falscher Wert.

---

21. 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`.

---

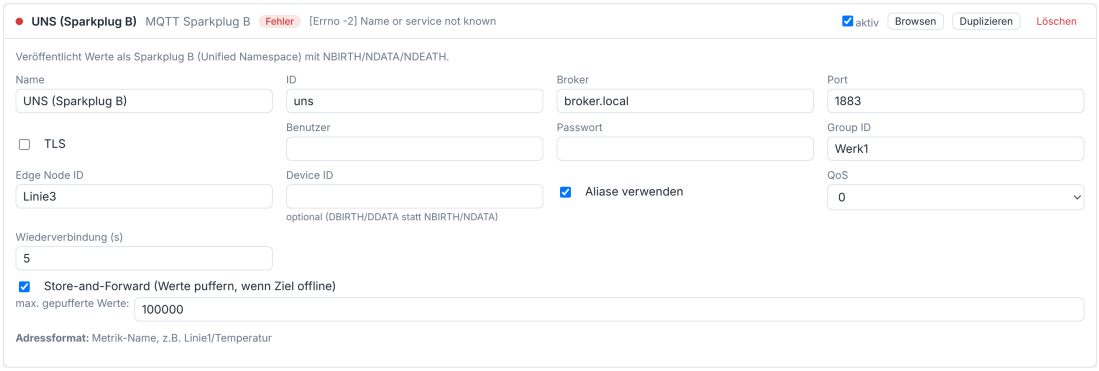
22. Verbindungstyp Sparkplug B

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

Feld	Bedeutung	Standard
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`.



Sparkplug B

---

23. 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

Feld	Bedeutung	Standard
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 34.

---

24. Verbindungstyp SQL

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

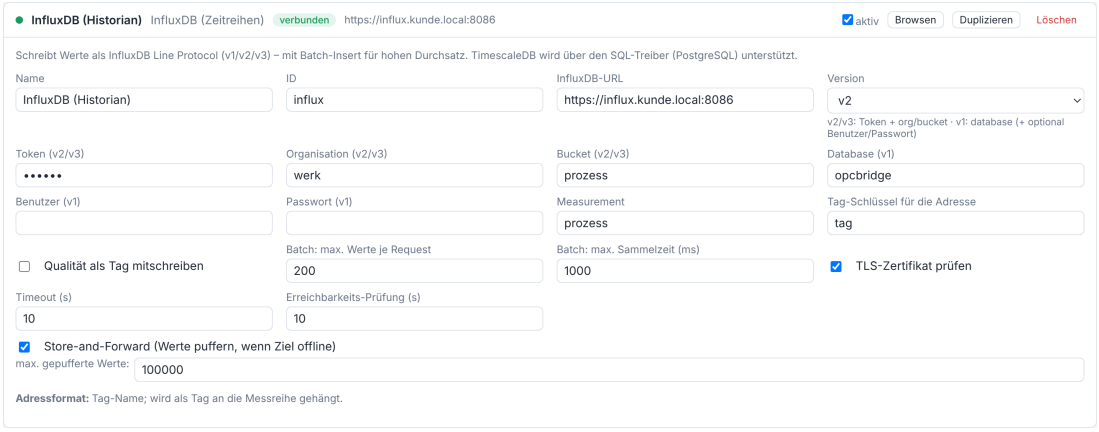
Feld	Bedeutung	Standard
Verbindungs-URL (SQLAlchemy)	z. B. `postgresql://user:pw@host/db`, `mssql+pyodbc://…`, `sqlite:///pfad.db`	`sqlite:///opcbridge_data.db`
Abfrageintervall (ms)	wie oft gelesen wird	2000
Lesemodus	`keyvalue` (Zeilen mit addr/value), `columns` (Spalten als Punkte), `off`	keyvalue
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
Batch: max. Zeilen je Insert	0 = einzeln; >1 schreibt mehrere Zeilen als **Multi-Row-Insert** in einer Transaktion	0
Batch: max. Sammelzeit (ms)	spätestens nach dieser Zeit wird auch ein unvollständiges Batch geschrieben	500
Verbindung erneuern nach (s)	Pool-Recycling	1800

Feld	Bedeutung	Standard
Wiederverbindung (s)	Wartezeit vor erneutem Versuch	5

---

25. Verbindungstyp InfluxDB

Rolle: Ausgabetreiber (Senke). Schreibt Werte als **InfluxDB Line Protocol** per HTTP – für InfluxDB **v2/v3** (Token + Organisation + Bucket) und **v1** (Database, optional Benutzer/Passwort). Für **TimescaleDB** nutzen Sie stattdessen den SQL-Treiber (Kapitel 24) mit einer PostgreSQL-URL.



InfluxDB-Verbindung

Feld	Bedeutung	Standard
InfluxDB-URL	z. B. `http://localhost:8086`	`http://localhost:8086`
Version	`v2` (Token + Org/Bucket) oder `v1` (Database, optional Benutzer/Passwort)	v2
Token (v2/v3)	API-Token	leer
Organisation (v2/v3)	Org-Name	leer
Bucket (v2/v3)	Ziel-Bucket	`opcbridge`
Database (v1)	Ziel-Datenbank	`opcbridge`
Benutzer / Passwort (v1)	optionale Anmeldung	leer
Measurement	Name der Messreihe	`opcbridge`
Tag-Schlüssel für die Adresse	Tag-Name, unter dem die Adresse abgelegt wird	`tag`
Qualität als Tag mitschreiben	Qualität zusätzlich als Tag	aus
Batch: max. Werte je Request	mehrere Werte als mehrzeiliges Line Protocol in einem Request	200

Feld	Bedeutung	Standard
Batch: max. Sammelzeit (ms)	spätestens nach dieser Zeit wird auch ein unvollständiges Batch gesendet	1000
TLS-Zertifikat prüfen	Zertifikatsprüfung	ein
Timeout (s)	Zeitüberschreitung	10
Erreichbarkeits-Prüfung (s)	Intervall des Health-Checks	10

Adressformat: frei gewählter Tag-Name; er wird unter dem konfigurierten **Tag-Schlüssel** an jede Messreihe gehängt.

---

26. Verbindungstyp Webhook

Rolle: Werte per HTTP an eine URL senden.

Feld	Bedeutung	Standard
Ziel-URL	Endpunkt	`https://erp.example.com/api/opcbridge`
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
Batch: max. Werte je Request	0 = einzeln senden; >1 bündelt mehrere Werte als **JSON-Array** in einem POST	0
Batch: max. Sammelzeit (ms)	spätestens nach dieser Zeit wird auch ein unvollständiges Batch gesendet	500
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}`).

---

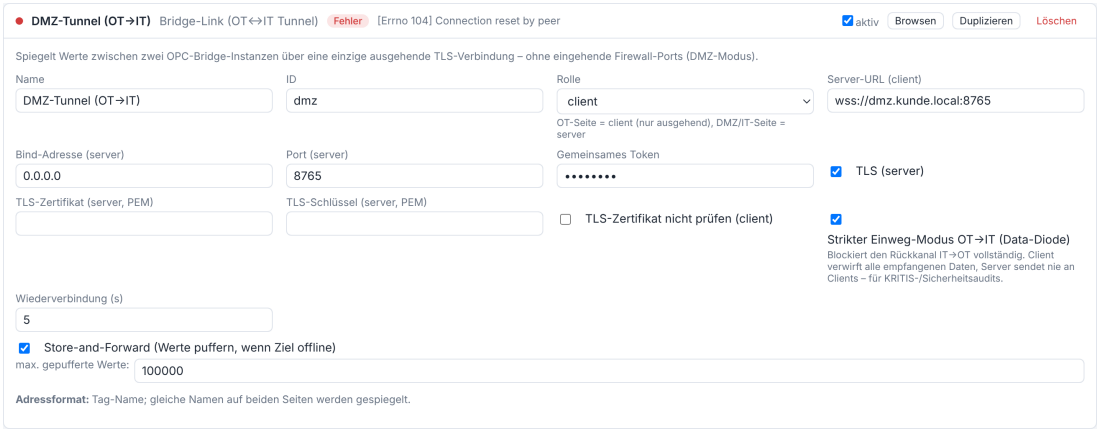
27. Verbindungstyp Bridge-Link

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

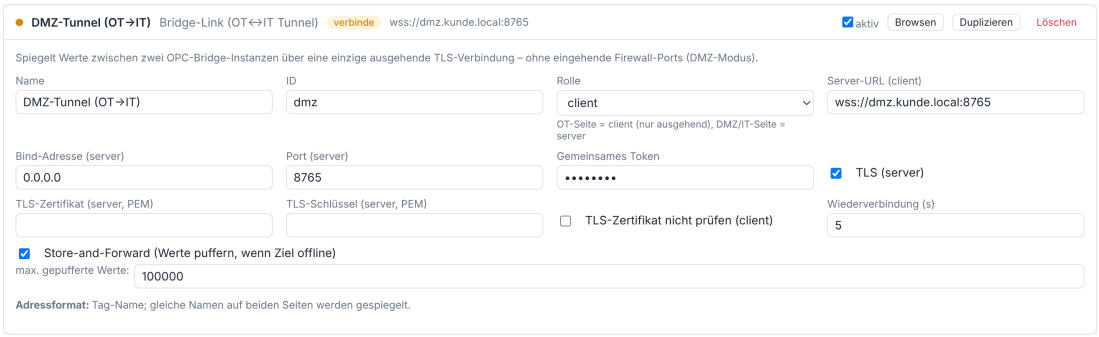
Feld	Bedeutung	Standard
Rolle	`client` (OT-Seite, nur ausgehend) oder `server` (DMZ/IT-Seite)	client
Server-URL (client)	Ziel des Clients, z. B. `wss://dmz-host:8765`	`wss://dmz-host:8765`
Bind-Adresse (server)	Netzwerkschnittstelle des Servers	0.0.0.0
Port (server)	Server-Port	8765
Gemeinsames Token	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
Strikter Einweg-Modus OT→IT (Data-Diode)	blockiert den Rückkanal IT→OT vollständig (siehe unten)	aus
Wiederverbindung (s)	Wartezeit vor erneutem Versuch	5

Adressformat: frei gewählter Tag-Name; **gleiche Namen** auf beiden Seiten werden gespiegelt. Anwendungsbeispiel siehe Kapitel 50.

Strikter Einweg-Modus (Data-Diode). Für KRITIS- und Hochsicherheitsumgebungen lässt sich der Rückkanal **IT→OT** vollständig blockieren: Ist *Strikter Einweg-Modus OT→IT* auf der Client- (OT-)Seite gesetzt, **verwirft** der Client jede empfangene Nachricht; ist er auf der Server- (IT-)Seite gesetzt, **sendet** der Server **nie** an Clients. Wird der Schalter auf beiden Seiten gesetzt, ist der Rückkanal nachweisbar und auf beiden Enden unterbunden – Werte fließen dann ausschließlich OT→IT. Das ist unabhängig von der Richtung der Mappings zu verstehen: Auch bei A↔B-Mappings bleibt die Netzwerk-Übertragung IT→OT gesperrt.



