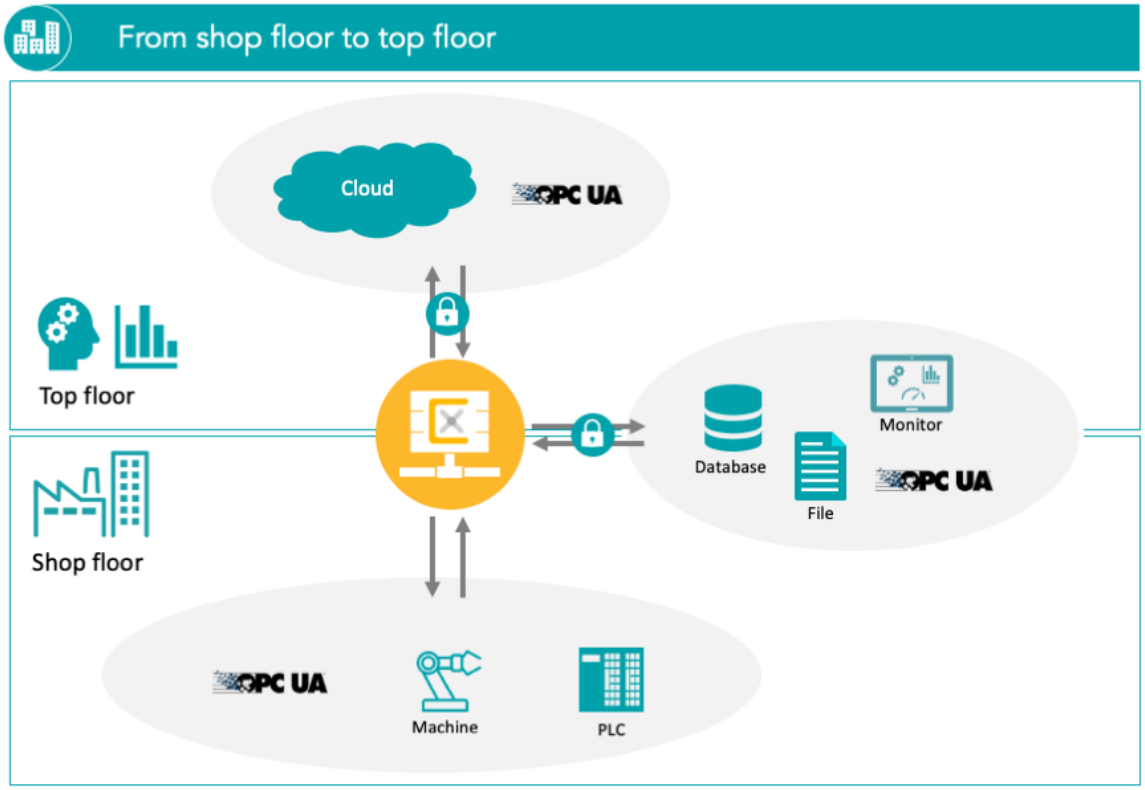
Inhaltsverzeichnis

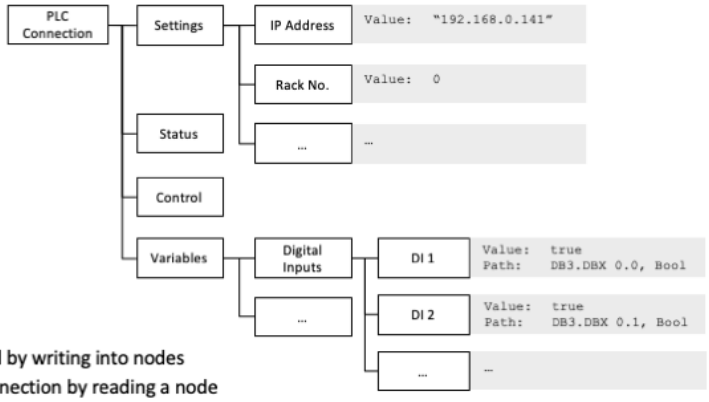
UKI-4.0UKI-4.0UKI-4.0 UKI-4.0UKI-4.0UKI-4.0 ® - Die Middleware für Industrie 4.0



The industrial Middleware for any type of connection.

Unified architecture over all interfaces

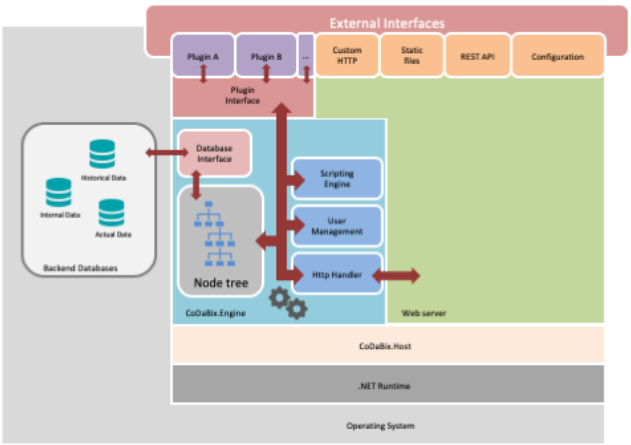
- Everything is a node
  - Only thing to handle is a node
  - Interface will not change
  - Completely mappable to **OPC UA**



- Examples
  - Controlling a connection
    - Settings are stored in nodes
      - Addresses
      - Timeouts
      - ...
    - Starting and stopping is performed by writing into nodes
    - Getting the current status of a connection by reading a node
  - Accessing process data
    - A PLC variable is represented by a node
  - Remote-Procedure-Call on the server
    - A Method node is called

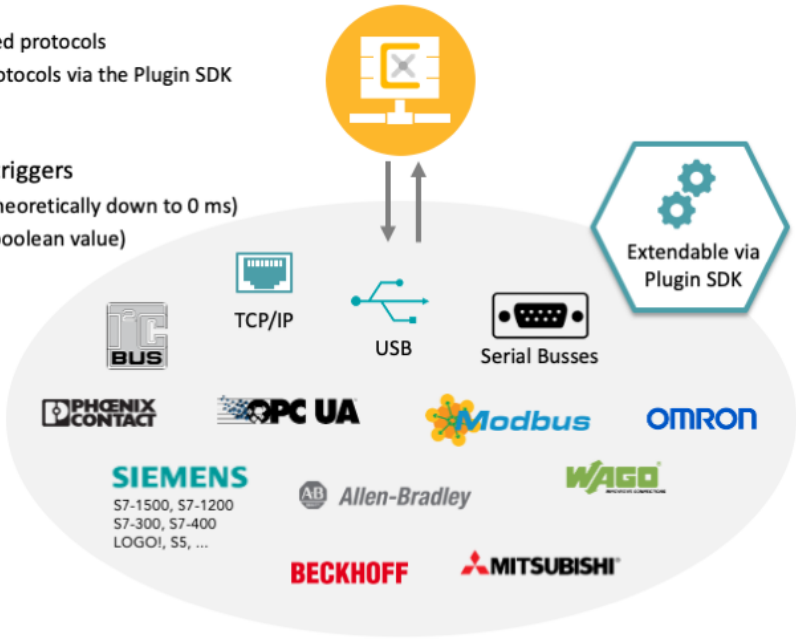
System architecture

- Modular plugin system
  - Easily extendable
  - No need to restart the system
  - Resource saving
- Integrated web server
  - Remote configuration via web technology
  - Deployment of web apps
  - Serving static files
- Database backend
  - Configuration storage
  - Historical data
- Process automation & customization
  - Online scripting engine
  - User-defined node structure



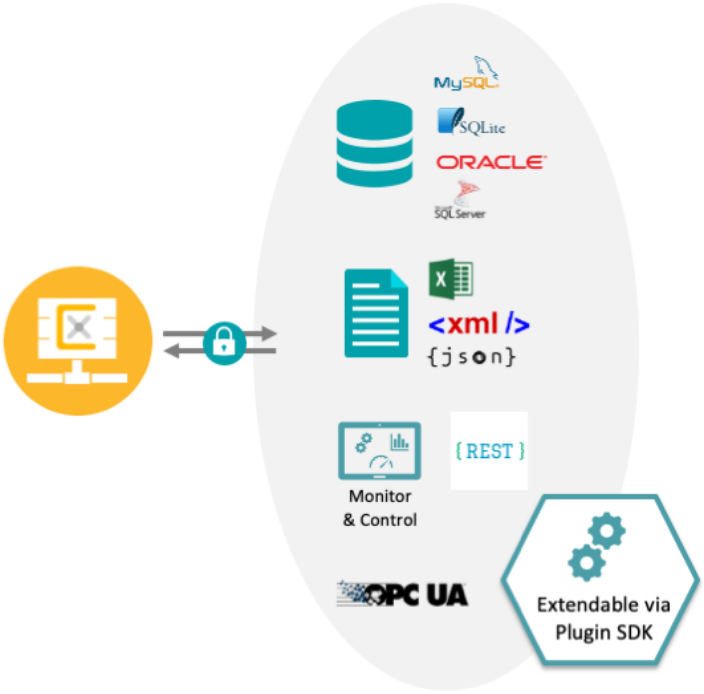
Connect to production machines and sensors

- Connect to any device
  - Plenty of already implemented protocols
  - Extendable by proprietary protocols via the Plugin SDK (upcoming feature)
- Read real time data based on triggers
  - Timer trigger (sample rates theoretically down to 0 ms)
  - Event trigger (e.g. edge of a boolean value)
  - Conditional trigger
- Create historical data
  - Integrated database
  - Snapshot a set of values
  - On value change or trigger based



Synchronize, exchange and monitor data

- Horizontal data exchange for e.g.
  - Synchronizing workstations
  - Sharing data between MES and machines
- Export to and import from
  - Existing database
  - Structured file
- Monitor and control data in real time
  - CoDaBix® Dashboard
  - Custom HMI based on REST JSON API

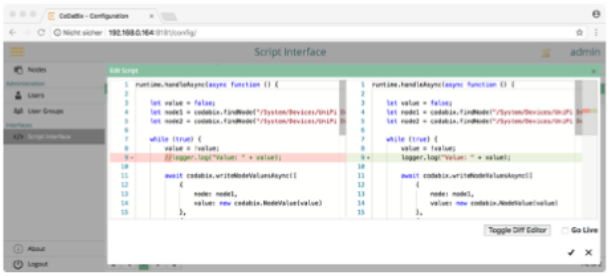


Access cloud services

- Publish your data
  - Convert data into required format
  - Execute on triggers
- Remote monitoring
  - Interpret data in real time and publish results
- Remote control
  - Fetch and validate data from cloud
  - Execute user-defined function (Remote Procedure Call)

Automate and customize with built-in Scripting Engine

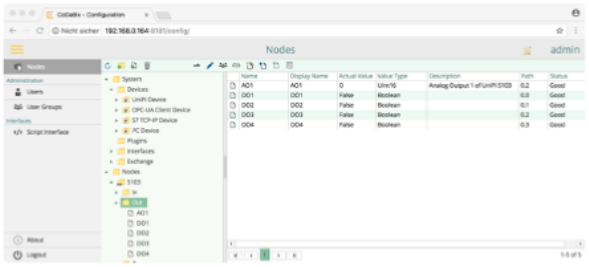
- TypeScript programming language
  - Standard JavaScript libraries available
  - Interface for node access
  - Compiles to .NET Intermediate Language at runtime
- Online editor
  - IntelliSense
    - Syntax highlighting
    - Tooltips
    - Autocompletion
  - Diff view



- Use cases
  - Create conditional triggers
  - Process data
  - Automate the control of machines and processes
  - Export data to files
  - Add custom functionality

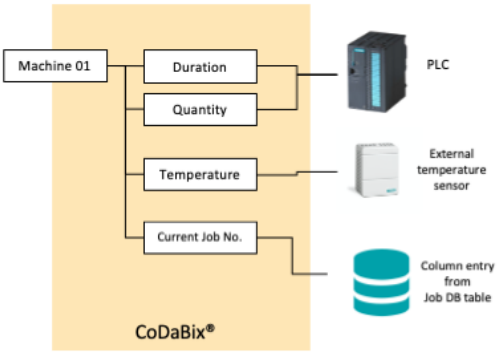
Configuration and administration

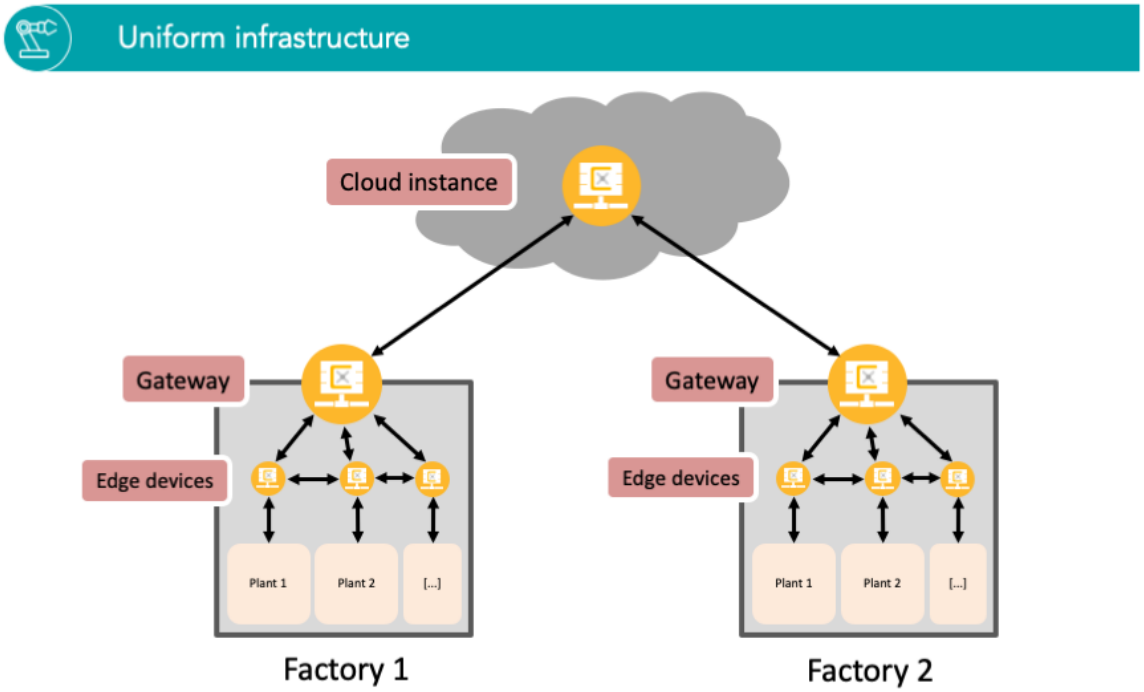
- Configuration via web interface
  - Remote access
  - Browser based
  - No compatibility issues
- Node management
  - Create node links
  - User defined node structure
  - Import and export configuration as XML
- Access control
  - User groups management
  - Configurable for every subtree
- Operation of CoDaBix®
  - Execution as system service
  - Backup functionality



Unify, harmonize and extend interfaces

- Node structuring and linking
  - Design interfaces according to requirements
  - Restructure machine data
  - Aggregate data from various sources
  - Decouple interface from underlying data source
- Custom node action handler
  - Implement virtual machine nodes
  - Handle data conversion and scaling on the fly
  - Create notifications on definable conditions

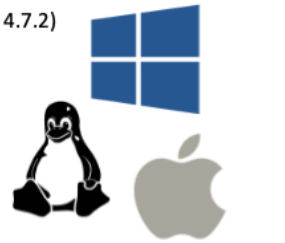


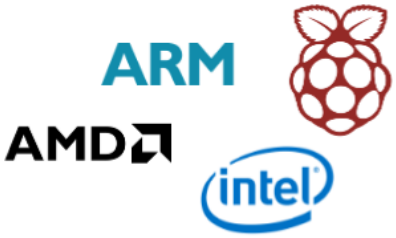

Uniform infrastructure
Cloud instance
Gateway
Edge devices
Plant 1
Plant 2
[...]
Factory 1
Gateway
Edge devices
Plant 1
Plant 2
[...]
Factory 2

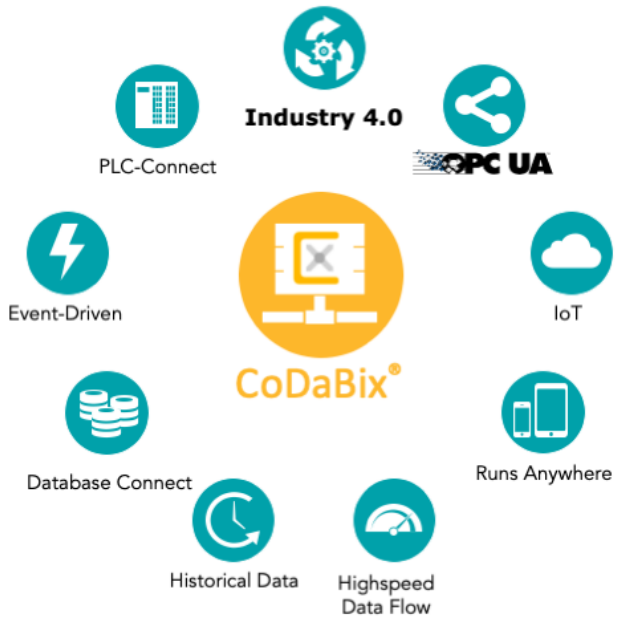
- Data collection
  - Collect and buffer data locally
  - Bundle and publish data
  - By-pass connection breakdowns
- Remote Management
  - Configure and manage plugins
  - Roll out updates
  - Configure Operating System properties
- Private Cloud
  - Keep your data local
  - Integrated historical database
  - Access data via interfaces
    - REST JSON API
    - OPC UA Server
    - MQTT

Supported systems

- Operating Systems
  - Windows 7 SP1, Windows 8.1, Windows 10 (with .NET Framework 4.7.2)
  - Windows Server 2008 R2 and upwards
  - Every OS supported by the .NET Core Runtime
    - Linux
      - Red Hat Enterprise Linux
      - CentOS
      - Oracle
      - Fedora
      - Debian
      - Ubuntu
      - Mint
      - openSUSE
      - Alpine Linux
    - Mac OS 10.13 and upwards
    - Docker Container
- Hardware
  - Recommended: Dual-Core CPU, 4 GB RAM
  - Runs on Raspberry Pi 2, 3 and 4
  - ARM32, ARM64, x86, x64 platforms







Previous Next

Was ist UKI-4.0 ? - Wo ist der Einsatz?

UKI-4.0 ist eine plattformunabhängige I4.0 Middleware, die zur Datenkommunikation, -erfassung und -verarbeitung verwendet wird.
Es beherrscht die wichtigsten proprietären Kommunikationsprotokolle, die im industriellen Umfeld im Einsatz sind.
Damit lassen sich SPS-Familien neuer und alter Generationen von Siemens, Mitsubishi, Allen Bradley, Bosch, Beckhoff, AEG, OMRON, WAGO und Geräte mit OPC Classic(DCOM) und OPC UA anbinden. Ebenso können Datenbanken (SQL, ...) und Dateien (CSV, XML, ...) integriert werden. Die Kommunikation ist immer bidirektional möglich.
Sämtliche Verbindungen, Einstellungen und Maschinendaten stehen in abstrahierter Form als Node-Graph zur Verfügung. Unabhängig von Maschinentyp und Hersteller können Variablen in Bezeichnung und Struktur je nach Anforderung der Anwendung harmonisiert und normiert werden. Alle Daten lassen sich adhoc per OPC UA Server und REST/JSON veröffentlichen.
UKI-4.0 lässt sich als Datensammler, aber auch als Kommunikations-Gateway einsetzen. Die Anbindung an MES, PPS, ERP ist gleichermaßen wie an I4.0 Clouds z.B. per OPC UA, MQTT realisierbar.
Mit Hilfe von Scripting lassen sich event-basiert und flexibel Datenerfassung, -verdichtung und Automationen implementieren (z.B. OEE, Rule Sets, ...).
UKI-4.0 kann sowohl auf Edge-Devices (z.B. Raspberry Pi) an der Maschine, auf einem Server im Rechenzentrum oder als Docker-Container in der Cloud ausgeführt werden. Die Konfiguration erfolgt per Web-Interface oder REST bzw. mit XML-Strukturen.

UKI-4.0 ® - Produktpräsentation

Plugins für Devices / Interfaces und Data Exchange

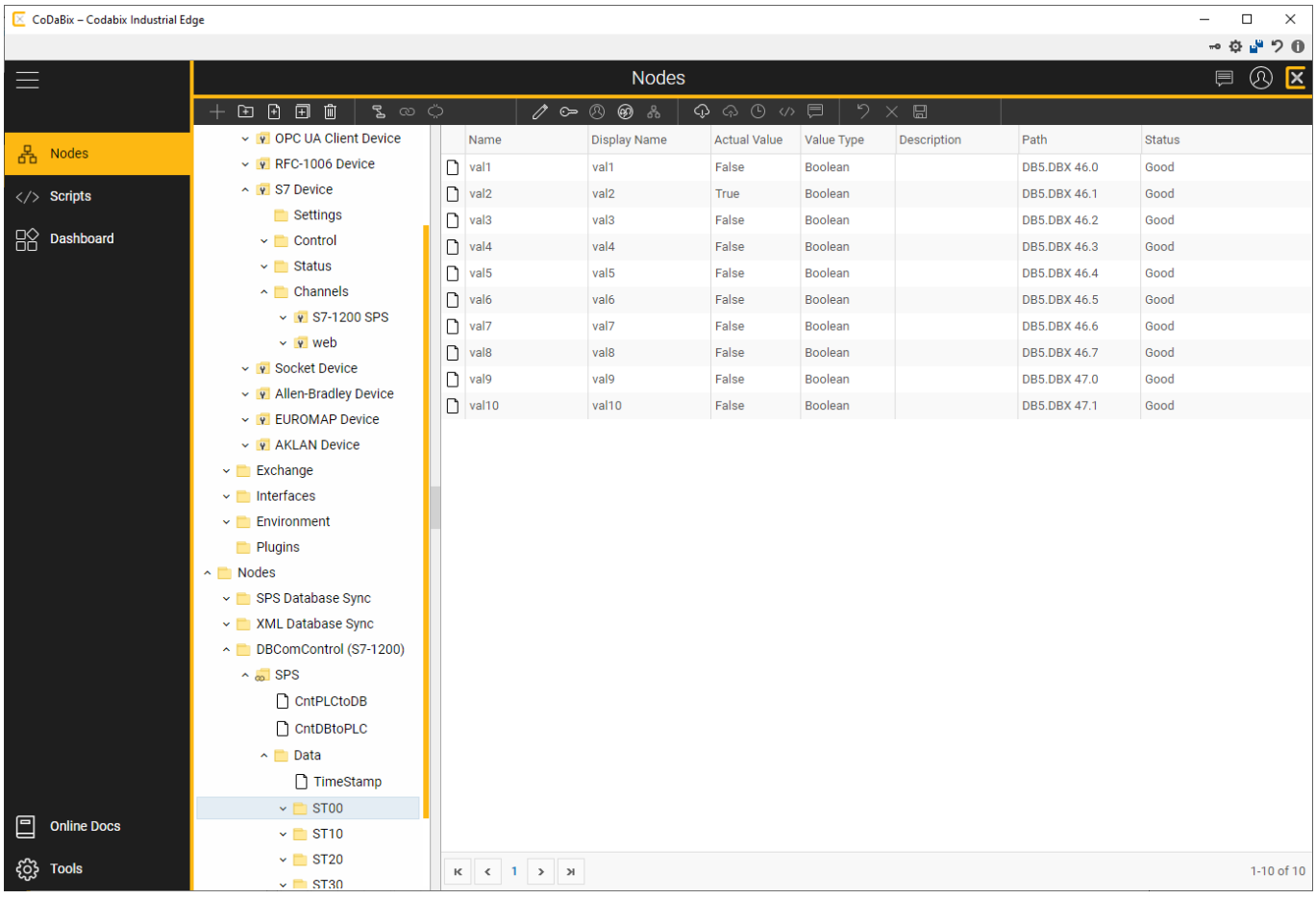
- OPC UA Server and Client
- OPC DA DCOM
- MQTT
- WEB Anwendung via REST/JSON Interface
- Online Scripting (Typescript)
- Device Treiber
    - Siemens SIMATIC S7 (importieren Sie die Variablen direkt aus Ihrem STEP7-Projekt)
    - Siemens SIMATIC S5
    - Siemens 3964R
    - SINUMERIK SL and PL
    - RFC-1006 (ISO on TCP)
    - Siemens SINEC H1
    - Siemens SINEC L1
    - Allen Bradley - Rockwell
    - ModBus
    - WAGO
    - EUROMAP 63
    - EUROMAP 77
    - EUROMAP 84

    - Beckhoff
    - Bosch (CL200, 300, 400, 500)
    - Schneider
    - Mitsubishi (Q-Serie, A, FX)
    - Omron (TCP and Serial)
    - UniPi (I2C)
    - AEG (A120/A250)
    - Phoenix PLC NEXT
    - RAW Socket TCP

- Datenbanken
    - MySQL
    - MariaDB
    - Microsoft SQL
    - Oracle SQL
    - PostgreSQL
    - ODBC Data bases
    - SQLite

- CSV / XML / Text Files
- SAP
- AKLAN (Audi)

Konfiguration



Die Konfiguration von UKI-4.0 ® erfolgt über das integrierte Webinterface.
Grundsätzlich ist die Parametrierung über ein XML-Config-File möglich. Das Format der XML-Config-Files ist frei verfügbar und entsprechend dokumentiert.

So lässt sich UKI-4.0 ® auf einfachem Weg mit vorhandenen Projektplanungs- / Parametrierungs-Applikationen (z.B. COMOS) kombinieren beziehungsweise verbinden.

Voraussetzungen

Betriebssysteme

- Windows
  - Workstation Windows 7/8/10 (32/64 Bit)
  - Server Windows Server 2008 R2/2012/2016/2019
- Linux
  - Debian 9 (Stretch) oder höher (x64, ARM64, ARM32)
    inklusive Derivate wie Raspberry Pi OS (für Raspberry Pi)
  - Fedora 32 oder höher (x64)
  - Ubuntu 18.04 oder höher (x64, ARM64, ARM32)
  - OpenSUSE Leap 15.0 oder höher (x64)

Plattformen

UKI-4.0 ist auf nahezu jeder Plattform lauffähig. Das kann ein Server / Desktop oder IPC mit Windows oder Linux sein.
Ebenso kommt Raspberry Pi (z.B. UniPI, KUNBUS), Siemens IOT2050, DELL-Edge Gateways oder ähnliche Plattformen in Frage.
UKI-4.0 kann auch im Docker betrieben werden. Die Anforderungen an CPU-Leistung, Arbeitsspeicher und Festplatte sind von gewünschtem Datendurchsatz und Datenvolumen abhängig.