Bridge-Link mit Data-Diode

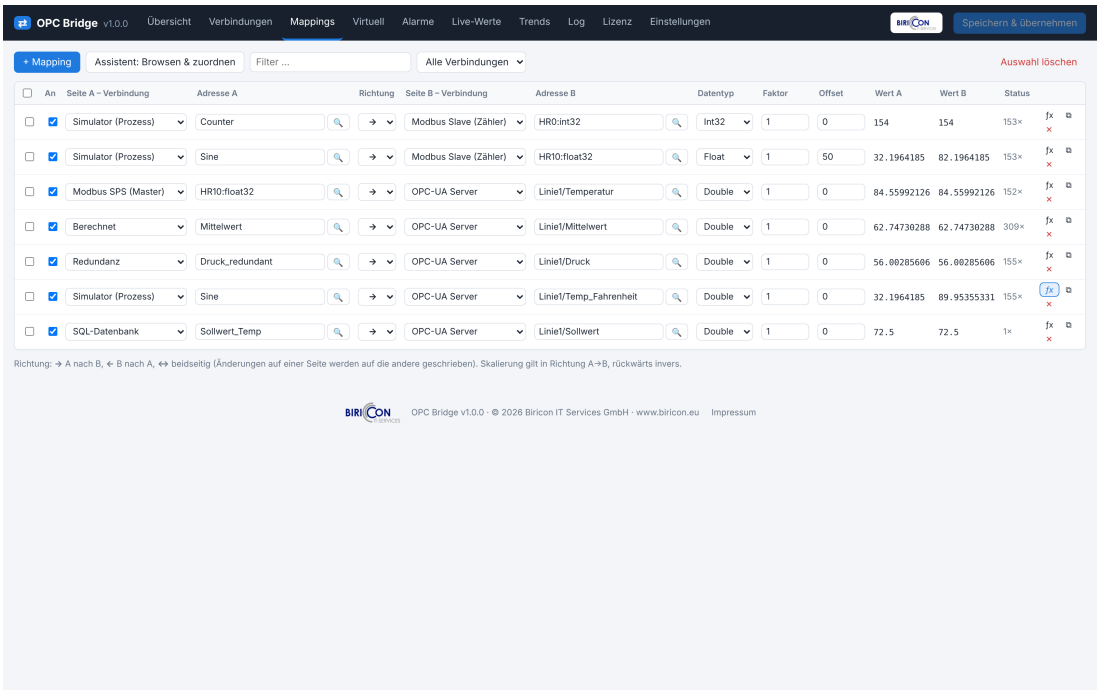


Bridge-Link / DMZ-Modus

---

28. Mappings und Datenfluss

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



Mappings

> »**Hinweis zum Bild**« Der Screenshot zeigt den Stand von Version 1.0; die Mapping-Tabelle zeigt inzwischen zusätzlich die Spalten **Faktor** und **Offset** (siehe unten).

Ein Mapping anlegen:

1. Reiter *Mappings* → **„+ Mapping"** klicken.
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**, **Faktor/Offset** und **Formel** (Kapitel 29) 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)
Faktor	Multiplikator, der auf den Rohwert angewendet wird (Standard 1)
Offset	Summand, der nach der Multiplikation addiert wird (Standard 0)
Formel	optionale zusätzliche Umrechnung (Variable `x`), wird nach Faktor/Offset angewendet
aktiv	Mapping ein-/ausschalten

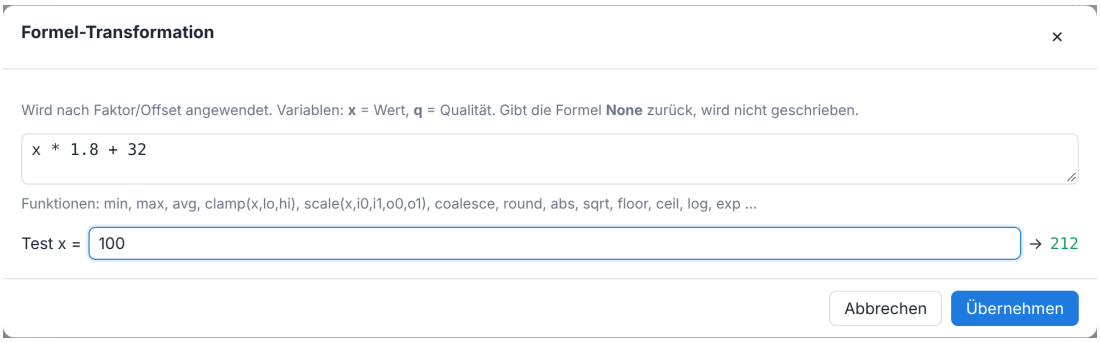
Faktor und Offset ergeben zusammen `wert * Faktor + Offset` – für die häufigsten linearen Skalierungen (z. B. Rohwert → Engineering-Einheit) reicht das ohne Formel. Bei Richtung A↔B wird beim Rückweg (B→A) automatisch umgekehrt skaliert.

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 30).

---

29. 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.



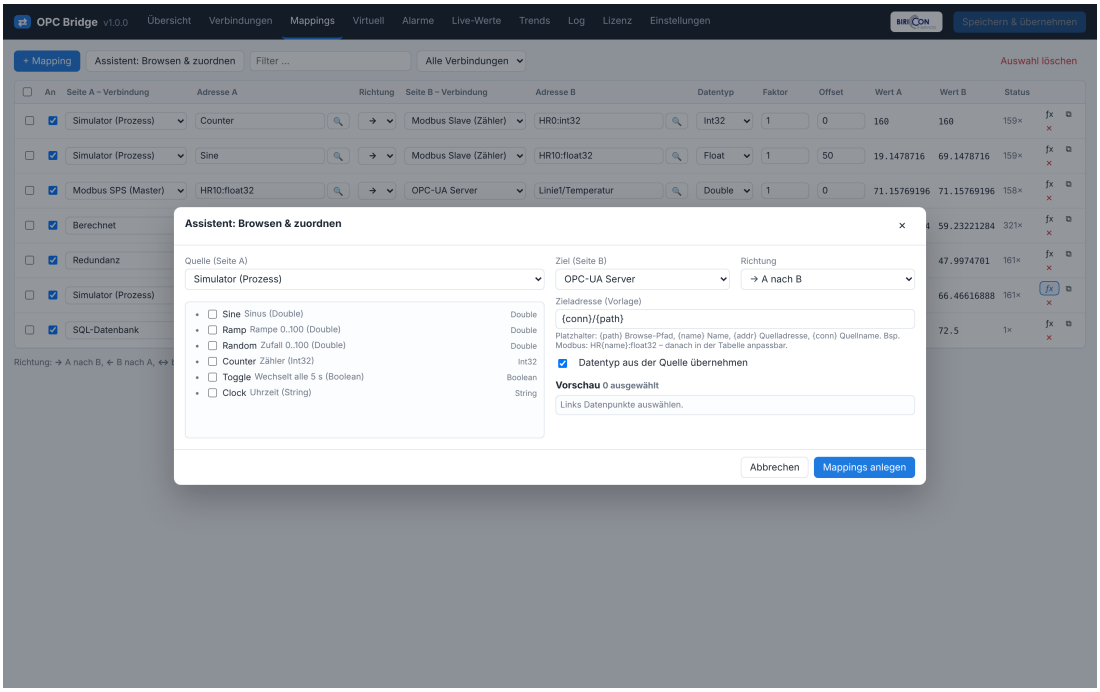
Formel

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.

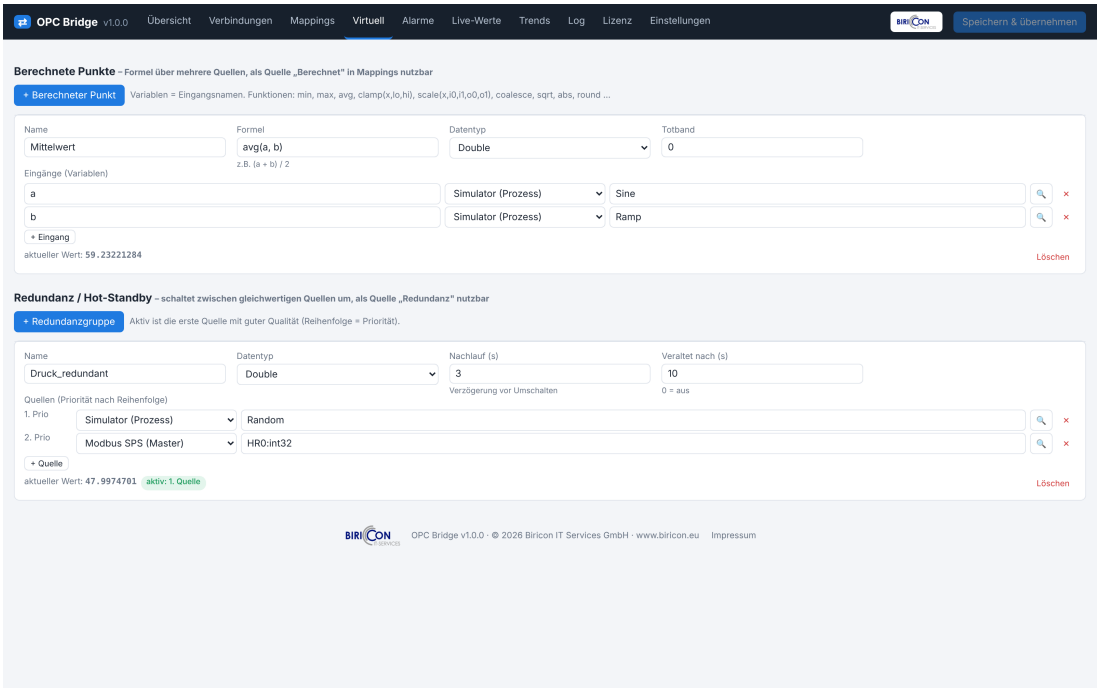
Assistent (Massen-Mapping). Über den Browse-Assistenten lassen sich viele Punkte auf einmal durchsuchen, auswählen und zuordnen – ideal bei großen Adressräumen.



Assistent

30. Virtuelle Quellen und Redundanz

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



Virtuelle Quellen

30.1 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.

30.2 Redundanz / Hot-Standby

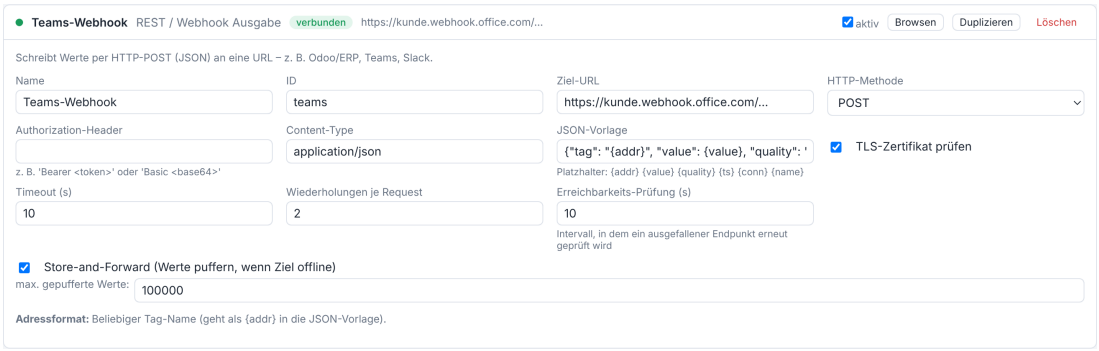
Eine Redundanzgruppe hat **mehrere Quellen für denselben Wert**. Die Bridge liefert automatisch die **beste verfügbare** Quelle und schaltet bei Ausfall oder schlechter Qualität auf die nächste um (Failover). Nutzbar z. B. bei redundanten Sensoren oder doppelt angebundenen Steuerungen.