Features

Unterstützte Geräte & Protokolle

- AKLAN
- Mitsubishi Melsec
- Modbus TCP
- OPC UA Client
- RAW TCP-Socket Messaging
- SIMATIC H1
- SIMATIC S7
- UniPi Neuron

Fließender Austausch von Daten

Interfaces & APIs

- OPC UA Server
- REST / JSON API
- Script Interface
- Integrierter Webserver

Dateien lesen und schreiben

Unterstützte Dateiformate:

- CSV
- XML
- JSON
- anwender-spezifische Formate mittels Script Interface

Individuelle Anpassung

- Script Interface
- REST / JSON API
- Benutzerdefinierte Datenstruktur

Monitoring und Visualisierung von

Produktionsdaten

- Integrierte historische Datenbank
- REST / JSON API
- Integrierter Webserver

Administration

Sicherheit

- Sicherer/Verschlüsselter Zugriff auf Webserver über HTTPS (TLS-Protokoll)
- Verschlüsselung von Passwörtern
- Zugriffsmanagement durch User und User-Groups
- Sandbox-Ausführung von UKI-4.0 -Daten möglich durch Einschränkung der Zugriffsrechte in den Settings (z.B. File Access Security)
    - Schützt Benutzerdateien und Meta-Settings wie Zugangsdaten der Backend-Datenbank oder Webserver-SSL-Zertifikate vor Zugriff durch Scripts oder Plugins

Konfiguration

- Backup und Wiederherstellen der gesamten Konfiguration
- Automatisieren von Aufgaben mittels REST / JSON API und Script Interface
- Export und Import der Konfiguration per XML-Dateien
- Datenstruktur vollständig individualisierbar

Installation von UKI-4.0UKI-4.0UKI-4.0 UKI-4.0UKI-4.0UKI-4.0

UKI-4.0 ist für folgende Betriebssysteme und Hardwarekonfigurationen verfügbar:

- UKI-4.0 **für Windows** (x64/x86)
- UKI-4.0 **für Linux** (x64/ARM64/ARM32)

UKI-4.0UKI-4.0 UKI-4.0UKI-4.0 für Windows

Sie können UKI-4.0 sowohl auf Workstation- als auch auf Serverversionen von Windows installieren. UKI-4.0 läuft sowohl auf 32-Bit- als auch auf 64-Bit-Versionen von Windows. Aus Stabilitätsgründen **empfehlen wir** allerdings, die **64-Bit**-Version von UKI-4.0 zu verwenden.

Systemanforderungen

Unterstützte Betriebssysteme

UKI-4.0 **für Windows** wird auf folgenden Betriebssystemen unterstützt:

Windows-Version	Workstation-Betriebssystem	Server-Betriebssystem
6.1.7601	Windows 7 SP1 [1)]	Windows Server 2008 R2 SP1 [2)] (Option „Vollständige Installation“)
6.2.9200	Windows 8 (Windows Embedded 8 Standard)	Windows Server 2012 (Option „Server mit grafischer Benutzeroberfläche“)
6.3.9600	Windows 8.1	Windows Server 2012 R2 (Option „Server mit grafischer Benutzeroberfläche“)
10.0.14393	Windows 10 [IoT] Enterprise 2016 LTSB	Windows Server 2016 (Option „Server mit Desktopdarstellung“)
10.0.16299	Windows 10 Version 1709 (Fall Creators Update)	
10.0.17134	Windows 10 Version 1803 (April 2018 Update)	
10.0.17763	Windows 10 [IoT] Enterprise 2019 LTSC	Windows Server 2019 (Option „Server mit Desktopdarstellung“, oder „Server Core“ mit installiertem Server Core App Compatibility FOD)
	Windows 10 Version 1809 (October 2018 Update)	Windows Server, Version 1809 (mit installiertem Server Core App Compatibility FOD)
10.0.18363	Windows 10 Version 1909 (November 2019 Update)	Windows Server, Version 1909 (mit installiertem Server Core App Compatibility FOD)
10.0.19041	Windows 10 Version 2004 (May 2020 Update)	Windows Server, Version 2004 (mit installiertem Server Core App Compatibility FOD)
10.0.19042	Windows 10 Version 20H2 (October 2020 Update)	Windows Server, Version 20H2 (mit installiertem Server Core App Compatibility FOD)
10.0.19043	Windows 10 Version 21H1	Windows Server, Version 21H1 (mit installiertem Server Core App Compatibility FOD)

Hardwareanforderungen

Empfohlen: 64-Bit Quad-Core CPU, 8 GB RAM

Anforderungen für die Backend-Datenbank

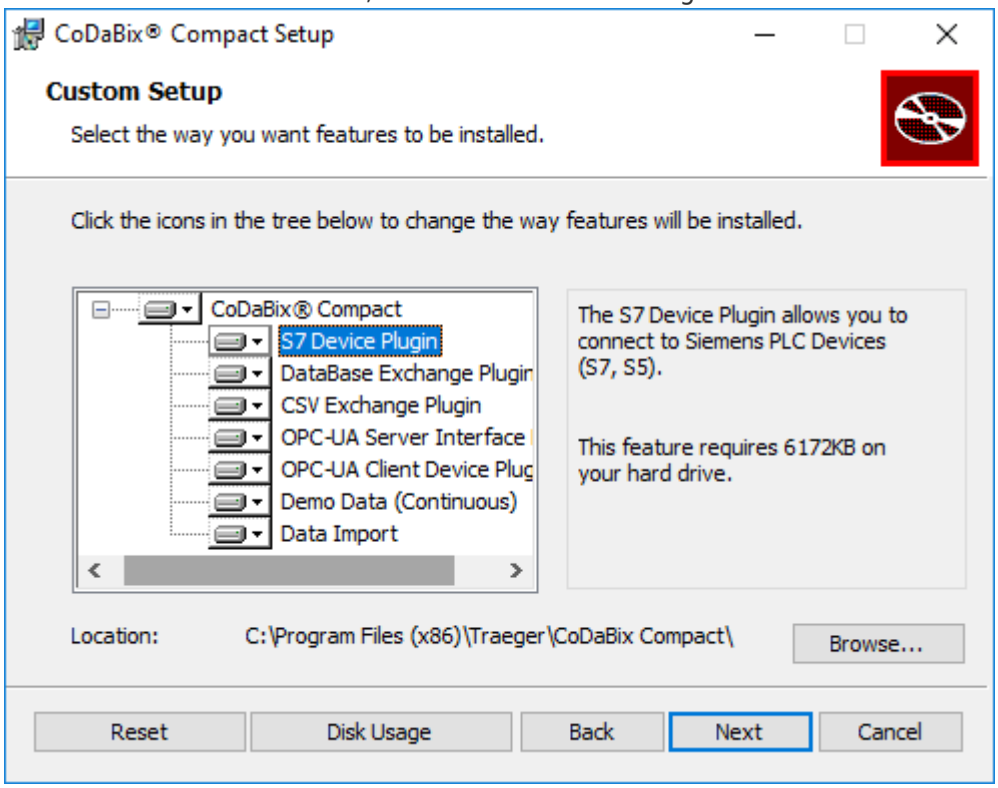
Standardmäßig benutzt UKI-4.0 eine **eingebettete Datenbank (SQLite)**, welche keine weiteren Anforderungen stellt.

Wenn Sie hingegen planen, MySQL, MariaDB oder Microsoft SQL Server als Backend-Datenbank zu nutzen, stellen Sie bitte sicher, dass Sie MySQL 8.0 oder höher, MariaDB 10.3 oder höher, bzw. Microsoft SQL Server 2012 oder höher verwenden.
Wir empfehlen, MySQL, MariaDB bzw. MSSQL auf der **selben Maschine** wie UKI-4.0 auszuführen.

UKI-4.0UKI-4.0 UKI-4.0UKI-4.0 Setup und erster Start

UKI-4.0 Setup

Um UKI-4.0 zu installieren, laden Sie bitte den UKI-4.0 Installer (MSI-Datei) herunter und führen ihn aus. Sie können dabei auswählen, welche Devices und Plugins mit UKI-4.0 installiert werden sollen.



Installieren und Deinstallieren von Plugins

Wenn Sie Ihre UKI-4.0 -Installation zu einem späteren Zeitpunkt ändern möchten (z. B. um Plugins hinzuzufügen oder zu entfernen), starten Sie einfach den UKI-4.0 Installer erneut (alternativ können Sie auch in der Windows-Systemsteuerung auf „Programme und Features“ gehen, dann „UKI-4.0 “ wählen und auf „Ändern“ klicken).
Sie können dann im Installer auf „Change“ klicken, um die Plugins von UKI-4.0 auszuwählen, die installiert oder deinstalliert werden sollen.

UKI-4.0 updaten

Wenn Sie bereits UKI-4.0 installiert haben und auf eine neuere Version updaten möchten, müssen Sie nicht extra die alte Version deinstallieren. Führen Sie einfach den UKI-4.0 Installer der neueren Version aus und dieser wird automatisch UKI-4.0 auf die neue Version updaten.

Hinweis: Wenn Sie UKI-4.0 als Dienst installiert hatten, müssen Sie nach dem Update den Dienst im UKI-4.0 Settings-Dialog erneut installieren.

Erster Start

Sie können UKI-4.0 starten, indem Sie auf die UKI-4.0 -Verknüpfung 

am Desktop doppelklicken, oder folgendes Kommando auf der Kommandozeile ausführen (z.B. unter Windows Server Core):

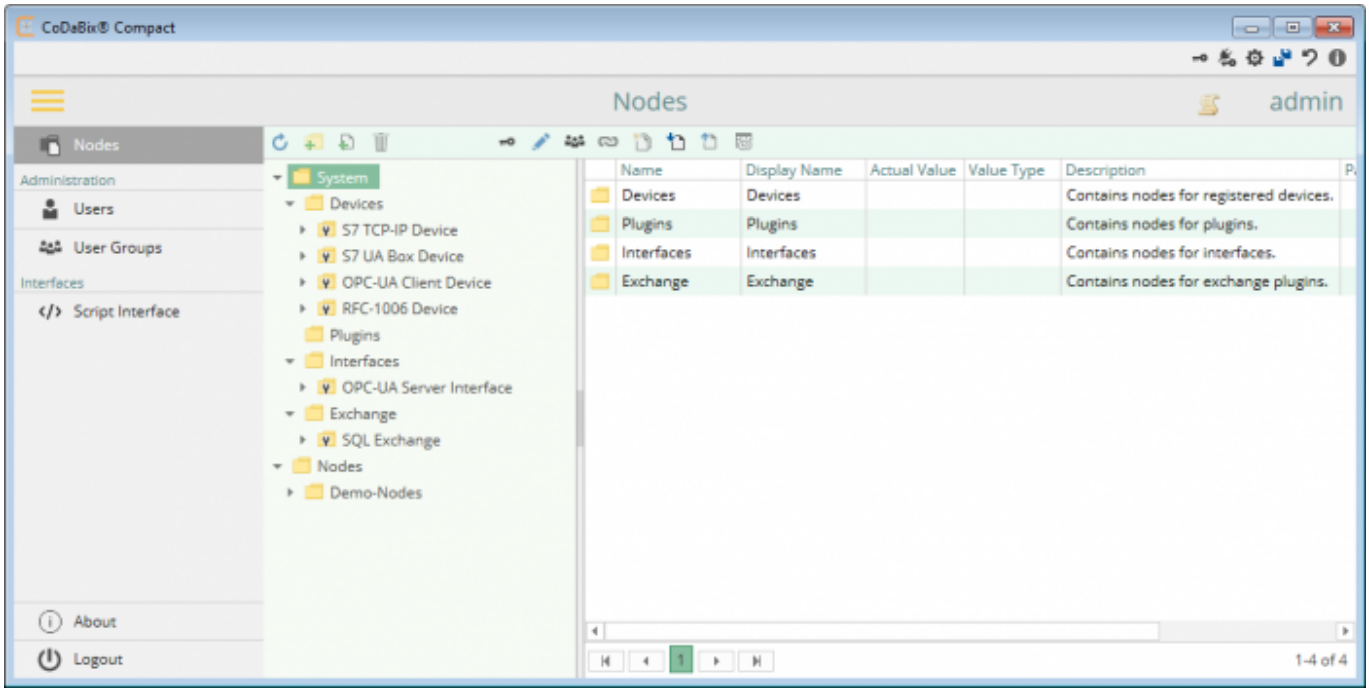
```
"%ProgramFiles%\TIS\UKI-4.0 \UKI-4.0 -ui.exe"
```

Wenn Sie UKI-4.0 das erste Mal starten, werden Sie zur Auswahl eines **Projektverzeichnisses** aufgefordert (idealerweise ein leerer Ordner), in welchem UKI-4.0 seine Daten (Einstellungen, Datenbank, Konfiguration, Logdateien usw.) abspeichern soll.

Nachdem Sie die Auswahl bestätigt haben, erscheint der **Settings-Dialog** (siehe nächster Abschnitt). Klicken Sie bitte auf OK, um die Einstellungen anzuwenden. Nun werden Sie aufgefordert, ein Admin-Passwort zu setzen. Dieses benötigen Sie später für die Webkonfiguration.

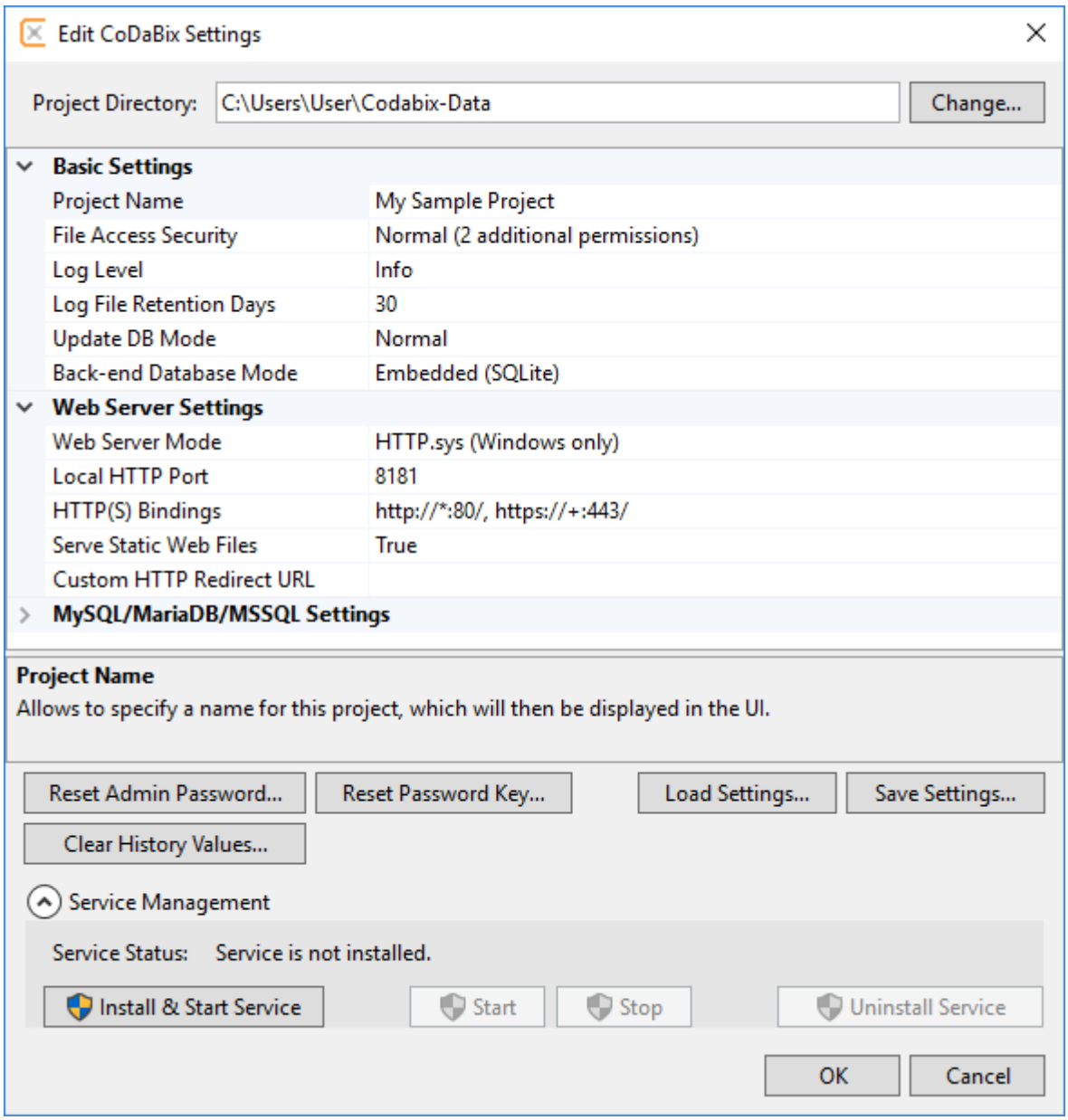
Hinweis: Möglicherweise wird ein Dialog der Windows Firewall angezeigt, in dem Sie gefragt werden, ob der Zugriff auf UKI-4.0 erlaubt werden soll. Dies kommt vom OPC UA Server Plugin, welches standardmäßig einen OPC UA Server auf Port 4840 erstellt. Wenn Sie auf „Zugriff erlauben“ klicken, können andere Geräte im Netzwerk auf den OPC-Server zugreifen.

Nachdem Sie das Passwort gesetzt haben und UKI-4.0 gestartet ist, erscheint der Login-Dialog der Webkonfiguration. Hier können Sie sich nun mit dem Benutzernamen **admin** und Ihrem zuvor gesetzten Passwort einloggen.



UKI-4.0UKI-4.0 UKI-4.0UKI-4.0 Projekteinstellungen

UKI-4.0 bietet Einstellungen für das ausgewählte Projekt an, die Sie konfigurieren können. Um diese zu editieren, klicken Sie auf das Zahnrad-Icon in der Symbolleiste oben rechts (⚙), über das sich der UKI-4.0 Settings-Dialog öffnet.



Basic Settings

Name	Beschreibung
Project Directory	Das Projektverzeichnis ist das Verzeichnis, in dem UKI-4.0 die Projekteinstellungen (alle der folgenden Einstellungen), die **Backend-Datenbank** (wenn *Back-end Database Mode* auf „Embedded (SQLite)“ eingestellt ist), Logdateien und Konfigurationsdateien für die Plugins ablegt. Es kann über die Umgebungsvariable `%`UKI-4.0 `ProjectDir%` abgerufen werden, z.B. in Scripts. Es enthält folgende Ordner: • `log`: Enthält die UKI-4.0 Runtime-Logdateien und Logdateien des Entity Models. • `plugins`: Enthält Konfigurationsdateien für Plugins. • `webfiles`: In diesem Ordner können Sie statische Dateien ablegen, die über den UKI-4.0 Webserver für mit `/webfiles/` anfangende URLs abrufbar sein sollen, falls die Einstellung *Serve Static Web Files* aktiviert ist. • `dashboard`: Ähnlich wie `webfiles`, aber für URLs die mit `/dashboard/` anfangen, wodurch das eingebettete Dashboard außer Kraft gesetzt und ein eigenes verwendet werden kann. • `userdata`: In diesem Ordner können Sie benutzerdefinierte Dateien ablegen (z.B. für die Verwendung in Scripts). Bei diesem Ordner ist garantiert, dass er in zukünftigen UKI-4.0 - Versionen nicht anderweitig verwendet wird. Beim Erstellen eines **Backups** werden die Inhalte dieser Unterordner im Backup mit abgelegt.

Name	Beschreibung
Project Name	Ermöglicht Ihnen, einen Namen für das aktuelle Projekt festzulegen, welcher in der Titelleiste der UKI-4.0 -Anwendung angezeigt wird und für den Standard-Dateinamen beim Erstellen eines Backups verwendet wird.
Access Security	Innerhalb von UKI-4.0 kann die Projektlogik auf Dateien Ihres Systems zugreifen, z.B. durch die Verwendung von File Nodes (auf die dann per HTTP-Access-URL oder über einen OPC UA Client zugegriffen werden kann), oder durch ein Script , das die `io.file`- und `io.directory`-Namespaces benutzt. Um einen Benutzer, der das UKI-4.0 -Adminpasswort hat (oder Scripts oder File-Nodes in UKI-4.0 erstellen kann), daran zu hindern, auf beliebige Dateien zuzugreifen (vor allem wenn UKI-4.0 als Dienst läuft), können Sie Zugriff auf bestimmte Pfade erlauben oder verweigern. Außerdem können Sie alternative Zugangsdaten angeben, die beim Zugriff auf den Pfad benutzt werden sollen (dies geschieht über Impersonation), oder alternativ den Pfad als Netzwerkresource hinzufügen. Standardmäßig ist der Level `Normal` gesetzt, welcher die Unterordner `plugins`, `log`, `userdata`, `webfiles`, `dashboard` innerhalb des **Projektverzeichnisses** für Lese- und Schreibzugriff zulässt.
Log Level	Gibt das Logging-Detaillevel an, das UKI-4.0 beim Schreiben von Logdateien benutzen soll. Es werden nur solche Einträge geschrieben, deren Schweregrad mindestens so hoch ist wie das angegebene Level. 'No' bedeutet, dass keine Logdateien erstellt werden.
Back-end Database Mode	Gibt an, welche Backend-Datenbank benutzt werden soll. Standardmäßig verwendet UKI-4.0 eine eingebettete Datenbank (SQLite), welche keine zusätzliche Konfiguration benötigt. Die eingebettete Datenbank wird in den Dateien UKI-4.0 `.db` und UKI-4.0 `historydb.db` im eingestellten Projektverzeichnis gespeichert. Sie können jedoch auch MySQL 8.0 oder höher, MariaDB 10.3 oder höher bzw. Microsoft SQL Server 2012 oder höher als Backend-Datenbank verwenden. Wenn Sie dementsprechend „MySQL/MariaDB“ oder „Microsoft SQL Server“ auswählen, müssen Sie die weiteren Einstellungen unter der Kategorie „MySQL/MariaDB/MSSQL Settings“ ausfüllen. Beim Erstellen eines **Backups** werden die Inhalte der Backend-Datenbank im Backup mit abgelegt. Sie können auch die aktuelle Backend-Datenbank von SQLite nach MySQL/MariaDB/MSSQL (oder von MySQL/MariaDB/MSSQL nach SQLite) migrieren, indem Sie ein Backup erstellen, den Database Mode umstellen und anschließend das Backup wiederherstellen.

Web Server Settings

Name	Beschreibung
Web Server Mode	Gibt den Modus des eingebetteten Webservers an, den UKI-4.0 zum Bereitstellen der Webkonfiguration, des REST/JSON Interfaces, der registrierten Script-HTTP-Handler sowie optional statischer Dateien verwendet. **Kestrel** ist über Socket-APIs implementiert und wird standardmäßig verwendet, da dieser generell die beste Performance bietet. Bei der Verwendung von HTTPS müssen SSL-Zertifikate als PFX- (PKCS #12)-Dateien (`.pfx`, `.p12`) bereitgestellt werden und werden in der UKI-4.0 -Settings-Datei abgelegt. Die **Windows HTTP Server API** wird unter Windows 10 Version 1607/Windows Server 2016 und höher unterstützt, und ermöglicht das Teilen von Ports mit anderen IIS-Websites. Bei der Verwendung von HTTPS müssen SSL-Zertifikate im Windows-Zertifikatsspeicher abgelegt werden (siehe unten). Um Remote HTTP Bindings nutzen zu können, muss UKI-4.0 als Dienst installiert werden.

Name	Beschreibung
Local HTTP Port	Gibt den Port an, den der eingebettete Webserver zum Lauschen auf lokale Verbindungen verwendet. Standardmäßig benutzt UKI-4.0 den Port **8181**. **Hinweis:** Wenn Sie **Windows HTTP Server API** als *Web Server Mode* verwenden und Ports ≤ 1024 (z. B. Port 80) verwenden möchten oder die UKI-4.0 Webkonfiguration oder das REST/JSON Interface auf anderen Geräten im Netzwerk aufrufen möchten, müssen Sie UKI-4.0 als Dienst installieren (siehe unten). Für Letzteres müssen Sie außerdem die „Service HTTP(S) Bindings"-Optionen öffnen und dort die Option „Use the Local Port as remote HTTP Binding" aktivieren. Anderenfalls sind nur Ports > 1024 möglich und auf UKI-4.0 kann über HTTP nur vom lokalen PC aus zugegriffen werden.
Service HTTP(S) Bindings	Sie können HTTP- und HTTPS-Bindings, bestehend aus Hostnamen, Port und SSL-Zertifikat (für HTTPS), für Remoteverbindungen aktivieren. Dies ermöglicht Ihnen, eine Verbindung zu UKI-4.0 von fremden Geräten herzustellen, optional über eine authentifizierte und verschlüsselte **TLS-Verbindung**. **Hinweis:** Um ein SSL-Zertifikat für den Webserver-Modus **Windows HTTP Server API** verwenden zu können, muss dieses im **Personal**- oder im **Web Hosting**-Zertifikatsspeicher des Kontos *Lokaler Computer* in Windows gespeichert sein und Sie müssen einen *privaten Schlüssel* für das Zertifikat besitzen. Zwischenzertifikate (auch CA-Zertifikat genannt) müssen im **Intermediate Certificate Authorities**-Zertifikatsspeicher gespeichert sein. Um Zertifikate unter Windows zu verwalten, können Sie MMC verwenden, indem Sie `certlm.msc` starten und dann den Ordner *Eigene Zertifikate* (bzw. *Webhosting*) für Ihr Zertifikat sowie *Zwischenzertifizierungsstellen* für Ihre Zwischenzertifikate öffnen; oder Sie können Powershell verwenden, wenn Sie zum Pfad `Cert:\LocalMachine\My` (oder `Cert:\LocalMachine\WebHosting`) für Ihr Zertifikat sowie `Cert:\LocalMachine\CA` für Ihre Zwischenzertifikate wechseln.
Serve Static Web Files	Falls aktiviert, liefert der eingebettete Webserver statische Dateien aus, die in dem `webfiles`-Ordner des Projektverzeichnisses platziert werden. Dies ermöglicht Ihnen in Kombination mit dem neuen Script-HTTP-Handler-Feature (siehe unten), Webapps zu entwickeln, die durch UKI-4.0 ausgeliefert werden (z.B. zur Visualisierung).
Custom HTTP Redirect URL	Wenn nicht angegeben, werden Request für den Root-Pfad (`/`) zur UKI-4.0 Webkonfiguration weitergeleitet. Alternativ können Sie eine eigene Weiterleitungs-URL angeben. Das ist nützlich, wenn Ihre Benutzer beim Aufruf der UKI-4.0 -Adresse im Browser direkt Ihre eigene Webapp zu sehen bekommen sollen, die durch den eingebetteten Webserver ausgeliefert wird.

MySQL/MariaDB/MSSQL Settings

(nur anwendbar wenn *Back-end Database Mode* auf „MySQL/MariaDB" oder „Microsoft SQL Server" gestellt ist)

Name	Beschreibung
Hostname	Gibt den Hostnamen des MySQL-/MariaDB-/MSSQL-Servers an. Wenn MSSQL verwendet wird und der Port leer ist, wird der Hostname als **Data Source** interpretiert (z.B. `<ComputerName>\<InstanceName>`). Andernfalls ist dies der Hostname für TCP-Verbindungen.
Port	Gibt den TCP-Port des MySQL-/MariaDB-/MSSQL-Servers an. Dieser kann leer sein, wodurch der Standard-Port verwendet wird (für MySQL/MariaDB), oder um den Hostnamen als Data Source zu verwenden (für MSSQL).
Database Name	Gibt den Namen der Datenbank an, welche UKI-4.0 auf dem MySQL-/MariaDB-/MSSQL-Server benutzen soll. Wenn die angegebene Datenbank nicht existiert, legt UKI-4.0 diese sowie alle dazugehörigen Tabellen automatisch an.
Login Name	Gibt den Benutzernamen an, unter welchem UKI-4.0 die Datenbankverbindung herstellen soll.
Password	Gibt das zugehörige Passwort des Benutzers an.