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

---

31. 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.



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.

---

32. Historie und Trends konfigurieren

Im Reiter **Einstellungen → Historie**:

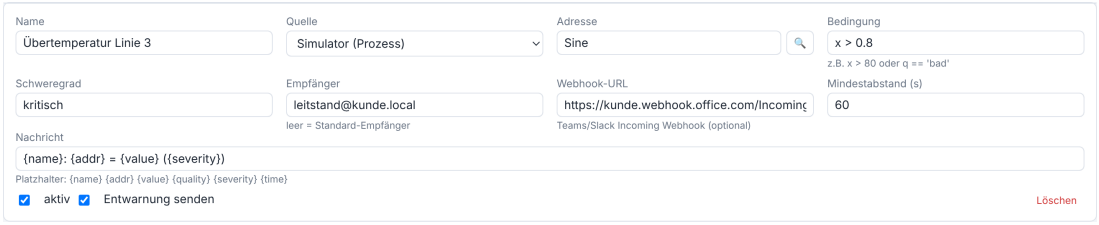
Feld	Bedeutung
Historie aktiv	Aufzeichnung ein/aus
Aufbewahrung (Tage)	wie lange Daten behalten werden
Min. Abstand je Punkt (s)	Drosselung (verhindert zu dichte Aufzeichnung)
Datenbankdatei	SQLite-Datei der Historie
Aufgezeichnete Punkte	Liste der Punkte mit optionalem **Totband**

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.

---

33. Alarme, E-Mail und Webhook

Im Reiter **Alarme** definieren Sie Benachrichtigungsregeln.



Alarmregel mit Webhook

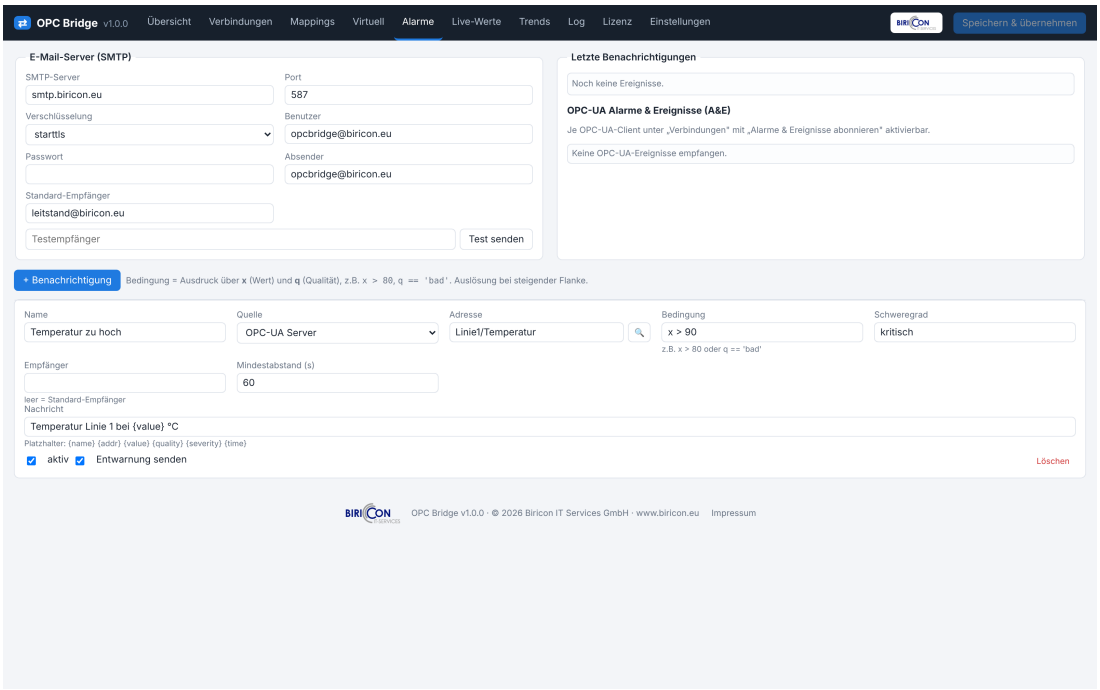
Felder einer Regel:

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: Die SMTP-Einstellungen liegen **im Reiter „Alarme"** selbst (nicht unter *Einstellungen*) – SMTP-Server, Port, Verschlüsselung (`none` / `starttls` / `ssl`), Benutzer, Passwort, Absender und 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.

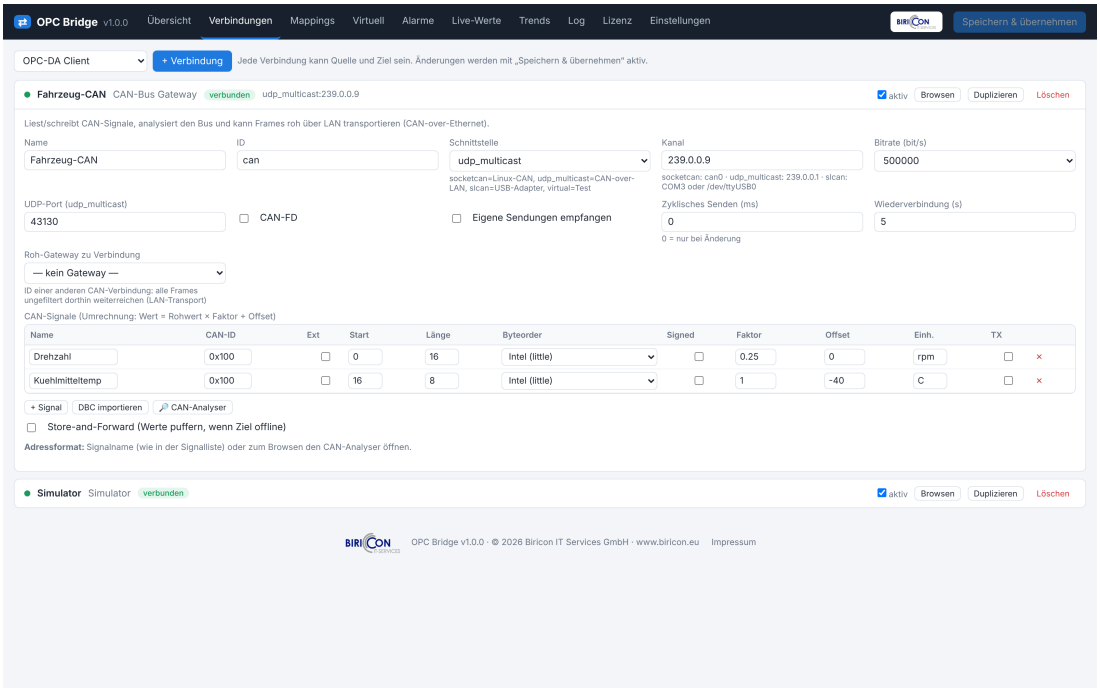


Alarme

> **»Hinweis zum Bild«** Der Screenshot zeigt den Stand von Version 1.0; der Reiter *Alarme* zeigt inzwischen zusätzlich den SMTP-Einstellungsblock (Host, Port, Verschlüsselung, Benutzer, Absender) direkt auf dieser Seite.

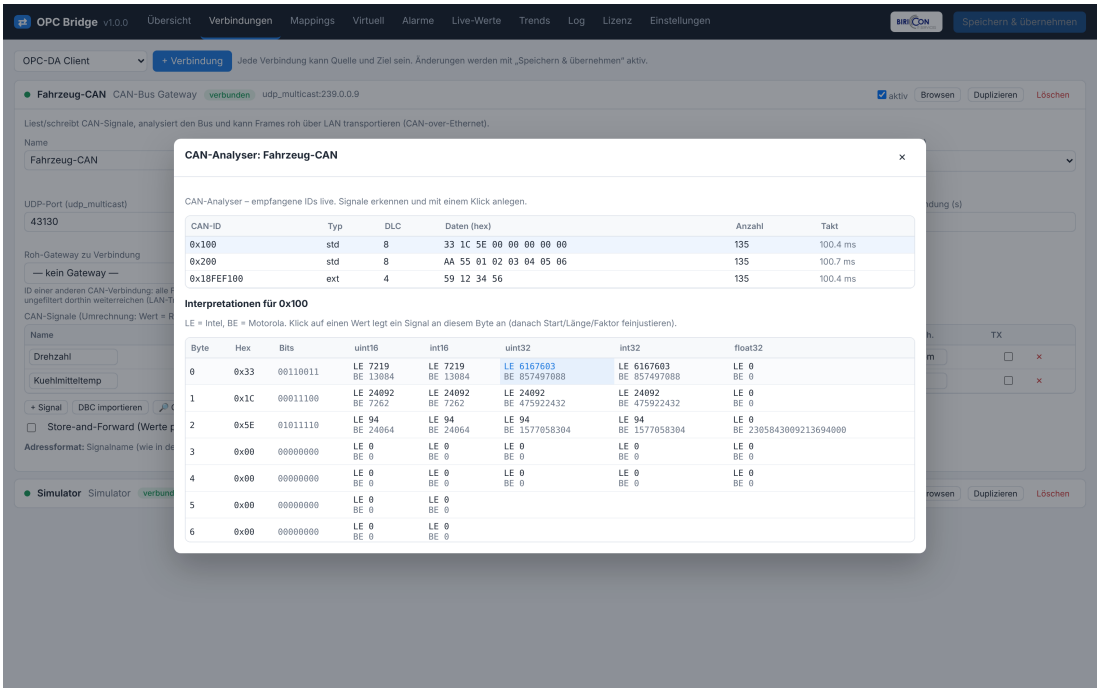
---

34. CAN-Analyser und Gateway



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.



CAN-Analyser

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.