UKI-4.0 als Dienst installieren

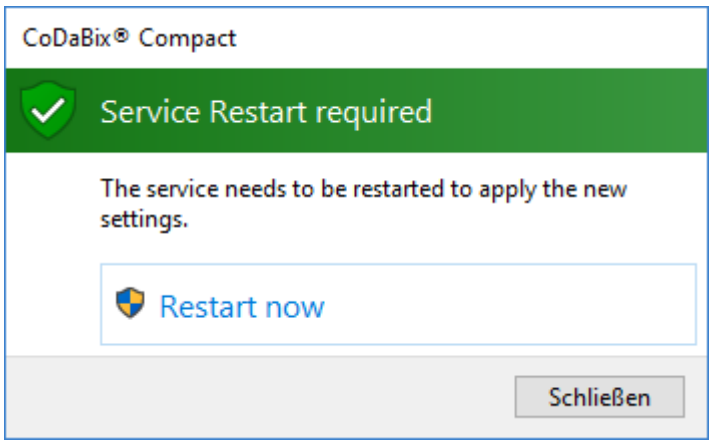
Sie können UKI-4.0 als Dienst installieren. Dadurch wird UKI-4.0 automatisch und dauerhaft im Hintergrund (wie auf einem Server) ausgeführt, ohne dass explizit die UKI-4.0 -Anwendung gestartet werden muss. Außerdem können dann andere Geräte im Netzwerk auf die UKI-4.0 -Instanz über HTTP (z. B. auf die Webkonfiguration oder das REST / JSON Interface) zugreifen, wenn Sie in den „Service HTTP(S) Bindings“-Optionen die Option „Use the Local Port as remote HTTP Binding“ aktivieren.
Das Ausführen als Dienst ermöglicht auch die Verwendung von HTTP-Ports ≤ 1024, z.B. Port 80.

Nachdem Sie den UKI-4.0 Settings-Dialog über das Zahnrad-Icon in der Symbolleiste oben rechts (⚙) geöffnet haben, zeigt dieser unten den Abschnitt „Service Management“ an, über den Sie UKI-4.0 als Dienst installieren, starten, stoppen und deinstallieren können:

Sie können UKI-4.0 als Dienst installieren und starten, indem Sie auf den Button „Install & Start Service“ klicken. In diesem Fall wird ein Dialog der Benutzerkontensteuerung (UAC) angezeigt, da für das Installieren von Diensten Administratorrechte erforderlich sind.

Um den Dienst wieder zu deinstallieren, klicken Sie auf den „Uninstall Service“-Button.

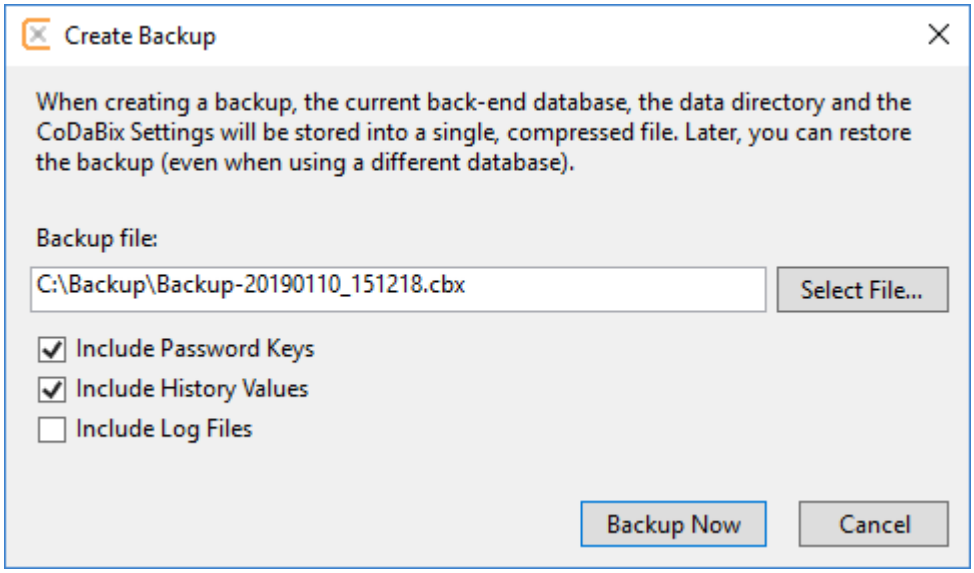
Hinweis: Wenn Sie eine Einstellung wie das Projektverzeichnis oder den HTTP-Port ändern, während UKI-4.0 als Dienst läuft, muss der Dienst neugestartet (oder neu installiert) werden, um die neuen Einstellungen anzuwenden. Dies wird durch den folgenden Dialog angezeigt, über den Sie den Dienst neustarten/neu installieren lassen können:



Hinweis: Der Dienst ist so konfiguriert, dass er im Fehlerfall automatisch neustartet. Dadurch wird sichergestellt, dass UKI-4.0 in einen sauberen, definierten Zustand neustarten kann, nachdem ein schwerwiegender Fehler aufgetreten ist.

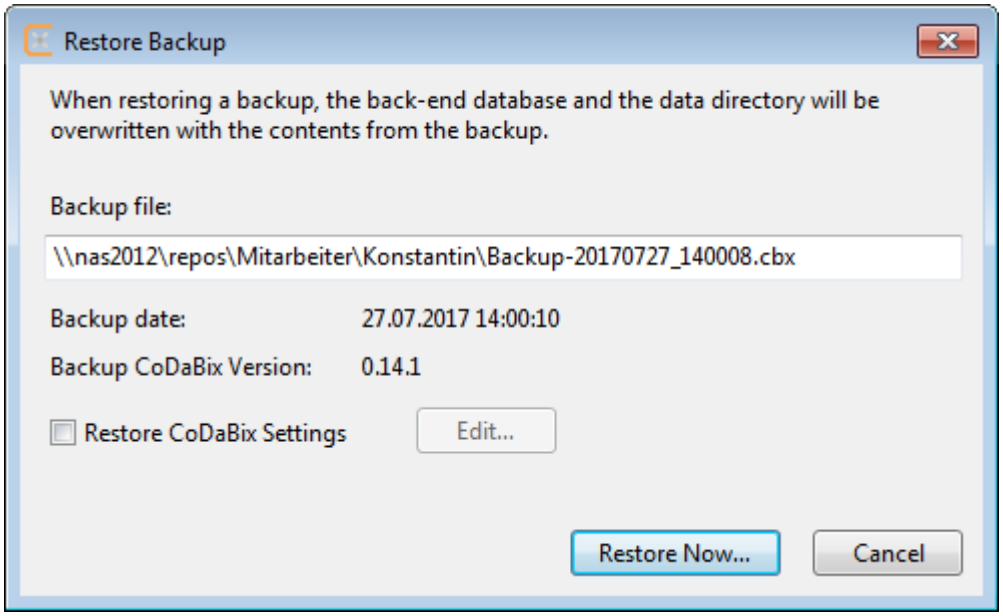
Backup erstellen

Beim Klick auf  erscheint folgender Dialog:



Backup wiederherstellen

Beim Klick auf  erscheint folgender Dialog:

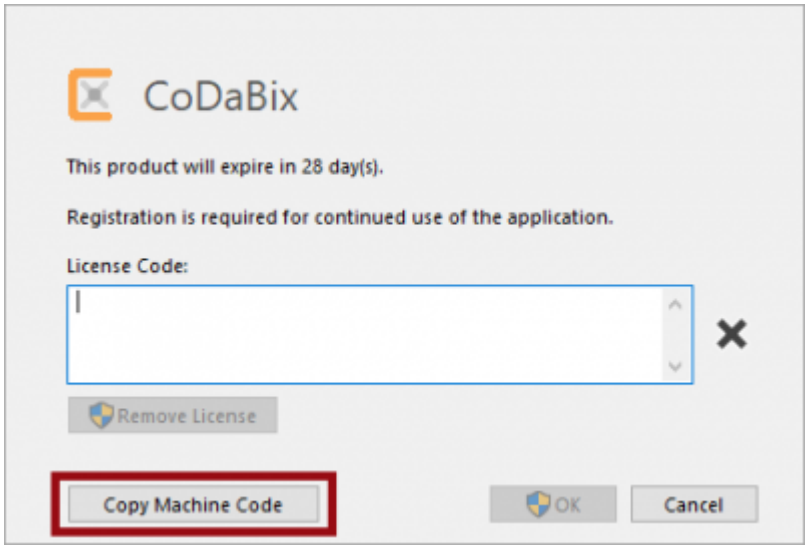


Lizenzverwaltung

Maschinencode

Wenn Sie eine Lizenz für UKI-4.0 bestellen, ist es notwendig, den Maschinencode derjenigen Maschine bereitzustellen, auf der UKI-4.0 läuft.

Um den Maschinencode zu erhalten, klicken Sie in der UKI-4.0 -Anwendung auf das Schlüsselsymbol ( ), um den Lizenzdialog zu öffnen. Klicken Sie anschließend auf den Button „Copy Machine Code“, um den lokalen Maschinencode in die Zwischenablage zu kopieren.



UKI-4.0UKI-4.0
UKI-4.0UKI-4.0 für Linux

Systemanforderungen

Unterstützte Betriebssysteme

UKI-4.0 **für Linux** wird auf den folgenden Linux-Distributionen unterstützt:

- Debian 9 (Stretch) oder höher (x64, ARM64, ARM32)
  - inklusive Derivate wie Raspberry Pi OS (für Raspberry Pi)
- Fedora 32 oder höher (x64)
- Ubuntu 18.04 oder höher (x64, ARM64, ARM32)
- OpenSUSE Leap 15.0 oder höher (x64)

Hardwareanforderungen

- **Raspberry Pi**:
  - UKI-4.0 für Linux (ARM64) kann auf einem Raspberry Pi 4 oder neuer (mit 64-Bit OS) ausgeführt werden.
  - UKI-4.0 für Linux (ARM32) kann auf einem Raspberry Pi 2 oder neuer ausgeführt werden.
  - Für optimale Performance empfehlen wir, UKI-4.0 (**ARM64**) auf einem **Raspberry Pi 4** (oder neuer) mit 4 GB oder 8 GB auf einem ARM64-OS auszuführen.
- Andere Maschinen:
  - Empfohlen: 64-Bit Quad-Core CPU (x64/ARM64), 8 GB RAM, 64-Bit OS und UKI-4.0

Anforderungen für die Backend-Datenbank

Standardmäßig benutzt UKI-4.0 eine **eingebettete Datenbank (SQLite)**, welche keine weiteren Anforderungen stellt.

Wenn Sie hingegen planen, MySQL, MariaDB oder Microsoft SQL Server als Backend-Datenbank zu nutzen, stellen Sie bitte sicher, dass Sie MySQL 8.0 oder höher, MariaDB 10.3 oder höher, bzw. Microsoft SQL Server 2012 oder höher verwenden.

Nicht unterstützte Features

Einige Features sind in UKI-4.0 für Linux nicht verfügbar. Dies beinhaltet:

- Windows-spezifische Plugins wie Melsec QJ Device Plugin, H1 Device Plugin, AKLAN Device Plugin
- Zugriff auf OPC Classic (COM)-Server über das OPC UA Client Device Plugin
- Impersonation für Dateien mit anderen Credentials (oder das Herstellen einer SMB-Netzwerkverbindung), z.B. für File-Nodes oder bei Benutzung des CSV Exchange Plugins
- GUI-Fenster (Sie können die UKI-4.0 Webkonfiguration in einem Browser öffnen, aber administrative Aufgaben wie das Einstellen des Projektverzeichnisses, Erstellen eines Backups usw. müssen in der UKI-4.0 -Shell-Konsolenanwendung ausgeführt werden)

Installieren von UKI-4.0UKI-4.0 UKI-4.0UKI-4.0 und erster Start

Installieren von UKI-4.0 für Linux

Um UKI-4.0 für Linux zu installieren, laden Sie die .setup-Datei herunter und führen Sie folgendes Kommando aus, um die Datei ausführbar zu machen:

```
chmod +x UKI-4.0-<platform>-<release-date>-<release-version>.setup
```

Starten Sie im nächsten Schritt das Setup:

```
sudo ./UKI-4.0-<platform>-<release-date>-<release-version>.setup
```

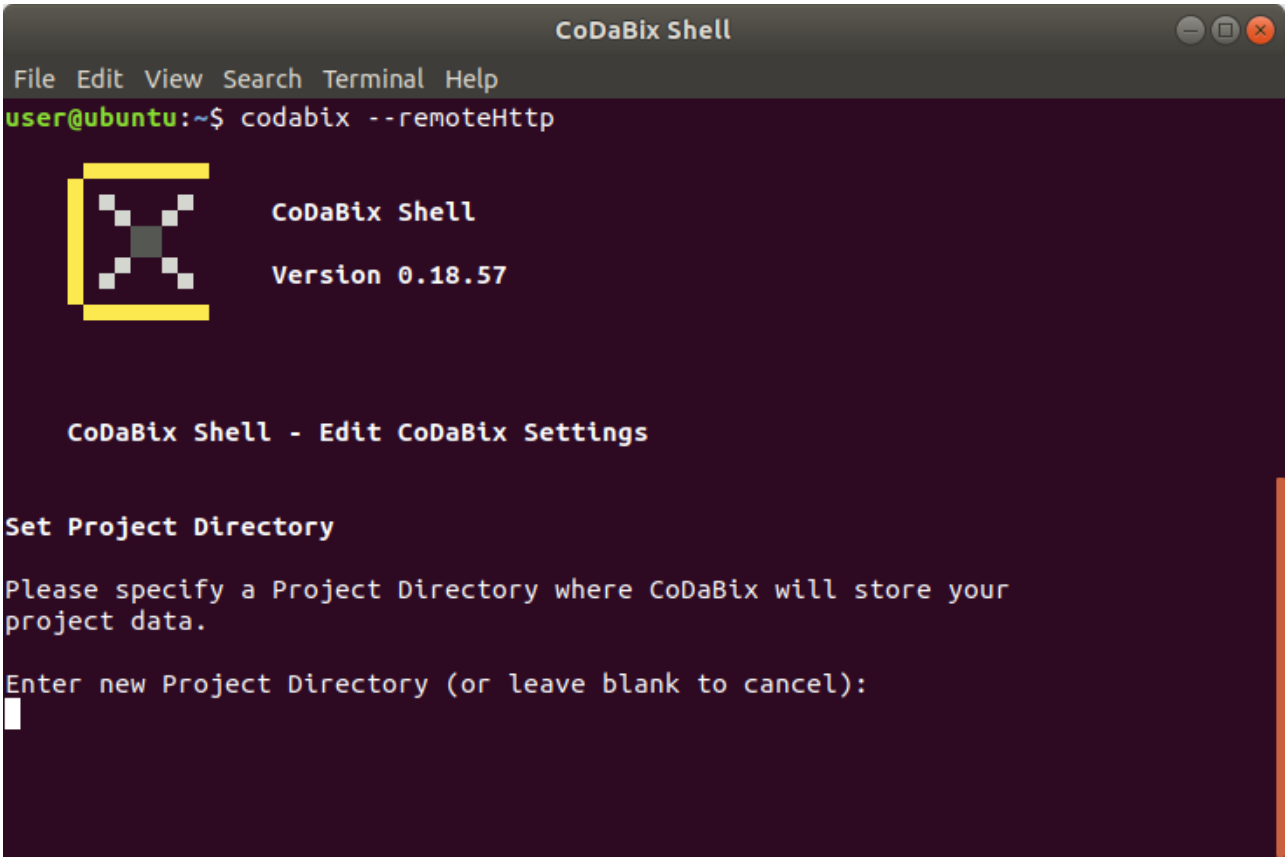
Das Setup leitet Sie durch die Installation.
Alte Installationen von UKI-4.0 werden nicht gelöscht, um bei Bedarf ein Roll-Back durchführen zu können. Das Backup, das für einen solchen Roll-Back notwendig ist, kann während des Setups erstellt werden.

Erster Start

Um die konsolenbasierte UKI-4.0 Shell zu starten, führen Sie das folgende Kommando aus:

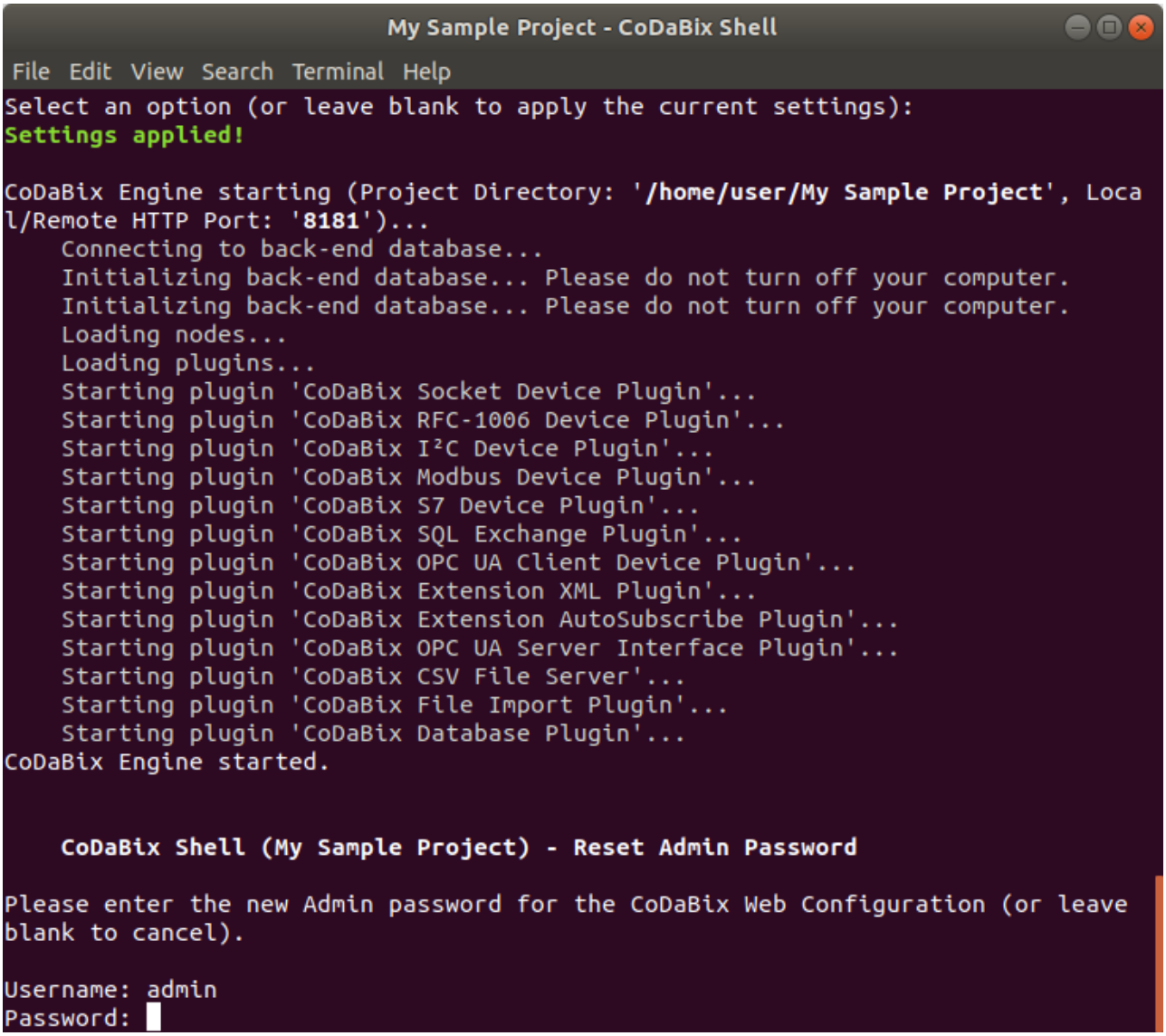
```
UKI-4.0
```

- UKI-4.0 sollte nun starten und nach einem Projektverzeichnis fragen:



- Geben den vollständigen Pfad ein, unter dem Sie das UKI-4.0 -Projektverzeichnis ablegen wollen (normalerweise ein Ordner in Ihrem Home-Verzeichnis) (siehe UKI-4.0 Projekteinstellungen). Drücken Sie anschließend zweimal Enter, um die Einstellungen anzuwenden und UKI-4.0 neuzustarten.

- UKI-4.0 bittet Sie nun darum, ein neues Admin-Passwort zu setzen; bitte geben Sie hier ein neues Passwort ein:



- Anschließend können Sie `http://localhost:8181/config/` in einem Browser auf Ihrer Maschine (wenn Ihre Linux-Distribution mit einer GUI installiert wurde) aufrufen, um die UKI-4.0 - Webkonfiguration zu öffnen, oder rufen Sie `http://<IP-Addresse>:8181/config/` in einem Browser auf einer anderen Maschine auf, um die Web-Konfiguration übers Netzwerk zu öffnen.

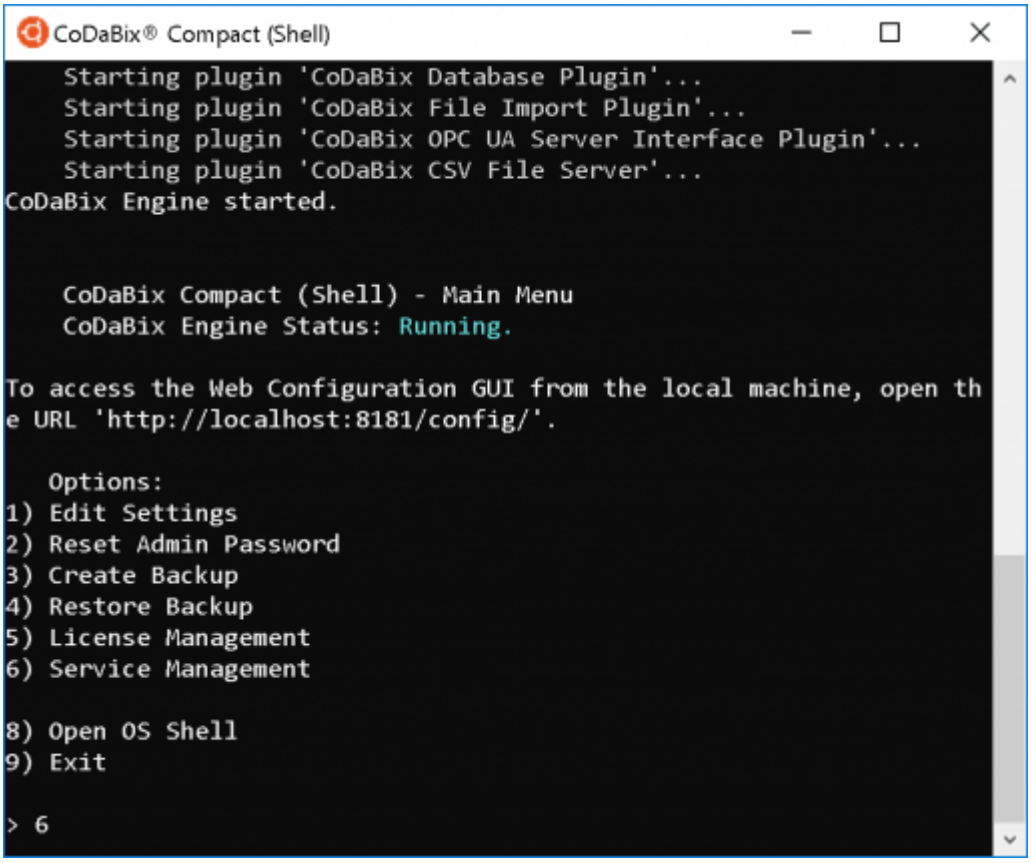
Verwenden von UKI-4.0UKI-4.0 UKI-4.0UKI-4.0 als Service

UKI-4.0 als Service installieren

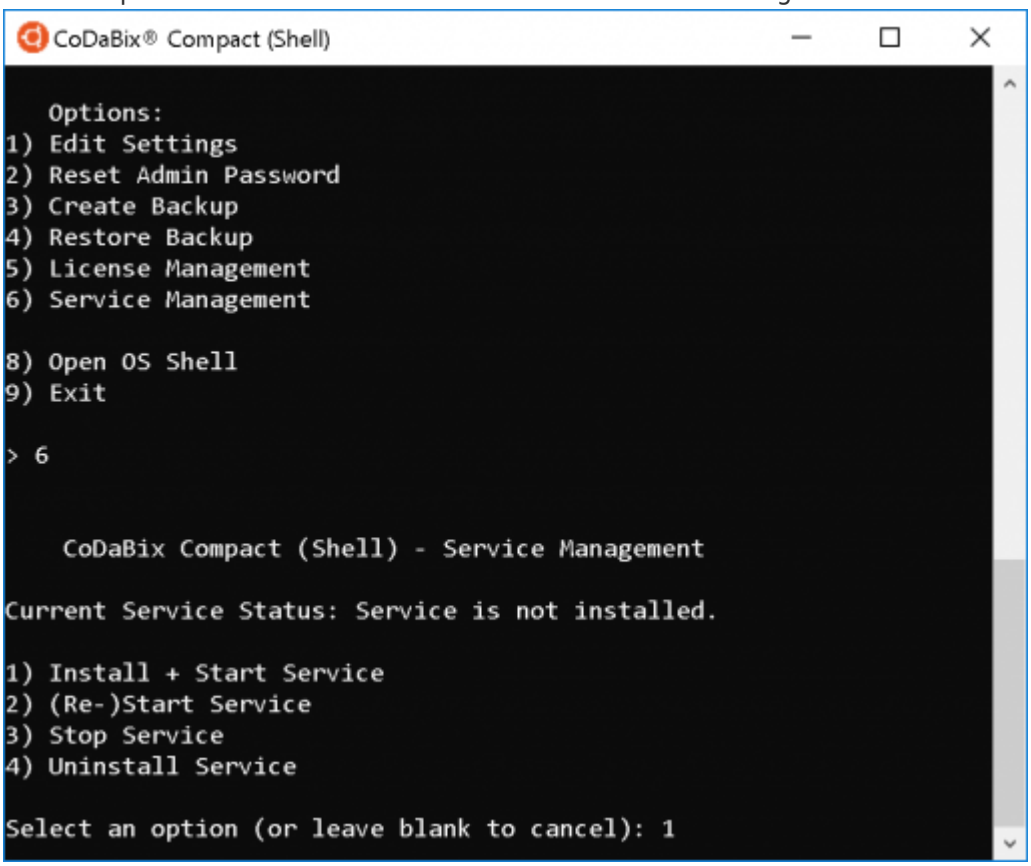
Um UKI-4.0 als Service zu installieren, starten Sie die UKI-4.0 Shell mit folgendem Kommando:

```
UKI-4.0
```