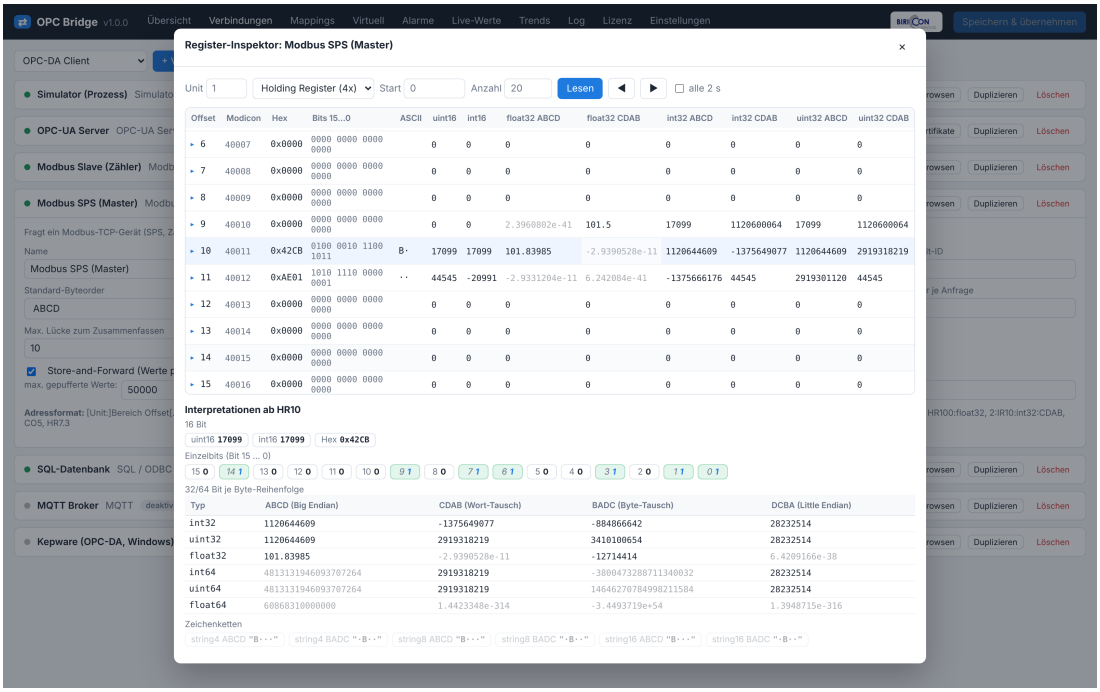
Roh-Gateway. Statt Signale zu extrahieren, kann der komplette Bus **ungefiltert über LAN** transportiert werden (Feld *Roh-Gateway zu Verbindung*). So lassen sich zwei entfernte CAN-Segmente über Ethernet koppeln (optional beidseitig).

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

---

35. 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**.



Register-Inspektor

Öffnen. Reiter *Verbindungen* → bei der jeweiligen **Modbus**-Verbindung auf **„Browsen"** klicken – der Register-Inspektor öffnet sich als Dialog für genau diese Verbindung (nur für aktive Modbus-Verbindungen verfügbar, sowohl Master als auch Slave).

Vorgehen:

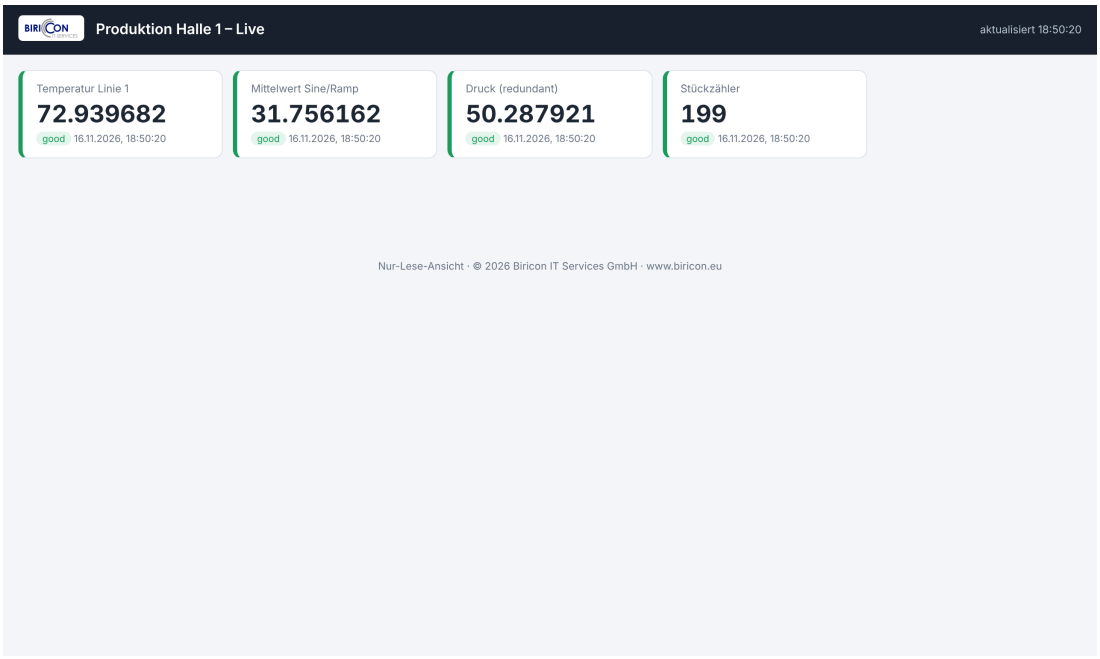
1. **Unit-ID**, **Bereich** – `CO` (Coils), `DI` (Discrete Inputs), `IR` (Input Registers) oder `HR` (Holding Registers) – sowie **Startadresse** und **Anzahl** wählen (Anzahl maximal 2000, Adresse 0…65535).
2. Bei `HR` / `IR` liest und zeigt der Inspektor je Registerpaar die Deutung als `int16`, `uint16`, `int32`, `uint32`, `float32` – in **allen** Byteorders (`ABCD`, `CDAB`, `BADC`, `DCBA`). Bei `CO` / `DI` werden die einzelnen Bits roh angezeigt.
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.

---

36. WebView

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



WebView

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

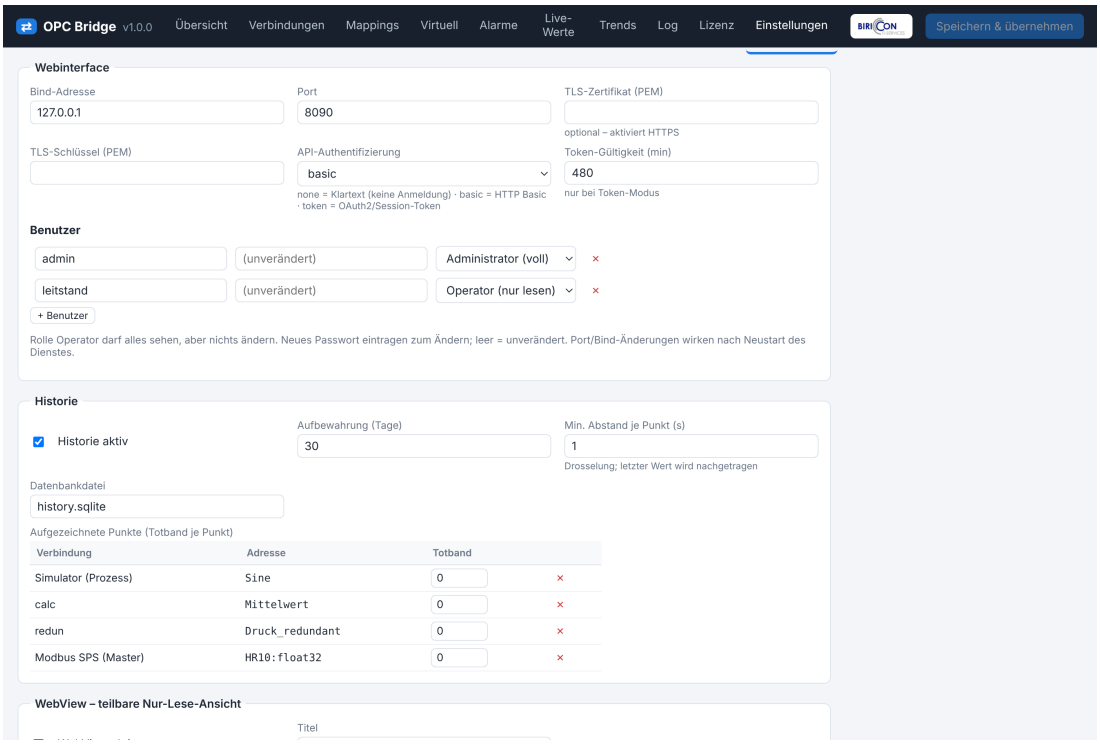
Feld	Bedeutung
WebView aktiv	Funktion ein/aus
Titel	Überschrift der Seite
Token	Geheimnis im Link (`/view?token=…`)
Punkte	welche Werte angezeigt werden

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

---

37. Benutzer, Rollen und API-Authentifizierung

Im Reiter **Einstellungen**.



API-Authentifizierung

> **»Hinweis zum Bild«** Der Screenshot zeigt den Stand von Version 1.0 (lokale Benutzer und Basic/Token-Auswahl); die Oberfläche zeigt inzwischen zusätzlich die Abschnitte **LDAP/Active Directory**, **OIDC/SSO** und **TOTP-2FA** (37.4–37.6).

37.1 Benutzer und Rollen

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

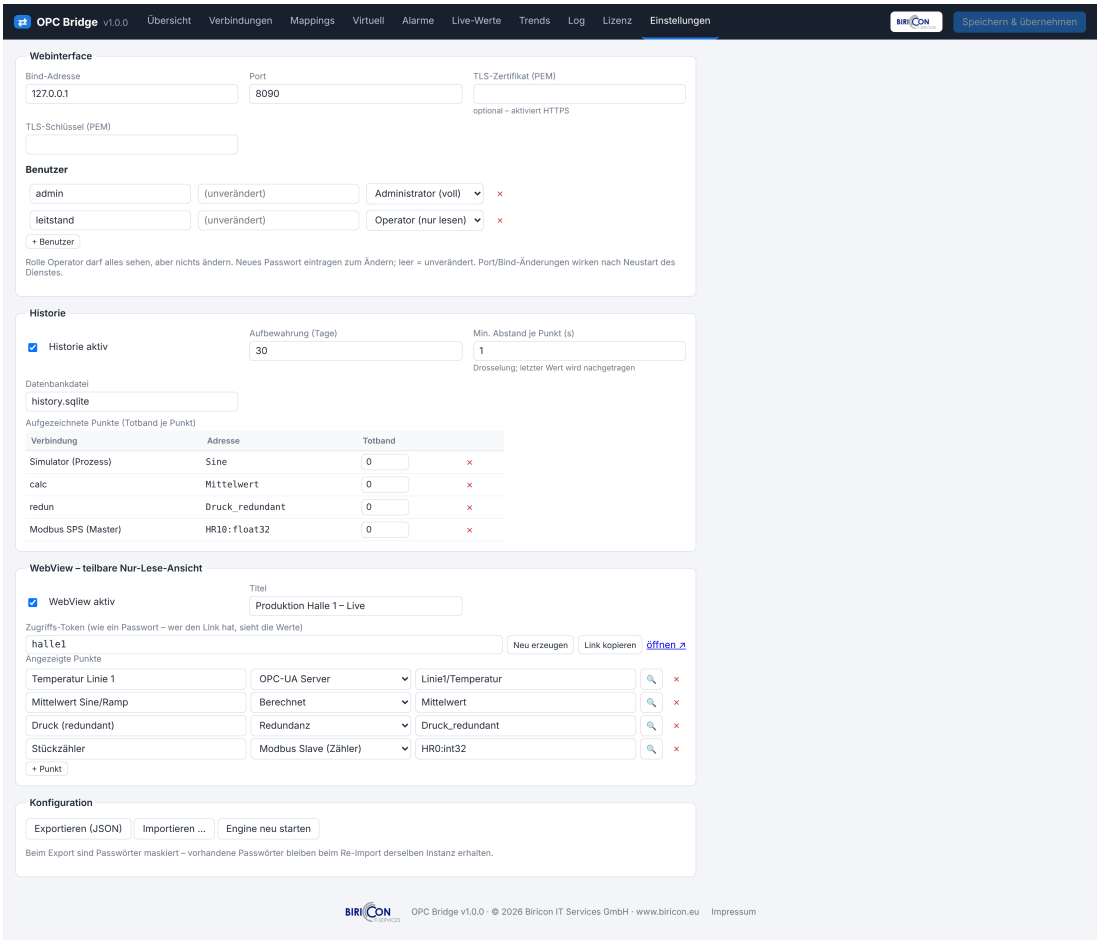
Passwörter werden als **PBKDF2-Hash** gespeichert. Neues Passwort eintragen zum Ändern; leer = unverändert. Schreibende Zugriffe sind serverseitig auf Administratoren beschränkt.

37.2 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).

> **»Hinweis zu 2FA im Modus „Basic"«** HTTP-Basic kann einen zweiten Faktor technisch nicht abfragen. Ein Benutzer mit aktivierter 2FA bekommt die Oberfläche im Modus `basic` dennoch angezeigt (sonst käme die Seite gar nicht zustande), muss sich für API-Aufrufe aber trotzdem über `/api/login` mit Benutzername, Passwort **und** TOTP-Code anmelden und den erhaltenen Bearer-Token verwenden – der reine Basic-Login genügt für 2FA-Benutzer nicht.

37.3 Webinterface-Einstellungen



Einstellungen

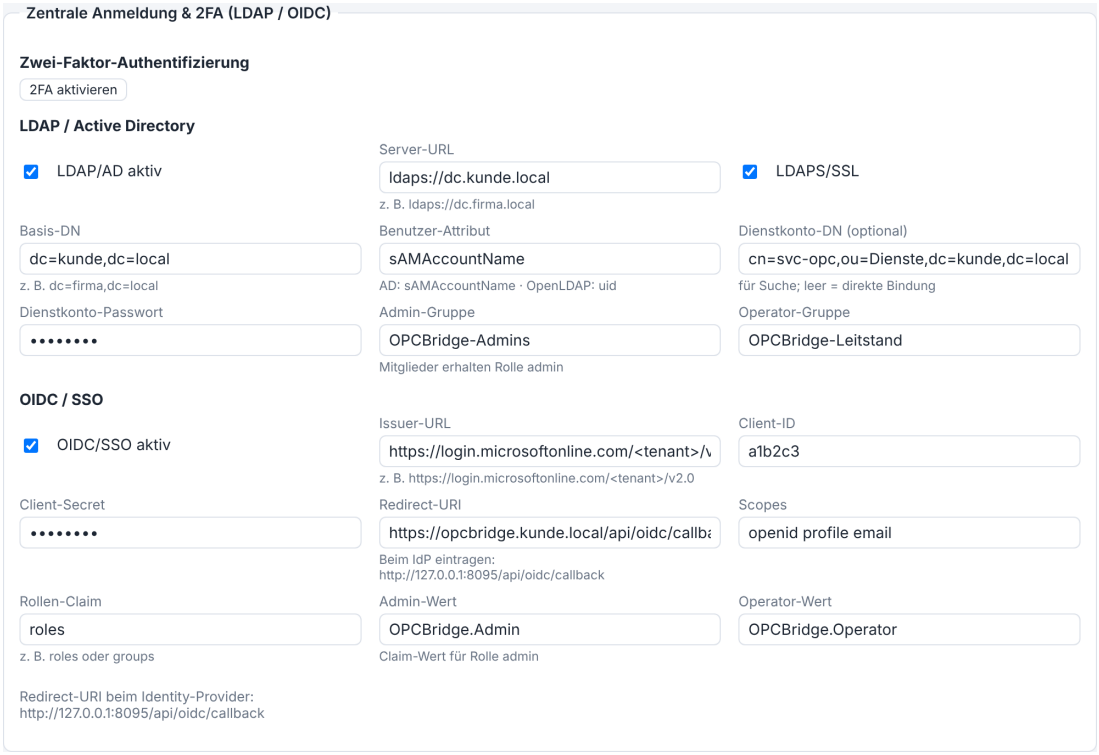
> **»Hinweis zum Bild«** Der Screenshot zeigt den Stand von Version 1.0; der Reiter *Einstellungen* zeigt inzwischen zusätzlich Abschnitte für **Online-Lizenz** und **Update-Prüfung** (Kapitel 41.5 bzw. 44.1). Die SMTP-Einstellungen sind – anders als es ein älterer Screenshot suggerieren könnte – **nicht** hier, sondern im Reiter *Alarme* (Kapitel 33) zu finden.

Feld	Bedeutung
Bind-Adresse / Port	Netzwerkschnittstelle und Port der Oberfläche (wirkt nach Dienst-Neustart)

Feld	Bedeutung
TLS-Zertifikat / -Schlüssel (PEM)	aktiviert HTTPS
API-Authentifizierung	siehe 37.2
Token-Gültigkeit (min)	Lebensdauer der Tokens im Token-Modus

37.4 Zentrale Anmeldung: LDAP/Active Directory

Für IT-Audits in größeren Betrieben lässt sich die Anmeldung an die zentrale Benutzerverwaltung anbinden, optional ergänzt um Zwei-Faktor-Authentifizierung.



LDAP/OIDC/2FA-Einstellungen

Anmeldung per Verzeichnisdienst mit Gruppen-zu-Rollen-Abbildung (Admin-/Operator-Gruppe), Dienstkonto-Suche oder direkter Bindung, sowie LDAPS/SSL. LDAP ist optional und standardmäßig deaktiviert; bestehende lokale Benutzer bleiben unverändert nutzbar.

37.5 OIDC / Single Sign-On

Authorization-Code-Flow gegen einen externen Identity-Provider (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; der Ablauf startet über `/api/oidc/start` .

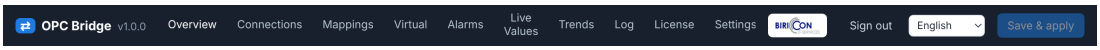
37.6 TOTP-Zwei-Faktor-Authentifizierung (2FA)

Jeder lokale Benutzer kann in den Einstellungen 2FA aktivieren (Authenticator-App per Secret/QR-Code, Verwaltung über `/api/2fa/status` , `/api/2fa/setup` , `/api/2fa/enable` , `/`

`api/2fa/disable`). Bei aktiver 2FA verlangt der Login zusätzlich einen 6-stelligen Code; die eigentliche Anmeldung erfolgt dann über den Token-Modus (`/api/login` mit `code`), siehe auch den Hinweis zu 2FA im Modus „Basic" in 37.2.

37.7 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.



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.

---

38. 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.

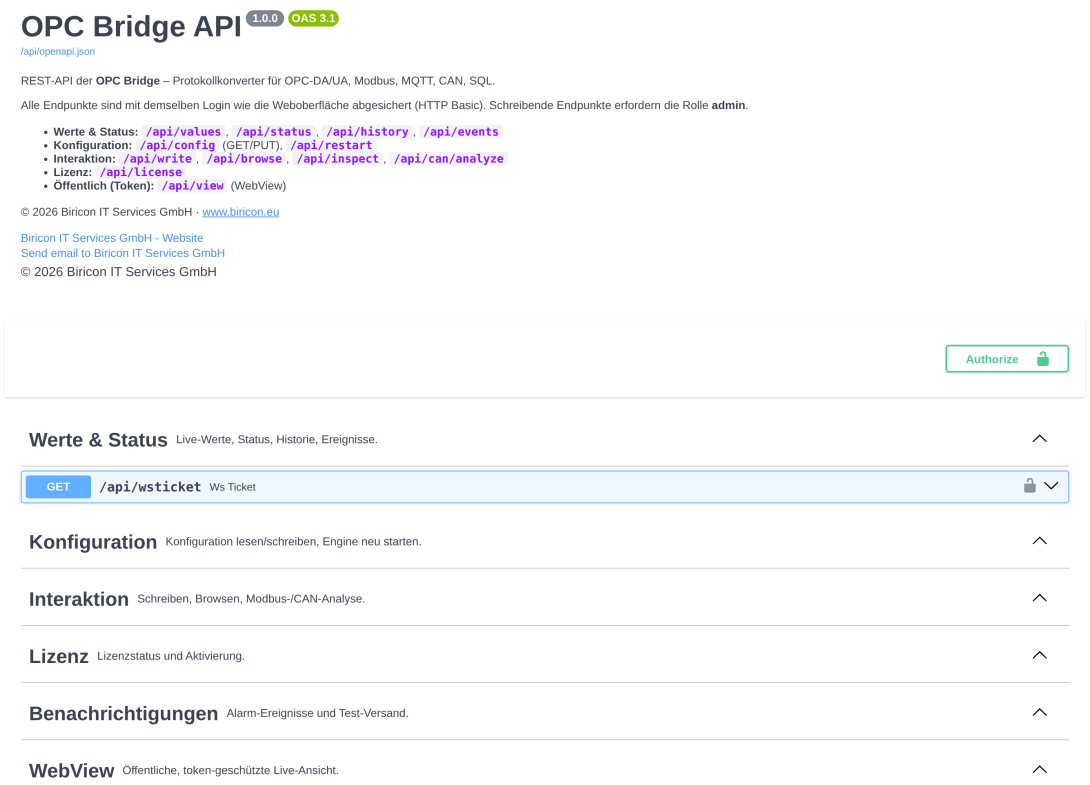
Technisch nutzt die WebSocket ein **kurzlebiges Ticket** (über `/api/wsticket`), das im gewählten Authentifizierungsmodus abgesichert ist.

Teil V – Schnittstellen und Integration

39. 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.