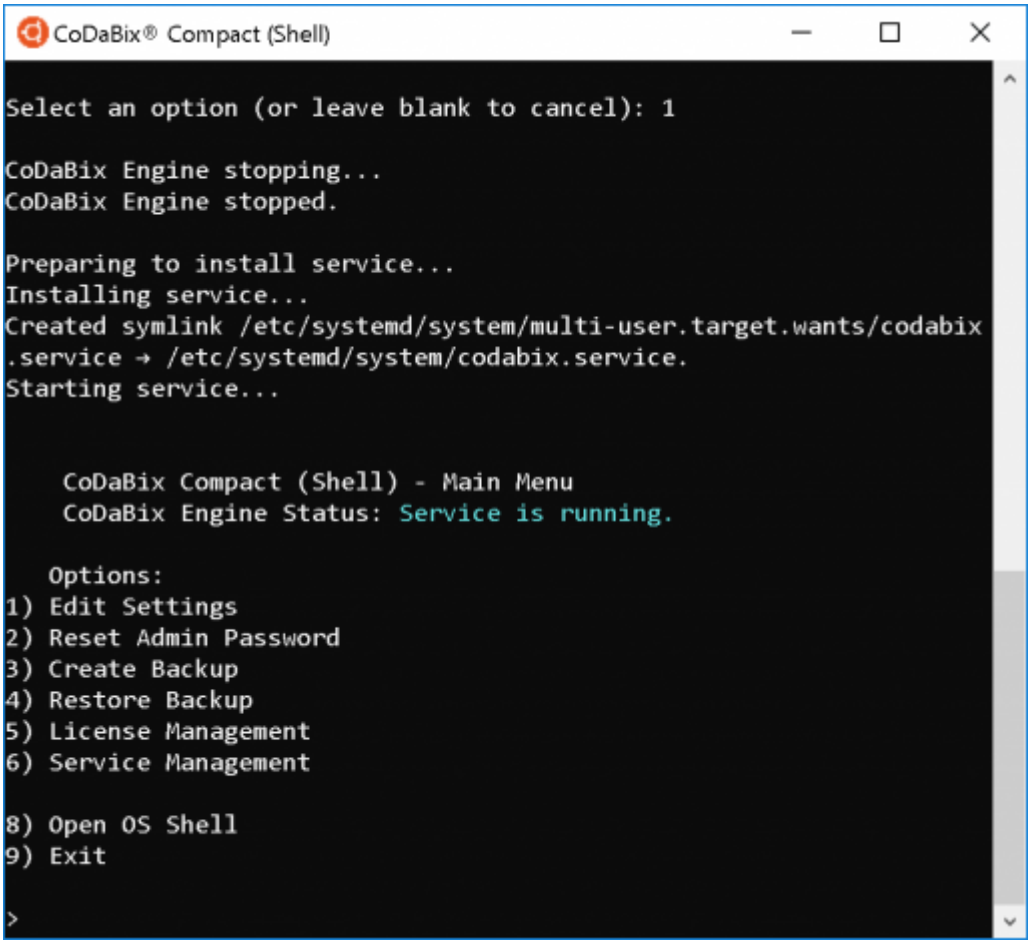
- Wählen Sie im Kommandozeilen-Menü die Option **6**, um das Service-Management aufzurufen:



- Mit der Option **1** installieren und starten Sie im Service-Management UKI-4.0 als Service:



- Nach erfolgreichem Start des Services wird der Status **Service is running** in der Kommandozeile angezeigt:



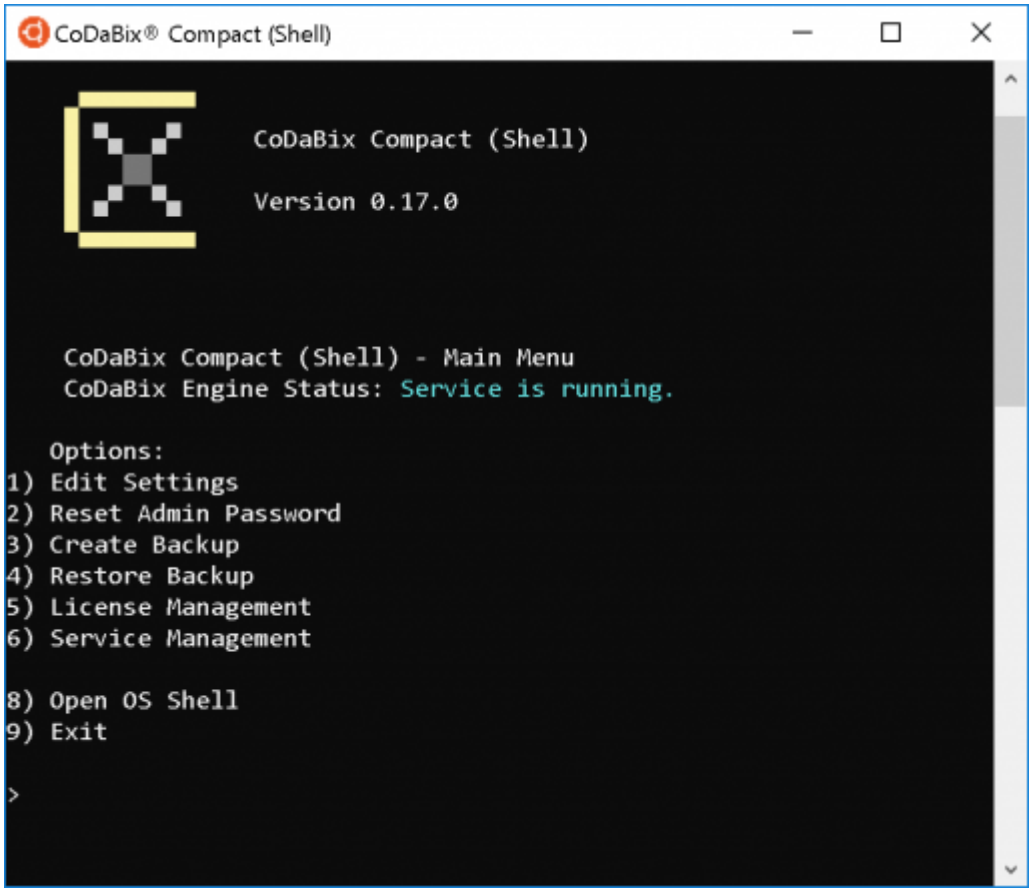
- Sie können nun die UKI-4.0 Shell mit **9** beenden und der Service läuft im Hintergrund weiter.
- Bei Neustart des Betriebssystems wird der Service zukünftig automatisch gestartet.

Status des UKI-4.0 Service

Um den aktuellen Status des UKI-4.0 Service (*Running*, *Stopped*) abzurufen, starten Sie die UKI-4.0 Shell:

```
UKI-4.0
```

Nach dem Start wird der aktuelle Status des Services angezeigt:

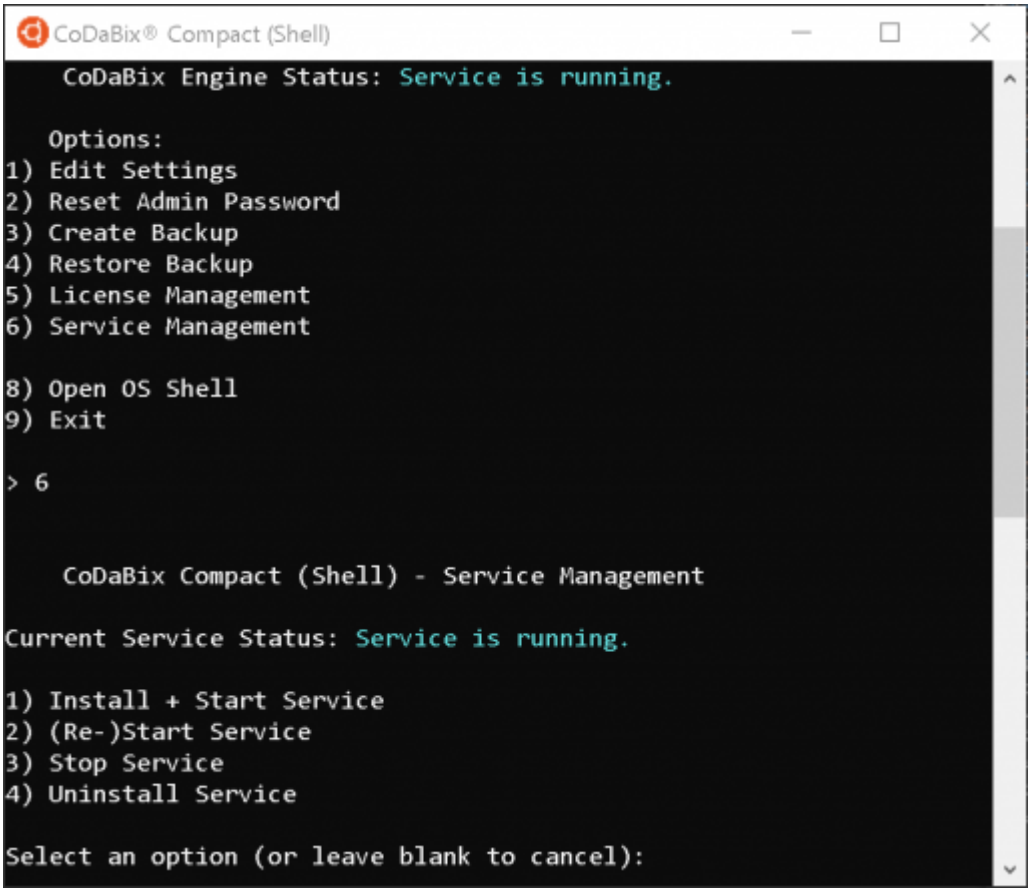


UKI-4.0 Service (neu)starten, stoppen und deinstallieren

Um den UKI-4.0 Service zu starten, stoppen oder deinstallieren, öffnen Sie die UKI-4.0 Shell:

```
UKI-4.0
```

- Hier rufen Sie mit **6** das Service-Management auf
- Wählen Sie nun die gewünschte Funktion:



Lizenzverwaltung

Maschinencode

Wenn Sie eine Lizenz für UKI-4.0 bestellen, ist es notwendig, den Maschinencode derjenigen Maschine bereitzustellen, auf der UKI-4.0 läuft.

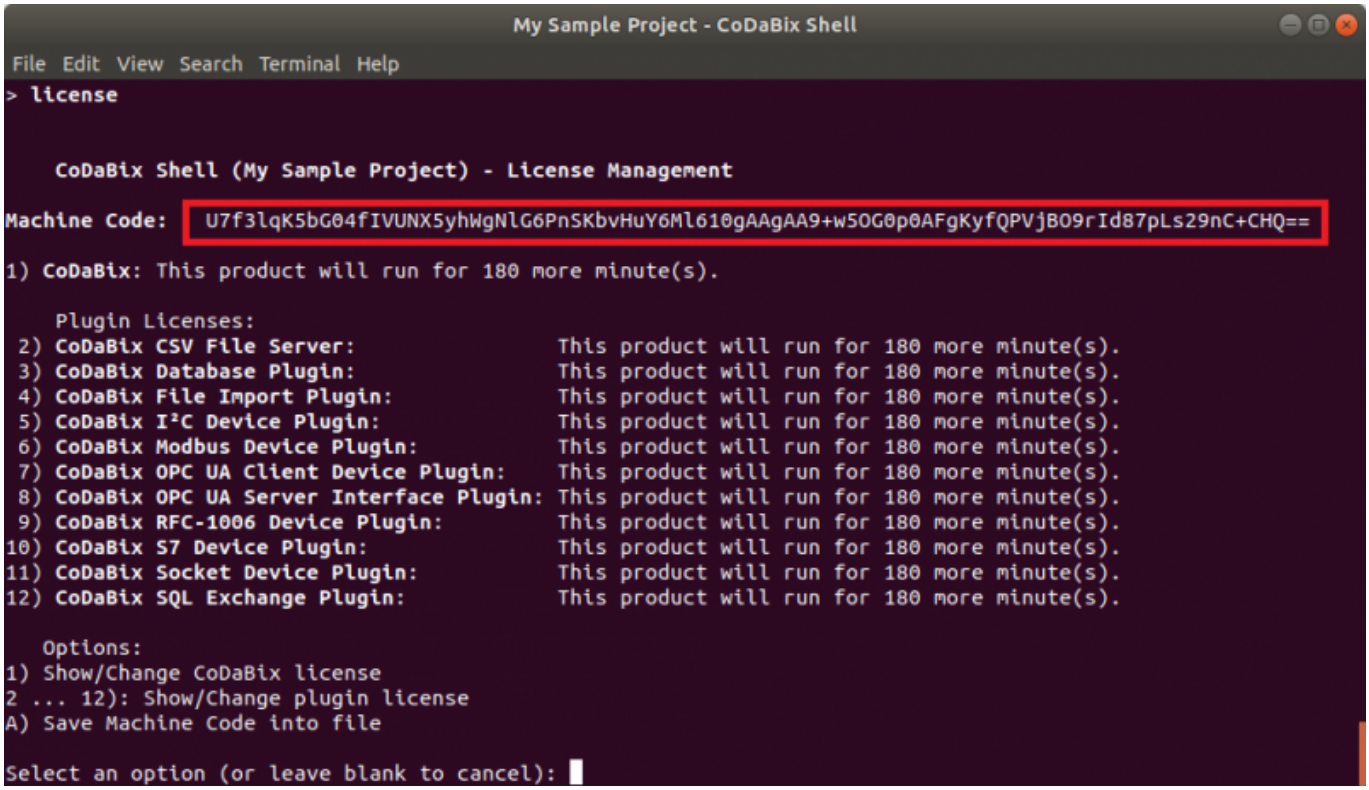
Um den Maschinencode zu erhalten, führen Sie das folgende Kommando auf der Kommandozeile (bash) aus:

```
UKI-4.0 --machinecode
```

Dies gibt den lokalen Maschinencode auf dem Terminal aus, z.B.:

```
$ UKI-4.0 --machinecode
U7f3lqK5bG04fIVUNX5yhWgNlG6PnSKbvHuY6Ml610gAAgAA9+w5OG0p0AFgKyfQPVjBO9rId87pLs29nC+CHQ==
```

Alternativ können Sie den Maschinencode in der UKI-4.0 -Shell-Anwendung anzeigen, indem Sie `license` im Hauptmenü eingeben, um die Lizenzverwaltung zu öffnen. Dort wird der lokale Maschinencode oben angezeigt:



Siemens IOT2050 Image

Diese Anleitung beschreibt die notwendigen Schritte, um das UKI-4.0 Siemens IOT2050 Image in Betrieb zu nehmen.

Inhalt des Images

Für dieses Image wurde das Siemens IOT2050 Example als Basisimage verwendet.

Folgende Komponenten wurden hinzugefügt:

- Komponenten zur DNS Discovery
  - Avahi Daemon
    - Dieser Dienst ermöglicht die sogenannte DNS-SD (DNS Service Discovery) basierend auf dem Apple Bonjour Protokoll. Dadurch lässt sich ein Gerät mit Hilfe seines Hostnamens und der Domäne „local“ im lokalen Netzwerk finden, ohne die IP-Adresse des Geräts zu kennen.
  - Automatischer Hostname
    - Während des Systemstarts wird ein automatischer Hostname generiert, der auf der Seriennummer des Geräts basiert. Jedes Gerät erhält dadurch einen eindeutigen Hostnamen, der sich mit Hilfe der Seriennummer, die auf dem Gehäuse aufgebracht ist, ermitteln lässt.
- UKI-4.0 + UKI-4.0 service
- Neues `root` Passwort: UKI-4.0 `-iot`

Schreiben des Images

Für das Schreiben des Images auf eine SD-Karte empfehlen wir, folgendes Programm zu verwenden:

https://www.balena.io/etcher/

Inbetriebnahme

Nachdem Sie das Image auf die SD-Karte geschrieben haben, setzen Sie die SD-Karte in das Siemens IOT2050 Gerät ein und stellen eine Stromversorgung her. Um per Netzwerk auf das Gerät zuzugreifen, muss sich der Computer, von welchem aus Sie die Verbindung aufbauen wollen, im selben Netzwerk befinden.

Automatischer Hostname

Der Hostname des Geräts wird nach folgendem Schema generiert:

```
UKI-4.0 -iot-<serialNumber>
```

Für die nächsten Schritte wird angenommen, dass Ihr Gerät die Seriennummer `M4A98757` besitzt. Stellen Sie also sicher, diese Nummer gegen die tatsächliche Seriennummber Ihres Geräts auszutauschen.

```
serialNumber = M4A98757
hostname = UKI-4.0 -iot-M4A98757
```

Um zu überprüfen, ob das Gerät erreichbar ist, können Sie folgenden Befehl auf einer Kommandozeile ausführen.

Achtung: Dazu muss auf Ihrem Computer der Apple Bonjour Dienst installiert sein.

```
$ ping UKI-4.0 -iot-M4A98757.local
```

Wenn dieser Ping-Befehl nicht erfolgreich war, ist das IOT2050 entweder noch nicht vollständig gestartet oder von Ihrem Netzwerk aus nicht erreichbar.

Zugriff auf die UKI-4.0 Webkonfiguration

Wenn der Ping-Befehl erfolgreich war, können Sie die UKI-4.0 Webkonfiguration in einem Browser unter der URL `http://`UKI-4.0 `-iot-M4A98757.local:8181` aufrufen.

Zugangsdaten:

- User: `admin`
- Passwort: UKI-4.0 `-iot`

Zugriff über SSH

Um über SSH auf das Gerät zuzugreifen, führen Sie folgenden Befehl auf der Kommandozeile aus:

```
$ ssh root@UKI-4.0 -iot-M4A98757.local
```

Zugangsdaten:

- User: `root`
- Passwort: UKI-4.0 `-iot`

[1] , [2]

Das Update **KB3063858** (x64 oder x86) wird benötigt.

Außerdem muss, um Lizenzen mit einem Maschinencode verwenden zu können, das Windows Management Framework 3.0 (**KB2506143**) installiert sein.

UKI-4.0UKI-4.0UKI-4.0 UKI-4.0UKI-4.0UKI-4.0 Webkonfiguration

UKI-4.0UKI-4.0 UKI-4.0UKI-4.0 Anwendung starten

1. Installieren Sie UKI-4.0 , siehe UKI-4.0 Installation.
2. Machen Sie die notwendigen Einstellungen UKI-4.0 Setup und erster Start.
3. Starten Sie UKI-4.0 durch Doppelklick auf das Icon 

, oder durch das Ausführen des folgenden Kommandos auf der Kommandozeile (z.B. unter Windows Server Core):
```
"%ProgramFiles%\TIS\UKI-4.0 \UKI-4.0 -ui.exe"
```
4. Der UKI-4.0 Login-Dialog öffnet sich.

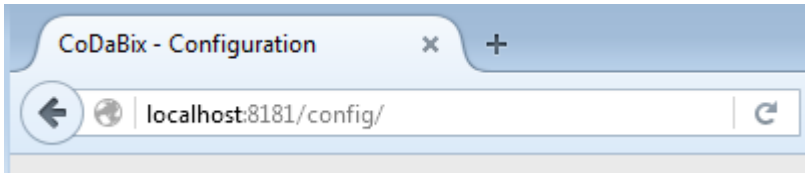
Konfiguration

UKI-4.0 kann über eine webbasierte GUI konfiguriert werden. Unter Windows verwendet UKI-4.0 einen eingebetteten Webserver, um die Webkonfiguration anzuzeigen.

Sie können die Webkonfiguration auch in Ihrem eigenen Browser anzeigen (z.B. Edge, Chrome, Firefox, Safari). Standardmäßig verwendet UKI-4.0 ein lokales HTTP-Binding (Standardport 8181), somit können Sie folgende URL verwenden:

```
http://localhost:8181/config/
```

Beispiel:

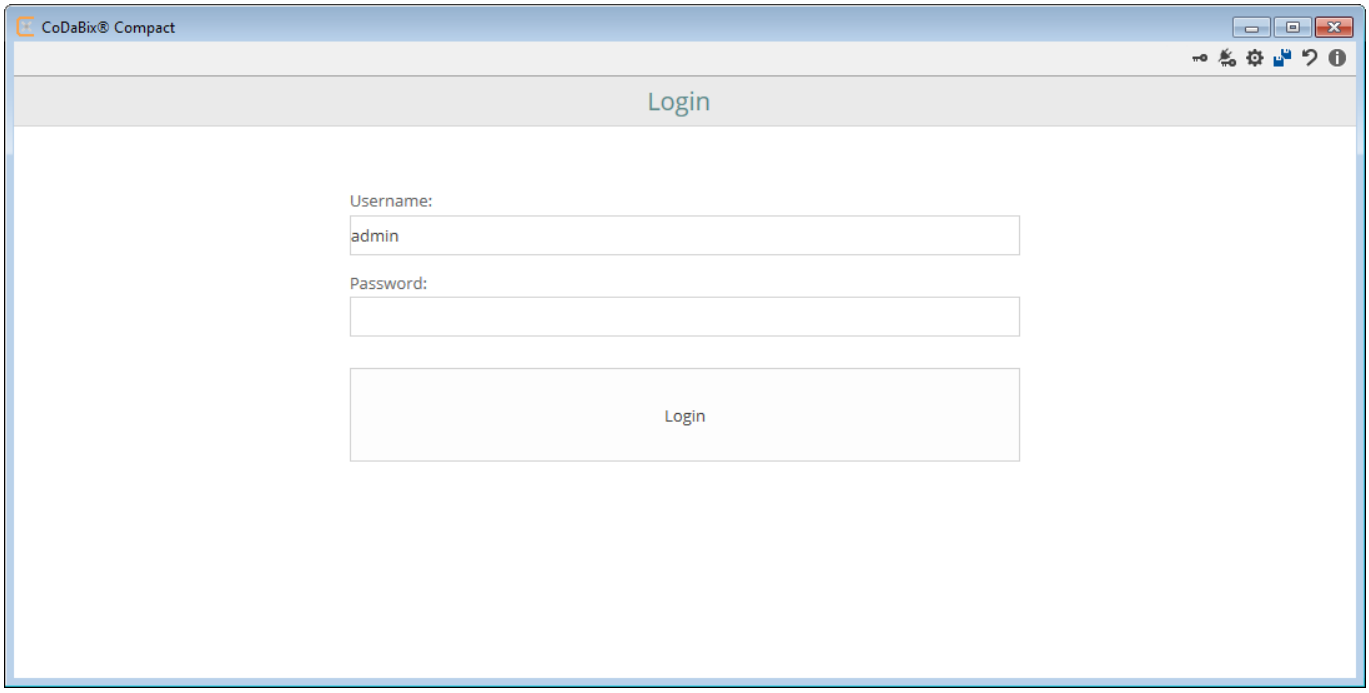


Beachten Sie: Wenn Sie auf die UKI-4.0 Webkonfiguration von einem **entfernten Rechner** aus zugreifen möchten, muss in den UKI-4.0 Settings ein Remote-HTTP-Binding hinzugefügt werden (bei UKI-4.0 für Linux ist dies standardmäßig der Fall).
Um auf UKI-4.0 mit einem Browser von einem entfernten Rechner zuzugreifen können Sie folgende URL verwenden:

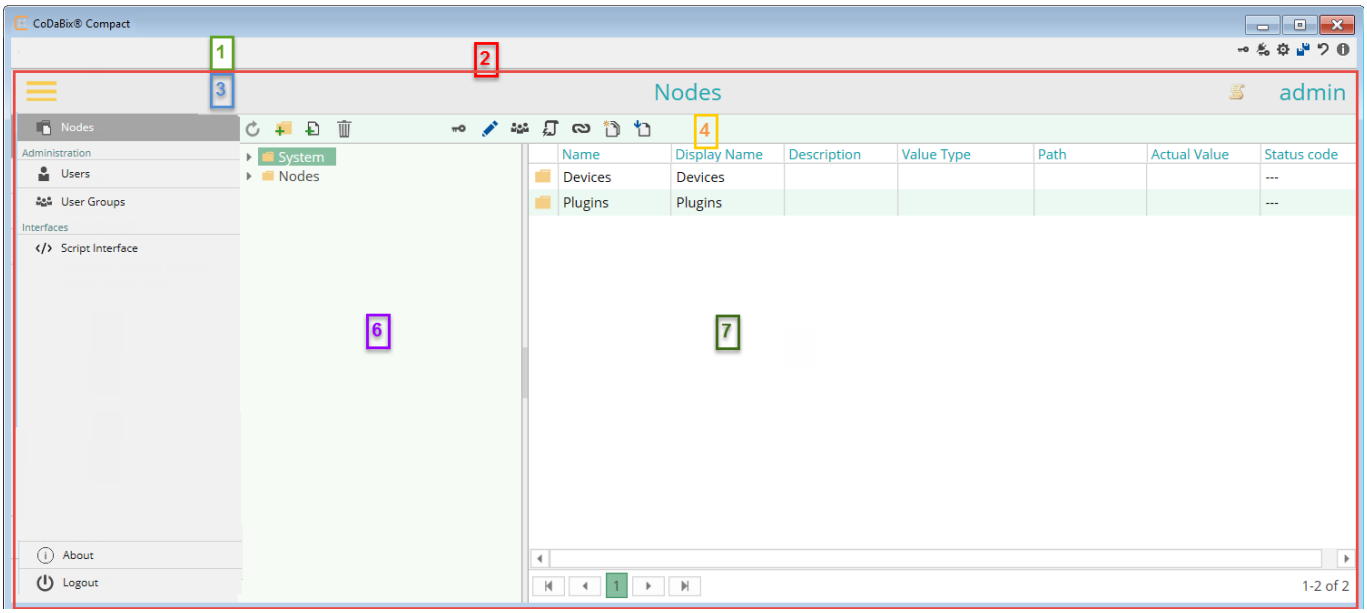
```
http://<hostname>:<port>/config/
```

Allgemein

Nach dem Start beziehungsweise Aufruf der URL wird der Login-Dialog angezeigt:



Nach dem Login gelangen Sie zum UKI-4.0 Hauptfenster.



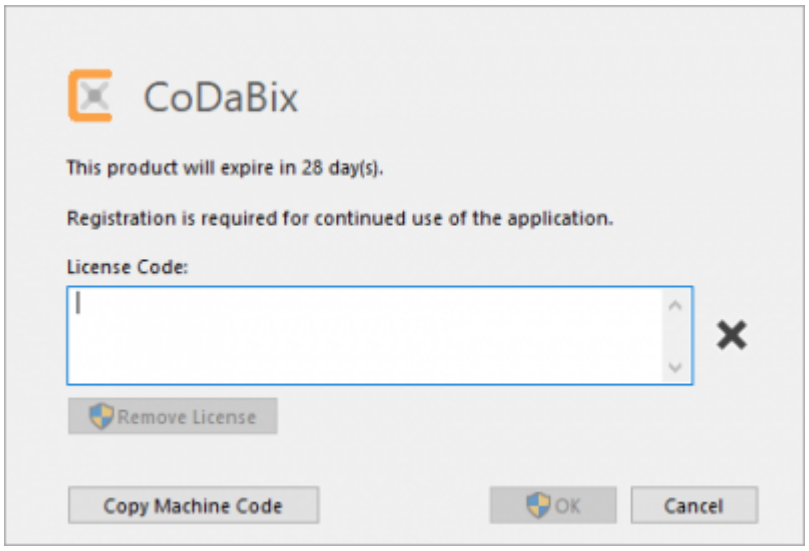
Die UKI-4.0 Webkonfiguration ist in folgende Bereiche unterteilt:

(1) UKI-4.0 Toolbar

Diese Toolbar ist nur sichtbar, wenn Sie die UKI-4.0 Applikation zur Konfiguration verwenden. Diese Leiste bezieht sich also nur auf die UKI-4.0 Applikation.

Icon	Beschreibung
	Öffnet den Lizenzdialog, in dem Sie die Lizenz eingeben, ändern oder löschen können
	Öffnet den Lizenzdialog, in dem Sie die Plugin Lizenz eingeben, ändern oder löschen können
	Siehe UKI-4.0 Projekteinstellungen
	Backup erstellen, siehe UKI-4.0 Projekteinstellungen
	Restore Backup, siehe UKI-4.0 Projekteinstellungen
	Zeigt alle Informationen der Lizenz und Anwendung

Lizenzdialog



Feld	Beschreibung
License Code:	Geben Sie hier Ihren Lizenzschlüssel ein
Remove License	Entfernt die Lizenz vom Computer
Copy Machine Code	Kopiert den Maschinencode in die Zwischenablage. Dieser wird für die Lizenzerstellung benötigt.
OK	Die eingegebene Lizenz wird auf den Computer gespeichert. Wenn dieses Feld ausgegraut ist, wurde kein gültiger Lizenzschlüssel eingegeben.

Über

Linker Bereich: Informationen der Anwendung
Rechter Bereich: Informationen zur aktuell installierten Lizenz

(2) Arbeitsbereich

Der Arbeitsbereich besteht aus einem eingebetteten Webbrowser (rote Umrandung).
Dieser Bereich unterteilt sich, je nach Menüwahl, in 3-4 Bereiche.

(3) Titelleiste

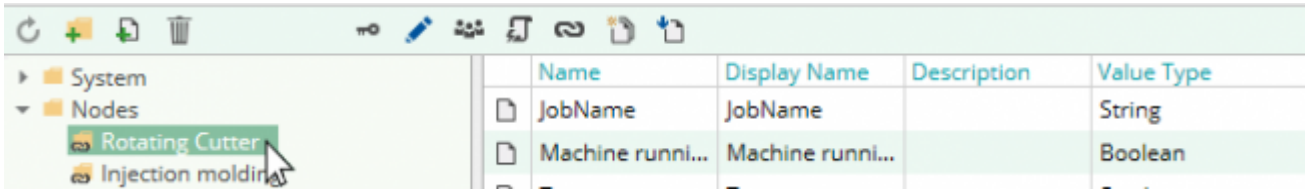
In der Titelleiste sind folgende Aktionen möglich und werden folgende Daten angezeigt:

Feld	Beschreibung
	Öffnet und schließt das Menü
Nodes	Anzeige des aktuell ausgewählten Menüs
admin	Anzeige des eingeloggten Benutzers

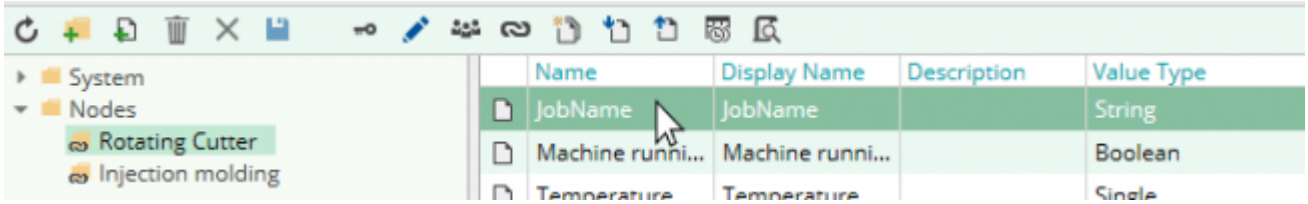
(4) Toolbar

Unterhalb der Titelleiste befindet Sie die Toolbar. Diese wird automatisch angepasst, je nachdem, in welchem Menü Sie sich befinden und welches Item Sie ausgewählt haben.

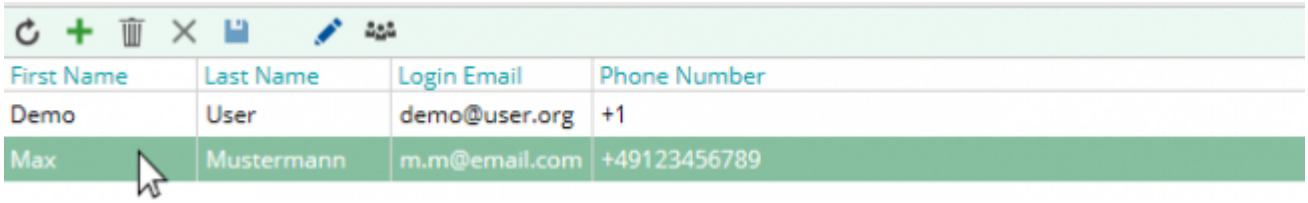
Foldernode:



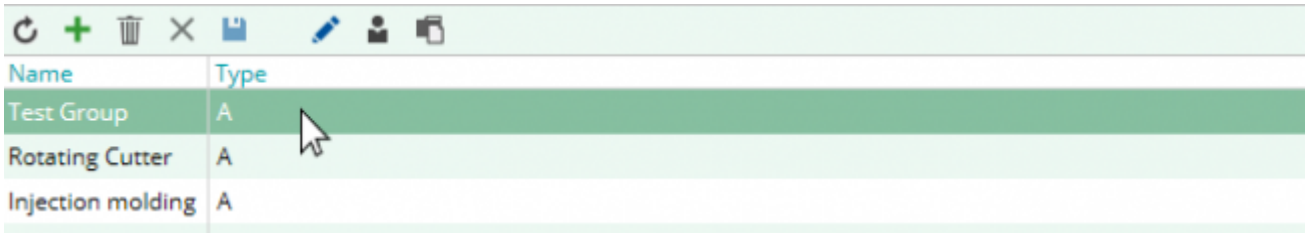
Datenpunktode:



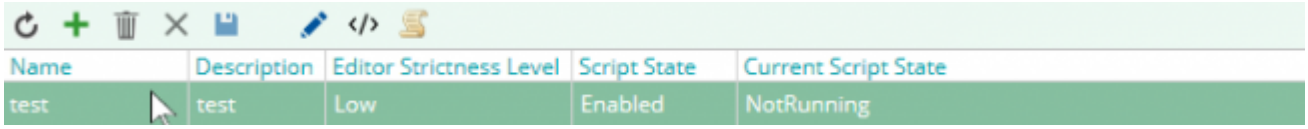
Benutzer:



Benutzergruppe:



Script Plugins:



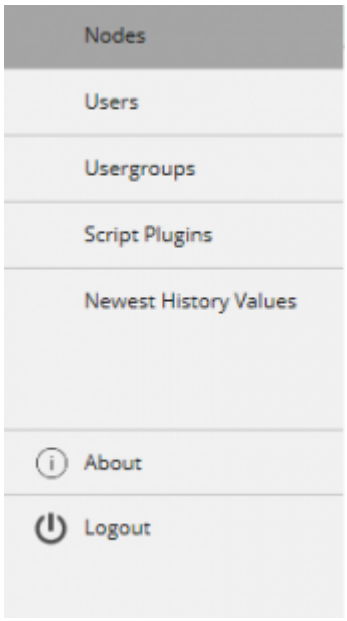
Allgemeines:

Ausgegraute Icons sind für das ausgewählte Item nicht anwendbar.

Icon	Beschreibung	Sichtbar in
	Ansicht auffrischen	alle Menüs
	UKI-4.0 Folder Node	Folder Node, Datenpunkt Node
	UKI-4.0 Datenpunkt Node	Folder Node, Datenpunkt Node

Icon	Beschreibung	Sichtbar in
	UKI-4.0 - verlinkte Folder Node	Folder Node, Datenpunkt Node
	UKI-4.0 - verlinkte Datenpunkt Node	Folder Node, Datenpunkt Node
	Alle Icons mit diesem Symbol fügen ein neues Item anhand eines Eingabedialoges hinzu	Folder Node, Datenpunkt Node, Benutzer, Benutzergruppe, Script Plugins
	Ausgewähltes Item löschen	Folder Node, Datenpunkt Node, Benutzer, Benutzergruppe, Script Plugins
	Aktion abbrechen	Datenpunkt Node, Benutzer, Benutzergruppe, Script Plugins
	Speichert den bearbeiteten Datensatz bzw. Datensätze	Datenpunkt Node, Benutzer, Benutzergruppe, Script Plugins
	Zugriff auf den selektierte Folder Node oder Datenpunkt Node	Folder Node, Datenpunkt Node
	Ausgewähltes Item editieren	Datenpunkt Node, Benutzer, Benutzergruppe, Script Plugins
	Benutzer hinzufügen oder entfernen	Benutzergruppe
	Folder Node oder Datenpunkt Node hinzufügen oder abwählen	Benutzergruppe
	Die Benutzergruppe für das ausgewählte Item setzen oder abwählen	Folder Node, Datenpunkt Node
	Link erstellen oder entfernen	Folder Node, Datenpunkt Node
	Dupliziert das ausgewählte Item	Folder Node, Datenpunkt Node
	Bei Klick auf ein Folder Node werden alle Werte der Datenpunkt Nodes geupdatet. Ist ein Datenpunkt Node ausgewählt, wird nur dieser Wert geupdated	Folder Node, Datenpunkt Node
	Datenpunkt Node Wert setzen	Datenpunkt Node
	Wenn verfügbar, werden die historischen Daten des Datenpunkt Nodes angezeigt	Datenpunkt Node
	Script bearbeiten	Script Plugin
	Loganzeige des ausgewählten Scripts	Script Plugin

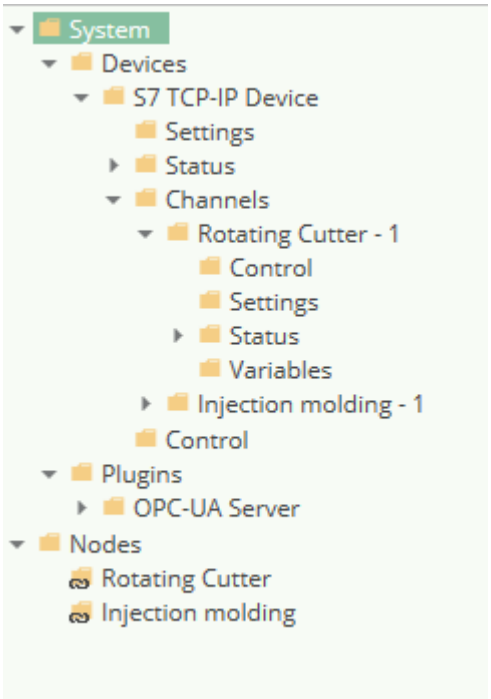
(5) Menü



Standardmäßig wird das Menü fest angezeigt.

Feld	Beschreibung
Nodes	Aktuelle Startseite. Folder Nodes und Datenpunkt Nodes bearbeiten
Users	Benutzer bearbeiten
Usergroups	Benutzergruppen bearbeiten
Script Plugins	Leichtgewichtiges JavaScript(TypeScript)-Plugin, dass UKI-4.0 um benutzerdefinierte Funktionen erweitert. Wird in einer sicheren Umgebung ausgeführt.
Newest History Values	Ansicht der zuletzt eingetragenen, historisch abgelegten Werte. Daten werden nur angezeigt, wenn ein Folder oder Datenpunkt Node die Speicherung von historischen Daten aktiviert hat.
About	Informationen über die Konfigurationsseite und gegebenenfalls UKI-4.0
Logout	Benutzer abmelden

(6) Node Tree



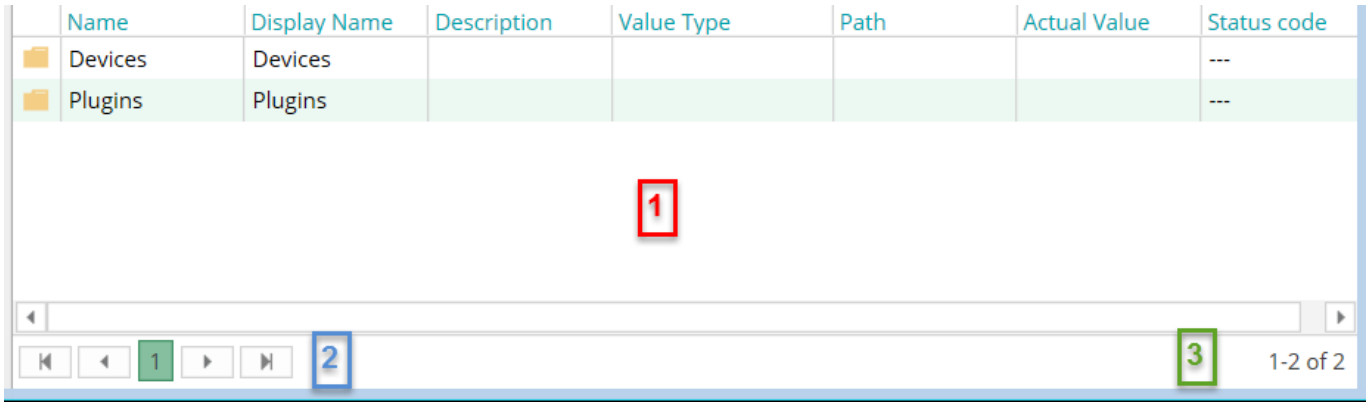
Hier werden die verfügbaren Folder Nodes in einer Baumstruktur dargestellt.

Aufbau des Node Baumes:

Node	Beschreibung	Zweck
System	System Nodes	Enthält alle Nodes die für den Betrieb von UKI-4.0 nötig sind. Die meisten Nodes und Variablen sind hier nicht editierbar. Fixe Folder Nodes sind: -System -Device -Plugins -Settings -Status -Channels -Variables -Control
Device	Device Nodes	Enthält alle Nodes der bei UKI-4.0 registrierten Devices.
S7 TCP-IP Device	S7-Device Plugin Container	Enthält alle Daten des Plugins, die für UKI-4.0 und den Anwender benötigt werden, z.B. Status, Channels, usw.

Node	Beschreibung	Zweck
Settings	Einstellungen des Plugins	Mögliche Einstellungen des Devices
Status	Status des Plugins	Zeigt allgemeine Informationen zum Plugin an, z.B. Fehler, Plugin gestartet
Channels	Node für alle Kanäle	Enthält alle definierten Kanäle des Plugins
Channel z.B. Rotating Cutter - 1	Definierter Kanal im Plugin	Enthält - die allgemeinen Einstellungen des Kanals - den Status des Kanals - alle definierten Datenpunkt Nodes
Plugin	Plugin Nodes	Sammelnode für alle Plugins, ähnlich wie Devices
Nodes	Sammelnode für benutzerdefinerte Nodes	Enthält alle Nodes und Variablen, die der Benutzer definiert hat

(7) Datenbereich



Nummer	Beschreibung
1	Hier werden die Daten des ausgewählten Menüs bzw. des Folder Nodes dargestellt
2	Navigationsleiste, hier schalten Sie die Ansichten der jeweiligen Daten durch. Mögliche Navigation: Seitenanfang eine Seite zurück eine Seite vor Seitenende
3	Anzahl aller Datensätze

Nodes

Was sind Nodes?

„Das OPC-Informationsmodell ist nicht mehr nur eine Hierarchie aus Ordnern, Items und Properties. Es ist ein sogenanntes Full-Mesh-Network aus Nodes, mit dem neben den Nutzdaten eines Nodes auch Meta- und Diagnoseinformationen repräsentiert werden. Ein Node ähnelt einem Objekt aus der objektorientierten Programmierung. Ein Node kann Attribute besitzen, die gelesen werden können (Data Access (DA), Historical Data Access (HDA)) ... Die Nodes werden sowohl für die Nutzdaten als auch alle anderen Arten von Metadaten verwendet. Der damit modellierte OPC-Adressraum beinhaltet nun auch ein Typmodell, mit dem sämtliche Datentypen spezifiziert werden.„
Quelle: wikipedia.org
UKI-4.0 ist ähnlich aufgebaut wie in OPC UA spezifiziert. Je nach Node Typ hat ein Node unterschiedliche Funktionen und Eigenschaften.

Es wird unterschieden zwischen:

- Folder Node

- Datenpunktnode
- Link Node
- Directory Node

Folder Nodes

Einen Folder Node kann man auch als Knoten bezeichnen, da er weitere Folder oder Datenpunktnodes enthält, aber selbst keinen Wert bereitstellen darf.
Eingestellte Eigenschaften werden automatisch auf die Child Nodes angewendet, außer diese definieren eine eigene Eigenschaft.

Datenpunktnodes

Der Datenpunktnode ist im UKI-4.0 wie eine Variable mit zusätzlichen Attributen (Properties).
Ein Datenpunktnode darf keine Child Nodes enthalten.

Link Nodes

Link Nodes besitzen nur die gemeinsamen Properties.
Ein Link Node zeigt auf einen anderen, bereits angelegten Node. So ist es möglich, sich aus verschiedenen Devices, Foldern und Datenpunktnodes einen individuellen Datensatz zusammenzubauen.

Directory Nodes

Der Directory Node ist ein Folder Node, der aber direkt auf ein physikalisches Verzeichnis zeigt und die Unterstruktur automatisch repräsentiert.

Node-Eigenschaften

Gemeinsame Eigenschaften

Name	Beschreibung	Datentyp
Lokale ID	Eindeutiger ID im UKI-4.0 System wird z.B. beim Zugriff über das REST-Interface benutzt	Long
Global ID	Eindeutige UKI-4.0 -übergreifende ID	GUID
Name	eindeutiger Name innerhalb des Parent Nodes für z.B. die OPC UA-Adressierung, JSON-Interface z.B. JobNumber	String
Display Name	Anzeigename für den Endanwender z.B. Job Nummer	String
Path	z.B.: SPS-Adresse (nach S7-Syntax, DB1000.DBB 500, Word) OPC UA Node (3:AirConditioner_1.State) Dateipfad (R:\\MachineData) **HINWEIS**: Bei Typ *Folder* wird der Pfad nicht behandelt	String
Max Value Age (ms)	Relevant für einen synchronen Lesevorgang. Wenn ein synchroner Lesevorgang keine **Time to live** angibt, wird anstelle dessen dieser Wert (in Millisekunden) benutzt. Das bedeutet, falls der derzeit gesetzte Nodewert **nicht** älter als diese Zeitangabe ist, wird er direkt zurückgegeben; ansonsten wird der Wert aus dem Device gelesen.	Integer

Nodetypen

Wenn Sie einen Folder Node anlegen, können Sie entscheiden zwischen den Typen:

- Folder
- Directory

Typ: Folder

Der Node ist ein regulärer Folder Node mit keinen besonderen zusätzlichen Eigenschaften.

Typ: Directory
Der Node ist ein Folder Node und repräsentiert ein physisches Verzeichnis im Dateisystem. Die Path-Eigenschaft des Nodes gibt den Verzeichnispfad an (dieser kann Umgebungsvariablen enthalten). Beim Browsen des Nodes (z.B. über die Webkonfiguration) werden Unterverzeichnisse und Dateien automatisch als entsprechende Folder Nodes (Typ **Directory**) sowie Datapoint Nodes (Typ **File**) mit dem zugehörigen Dateipfad in der Path-Eigenschaft erstellt und abgeglichen.

Hinweis: Der angegebene Pfad unterliegt den Einschränkungen der **Access Security**, die in den UKI-4.0 Projekteinstellungen definiert wurden.

Hinweis: Unter Windows 10 Version 1511 und älter (sowie Windows Server 2012 R2 und älter), z.B. unter Windows 7, ist die maximale Pfadlänge auf 260 Zeichen (`MAX_PATH`) begrenzt. Unter Windows 10 Version 1607 und höher (sowie Windows Server 2016 und höher) können Sie längere Pfade benutzen. Dazu müssen Sie jedoch erst die Einstellung „Lange Win32-Pfade aktivieren“ in den Windows-Gruppenrichtlinien aktivieren, siehe Enabling Win32 Long Path Support.

Datenpunktnodes

Zusätzlich zu den oben erwähnten gemeinsamen Eigenschaften besitzt der Datenpunktnode noch folgende Eigenschaften:

Name	Beschreibung	Datentyp
Description	Beschreibung des Datenpunktes	String
Location	Wird bei der Auswertung von Conditions verwendet. Dient zur Anzeige z.B. des Ursprungs der Meldung. Beliebig vergebbar	String
Value Types	Datentyp des Nodes, weitere Informationen siehe unterhalb Tabelle Value Types	enum
Min Value	Minimaler Wert, wird für die Auswertung von Conditions benötigt	Double
Max Value	Maximaler Wert, wird für die Auswertung von Conditions benötigt	Double
Hysteresis	Schwellenwert der bei der Auswertung von Conditions verwendet wird, wenn z.B. kleinere Temparaturschwankungen ausgeglichen werden sollen.	Double
Scaling Factor	Faktor, der auf den aktuell gelesenen Wert aufgerechnet wird	Double
Scaling Offset	Wert, der auf den aktuell gelesenen Wert addiert wird	Double
Unit	Einheit des Wertes z.B. °C	String
Precision	Anzahl der Nachkommastellen	Zahl
History Options	Gibt an, wie für diesen Node historische Werte erfasst und in die Datenbank geschrieben werden sollen. **No**: Es werden keine historischen Werte für diesen Node erfasst. **Yes; only on Value Change**: Wenn für den Node ein **History Interval** festgelegt ist, werden historische Werte regelmäßig vom *History Timer* aus dem Istwert im angegebenen History-Intervall erfasst; ansonsten werden historische Wert beim Schreiben eines Werts in den Node erfasst. Ein erfasster historischer Wert wird nur dann in die Datenbank geschrieben, wenn dieser sich vom zuletzt geschriebenen historischen Wert dieses Nodes unterscheidet. Zusätzlich wird im Falle einer Device-Variablen eine Subscription für diesen Node erstellt. **Yes**: Wenn für den Node ein **History Interval** festgelegt ist, werden historische Werte regelmäßig vom *History Timer* aus dem Istwert im angegebenen History-Intervall erfasst; ansonsten werden historische Wert beim Schreiben eines Werts in den Node erfasst. Erfasste historische Werte werden in die Datenbank geschrieben. Zusätzlich wird im Falle einer Device-Variablen eine Subscription für diesen Node erstellt.	enum
History Interval	Gibt das Intervall an, in dem historische Daten erfasst werden sollen.	Enum

Name	Beschreibung	Datentyp
History Resolution	Gibt die Auflösung an, auf die numerische historische Werte beim Erfassen gerundet werden.	Double

Beachten Sie:

- Um festzustellen, ob sich ein erfasster historischer Wert von letzten historischen Wert unterscheidet (für die Einstellung **Yes; only on Value Change)**, werden nur die Properties `Value` und `Status` verwendet, nicht jedoch der `Timestamp` (CreationTimestamp).
- Wenn die **History Options** eines Nodes von **No** auf **Yes; only on Value Change** geändert werden während UKI-4.0 läuft, wird der erste erfasste historische Wert nach dieser Umstellung in jedem Fall in die Datenbank geschrieben, auch wenn dieser sich nicht vom zuletzt geschriebenen historischen Wert für diesen Node unterscheidet (außer wenn UKI-4.0 zwischen der Umstellung der History Options und dem Verstreichen des Intervalls heruntergefahren wird - siehe nächster Punkt).
- Wenn die **History Options** eines Nodes auf **Yes; only on Value Change** gestellt sind und UKI-4.0 heruntergefahren und später wieder gestartet wird, wird nach dem Neustart der erste erfasste historische Wert eines Nodes nicht in die Datenbank geschrieben, wenn dieser sich nicht vom zuletzt geschriebenen historischen Wert für diesen Node unterscheidet.
- Wenn die **History Options** eines Nodes auf **Yes; only on Value Change** gestellt sind (oder kein **History Interval** festgelegt ist), wird beim Erfassen eines historischen Werts der *Creation Timestamp* des Istwerts beibehalten; ansonsten wird der *Creation Timestamp* auf die aktuelle Uhrzeit gesetzt wird.
- Die Einstellung **Yes; only on Value Change** entspricht in einem **Script** dem Wert UKI-4.0 `.NodeHistoryOptions.Subscription |` UKI-4.0 `.NodeHistoryOptions.ValueChange` (enthält UKI-4.0 `.NodeHistoryOptions.Active`), und die Einstellung **Yes** entspricht dem Wert UKI-4.0 `.NodeHistoryOptions.Subscription` (enthält UKI-4.0 `.NodeHistoryOptions.Active`).
- Historische Werte werden in die Datenbank **asynchron** im Hintergrund geschrieben. Dies bedeutet, dass nach dem Schreiben eines Wertes in einen Node (bei welchem kein *History Interval* festgelegt ist) der erfasste historische Wert noch nicht in der Datenbank sein muss, selbst nachdem die Schreiboperation abgeschlossen ist. Dies ist insbesondere dann der Fall, wenn Sie die Einstellung *Update DB Mode* in den UKI-4.0 Settings auf **Restricted** gestellt haben, da erfasste historische Werte dann nur alle 5 Sekunden in die Datenbank geschrieben werden.

Value Types

UKI-4.0 stellt folgende Werttypen bereit:

Name	entspricht Datentyp	Länge in Bits
Blob	binär, optional Angabe von Dateiname und MIME-Typ möglich	beliebig
String	String	beliebig
Null	ohne Wert	0
Boolean	Bool	1
SByte	signed Byte	8
Byte	unsigend Byte	8
Int16	signed Integer	16
UInt16	unsigned Integer	16
Int32	signed Integer	32
UInt32	unsinged Integer	32
Int64	signed Integer	64
UInt64	unsigned Integer	64
Single	single Floating Point	32
Double	double Floating Point	64

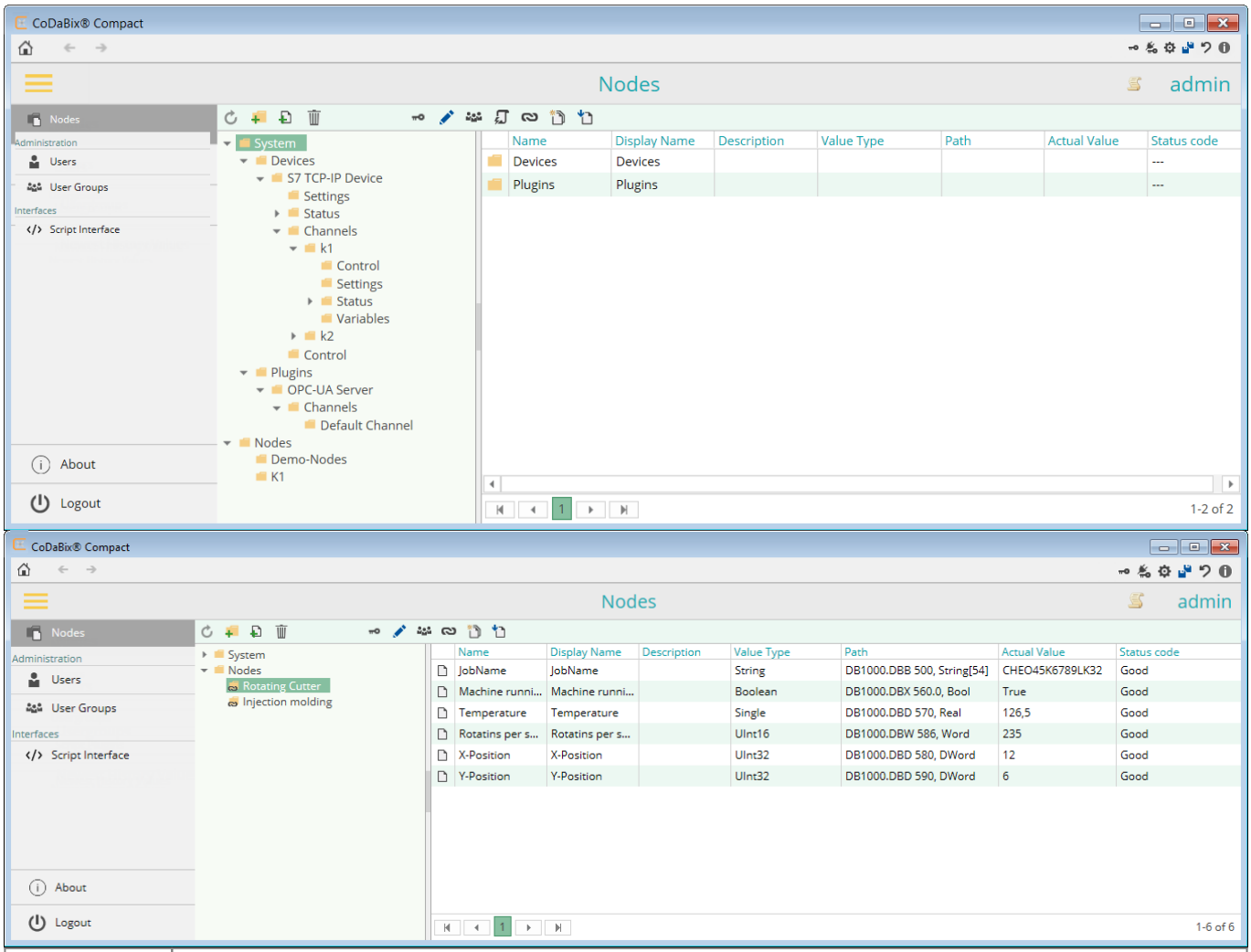
Name	entspricht Datentyp	Länge in Bits
File	Der Node repräsentiert eine physische Datei, auf die mittels OPC UA Client oder per HTTP Access URL zugegriffen werden kann. Die „Path“-Eigenschaft des Nodes enthält den Dateipfad. **Hinweis:** Der angegebene Pfad unterliegt den Einschränkungen der **Access Security**, die in den UKI-4.0 Projekteinstellungen definiert wurden.	