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 37.2): keine (Klartext), HTTP-Basic oder Bearer-Token. Beispiel (Token holen und verwenden):

```
# Token holen (Token-Modus): /api/login erwartet JSON im Body, nicht HTTP-Basic.
# "code" ist nur bei aktiver 2FA nötig (TOTP-Code) und kann sonst entfallen.
curl -X POST http://host:8080/api/login \
```

```
     -H "Content-Type: application/json" \
     -d '{"username":"admin","password":"geheim"}'
# -> {"token":"...","token_type":"bearer","expires_in":28800,"role":"admin"}

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

> »**Hinweis**« `POST /api/token` steht zusätzlich als klassischer **OAuth2-Password-Flow** (Formular statt JSON) bereit – u. a. für die „Authorize"-Schaltfläche der eingebauten Swagger-Dokumentation und generische OAuth2-Clients.

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

---

40. 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**« Nur die **Web-Benutzer-Passwörter** (`web.users[].password_hash`) werden als PBKDF2-**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 43).

> »**Achtung – Klartext-Zugangsdaten in config.json**« Zugangsdaten zu **Fremdsystemen** stehen dagegen **im Klartext** in `config.json` : SQL-Connection-Strings (DB-Passwörter), MQTT-/ Bridge-Link-/InfluxDB-Benutzer-Passwörter und -Token, Webhook- `Authorization` -Header, WebView-Token, SMTP-Passwort, der Online-Lizenz-Rollout-Token usw. Das ist eine **bewusste Designentscheidung** (die Bridge muss diese Geheimnisse zur Laufzeit im Klartext an die jeweiligen Treiber übergeben

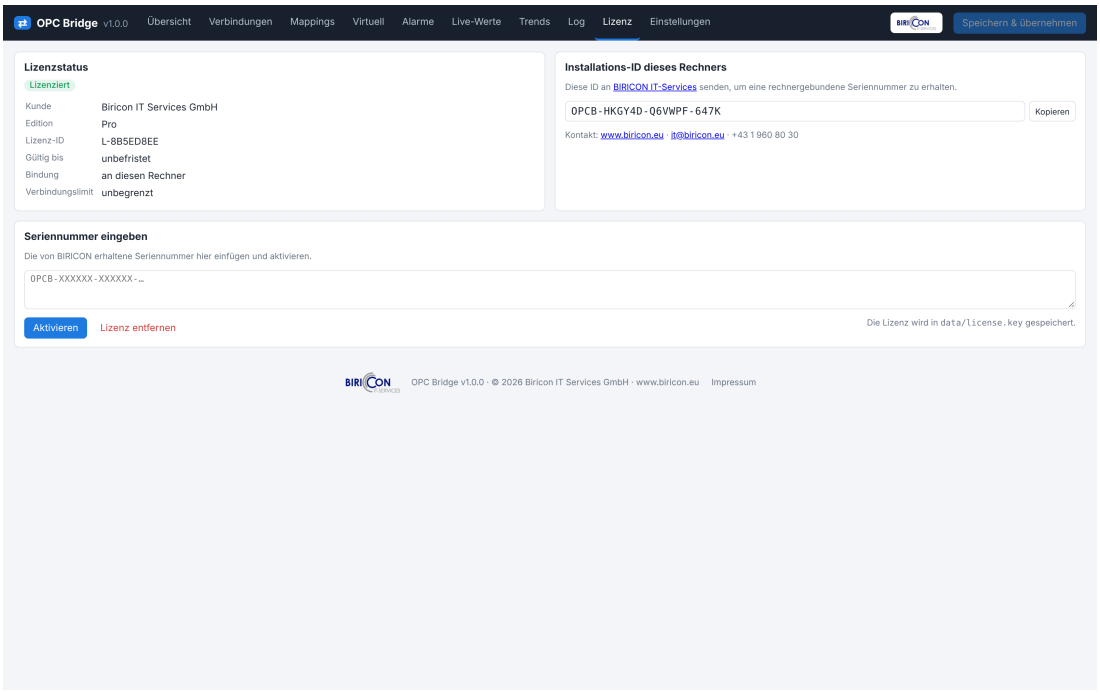
> können) und **keine Dokumentationslücke** – aber jeder, der die Datei sichert, kopiert oder weitergibt, muss sich dessen bewusst sein. Die **Web-API** maskiert diese Felder beim Lesen (`GET /api/config` ersetzt Passwörter, Token, Webhook-`auth_header` und das Passwort in SQL-Verbindungs-URLs durch `********`) – das gilt aber nur für die API-Antwort, **nicht** für die Datei selbst. Behandeln Sie `config.json` und ihre Backups entsprechend vertraulich (Dateiberechtigungen, Transportverschlüsselung).

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. Für ein Repository mit mehreren Mitlesern ist angesichts der Klartext-Zugangsdaten oben ein privates Repository bzw. ein verschlüsselter Secret-Store vorzuziehen.

Teil VI – Betrieb

41. Lizenzierung

Die OPC Bridge kennt zwei – parallel nutzbare – Lizenzverfahren: die bewährte **rechnergebundene Offline-Seriennummer** (Ed25519-signiert, offline prüfbar, fälschungssicher) und eine optionale **Online-Lizenz** gegen den Biricon-Lizenzserver.



Lizenz

> »**Hinweis zum Bild**« Der Screenshot zeigt den Stand der Offline-Lizenz von Version 1.0; die Lizenzseite zeigt inzwischen zusätzlich den gesamten Abschnitt **Online-Lizenz** (Standort-Adresse, Lizenz-ID, Rollout-Token, Strikt-Option, Maschinen-ID) aus 41.5.

41.1 Demo-Modus

Ohne gültige Lizenz (weder Offline-Seriennummer noch online erteilte Lizenz) läuft die Bridge im **Demo-Modus**:

- **einmalig 60 Minuten** volle Funktion – die Uhr beginnt beim **allerersten Start** der Installation und wird dauerhaft gespeichert (`demo_started_at.txt` im Datenverzeichnis). Nach Ablauf dieser 60 Minuten wird die Weiterleitung gestoppt (die Oberfläche bleibt erreichbar) – und zwar **endgültig**, nicht nur bis zum nächsten Start.
- dauerhaft **maximal 2 Verbindungen**.

> **»Achtung«** Ein Neustart des Dienstes/Containers **verlängert die Demo-Zeit nicht**. Die 60 Minuten laufen ab dem ersten jemals beobachteten Start einmalig ab, unabhängig davon, wie oft die Bridge danach neu gestartet wird. Für den Dauerbetrieb ist eine Lizenz erforderlich.

41.2 Offline-Aktivierung

1. Reiter *Lizenz* öffnen und die **Installations-ID** ablesen (sie ist an diesen Rechner gebunden; in Docker an die im Datenverzeichnis/Volume persistierte Maschinenkennung, siehe unten – sie überlebt eine Container-Neuerstellung).
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.

41.3 Lizenzarten und Edition/Verbindungslimit

- **Unbefristet:** dauerhaft gültig.
- **Befristet (Jahres-/Wartungslizenz):** mit Ablaufdatum; rechtzeitig verlängern.
- Jede Lizenz (offline wie online) kann eine **Edition** und ein **Verbindungslimit** (`max_connections`) tragen; `0` bedeutet unbegrenzt. Ohne gültige Lizenz gilt das Demo-Limit von 2 Verbindungen.

41.4 Ablauf und Widerruf – sofortiger Stopp ohne Gnadenfrist

Läuft eine zuvor gültige Lizenz **ab** (Offline-Ablaufdatum oder von der Online-Prüfung gemeldet) oder wird sie **widerrufen** (z. B. `lizenz_widerrufen` , serverseitig erkennbar bei nicht fristgerechter Zahlung), **stoppt die Weiterleitung sofort** – ohne weitere Gnadenfrist und ohne dass ein Neustart nötig wäre; die Prüfung läuft auch während des laufenden Betriebs (nicht nur beim Start). Das unterscheidet sich bewusst vom „nie lizenziert"-Fall (Demo), für den es die 60-minütige Demo-Zeit gibt. Manuelles Schreiben über Oberfläche/API ist von derselben Sperre betroffen wie das automatische Mapping-Routing.

41.5 Online-Lizenz, automatische Aktivierung und Klon-Erkennung

Zusätzlich zur Offline-Seriennummer kann die OPC Bridge die Lizenz **online** gegen den Biricon-Lizenzserver prüfen und aktivieren – **automatisch, sobald Internet verfügbar ist**. Das Offline-Verfahren bleibt unverändert und greift weiterhin in abgeschotteten Netzen; beide Verfahren dürfen parallel bestehen. Konfiguration unter *Einstellungen → Online-Lizenz*:

Feld	Bedeutung
Aktiv	schaltet die Online-Prüfung ein (Standard: aus)

Feld	Bedeutung
Server	Lizenzserver-URL (Standard: `https://www.biricon.eu`)
Standort-Adresse	feste `https://host[:port]`-Adresse der Anlage, die bei der Lizenz im Lizenzserver hinterlegt ist
Lizenz-ID (Auto-Aktivierung)	Format `XXXXX-XXXXX-XXXXX-XXXXX`; ohne Seriennummer gesetzt löst sie die Selbstaktivierung aus
Rollout-Token (optional)	Geheimnis für automatisierte Massen-Aktivierung vieler Installationen
Strikt	ohne erfolgreiche Online-Prüfung innerhalb der Karenz endet die Lizenz (siehe unten)

- **Zwei Betriebsarten:** Ist bereits eine gültige **Offline-Seriennummer** installiert, prüft die Online-Funktion nur deren **Lizenz-ID** online nach (`POST /licence/v1/pruefen`, Modus „Prüfung“). Ist **keine** Seriennummer installiert, aber eine **Lizenz-ID** oder ein **Rollout-Token** hinterlegt, aktiviert sich die Bridge **selbst** (`POST /licence/v1/aktivieren`, Modus „Aktivierung”) – ganz ohne manuell eingegebene Seriennummer. Danach wird regelmäßig (mindestens täglich, je nach Servervorgabe auch öfter) erneut geprüft. Manuelles Nachprüfen/Aktivieren: Knopf auf der Lizenzseite bzw. `POST /api/license/online/activate`.
- **Rechnerbindung & Klon-Erkennung:** Die Bridge berechnet eine **Maschinen-ID** (SHA-256 aus einer gerätespezifischen Kennung; der Rohwert verlässt den Rechner nie – unter Docker wird dafür eine einmal erzeugte Kennung **im Datenverzeichnis/Volume** `/data` abgelegt und danach immer wiederverwendet, statt sich bei jeder Container-Neuerstellung instabil aus `/etc/machine-id` o. Ä. neu abzuleiten). Jeder Platz der **Gerätegrenze** ist genau eine Maschine. Wird ein Rechner neu aufgesetzt oder eine VM geklont und erhält dabei eine **neue** Kennung, gilt er als **neue Maschine** und braucht eine neue Freischaltung (Backend → „Lizenz umziehen"). Ein bit-identischer Klon teilt die Kennung und belegt denselben Platz. Die eigene **Maschinen-ID** ist auf der Lizenzseite kopierbar.
- **Fälschungs-/Replay-Schutz:** Jede Prüfung sendet eine neue Zufalls-Nonce, die der Server signiert (ES256) zurückgibt; eine alte Antwort kann nicht wiedereingespielt werden.
- **Karenz bei Netzausfall:** Ein reiner **Netzfehler** (kein Netz, 429, 503) sperrt die Lizenz **nie** – es gilt die zuletzt erfolgreich geprüfte, signierte Antwort bis zu deren `karenz_bis`-Zeitpunkt weiter (nie über das eigentliche Ablaufdatum `gueltig_bis` hinaus). Nur mit aktivierter Option **„Strikt"** endet die Lizenz, wenn die letzte erfolgreiche Prüfung länger als die Karenz zurückliegt. Eine **gültige Absage** des Servers (z. B. `lizenz_widerrufen`, `lizenz_abgelaufen`, `adresse_falsch`, `maschine_nicht_lizenziert`) sperrt dagegen **sofort**, ohne Karenz (siehe 41.4).

---

42. Sicherheit und Härtung

> »**Achtung**« Werkseitig bindet die Weboberfläche auf `0.0.0.0:8080` und spricht **Klartext-HTTP**, solange kein TLS-Zertifikat/-Schlüssel hinterlegt ist. Im Auslieferzustand ist das Webinterface also von jeder Netzwerkschnittstelle unverschlüsselt erreichbar – für den Produktivbetrieb unbedingt die folgenden Punkte beachten.

> »**Achtung – Modus „Klartext" (`none`)**« Im API-Authentifizierungsmodus `none` ist **JEDER**, der das Webinterface erreichen kann, faktisch **Administrator ohne jede Anmeldung** – es gibt in diesem Modus keine Rollenprüfung. Dieser Modus ist **ausschließlich** für ein physisch abgeschottetes Maschinennetz vertretbar, niemals für ein Netz mit IT-Zugriff oder Internet.

Empfehlungen für den Produktivbetrieb:

1. **Passwörter ändern.** Erststart-Passwort sofort ändern; starke Passwörter verwenden. Die Datei `initial_admin_password.txt` (Anhang F) sollte nach dem ersten erfolgreichen Login **gelöscht** werden – sie liegt im Klartext im Datenverzeichnis.
2. **Rollen nutzen.** Für Beobachter nur die Rolle **Operator** vergeben.
3. **API-Authentifizierung** mindestens auf **Basic**, besser **Token** stellen. **Klartext** (`none`) nur im physisch abgeschotteten Netz (siehe Hinweis oben).
4. **Zentrale Anmeldung/2FA nutzen, wenn vorhanden.** Für Audits in größeren Betrieben stehen optional **LDAP/Active Directory**, **OIDC/SSO** und **TOTP-2FA** für lokale Benutzer zur Verfügung (Kapitel 37.4–37.6). Alle drei sind standardmäßig deaktiviert.
5. **HTTPS aktivieren.** TLS-Zertifikat/-Schlüssel hinterlegen oder einen Reverse-Proxy (nginx) mit TLS davorsetzen – siehe Achtung-Hinweis oben.
6. **OPC-UA absichern.** Security-Policy `Basic256Sha256 SignAndEncrypt`, Truststore statt `trust_all`, Benutzer statt `anonymous=readwrite`. Pro OPC-UA-Verbindung verwaltet die Bridge einen eigenen **PKI-Ordner** (eigenes Zertifikat `own_cert.der`, Unterordner `trusted/` und `rejected/`); von einem fremden Server abgelehnte/unbekannte Zertifikate landen zunächst in `rejected/` und können über *Verbindungen* bzw. `GET` / `POST /api/pki/{conn}[...]` eingesehen, in den Trust-Store verschoben (`trust`) oder gelöscht werden.
7. **Netzsegmentierung.** OT und IT trennen; für die Kopplung den **Bridge-Link** (DMZ-Modus, bei Bedarf mit **Data-Diode**) verwenden, damit keine eingehenden OT-Ports nötig sind.
8. **WebView** nur mit starkem Token und HTTPS; nur nötige Punkte freigeben.
9. **Ports minimieren.** Nur benötigte Dienste (OPC-UA-Server, Modbus-Slave) aktivieren und in der Firewall öffnen.
10. **Updates** zeitnah einspielen (Kapitel 44). Die Update-Prüfung ist werkseitig **aktiv** und fragt **täglich** `www.biricon.eu` ab (siehe unten).

11. **Ausgehende Verbindungen im Blick behalten.** Auch in einem sonst abgeschotteten Netz baut die Bridge selbst ausgehende Verbindungen auf, wenn die jeweilige Funktion aktiviert ist:
    - die **Update-Prüfung** (täglich gegen `www.biricon.eu`, abschaltbar),
    - die **Online-Lizenzprüfung**, falls konfiguriert (Kapitel 41.5, gegen den Lizenzserver, standardmäßig aus),
    - konfigurierte **Ausgabe-Verbindungen** mit externem Ziel: Webhook-URL, MQTT-/Sparkplug- Broker, SQL-/InfluxDB-Zielserver, Bridge-Link-Client zur Gegenstelle. In einem Air-Gap-/Hochsicherheitsnetz sind Update-Prüfung und Online-Lizenz bewusst zu deaktivieren, wenn kein ausgehender Internetzugriff gewünscht ist.

---

43. Datensicherung und Wiederherstellung

43.1 Export/Import über die Weboberfläche

Für reine **Konfigurations**-Backups (ohne Historie/Lizenz/Zertifikate) genügt ein Export über die API: `GET /api/config` liefert die vollständige Konfiguration als JSON – allerdings mit **maskierten Geheimnissen** (Passwörter, Token, Webhook-`auth_header`, das Passwort in SQL-Verbindungs-URLs sowie das WebView-Token werden als `********` zurückgegeben, siehe Kapitel 40). Ein Export in dieser Form eignet sich daher zur **Dokumentation/Versionsverwaltung**, nicht aber als vollständiges Wiederherstellungs-Backup – beim Zurückspielen über `PUT /api/config` bleiben zuvor maskierte, unverändert gelassene Geheimnisse erhalten, **neue** Geheimnisse müssen aber erneut eingetragen werden.

43.2 Vollständiges Backup (Dateiebene)

Was sichern? Für ein **vollständiges** Backup (inklusive Lizenz, Historie, Store-and-Forward-Puffer, Zertifikaten und Demo-/Update-/Lizenzzustand) sichern Sie das gesamte **Datenverzeichnis** (Anhang F). Es enthält insbesondere:

Datei/Ordner	Inhalt
`config.json`	die gesamte Konfiguration (Verbindungen, Mappings, Benutzer-Hashes, Klartext-Zugangsdaten zu Fremdsystemen – siehe Kapitel 40)
`license.key`	die Offline-Seriennummer
`history.sqlite` (bzw. der konfigurierte Dateiname)	die Historien-Datenbank
`sf_queue.sqlite`	die persistente Store-and-Forward-Warteschlange
`online_state.json`	Zustand der Online-Lizenzprüfung (letzte signierte Server-Antwort)

Datei/Ordner	Inhalt
`update_state.json`	Zustand der Update-Prüfung
`machine_id`	die persistierte Maschinenkennung für die Rechnerbindung (wichtig in Docker!)
`demo_started_at.txt`	Zeitpunkt des allerersten Starts (Demo-Zeitfenster, Kapitel 41.1)
`pki/`	OPC-UA-Zertifikate je Verbindung (eigenes Zertifikat, Trust-/Reject-Store, Kapitel 42)
`opcbridge.log`, `service.log`	Protokolldateien (für ein Backup nicht erforderlich, aber unschädlich)

> **»Achtung«** `machine_id` sollte bei einer Wiederherstellung **mitgesichert und zurückgespielt** werden, wenn eine bestehende Lizenz erhalten bleiben soll – insbesondere in Docker, wo diese Datei die Rechnerbindung über eine Container-Neuerstellung hinweg stabil hält (Kapitel 41.5).

Sicherung unter Linux (Beispiel):

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

Sicherung unter Windows (Beispiel, PowerShell):

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

Sicherung unter Docker (Beispiel): Das Datenverzeichnis liegt im benannten Volume `opcbridge-data`; es lässt sich unabhängig vom Container sichern:

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

Wiederherstellung: Dienst/Container stoppen, Datenverzeichnis bzw. Volume aus dem Backup zurückspielen, Dienst/Container starten. Da die **Lizenz rechnergebunden** ist, funktioniert sie nur auf demselben Rechner (bzw. mit derselben, mitgesicherten `machine_id`); bei einem neuen Rechner ohne übernommene `machine_id` ist eine neue Lizenz anzufordern.

> **»Tipp«** Für reine Konfig-Backups ohne die übrigen Dateien genügt `config.json`. Legen Sie diese Datei in eine Versionsverwaltung, um Änderungen nachvollziehen zu können – beachten Sie dabei aber die Klartext-Zugangsdaten aus Kapitel 40.

---

44. Update und Migration

44.1 Update-Prüfung und Ein-Klick-Installation

Die Bridge prüft **werkseitig aktiv** und **täglich** (alle 86 400 Sekunden) gegen eine **signierte Versionsdatei** (`version.json`, ES256 mit dem Biricon-Update-Schlüssel) bei `www.biricon.eu`, ob eine neuere Version verfügbar ist. Die laufende Version sehen Sie auf der Lizenz-/Update-Seite der Oberfläche oder über `GET /api/update` bzw. `GET /api/meta`. Ist eine neuere Version verfügbar, erscheint sie mit Versionshinweisen und – sofern ein signiertes Installationspaket für die laufende Plattform hinterlegt ist – einem Knopf **„Jetzt installieren"** (`POST /api/update/install`, nur Administratoren).

- **Sicher:** Nur eine mit dem Biricon-Update-Schlüssel **signierte** Versionsdatei wird verwendet; das gemeldete Paket wird vor der Installation gegen den mitsignierten **SHA-256** geprüft.
- **Manipulationsschutz der ausgelieferten Dateien:** Jedes Release enthält eine **signierte Hash-Liste** (`integrity.json`) aller ausgelieferten Programmdateien. Weicht beim Start oder bei der periodischen Nachprüfung eine Datei vom signierten Hash ab, fehlt die Signatur, oder taucht eine fremde Binärdatei im Programmverzeichnis auf, verweigert ein Release-Build entweder den Start oder fällt auf den Lizenzstatus „nicht erteilt" (Demo) zurück – je danach, ob die Prüfung beim Start oder im laufenden Betrieb fehlschlägt.
- **Windows:** Das Setup wird **still** ausgeführt (Ein-Klick); der Dienst wird dabei ersetzt.
- **Debian/Ubuntu:** `apt-get install -y <paket>` wird ausgeführt. **Läuft der Dienst unter systemd mit `NoNewPrivileges=true`** (wie im mitgelieferten Unit, Standard), kann **kein** `sudo` – auch mit passender `sudoers`-Regel – aus dem Dienst heraus jemals funktionieren. Die Bridge erkennt das vorher und versucht es in diesem Fall **gar nicht erst**, sondern liefert sofort die klare Meldung „Der Dienst läuft mit systemd NoNewPrivileges – sudo kann von dort aus nie funktionieren. Bitte manuell: `sudo apt-get install -y <paket>`" – das Update ist dann **manuell** auf der Konsole einzuspielen (siehe 44.2).
- **Docker:** Es gibt keinen „Ein-Klick"-Mechanismus im Container; die Oberfläche weist auf das neu zu ziehende Image hin (`docker compose pull`, siehe 44.3).
- Ohne hinterlegtes signiertes Paket für die laufende Plattform bleibt es beim reinen Hinweis mit Download-Link.
- **Abschalten:** Die Update-Prüfung lässt sich in den Einstellungen deaktivieren (z. B. in einem Air-Gap-Netz ohne Internetzugriff, siehe Kapitel 42).

44.2 Windows

Neues Setup ausführen – es aktualisiert das Programm und behält Daten und Lizenz. Vorher ein Backup des Datenverzeichnisses anlegen (Kapitel 43). **Deinstallation:** über *Einstellungen* → *Apps* → *Apps & Features* (bzw. *Systemsteuerung* → *Programme und Funktionen*) → „OPC Bridge" → **Deinstallieren**; das Datenverzeichnis unter `%ProgramData%\OPCBridge\` bleibt dabei standardmäßig erhalten.

44.3 Debian/Ubuntu

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

Das Paket aktualisiert das Programm; Daten unter `/var/lib/opcbridge/` bleiben erhalten. Die laufende Version lässt sich mit `dpkg -s opcbridge | grep Version` oder über die Oberfläche prüfen.

44.4 Docker