Von jedem Datentyp kann auch ein Array erstellt werden. Ausnahmen sind die Datentypen:

- Null
- File

Die Länge des Arrays wird beim Schreiben des Datenpunktnodes bestimmt.

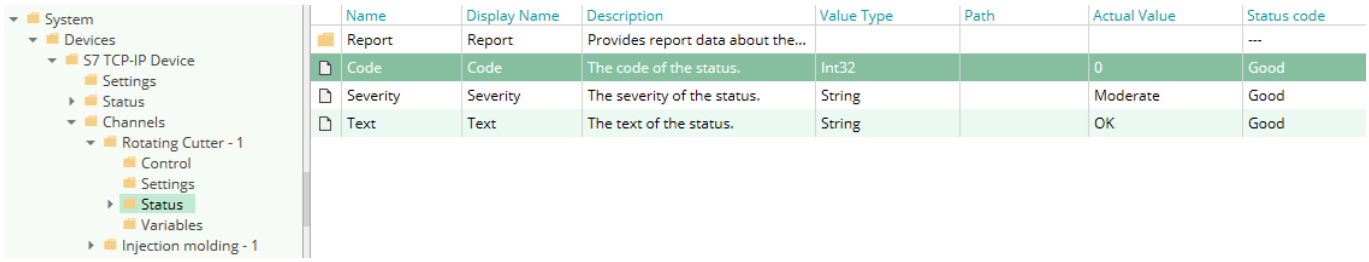
Nodeansicht



Feld	Beschreibung
Name	intern eindeutiger Name des Foldernodes
Display Name	Anzeigename des Nodes
Description	Beschreibung zum Node
Value Type	Datentyp bzw. -art des Nodes. Siehe Eigenschaften
Path	z.B. SPS-Adresse, OPC-Node, Dateipfad zum Node. Siehe Eigenschaften
Actual Value	Wert nach Klick auf . Es werden keine Livedaten automatisch angezeigt. Siehe (4) Toolbar

Feld	Beschreibung
Status	Aktueller Status des Nodes. Mögliche Stati: - Good - Bad

Status bei Device:



Unter System / Devices / Device Plugin / Channels / Channel / Status finden Sie den Status für den einzelnen Kanal mit der dazugehörigen Meldung.
Bei einem Code >= 0 läuft der Kanal ok.

Nodestruktur

Es gibt Systemnodes und Benutzernodes.
Systemnodes sind fest vorgegebene Strukturen, die für den Betrieb von UKI-4.0 und dessen Plugins nötig sind.
Benutzernodes werden vom Benutzer nach seinen Bedürfnissen angelegt und verwaltet.

Aufbau Systemnodes:

- System
  - Devices
    - Device Plugin
      - Settings *(Einstellungen zum Plugin)*
      - Status *(Status des Plugins)*
      - Channels *(Definierte Kanäle)*
        - Channel 1 *(definierter Kanal)*
          - Control *(Datenpunktnodes zur Steuerung des Kanals)*
          - Settings *(Einstellungen des Kanals)*
          - Status *(Statusinformationen des Kanals)*
          - Variables *(Definierte Variablen)*
        - Channel 2
          - Control ...
      - Control *(Datenpunktnodes für die Steuerung des Plugins)*
  - Plugins
    - Plugin
      - Settings
      - Status
      - Channels
        - Channel 1
          - Control
          - Settings
          - Status
          - Variables
        - Channel 2
          - Control ...

- Control

Aufbau Benutzernodes:

- Nodes
  - User Folder Node
    - User Datenpunktnode
    - User Folder Node
    - User Datenpunktnode
    - User Link Node
  - User Folder Node...

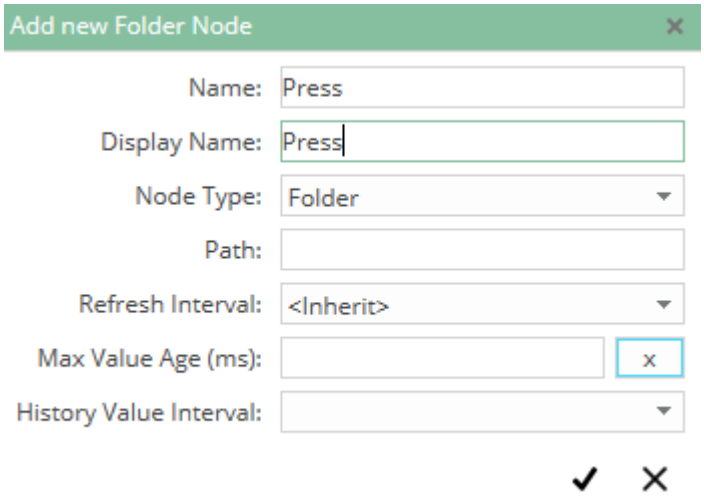
Siehe auch (6) Node Tree.

Node erstellen

In der *Nodes* Ansicht können Sie unter Nodes eigene Nodes hinzufügen.
Um einen Node anlegen zu können, müssen Sie immer erst den übergeordneten Node auswählen. In der Toolbar finden Sie folgende Icons zum hinzufügen von Nodes:

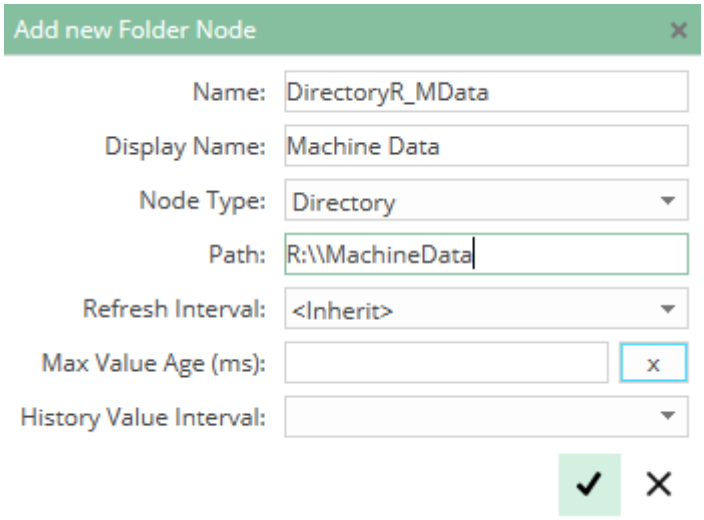
	Folder oder Directory Nodes hinzufügen
	Datenpunktnode hinzufügen

Folder Node hinzufügen



- Name eingeben z.B. Press
- Display Name eingeben z.B. Press
- Node Type: „Folder“ auswählen
- ggf. Intervalle setzen
- mit ✔ Node anlegen

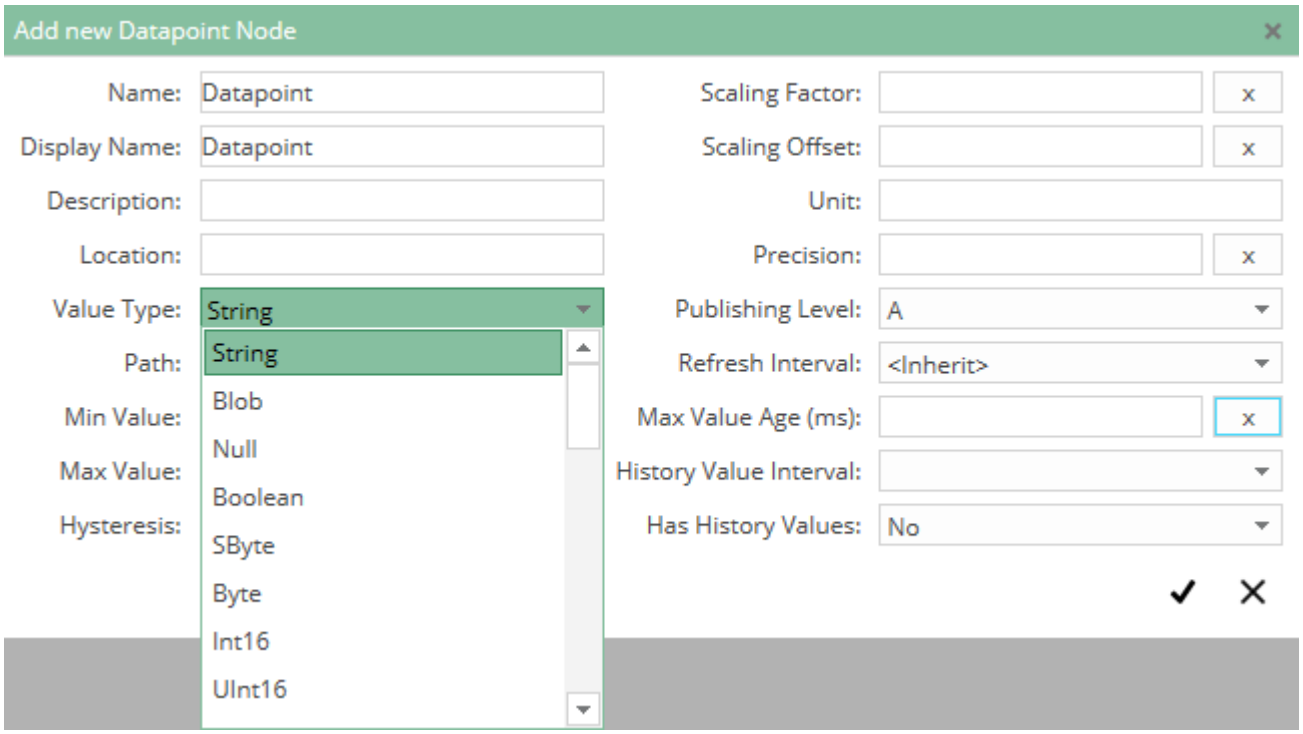
Directory Node hinzufügen



- Name eingeben z.B. DirectoryR_MData
- Display Name eingeben z.B. Machine Data
- Node Type: „Directory“ auswählen
- Path: Verzeichnispfad eingeben z.B. R:\\MachineData
- ggf. Intervalle setzen
- mit ✔ Node anlegen

tipp	Damit Sie die Unterordner und Dateien angezeigt bekommen, müssen Sie die Ansicht auffrischen, z.B. durch Klick auf „User“ und wieder zurück auf „Nodes“. Je nach Verzeichnisgröße kann es vorkommen, dass Sie nicht sofort alle Dateien und Untervezeichnisse sehen.

Datepunktnode hinzufügen

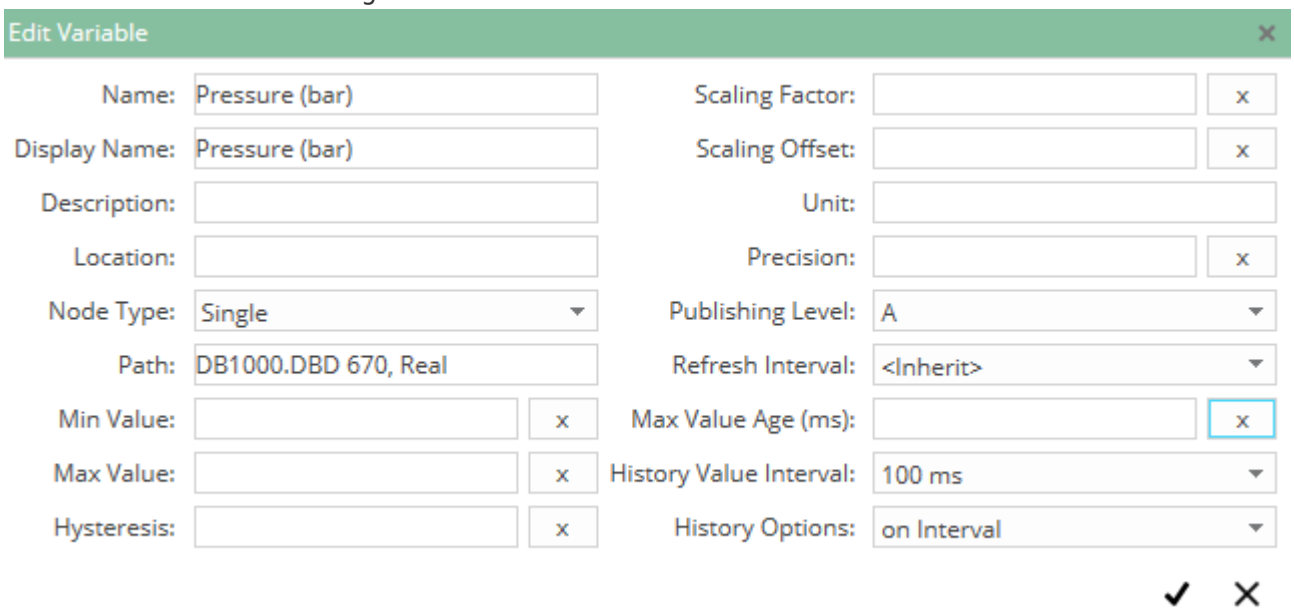


* Name eingeben z.B. Pressure (bar)

- Display Name eingeben z.B. Pressure (bar)
- Value Type auswählen z.B. Single
- ggf. Grenzen, Skalierungen, historische Daten (wie z.B. Min Value = 20,5, Scaling Factor = 2,398 ...) eingeben

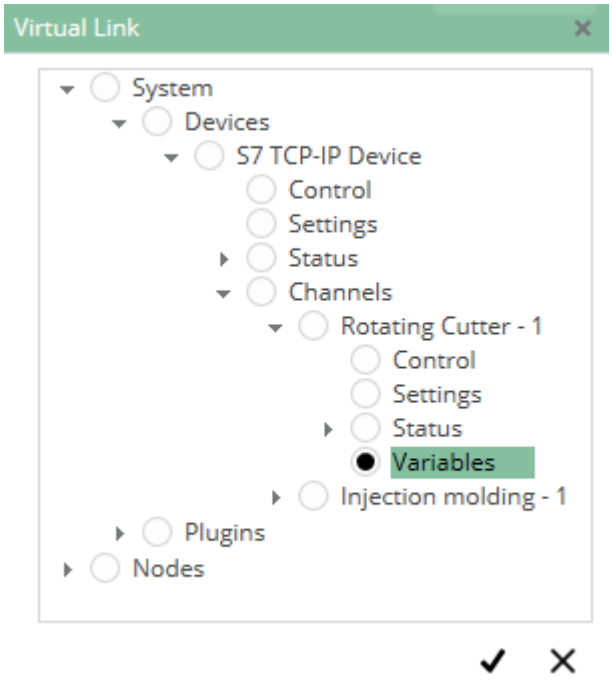
- Mit ✔ Node anlegen

Z.B. könnte die Node wie folgt aussehen:



Node verlinken

Für das ausgewählte Item wird folgender Dialog geöffnet:



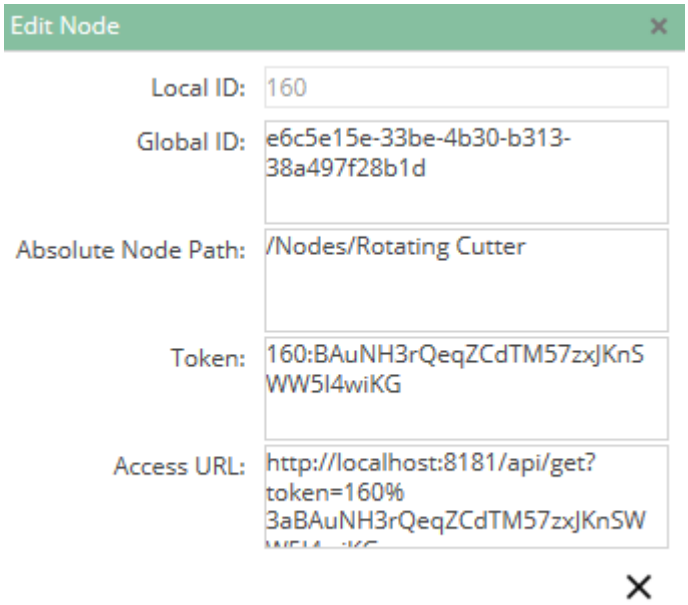
Das ausgewählte Item wird auf den selektierten Node in der Baumstruktur verlinked. Folgende Icons sind möglich:

	verlinkter Folder Node
	verlinkter Datenpunktnode

Wird auf einen Datenpunktnode einn Folder Node verlinkt, wird dieser automatisch zum Folder Node umgewandelt.

Nodezugriff

Für das ausgewählte Item wird folgender Dialog geöffnet:



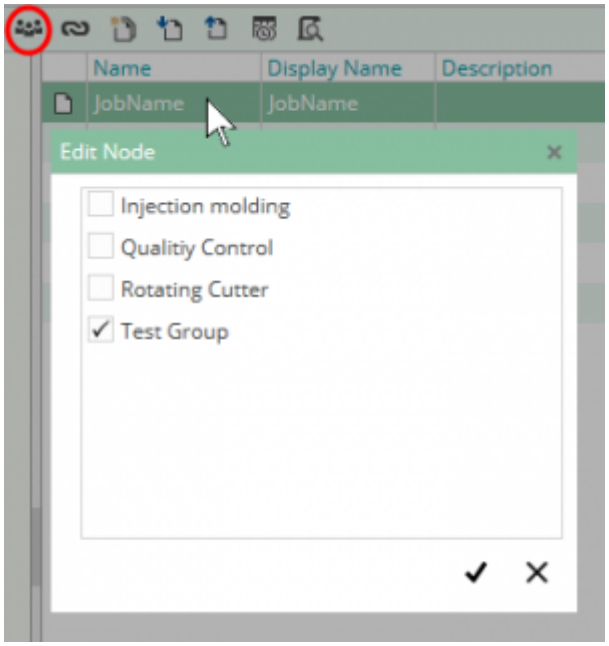
Feld	Beschreibung
Local ID	Eindeutiger, UKI-4.0 -interner ID des ausgewählten Items. Siehe Abkürzungen / Glossar
Global ID	Der GUID identifiziert den Node eindeutig über das System hinaus, z.B. Einsatz von UKI-4.0 an mehreren Standorten, die zusammen einen Datensatz bilden sollen. Abbkürzungen / Glossar
Absolute Node Path	Pfad für den Zugriff auf den Node. Bei verlinkten Datenpunkten wird die Pfadanzeige zum Ursprung des Datenpunktes angezeigt.
Token	Verhält sich wie ein Passwort für JSON, um Zugriff auf den Node zu erhalten.
Access URL	Aktuellen Wert (Actual Value) über http direkt auslesen und in einem Browser anzeigen.

Einem Benutzer einen Nodezugriff hinzufügen / entfernen

Ein Nodezugriff kann nur einer Benutzergruppe hinzugefügt bzw. entfernt werden.
Es gibt zwei Möglichkeiten, dem Benutzer einen Node per Benutzergruppe hinzuzufügen:

- über die Nodeansicht
- über die Benutzergruppe direkt

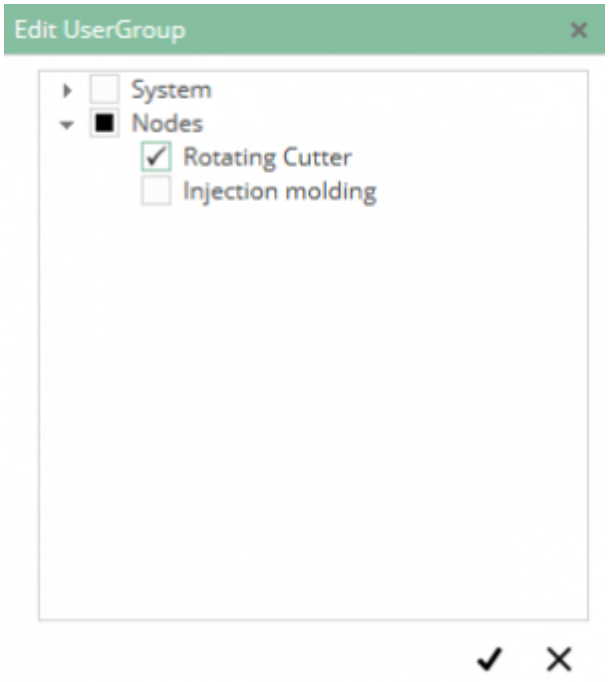
Über Nodeansicht hinzufügen / entfernen



Bei ausgewählten Folder Nodes erben automatisch alle darunter liegenden Nodes die Benutzergruppe.

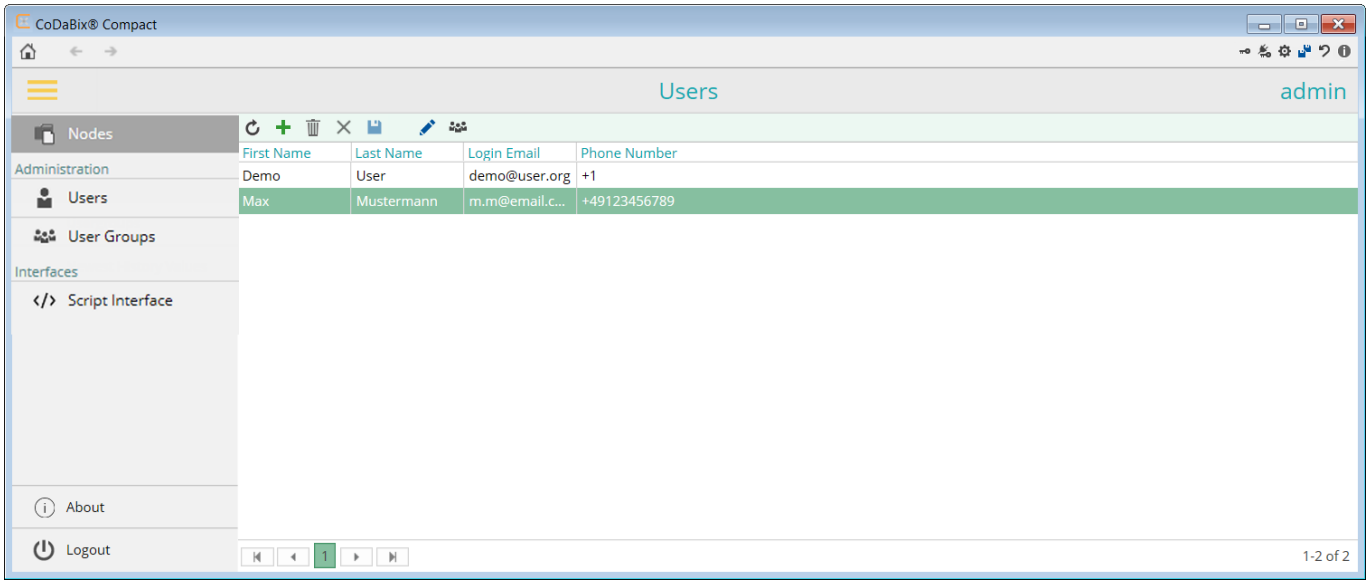
Die Benutzergruppe wird nicht in den darunterliegenden Nodes angezeigt, wenn Sie auf klicken. Somit haben Sie die Möglichkeit, einem bestimmten Folder bzw. Datenpunktnode explizit die Benutzergruppe zu setzten, falls z.B. dem übergeordneten Node der Zugriff für die Gruppe entzogen wird.

Über Benutzergruppe hinzufügen / entfernen



Alle bereits ausgewählten Folder und Datenpunktnodes werden per Haken angezeigt. Wenn Sie einen Folder Node auswählen, werden automatisch alle darunter liegenden Nodes mit aus- bzw. abgewählt.

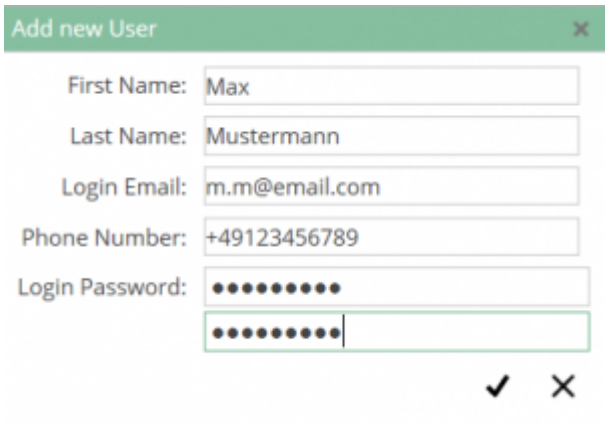
Benutzer



Hier legen Sie die Benutzer an, die Zugriff via OPC UA Client auf die bereitgestellten Daten von UKI-4.0 haben sollen.
Nur der Admin hat das Recht, auf die Konfigurationsseite zuzugreifen.
Funktionen:

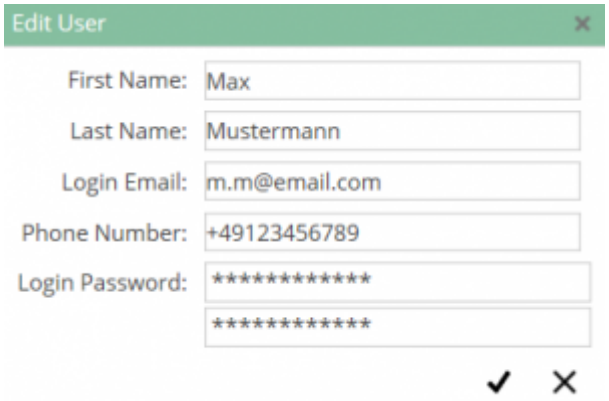
- Benutzer hinzufügen / editieren / löschen
- Benutzerguppe(n) zuweisen

Benutzer hinzufügen



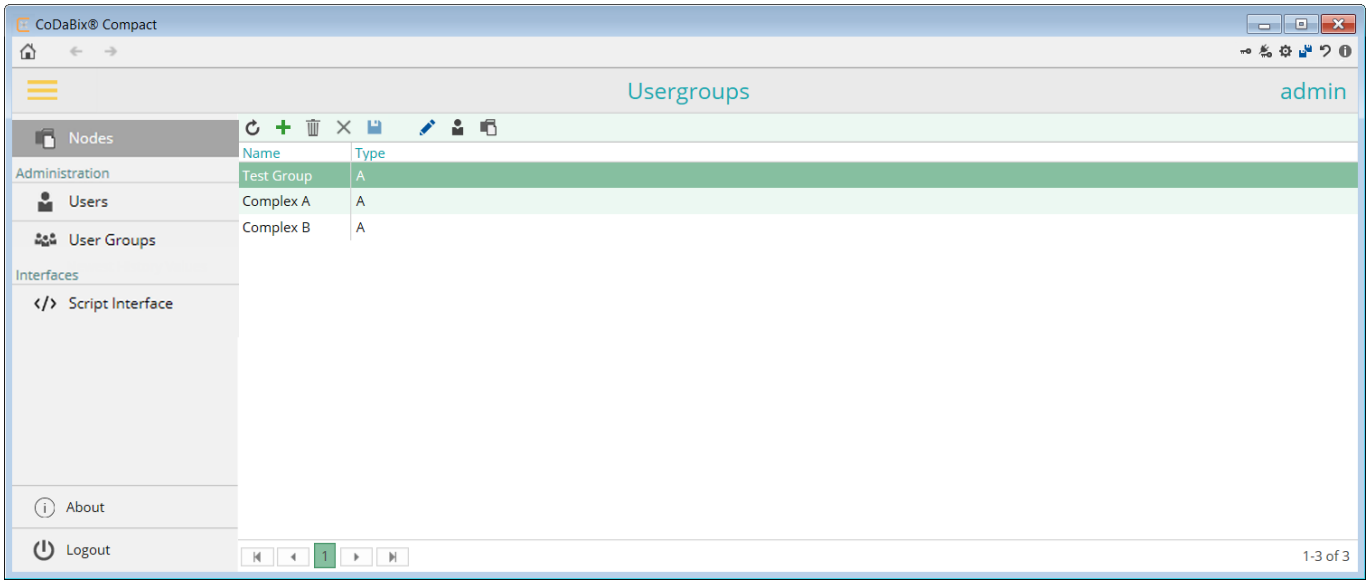
Feld	Beschreibung
First Name	Vorname
Last Name	Nachname
Login Email	Email-Adresse: Darf nur einmal vorkommen
Phone Number	Telefonnummer: Darf nur einmal vorkommen
Login Password	Passwort mit Wiederholung

Benutzer editieren



Es gelten dieselben Einstellungen wie bei Benutzer hinzufügen.

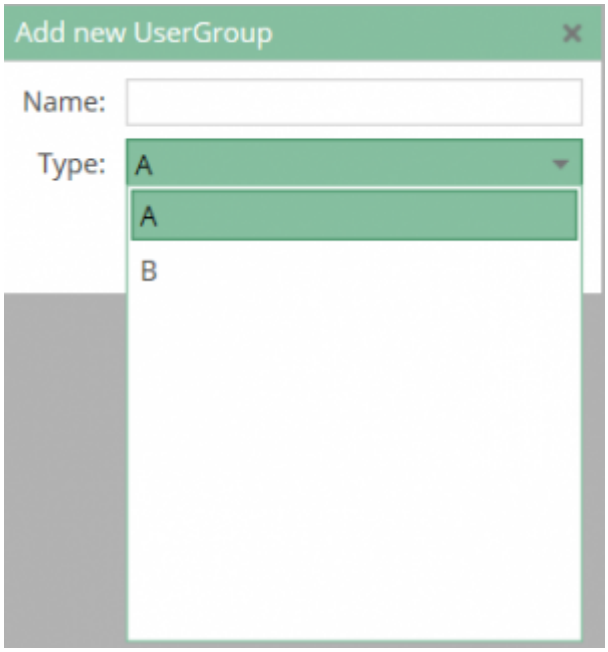
Benutzergruppe



Einem Benutzer kann nicht direkt ein Node zugewiesen werden. Dafür wird immer eine Benutzergruppe benötigt. Dadurch wird die Verwaltung der Nodezugriffe vereinfacht.
Funktionen:

- Benutzergruppe hinzufügen / editieren / löschen
- Benutzer hinzufügen / entfernen
- Node hinzufügen / entfernen

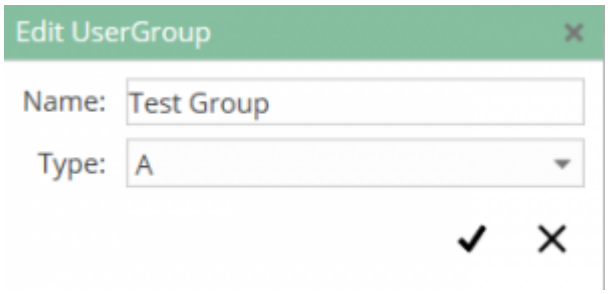
Benutzergruppe hinzufügen



Feld	Beschreibung
Name	Anzeigename der Gruppe
Type	A oder B, aktuell nicht benutzt

Im Moment wird der Benutzertyp nicht verwendet.

Benutzergruppe editieren



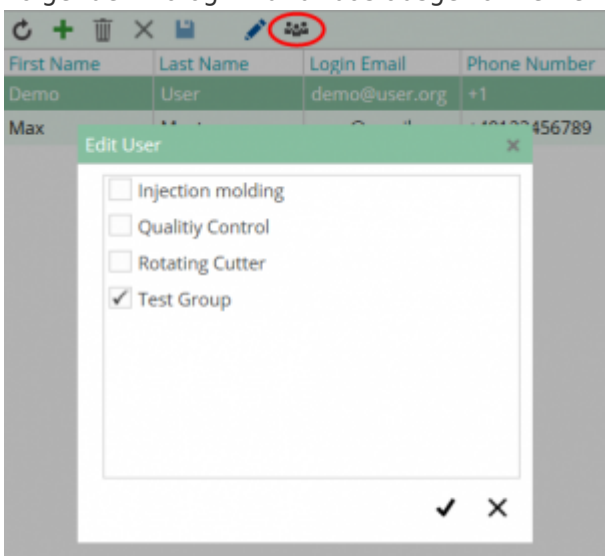
Es gelten dieselben Einstellungen wie bei Benutzergruppe hinzufügen.

Benutzer zur Benutzergruppe hinzufügen

Es gibt zwei Möglichkeiten einem Benutzer zur Benutzergruppe hinzuzufügen:

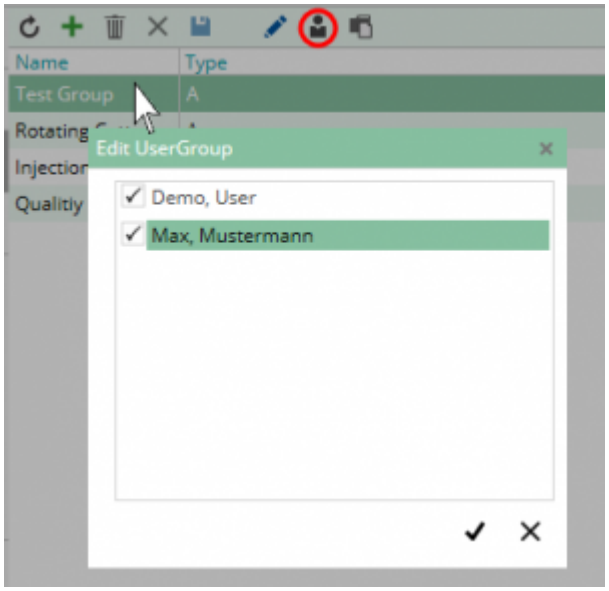
Über das Benutzermenü:

Folgender Dialog wird für das ausgewählte Item angezeigt:



Hier können Sie die Benuztergruppe(n) für das ausgewählte Item hinzufügen oder abwählen.

Über die Benutzergruppe:
Bei Klick auf das Benutzericon wird für die selektierte Benutzergruppe folgender Dialog angezeigt:



Alle verfügbaren Benutzer können hier an- bzw. abgewählt werden.

Plugins

UKI-4.0 kann mittels Plugins um Funktionen erweitert werden. Diese Plugins sind durch ihre Klassifizierung gruppiert. Je nach Kategorie bietet das Plugin eine spezielle Reihe von Dienstleistungen, Organisationen und Knoten. Einige Plugins können Konfigurationsdateien zur Verfügung stellen und zusätzlich, wie die Umstände es erfordern, eine Konfigurationsanwendung enthalten.

Device Plugins

Alle Device Plugins verwenden das UKI-4.0 Device Model. Jedes Gerät das zur Verfügung gestellt wird, wird vom UKI-4.0 Device Manager registriert und verwaltet. Siehe Device Plugins.

S7 Device
Verbindung zur Siemens SIMATIC S7. Siehe S7 Device Plugin.

Exchange Plugins

Alle Exchange Plugins benutzen das UKI-4.0 Storage Model. Jedes Exchange Plugin, das bereitgestellt wird, wird vom UKI-4.0 Storage Manager registriert und verwaltet. Siehe Exchange Plugins.

Datenbank
UKI-4.0 unterstützt die Verbindung zu Datenbanken. Weiteres finden Sie unter Datenbank Plugin.

CSV
Weitere Informationen zum Handling mit CSV-Dateien finden Sie unter CSV Exchange Plugin.

Interface Plugins

UKI-4.0 enthält eine API-Schnittstelle. Weiteres unter Interface Plugins.

REST
Die REST-Schnittstelle ermöglicht den Zugriff auf den Knoten UKI-4.0 via HTTP-Request formatiert als JSON-Objekt. Weiteres unter REST Interface Plugin

Script Plugins

Script Plugins erweitern UKI-4.0 um weitere Funktionalitäten via JavaScript Editor. Sie werden in einer sicheren Umgebung ausgeführt.
Siehe Script Interface Plugin.

OPC UA Server Interface
UKI-4.0 enthält einen OPC UA Server zum Austausch der Daten. Weiters unter OPC UA Server Interface Plugin.

Plugins

Für UKI-4.0 ® sind verschiedene Plugins verfügbar, die durch ihre Klassifizierung gruppiert sind. Je nach Kategorie bietet das Plugin eine spezielle Reihe von Diensten, Entitäten und Nodes an. Einige Plugins können Konfigurationsdateien zur Verfügung stellen und zusätzlich, je nachdem wie es die Umstände erfordern, eine Konfigurationsanwendung.

Verfügbare Plugins

Device Plugins

- Allgemein
- AKLAN Device Plugin
- Allen-Bradley Device Plugin
- EUROMAP Device Plugin
- H1 Device Plugin
- Melsec QJ Device Plugin
- Modbus Device Plugin
- OPC UA Client Device Plugin
- S7 Device Plugin
- MQTT Client Device Plugin
- UniPi Device Plugin

Exchange Plugins

- Allgemein
- CSV Exchange Plugin
- Database Plugin
- SQL Exchange Plugin
- XML Exchange Plugin

Interface Plugins

- Allgemein
- OPC UA Server Interface Plugin
- REST Interface Plugin
- Script Interface Plugin

Aktivierung

Während des Startvorgangs von UKI-4.0 ® werden alle in dem Pluginverzeichnis befindlichen Plugins (<UKI-4.0 `PluginsDir`>) rekursiv durchlaufen, geladen, instanziiert und in alphabetischer Reihenfolge gestartet. Jedes Plugin, das sich nicht in dem Pluginverzeichnis oder in einem Unterverzeichnis befindet, wird weder geladen noch vom UKI-4.0 ® Plugin Manager gestartet.
Während die UKI-4.0 ® Engine gestartet wird, ist es möglich, ein Plugin neu zu starten. Dies geschieht, indem eine Anfrage mittels Plugin-Proxyinstanz an den UKI-4.0 ® Plugin Manager gesendet wird (beachten Sie, dass diese Funktion ohne Vorankündigung die Änderungen vornimmt).

Zustandsautomat

Während des Startvorgangs und während der Laufzeit von UKI-4.0 ® kann das Plugin durch verschiedene Zustände laufen. Jeder Status beschränkt den möglichen Übergang von einem Zustand in den anderen, während einige von ihnen aufeinanderfolgend auftreten. Die Plugininstanz bleibt selbst in dem Zustand Created, nachdem der UKI-4.0 ® Plugin-Manager diese geladen und instanziiert hat.
Nachdem der Plugin Manager alle Plugins geladen und instanziiert hat, ruft die Startsequenz die Startsequenz der Plugins auf. Während des Startvorgangs wird der Status auf Starting gesetzt. Nach erfolgreichem Start des Plugins wird der Status auf Started gewechselt. Wenn das Plugin nicht gestartet werden kann, wird der Status wieder auf den Wert, in dem er vor Starting war, gesetzt. Nur Plugins mit dem Status Started können gestoppt werden.

Ein Plugin wird während der Neustartsequenz oder beim Herunterfahren des UKI-4.0 ® Plugin Managers gestoppt. Das Plugin verarbeitet dann seine Stopsequenz (nur im Zustand Started). Während dieser Sequenz ändert sich der Pluginstatus zu Stopping. Bei erfolgreichem Beenden des Plugins wird der Status auf Stopped gesetzt. Wenn das Plugin nicht gestoppt werden kann, fällt es in seinen Zustand zurück, bevor es zu Stopping geändert wurde.

Fehlerdiagnose

Die zur Verfügung gestellten Diagnoseinformationen von einem Plugin hängen von der Klassifizierung des Plugins ab und ob das Plugin spezifische Diagnoseinformationen selbst zur Verfügung stellt. Das S7 Device Plugin z.B. ist ein Device Plugin und stellt Statusinformationen und andere Diagnoseinformationen über sein UKI-4.0 ® Device, seinen UKI-4.0 ® Device Channel und seine S7 Device Variablen bereit.

Speicherort

Alle Plugin-spezifischen Dateien werden in einem eigenen Pluginverzeichnis innerhalb des `Plugin`-Verzeichnisses abgelegt, welches sich im UKI-4.0 ® Installationsverzeichnis befindet (ausgewählt während der Installation). Die folgende Liste zeigt die Hierarchie:

- <UKI-4.0 `InstallDir>`
  - `plugins\`
    - `<PluginName>`
      - `<PluginAssembly>`
      - `…`

Zum Beispiel:

- `C:\Program\Files\TIS\`UKI-4.0 `\`
  - `plugins\`
    - `S7Device\`
      - UKI-4.0 `.S7DevicePlugin.dll`
      - `…`

Klassifizierung

Device Plugins

Alle als Device Plugin klassifizierten Plugins binden physische oder logische Geräte in UKI-4.0 ® ein. Ein

Gerät selbst kann jede Art von Ressource sein, die durch das Kanalmodell, welches durch das UKI-4.0 ® Device Modell definiert ist, für UKI-4.0 ® zugänglich gemacht wird.

Im Allgemeinen definiert ein Device Plugin einen spezialisierten Typen eines UKI-4.0 ® Devices unter Verwendung des UKI-4.0 ® Device Models (für mehr Informationen siehe Device Plugins) und stellt dadurch die nötige Zugangsschicht für diese Art von Gerät bereit.

Exchange Plugins

Alle als Exchange Plugin klassifizierten Plugins, verbinden Storage Engines mit UKI-4.0 ®. Eine Storage Engine selbst kann jede Art von Datenbank-Managementsystem sein (DBMS), dessen Instanzen durch das relationale Modell unter Verwendung vom UKI-4.0 ® Exchange Modell zugänglich sind. Die Storage Engine selbst muss mindestens ein primitives relationales Modell zur Verfügung stellen, z.B. eine Tabelle (= Einheit in DBMS).

Im Allgemeinen definiert ein Exchange Plugin einen spezialisierten Typen eines UKI-4.0 ® Storages unter Verwendung des UKI-4.0 ® Storage Models (weitere Informationen finden Sie unter Exchange Plugins) und stellt dadurch die notwendige Zugangsschicht für die Art von Storage Engine bereit.

Interface Plugins

Alle als Interface Plugin klassifizierten Plugins verbinden UKI-4.0 ® mit anderen Plattformen und Technologien. Das Interface kann jede Art von Sprache, Dienst, Protokoll etc. sein. Das API Modell muss nur die spezielle Umgebung des Interfaces auf die von UKI-4.0 ® definierte API-Schnittstelle abbilden.

Im Allgemeinen definiert ein Interface Plugin einen spezialisierten Typen einer Plattform oder Technologie-API (weitere Informationen unter Interface Plugins) und stellt dadurch die notwendige Zugangsschicht für die UKI-4.0 ® Programmierschnittstelle bereit.

Konfiguration

Verwenden einer Anwendung

Speicherort:

Im Falle, dass das Plugin eine Konfigurationsanwendung zur Verfügung stellt, kann sich diese im `<`UKI-4.0 `InstallDir>`-Verzeichnis befinden. Die nachfolgende Liste zeigt die Hierachie:

- `<`UKI-4.0 `InstallDir>`
  - `<PluginConfigurationAppName>`
  - `…`

um Beispiel:

- `C:\Program Files\TIS\`UKI-4.0 `\`
  - UKI-4.0 `.S7DevicePlugin.Configurator.exe`
  - `…`

Verwenden einer Konfigurationsdatei

Speicherort

Falls das Plugin eine Konfigurationsdatei benutzt, kann diese im Pluginverzeichnis innerhalb des UKI-4.0 - Projektverzeichnisses (eingestellt in UKI-4.0) liegen. Die nachfolgende Liste zeigt die Hierachie:

- <UKI-4.0 ProjectDir>
  - plugins\
    - <PluginConfigurationFileName>
    - ...

Zum Beispiel:

- D:\Data\TIS\UKI-4.0 .Data\
  - UKI-4.0 .S7DevicePlugin.Settings.xml
  - [UKI-4.0 .S7DevicePlugin.Settings.xsd]
  - …

Struktur

Die XML-basierte Plugin-Konfigurationsdatei definiert mindestens das „PluginSettings“ Element, während die weiteren definierten Elemente von der Klassifizierung des Plugins und eigener benutzerdefinierter Elemente und Attribute abhängen.

```
<?xml version="1.0" encoding="utf-8" ?>
<PluginSettings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <!-- Plugin specific child elements  -->
</PluginSettings>
```

Objektattribute

Einige Elemente in der Konfigurationsdatei erstellen eine Entität in UKI-4.0 ® oder symbolisieren diese. Für solche Elemente sind bestimmte Attribute reserviert, um solche Entitäten leichter zu warten.

Entität `Identifier` Attribut
Dieses Attribut ist vom Typ GUID. Dieser ID wird beim Erstellen der Entität von UKI-4.0 ® erstellt. Durch diese Kennung (= in UKI-4.0 ® der Global Node Identifier) ist es möglich, die Entität eindeutig zu identifizieren und dem Konfigurationselement eindeutig zuzuordnen.

Falls ein Konfigurationselement fehlt oder das mit dem ID zu identifizierende Element bzw. Attribut nicht gefunden wird, wird angenommen, dass eine neue Entität erstellt werden soll. Wenn eine neues Konfigurationselement erzeugt werden soll, ist es nicht nötig, das „Identifier“ Attribut zu setzen. Soll ein bestehendes Element geändert werden, ist es unerlässlich, das „Identifier“ Attribut zu setzten.

Entität `ChangeType` Attribut
Die folgenden Werte gelten für alle Attribute dieser Art:

Wert	Beschreibung
„None“	Das Entitäts-Konfigurationselement wurde nicht rekursiv geändert.
„Created“	Das Entitäts-Konfigurationselement ist neu und erfordert, dass eine neue Entität geschaffen werden muss.
„Updated“	Das Entitäts-Konfigurationselement wurde geändert und erfordert, dass die dargestellte Entität aktualisiert werden soll.
„Deleted“	Das Entitäts-Konfigurationselement ist veraltet und erfordert, dass die dargestellte Entität gelöscht werden soll.

Für den Fall, dass das Attribut den Wert hat:

- „None“: Keine besondere Aktion muss erfolgen und der Wert wird ignoriert. Das Plugin evaluiert anschließend das Attribut.
- „Created“: Das `Identifier` Attribut wird benutzt (falls gesetzt), um die Entität zu finden. Wird diese Entität nicht gefunden, wird eine neue Entität erstellt. Der Global Node Identifier der Entität wird dem `Identifier` Attribut hinzugefügt. Das Plugin evaluiert anschließend das Attribut.

- `„Updated“`: Das `Identifier` Attribut wird benutzt, um die Entität zu finden. Wird diese Entität nicht gefunden, wird eine neue Entität erstellt. Der übergebene Wert wird upgedatet. Das Plugin evaluiert anschließend das Attribut.
- `„Deleted“`: Das `Identifier` Attribut wird benutzt, um die Entität zu finden. Wird diese Entität gefunden, wird diese gelöscht. Nachfolgend überprüft das Plugin, ob das Attribute entfernt wurde.

Synchronisation

Sobald ein Plugin durch den UKI-4.0 ® Plugin Manager geladen und gestartet wurde, wird die Konfigurationsdatei (falls vorhanden) eingelesen und vom Plugin mit den entsprechenden UKI-4.0 ® Entitäten synchronisiert.

Wird die Konfigurationsdatei geändert, informiert der Plugin Manager das entsprechende Plugin. Das Plugin entscheidet daraufhin, ob ein Neustart oder eine Synchronisation der entsprechenden Konfigurationsentitäten stattfindet.

Wird mindestens eine Konfigurationsentität geändert, muss das Plugin das Entity Change Event behandeln und die Entität(en) mit der Konfigurationsdatei synchronisieren.

Device Plugins

Alle Device Plugins benutzen das UKI-4.0 ® Device Modell. Jedes Gerät das zur Verfügung gestellt wird, wird vom UKI-4.0 ® Device Manager registriert und verwaltet.
Mögliche Geräte:

- Physische Geräte wie:
  - SIMATIC S7 SPSen (siehe S7 Device Plugin)
  - Allen Bradley SPSen
  - Mitsubishi SPSen (siehe Melsec QJ Device Plugin)
  - Raspberry Pi erreichbar via TCP/IP
  - Arduino- / Netduino-Projekte erreichbar z.B. via USB
- Logische Geräte wie:
  - Dateien im lokalen oder Remote-Dateisystem, z.B. eine Verbindung zu einem Remote-Laufwerk - das ist eine Datei in einem Cloud-Laufwerk oder eine Datei, die via (S)FTP zugänglich sein könnte - muss noch eingerichtet und solange bereitgehalten werden, während die Daten Zugriff auf die Datei nehmen und vom System verwaltet werden.
  - Dienstleistungen wie lokale Dienste über Shared Memory oder Remote Services, zugänglich über RPC, REST, DCOM oder andere Arten von Protokollen wie SOAP. Ein solcher lokaler / Remote Service kann auch ein OPC Server sein (siehe OPC UA Client Device Plugin).

Device Modell

Das UKI-4.0 ® Device Modell erweitert das grundlegende UKI-4.0 ® Entity Modell um für Devices typische Entities. Dabei definiert ein Device Entity die untergeordneten Entities zur Steuerung (engl. control), für Einstellungen (engl. settings), für den Status des Plugins und die diversen Kanäle (engl. channels), über die das Plugin die unterstützten Geräte an UKI-4.0 ® anbindet.

Konfiguration

Jedes mit UKI-4.0 ® ausgelieferte Device Plugin lässt sich in der UKI-4.0 ® Host Anwendung konfigurieren. Verfügt ein Plugin über Konfigurationsparameter, dann können diese im entsprechenden Device Entity modifiziert werden.

UKI-4.0UKI-4.0 UKI-4.0UKI-4.0 ® verwenden

Die gesamte Konfiguration aller Device Plugins finden Sie unter dem Nodepfad `/System/Devices`. Dieser Wurzelnode der Device Plugins ermöglicht die vollständige Konfiguration der Device Plugins, vorausgesetzt, dass eines der aktiv verwendeten Device Plugins eigene Device Entities zur Konfiguration bereitstellt.

Anwendung verwenden

Speicherort

Falls ein Plugin eine Anwendung für die Konfiguration zur Verfügung stellt, kann es sich unter dem UKI-4.0 ® Installationsverzeichnis befinden (ausgewählt während der Installation). Die folgende Liste zeigt die Hierarchie:

- <UKI-4.0 InstallDir>
  - <PluginConfigurationAppName>
  - …

Zum Beispiel:

- C:\Program Files\TIS\UKI-4.0 \
  - UKI-4.0 .S7DevicePlugin.Configurator.exe
  - …

Konfigurationsdatei verwenden

Speicherort

Falls ein Plugin eine Konfigurationsdatei zur Verfügung stellt, kann diese in einem pluginspezifischen Verzeichnis liegen. Dieses Verzeichnis ist wiederum innerhalb des „plugins“-Verzeichnisses im UKI-4.0 ® Datenverzeichnis (konfiguriert in UKI-4.0). Die folgende Liste zeigt die Hierarchie:

- <UKI-4.0 DataDir>
  - plugins\
    - <PluginConfigurationFileName>
    - …

Zum Beispiel:

- D:\Data\TIS\UKI-4.0 .Data\
  - UKI-4.0 .S7DevicePlugin.Settings.xml
  - [UKI-4.0 .S7DevicePlugin.Settings.xsd]
  - …

Struktur

Jedes UKI-4.0 ® Plugin definiert die Wurzel seiner Elementstruktur durch das `PluginSettings` Element, wie in Konfiguration - Verwenden einer Konfigurationsdatei beschrieben. Ein Device Plugin erweitert seine XML-Struktur anhand des `Device` Elements, während die weiteren definierten Child Elemente abhängig sind von der Implementierung des Plugins und seiner eigenen definierten Elemente und Attribute.

Device Element
Das `Device` Element dient als Container für das `Channels` Element. Dieses Element konfiguriert den kompletten Typ des Devices, das durch das Device Plugin bereitgestellt werden soll, während die weiteren definierten Child Elemente, eigens definierten Elemente und Attribute von der Implementierung im Plugin abhängig sind.

Das `Device` Element kann wie folgt aussehen:

```
<Device>
  <Channels />
</Device>
```

Channels Element
Das `Channels` Element dient als Container für ein oder mehrere `Channel` Elemente. Dieses Element pflegt alle Channels, die mit dem Typ in Verbindnung stehen und durch das Device Plugin bereitgestellt werden, während die weiteren Child Elemente von der Implementierung der eigenen Elemente und Attribute im Plugin abhängig sind.

Das `Channels` Element kann wie folgt aussehen:

```
<Channels>
  <!-- 0-n Channel elements -->
</Channels>
```

Channel Element

Das `Channel` Element dient als Container für die `Settings` Elemente, während die weiteren Child Elemente von der Implementierung der eigenen Elemente und Attribute im Plugin abhängig sind.

Jedes `Channel` Element stellt die folgende Liste von Attributen bereit:

	Verpflichtent	Type	Purpose
Identifier	nein	GUID	Der generische eindeutige Identifizierer des Kanals.
Name	ja	String	Der eindeutige Name (im `Channels` Element) des Kanals.

Das `Channel` Element kann wie folgt aussehen:

```
<Channel Name="Channel No. 1">
  <Settings />
</Channel>
```

Settings Element

Das `Settings` Element definiert das Attribut, um den Kanal einzurichten. Dieses Attribut konfiguriert die Kanalverbindungsparameter, während die weiteren Child Elemente von der Implementierung der eigenen Elemente und Attribute im Plugin abhängig sind.