```
docker compose pull
docker compose up -d
```

Neues Image beziehen und Container neu erstellen – das benannte Volume `opcbridge-data` (Kapitel 8) bleibt davon unberührt bestehen.

44.5 Rückweg (Downgrade)

Die Bridge selbst bietet **keine** automatische Downgrade-Funktion – der Ein-Klick-Updater installiert ausschließlich die als *neuer* gemeldete Version. Für einen Rückweg auf eine ältere Version:

- **Windows:** das ältere `OPCBridge-Setup-<alte-version>.exe` manuell ausführen.
- **Debian/Ubuntu:** `sudo apt install ./opcbridge_<alte-version>_amd64.deb` (bzw. `dpkg -i`).
- **Docker:** ein älteres Image-Tag referenzieren und `docker compose up -d` erneut ausführen.

Legen Sie vor einem Downgrade ein Backup des Datenverzeichnisses an (Kapitel 43) und prüfen Sie nach dem Rückweg, ob die Konfigurationsdatei vom neueren Format noch verstanden wird – im Zweifel eine zuvor gesicherte ältere `config.json` zurückspielen.

44.6 Migration auf anderen Rechner

Datenverzeichnis kopieren (inkl. `machine_id`, falls die Lizenzbindung übernommen werden soll – siehe Kapitel 43), Programm installieren. Ohne übernommene `machine_id` bzw. bei echtem Hardwaretausch ist eine **neue Lizenz** anzufordern (rechnergebunden).

---

45. 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.

---

46. 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	Passwort unbekannt	Dienst stoppen, in `config.json` den Abschnitt `"web": {"users": [...]}` auf `"users": []` leeren, Dienst wieder starten – die Bridge legt dann wie beim allerersten Start automatisch einen neuen `admin` mit Zufallspasswort an und schreibt es erneut nach `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 „BadSecurityChecksFailed"	Zertifikat nicht vertraut	Zertifikate gegenseitig vertrauen (Truststore) oder Policy anpassen.
S7-Verbindung schlägt fehl	falsches Rack/Slot, falsche Verbindungsart,	Rack/Slot gemäß CPU-Typ prüfen (S7-1200/1500: 0/1 · S7-300: 0/2); Verbindungsart `pg` / `op` / `basic`

Symptom	Mögliche Ursache	Lösung
	Firewall/Port 102, `python-snap7` fehlt	testen; Port 102 (RFC1006) freigeben; Log auf „snap7 nicht installiert" prüfen.
PKI-Zertifikat eines OPC-UA-Servers abgelehnt	Zertifikat (noch) nicht vertraut	Landet automatisch im `rejected`-Ordner der Verbindung; über *Verbindungen* bzw. `GET /api/pki/{conn}` prüfen und per `POST /api/pki/{conn}/trust` in den Trust-Store verschieben – wirksam nach „Speichern & übernehmen"/ Neustart der Verbindung.
Lizenzlimit erreicht – Verbindung startet nicht	Demo- oder Lizenzlimit an Verbindungen überschritten	Im Log erscheint „Verbindung … nicht gestartet: Lizenz erlaubt max. N Verbindungen"; betroffene Verbindung deaktivieren/löschen oder Lizenz mit höherem Verbindungslimit aktivieren (Kapitel 41). `GET /api/status` / `/api/license` zeigt `blocked_connections`.
2FA-Benutzer im Modus „Basic" kann sich nicht anmelden	HTTP-Basic kennt keinen zweiten Faktor	Die Seite selbst lädt trotzdem; für API-Zugriffe über `/api/login` mit Benutzername, Passwort **und** TOTP-Code anmelden und den erhaltenen Bearer-Token verwenden (Kapitel 37.2/37.6) – reines Basic-Login genügt nicht.
Änderung wirkt nicht	nicht übernommen	„Speichern & übernehmen" klicken.
E-Mail/Webhook kommt nicht	SMTP/URL falsch	SMTP-Daten im Reiter **Alarme** (nicht Einstellungen!) 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).
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.
		Lizenz aktivieren (Kapitel 41) – ein weiterer Neustart verlängert die

Symptom	Mögliche Ursache	Lösung
Demo stoppt nach 60 Minuten – endgültig, auch nach Neustart	keine Lizenz; die einmalige Demo-Zeit ist bereits abgelaufen	Demo-Zeit **nicht**, sie läuft einmalig seit dem allerersten Start.
Weiterleitung stoppt plötzlich im laufenden Betrieb	Lizenz während des Betriebs abgelaufen/ widerrufen (Offline- oder Online-Prüfung)	Es gibt keine Gnadenfrist mehr (Kapitel 41.4); Lizenzstatus über *Lizenz* bzw. `GET /api/license` prüfen und erneuern.
Dienst startet nicht (Linux)	Startfehler/ungültige config.json	`journalctl -u opcbridge -f`; config.json aus Backup.
Dienst startet nicht (Windows)	Startfehler, ungültige config.json, Port belegt	`service.log` und `opcbridge.log` im Datenverzeichnis (`%ProgramData%\OPCBridge\`) prüfen (Anhang F); ggf. config.json aus Backup; Port-Konflikt (z. B. 8080) prüfen.
Windows-Dienst lässt sich nicht starten/stoppen	Dienst hängt, Berechtigungsproblem	`services.msc` öffnen, Dienst „OPCBridge" prüfen; als Administrator ausführen; `service.log` auf Fehler der Startumgebung prüfen.
Ein-Klick-Update unter Debian schlägt mit Hinweis auf „NoNewPrivileges" fehl	systemd-Unit verhindert jede Rechteausweitung inkl. sudo	Erwartetes Verhalten (Kapitel 44.1) – Update manuell auf der Konsole einspielen: `sudo apt-get install -y <paket>` bzw. `sudo apt install ./opcbridge_<version>_amd64.deb`.

Teil VII – Anwendungsbeispiele

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

47. 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`.

---

48. 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 Byteorder 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.**

---

49. 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.**

---

50. 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	`+`, `-`, `*`, `/`, `//` (Ganzzahldivision), `%` (Modulo), `**` (Potenz)
Vergleich/Logik	`>`, `<`, `>=`, `<=`, `==`, `!=`, `and`, `or`, `not`
Bitoperatoren	`&` (UND), `|` (ODER), `^` (XOR), `~` (NOT), `<<`, `>>` (Schieben)
Runden	`round(x, n)`, `floor(x)`, `ceil(x)`, `int(x)`, `float(x)`, `bool(x)`, `str(x)`, `abs(x)`

Kategorie	Funktionen / Operatoren
Begrenzen/ Skalieren	`clamp(x, min, max)`, `scale(x, in_min, in_max, out_min, out_max)`
Min/Max/Summe/ Mittel	`min(a, b, …)`, `max(a, b, …)`, `sum(a, b, …)`, `avg(a, b, …)` (= `mean`)
Nullwerte behandeln	`coalesce(a, b, …)` – liefert den ersten Wert, der nicht `None` / `null` ist
Bedingung	`x if Bedingung else y` (Ternär-Ausdruck) sowie `ifelse(Bedingung, a, b)`
Logische Hilfsfunktionen	`and_(a, b, …)` (alle wahr), `or_(a, b, …)` (mindestens eine wahr)
Mathematik	`sqrt(x)`, `pow(x, y)`, `exp(x)`, `log(x)` (natürlich), `log10(x)`, `sin`, `cos`, `tan`
Konstanten	`pi`, `e`, `true` / `false` / `True` / `False`, `None` / `null`
Länge	`len(x)` (z. B. für Zeichenketten)

Beispiele: `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>0, x, 0)`, `avg(a, b, c)`, `coalesce(a, b, 0)`, `(x & 0xFF00) >> 8`.

> »**Hinweis**« Der Formel-Interpreter ist aus Sicherheitsgründen auf eine feste Liste von Operatoren und Funktionen beschränkt (kein Attributzugriff, keine Schleifen, kein Import, kein Dateizugriff, keine Systemaufrufe) – es lässt sich also kein beliebiger Code ausführen.

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		OIDC-/SSO-Anmeldung

Methode	Pfad	Zweck
	`/api/oidc/start`, `/api/oidc/callback`	
GET	`/api/meta`	Produkt-/Versionsinfo
GET	`/api/status`	Status aller Verbindungen/Mappings
GET	`/api/values`	aktuelle Werte
GET	`/api/log`	Log-Meldungen
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, Geheimnisse beim GET maskiert)
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/opcda/servers`	installierte OPC-DA-Server auflisten (Windows)
POST	`/api/can/analyze`	CAN-Analyse
POST	`/api/can/dbc`	DBC-Import (Admin)
GET/POST/DELETE	`/api/license`	Lizenzstatus / aktivieren / entfernen
POST	`/api/license/online/activate`	Online-Lizenzprüfung/-Aktivierung jetzt ausführen (Admin)
GET	`/api/update`	Update-Status (verfügbare Version, installierbar?)
POST	`/api/update/install`	Ein-Klick-Installation der gemeldeten Version (Admin)
GET	`/api/pki/{conn}`	PKI-Status einer OPC-UA-Verbindung (eigenes Zertifikat, trusted/rejected)
POST	`/api/pki/{conn}/trust`	abgelehntes Zertifikat in den Trust-Store verschieben (Admin)

Methode	Pfad	Zweck
POST	`/api/pki/{conn}/delete`	Zertifikat aus trusted/rejected löschen (Admin)
GET	`/api/pki/{conn}/own_cert.der`	eigenes Zertifikat der Verbindung herunterladen
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
`uint64`	64-Bit ohne Vorzeichen	sehr groß	4
`float32`	32-Bit Gleitkomma (IEEE 754)	ca. $\pm 3{,}4 \cdot 10^{38}$	2
`float64`	64-Bit Gleitkomma (IEEE 754)	hohe Präzision	4
`stringN`	Text (N Zeichen)	je nach Länge	n (2 Zeichen/Register)

> »**Hinweis**« `double` ist kein gültiges Kürzel – verwenden Sie `float64`.

E. Standard-Ports

Dienst	Port	Hinweis
Webinterface / REST	8080	konfigurierbar
OPC-UA-Server	4840	wenn aktiviert
Modbus-Slave	502	wenn aktiviert; im mitgelieferten `docker-compose.yml` auf Host-Port **1502** veröffentlicht
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	benanntes Volume `opcbridge-data` (Mountpoint `/data` im Container)	`/data/initial_admin_password.txt`

Im Datenverzeichnis liegen u. a.:

Datei/Ordner	Inhalt
`config.json`	Konfiguration
`license.key`	Offline-Lizenz (Seriennummer)
`history.sqlite` (konfigurierbar)	Historien-Datenbank
`sf_queue.sqlite`	persistente Store-and-Forward-Warteschlange
`online_state.json`	Zustand der Online-Lizenzprüfung
`update_state.json`	Zustand der Update-Prüfung
`machine_id`	persistierte Maschinenkennung (Rechnerbindung, wichtig in Docker)
`demo_started_at.txt`	Zeitpunkt des allerersten Starts (Demo-Zeitfenster)

Datei/Ordner	Inhalt
`pki/`	OPC-UA-Zertifikate je Verbindung (eigenes Zertifikat, trusted/rejected)
`opcbridge.log`	rotierendes Anwendungslog
`service.log`	nur Windows: Standardausgabe/-fehler des Windows-Dienstes
`initial_admin_password.txt`	Erststart-Passwort (nach erstem Login löschen)

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)

Begriff	Bedeutung
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 43.

OPC Bridge – Technisches Betriebshandbuch Version 1.1.0 · © 2026 **Biricon IT Services GmbH** · www.biricon.eu

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