Das `Settings` Element kann wie folgt aussehen:

```
<Settings />
```

Beispiel Konfigurationsdatei

DevicePlugin.Settings.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<PluginSettings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Device>
    <Channels>
      <Channel>
        <Settings />
        <!-- Plugin specific child elements -->
      </Channel>
    </Channels>
  </Device>
  <!-- Plugin specific child elements  -->
</PluginSettings>
```

Allen-Bradley Device Plugin

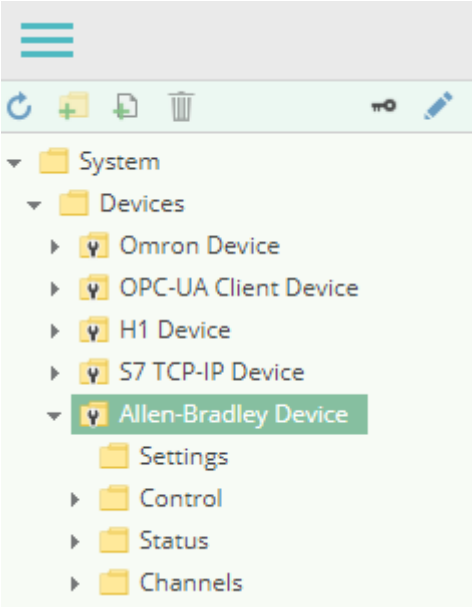
Das Allen-Bradley Device Plugin ermöglicht das Lesen und Schreiben von Daten von Allen-Bradley-SPS-Geräten über EtherNet/IP.

Folgende Gerätetypen werden unterstützt:

- ControlLogix/CompactLogix
- Micro800
- MicroLogix
- PLC-5
- SLC 500

Konfiguration

Die gesamte Allen-Bradley Device Plugin-Konfiguration befindet sich unter dem Nodepfad `/System/Devices/Allen-Bradley Device`.



Channel

Ein Allen-Bradley Device Channel repräsentiert die Verbindung zu einer Allen-Bradley SPS.

Settings

Address

IP-Adresse der Allen-Bradley-SPS.

Device Type

Der Gerätetyp der Allen-Bradley-SPS. Es werden folgende Gerätetypen unterstützt:

- ControlLogix/CompactLogix
- Micro800

- MicroLogix
- PLC-5
- SLC 500

CIP Path

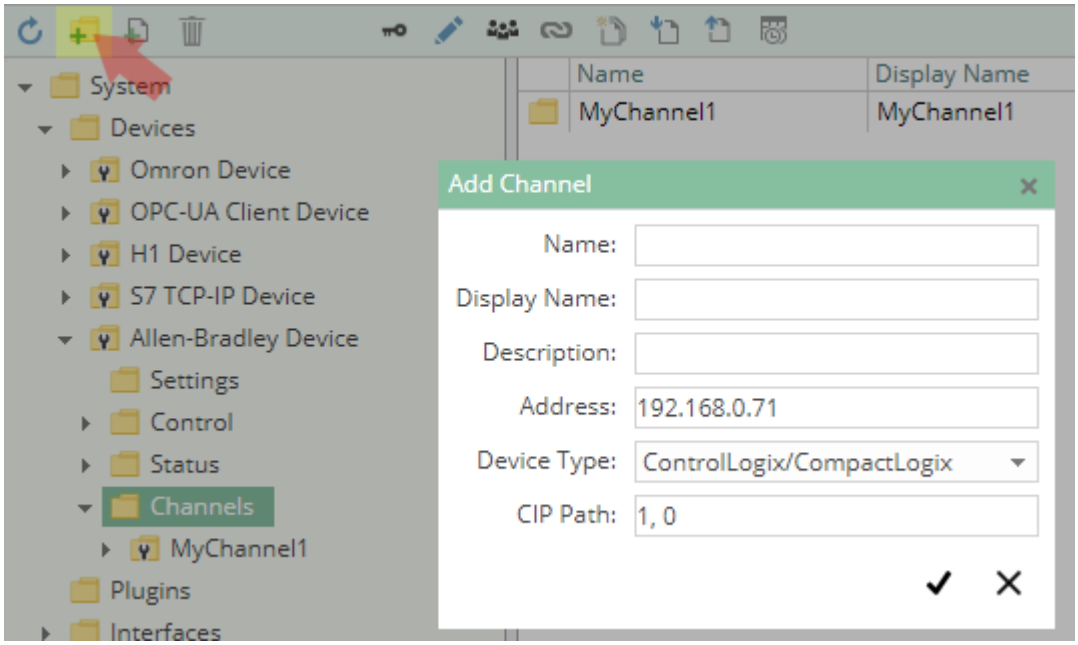
Der CIP-Pfad zur SPS. Dieses Feld muss angegeben werden bei ControlLogix/CompactLogix sowie bei Verwendung einer DH+-Protokollbrücke (d.h. ein DHRIO-Modul).
Dieses Feld kann das Format `A,B` haben, wobei `A` den Porttyp angibt (1=Backplane), und `B` den Slot angibt, in dem die CPU gesteckt ist.

Beispiel: `1,0`

Operation Timeout

Das Timeout, das bei Lese- und Schreiboperationen angewendet werden soll.

Hinzufügen eines Channels



Um einen neuen Allen-Bradley-Channel zu erstellen, gehen Sie wie folgt vor:

1. Fügen Sie einen Folder Node unter dem Node `Allen-Bradley Device/Channels` hinzu, oder machen Sie einen Rechtsklick auf den `Allen-Bradley Device/Channels`-Node und wählen Sie `Add Channel` aus.
2. Tragen Sie im `Add Channel`-Dialog die Settings für die Allen-Bradley-Verbindung ein.
3. Nachdem Sie „Save“ geklickt haben, wird die Channel-Node erstellt.
4. Sie können den Kanal starten, indem Sie die Channel-Node auswählen und den Startbutton klicken.

Variablen

Unter dem `Variables`-Node können Sie Datenpunktnodes erstellen, die aus der Allen-Bradley-SPS als **Tags** gelesen und in diese geschrieben werden.
Die `Value Type`-Eigenschaft muss dabei auf den entsprechenden Typen des Tags festgelegt werden.
Aktuell werden folgenden Tag-Typen unterstützt:

Tag-Typ	UKI-4.0 -Typ
SINT	SByte bzw. SByte-Array
USINT	Byte bzw. Byte-Array
INT	Int16 bzw. Int16-Array
UINT	UInt16 bzw. UInt16-Array
DINT	Int32 bzw. Int32-Array
UDINT	UInt32 bzw. UInt32-Array
LINT	Int64 bzw. Int64-Array
ULINT	UInt64 bzw. UInt64-Array
REAL	Single bzw. Single-Array
STRING (oder benutzerdefinierter String-Datentyp)	String bzw. String-Array

Bei **String**-Datentypen wird ein ASCII/ISO-8859-1-ähnliches Encoding angenommen (das High Byte des 16-Bit-Chars wird abgeschnitten). Dabei werden zwar Surrogate-Pairs (für Characters außerhalb der Unicode-BMP) nicht korrekt behandelt, jedoch ist damit die Anzahl der Bytes gleich der Länge des Strings.

Beachten Sie: Das jeweilige signierte/unsignierte Pendant eines Datentyps kann auch anstelle dessen verwendet werden. Z.B. kann ein SINT auch als Byte statt als SByte verwendet werden; jedoch wird das MSB (most significant bit) dann anders interpretiert, da dieses bei einem signierten Datentypen das Vorzeichenbit ist.

Path

Über die Path-Eigenschaft des Nodes wird der Name des Tags, optional eine Typangabe, sowie (bei Arrays) optional ein Offset und die Länge der Daten definiert:

```
<TagName>
<TagName>, <Length>
<TagName>, <Type>
<TagName>, <Type>[<Length>]
```

Platzhalter	Beschreibung
<TagName>	Der vollständige Name des Tags. Falls das Tag ein Programm-Tag ist, muss als Präfix Program:<Programmname>. mit dem entsprechenden Programmnamen verwendet werden. Falls das Tag eine Struktur ist, kann nach einem Punkt (.) der Name des zu verwendenden Feldes angegeben werden. Falls das Tag (oder ein Struktur-Feld) ein Array ist, kann mit der Syntax [Offset] das Array-Offset angegeben werden.
<Length>	Falls das Tag ein Array ist, kann hier die Länge des Arrays angegeben werden. Diese wird nur beim Lesen verwendet; beim Schreiben wird die Länge aus dem zu schreibenden Array ermittelt.
<Type>	Falls vorhanden, gibt eine Typspezifikation an, wenn diese nicht aus dem UKI-4.0 -Typen abgeleitet werden kann. Aktuell können folgende Typen angegeben werden: • STRING:<MaxLen>: Gibt die maximale Anzahl an String-Characters an, wenn die Variable vom Typ String bzw. String-Array ist. Dies kann bei benutzerdefinierten String-Datentypen verwendet werden; ansonsten wird der vordefinierte STRING-Datentyp mit max. 82 Characters verwendet.

Beispiele

Value Type	Path	Erklärung
`Int16`	`Local:1:O.Data`	`Data`-Feld (`INT`) des digitalen Ausgangs (`Int16` enthält Bitmaske der 16 digitalen Ausgänge (Digital Output Points))
`Int16`	`Local:1:I.Data`	`Data`-Feld (`INT`) des digitalen Eingangs (`Int16` enthält Bitmaske der 16 digitalen Eingänge (Digital Input Points))
`Int32`	`MyControllerTag`	Controller-Tag `MyControllerTag` (`DINT`)
`Int32`	`Program:MainProgram.MyTag1.MyField1`	Vom Programm `MainProgram` das Tag `MyTag1` (Struktur) mit dem Feld `MyField1` (`DINT`)
`Byte-Array`	`Program:MainProgram.MyTag2, 20`	Vom Programm `MainProgram` das Tag `MyTag2` (`SINT[20]`) mit Länge 20 (Bereich 0..20)
`Byte-Array`	`Program:MainProgram.MyTag2[2], 16`	Vom Programm `MainProgram` das Tag `MyTag2` (`SINT[20]`) ab Index 2 mit Länge 16 (Bereich 2..18)
`Byte-Array`	`Program:MainProgram.MyTag3[4].MyField1[1], 5`	Vom Programm `MainProgram` das Tag `MyTag3` (Struktur`[5]`) bei Index 4 das Feld `MyField1` (`SInt[10]`) ab Index 1 mit Länge 5 (Bereich 1..6)
`String`	`MyStringTag1`	Controller-Tag `MyStringTag1` (`STRING` mit max. 82 Characters)
`String`	`MyStringTag2, STRING:20`	Controller-Tag `MyStringTag2` (benutzerdefinierter String-Typ mit max. 20 Characters)
`String-Array`	`MyStruct.MyStringTag3[2], STRING:20[3]`	Controller-Tag `MyStruct` (Struktur) mit dem Feld `MyStringTag3` (benutzerdefinierter String-Typ mit max. 20 Characters) ab Index 2 mit Länge 3 (Bereich 2..5)

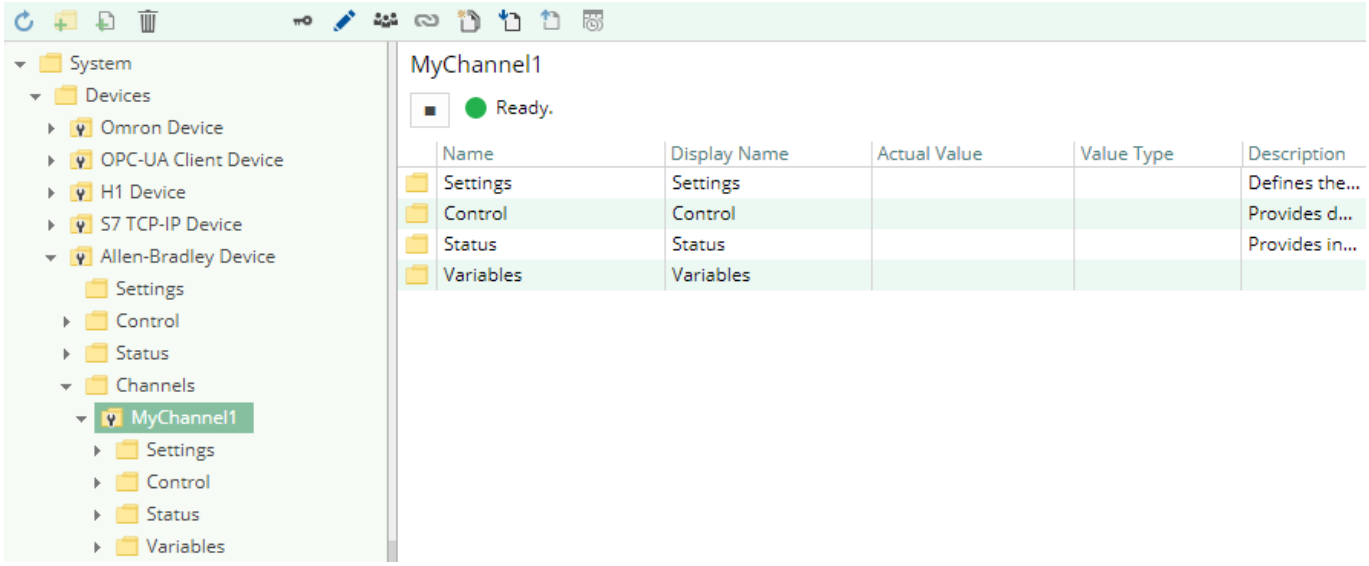
Name	Display Name	Actual Value	Value Type	Description	Path	Status
Output	Output	234	Int16		Local:1:O.Data	Good
Input	Input	0	Int16		Local:1:I.Data	Good
InputFault	InputFault	0	Int32		Local:1:I.Fault	Good
MyRealControllerTag	MyRealControllerTag	0	Single		MyRealControllerTag	Good
Tag1	Tag1	20	Int32		Program:MainProgram.Tag1	Good
Tag4	Tag4	30,5678	Single		Program:MainProgram.Tag4_REAL	Good
Array	Array	[10, 20, 30, 99, 200]	Int32-Array		Program:MainProgram.Tag5Array[0], 5	Good
StructTest1	StructTest1	9999	Int16		Program:MainProgram.MyStructTag1.MyField1	Good
StructTest2	StructTest2	3,1415	Single		Program:MainProgram.MyStructTag1.MyField2	Good

Fehlerdiagnose

Das Allen-Bradley Device Plugin liefert je nach zu untersuchender Schicht verschiedene Statusinformationen. Generell werden die kanalbasierten Diagnoseinformationen durch den Verbindungsstatus des Channels zur SPS produziert. Die variablenbasierten Diagnoseinformationen werden während des Lese-/Schreibzugriffs auf die verschiedenen Variablen produziert.

Kanal

Um den Status des Allen-Bradley-Kanals zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf das folgende Bild:



Das obige Bild zeigt das Bedienfeld des Allen-Bradley-Kanals, das alle statusrelevanten Informationen anzeigt. Das Bedienfeld aktualisiert automatisch seine Statusinformation, wenn ein neuer Status verfügbar ist.

Statuskreis

Farbe	Bedeutung
	Der Kanal ist gestoppt. Klicken Sie den ►-Button, um ihn zu starten.
	Der Kanal startet oder stoppt gerade oder wartet auf den Verbindungsaufbau.
	Der Kanal ist bereit für Lese-/Schreiboperationen. Sie ihn können durch Klick auf den ■-Button stoppen.
	Der Kanal läuft, aber die Verbindung ist momentan fehlerhaft. Bitte überprüfen Sie den Statustext für weitere Informationen.

Variablen

Um den Status der verschiedenen Variablen zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf die in UKI-4.0 angezeigte `Status`-Eigenschaft der Spalte. Benutzen Sie den Button „Read actual Value“, um die Werte von der SPS auszulesen und das Ergebnis in den Variablen zu speichern.

Name	Display Name	Actual Value	Value Type	Description	Path	Status
Output	Output		Int16		Local:2:O.Data	Bad: PLCTAG_ERR_NOT_FOUND
Input	Input		Int16		Local:1:I.Data2	Bad: PLCTAG_ERR_BAD_PARAM
InputFault	InputFault		Int32		Local:1:I.	Bad: PLCTAG_ERR_TOO_LARGE
MyRealControllerTag	MyRealControllerTag		Single-Array		MyRealControllerTag, 5	Bad: PLCTAG_ERR_OUT_OF_BOUNDS
Tag2	Tag2		Int32		Program:MainProgram.Tag1	Bad: The operation has timed out.
Tag1	Tag1	20	Int32		Program:MainProgram.Tag1	Good

Häufig vorkommende Fehlertexte:

- **PLCTAG_ERR_NOT_FOUND**: Das Tag wurde in der SPS nicht gefunden. Überprüfen Sie, ob der Name richtig geschrieben wurde.
- **PLCTAG_ERR_BAD_PARAM**: Der Typ oder die Syntax des Tags sind nicht korrekt. Überprüfen Sie den Tagnamen und ob der richtige Value Type verwendet wird.
- **PLCTAG_ERR_TOO_LARGE**: Der gelesene Wert kann nicht in eine Variable von diesem Typ geschrieben, da mehr Daten geliefert wurden, als hineinpassen. Wählen Sie einen Value Type, der

dem SPS-Typ entspricht. Falls das Tag eine Struktur ist, geben Sie bitte den Namen des Feldes an, auf das Sie zugreifen möchten.
- **PLCTAG_ERR_OUT_OF_BOUNDS**: Es wurde versucht, auf Arrayindizes zuzugreifen, die außerhalb des Arraybereichs liegen. Überprüfen Sie die angegebene Länge sowie den Offset des Arrays.
- **The operation has timed out.**: Der Zugriff zur SPS hat ein bestimmtes Zeitlimit überschritten, oder die SPS ist nicht erreichbar.

Logdatei

Alle kanalbezogenen Statusinformationen werden auch in die kanalspezifische Logdatei im `[LoggingFolder]` protokolliert. Jede Logdatei wird nach dem Namensschema `Allen-Bradley Device.<ChannelName>.log` benannt.

Der Inhalt einer solchen Logdatei kann wie folgt aussehen:

```
...
2018-04-11 11:32:37.0 +2: [Error] Error (Severity=High): Code=[-1], Text=[The operation has
timed-out.], Details=[]
...
```

Entities

Wie jedes Device Plugin erweitert das Allen-Bradley Device Plugin das UKI-4.0 Device Modell.

Device

Der Device Typ `AllenBradleyDevice` des Plugins definiert auch den `AllenBradleyDeviceChannel` und erweitert somit die grundlegenden UKI-4.0 `Device` und UKI-4.0 `DeviceChannel` Entities. Während das `AllenBradleyDevice` nur eine Konkretisierung des UKI-4.0 `Device` darstellt, erweitert der `AllenBradleyDeviceChannel` den UKI-4.0 `DeviceChannel` mit den Allen-Bradley Variable Entities.

Channel

Der Kanal wird von einem Channel Worker behandelt, der eine TCP-Socket-Verbindung zur SPS herstellt.

Der Worker liest standardmäßig keine Werte. Wenn ein Anwender oder Plugin in UKI-4.0 einen synchronen Lesevorgang der `Channel`-Variablen anfordert (z.B. mit der „Read actual value“-Funktion in der UKI-4.0 Webkonfiguration), liest der Channel Worker diese aus der SPS und schreibt diese in die entsprechenden UKI-4.0 -Nodes.

Ähnlich schreibt der Channel Worker auch die Werte in die SPS, wenn ein Client oder Plugin Werte in die `Channel`-Variablen schreibt.

Damit eine Allen-Bradley-Variable regelmäßig gelesen wird, können Sie in der Webkonfiguration bei dem Node „History Options“ auf `Yes` stellen (was eine interne Subscription erstellt), oder Sie können zum Beispiel einen OPC UA Client verwenden, der mit dem OPC UA Server Plugin verbunden ist und damit eine Subscription für die Allen-Bradley Variablennodes erstellen. In diesen Fällen liest der Channel Worker in regelmäßigen Intervallen die Variablen von der SPS und schreibt den neuen Wert nach einer Wertänderung automatisch in dden entsprechenden UKI-4.0 -Node.

Ordner & Dateien

Ordner

Name	Pfad	Zweck / Verwendung
AssemblyFolder	<UKI-4.0 InstallDir>/plugins/AllenBradleyDevicePlugin/	Beinhaltet die Plugin-Assemblydatei.
ConfigFolder	<UKI-4.0 ProjectDir>/plugins/AllenBradleyDevicePlugin/	Beinhaltet die Plugin-Konfigurationsdatei.
LoggingFolder	<UKI-4.0 ProjectDir>/log/	Beinhaltet die Plugin-Logdateien.

Dateien

Typ	Pfad	Zweck / Verwendung
Assembly	[AssemblyFolder]/UKI-4.0 .AllenBradleyDevicePlugin.dll	Die Plugin-Assembly Datei.
Logging	[LoggingFolder]/Allen-Bradley Device.<ChannelName>.log	Die Logdatei.

Versionsinformation

Dieses Dokument

Datum	2019-11-11
Version	1.0

Plugin

Name	Allen-Bradley Device Plugin
Node	/System/Devices/Allen-Bradley Device
Version	1.0.0

Assembly

Name	UKI-4.0 .AllenBradleyDevicePlugin.dll
Datum	2019-11-11
Version	1.0.0.0

EUROMAP Device Plugin

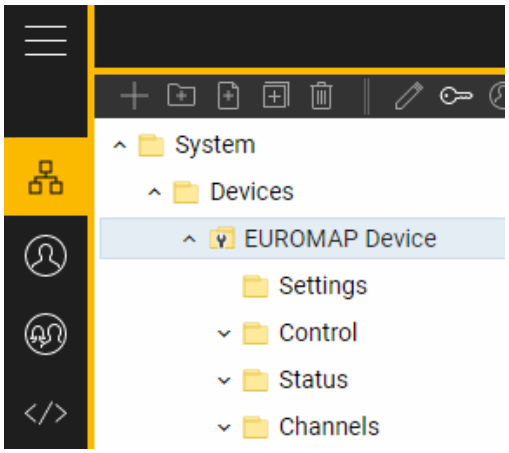
Das EUROMAP Device Plugin ermöglicht das Lesen und Schreiben von Werten von Geräten, die die **EUROMAP 63**-Spezifikation implementieren (dateibasierter Zugriff):

- Lesen von Werten über `REPORT`-Kommandos
- Schreiben von Werten über `SET`-Kommandos
- Browsen der Maschine über `GETID`-Kommandos

Beachten Sie: Das EUROMAP Device Plugin verwendet dateibasierten Zugriff; daher muss das **Session Directory** für UKI-4.0 zugreifbar sein, damit das Plugin funktioniert. Sie müssen daher eine Erlaubnis zum Zugriff auf diesen Pfad in der **Access Security**-Sektion der UKI-4.0 Projekteinstellungen hinzufügen.

Konfiguration

Die gesamte EUROMAP Device Plugin-Konfiguration befindet sich unter dem Nodepfad `/System/Devices/EUROMAP Device`.



Channel

Ein EUROMAP Device Channel repräsentiert die Verbindung zu einer EUROMAP-Maschine.

Settings

Protocol

Das zu verwendende Protokoll (derzeit wird nur EUROMAP 63 unterstützt).

Session Path

Der Pfad zum Session-Verzeichnis im lokalen Dateisystem.
Beachten Sie: Damit UKI-4.0 auf den Pfad zugreifen darf, müssen Sie eine Lese-+Schreibberechtigung in der **Access Security**-Sektion der UKI-4.0 -Projekteinstellungen hinzufügen.

Min Session Number

Die kleinste zu verwendende Session-Zahl.

Max Session Number

Die größte (exklusiv) zu verwendende Session-Zahl.

Encoding

Das Encoding, das beim Lesen von Antwortdateien der Maschine verwendet werden soll, um Zeichen außerhalb des ASCII-Bereichs zu unterstützen.

Line Ending

Das zu verwendende Zeilenende. (Diese Einstellung wird derzeit ignoriert; es wird immer CR+LF verwendet.)

List Delimiter

Der zu verwendende Listendelimiter. (Diese Einstellung wird derzeit ignoriert; es wird immer „,“ verwendet.)

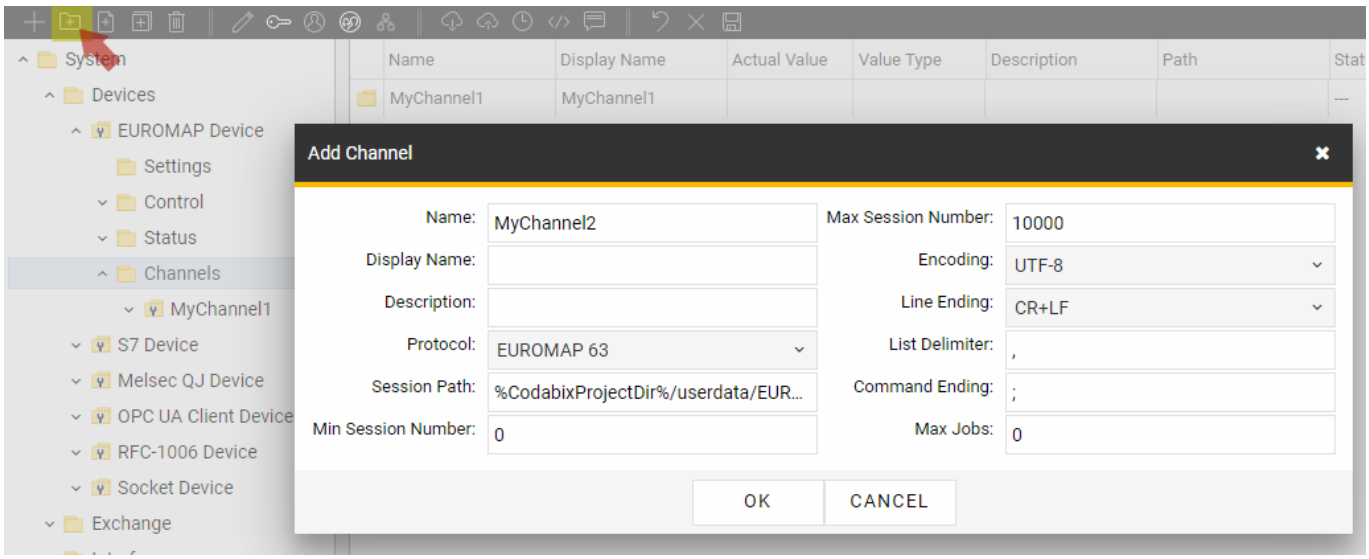
Command Ending

Das zu verwendende Kommando-Ende-Zeichen. (Diese Einstellung wird derzeit ignoriert; es wird immer „;“ verwendet.)

Max Jobs

Die maximale Anzahl an Jobs, die zur gleichen Zeit aktiv sein können. (Diese Einstellung wird derzeit nicht verwendet.)

Hinzufügen eines Channels



Um einen neuen EUROMAP-Channel zu erstellen, gehen Sie wie folgt vor:

1. Fügen Sie einen Folder Node unter dem Node `EUROMAP Device/Channels` hinzu, oder machen Sie einen Rechtsklick auf den `EUROMAP Device/Channels`-Node und wählen Sie `Add Channel` aus.
2. Tragen Sie im `Add Channel`-Dialog die Settings für die EUROMAP-Verbindung ein.
3. Nachdem Sie „Save“ geklickt haben, wird die Channel-Node erstellt.
4. Sie können den Kanal starten, indem Sie die Channel-Node auswählen und den Startbutton klicken.

Parameter

Unter dem `Parameters`-Node können Sie Datenpunktnodes erstellen, die aus der EUROMAP-Maschine gelesen und diese geschrieben werden können.

Unterstützte Node-Werttypen:

- `String`
- Numerische Typen wie `Int32`, `Double`
- `Boolean`

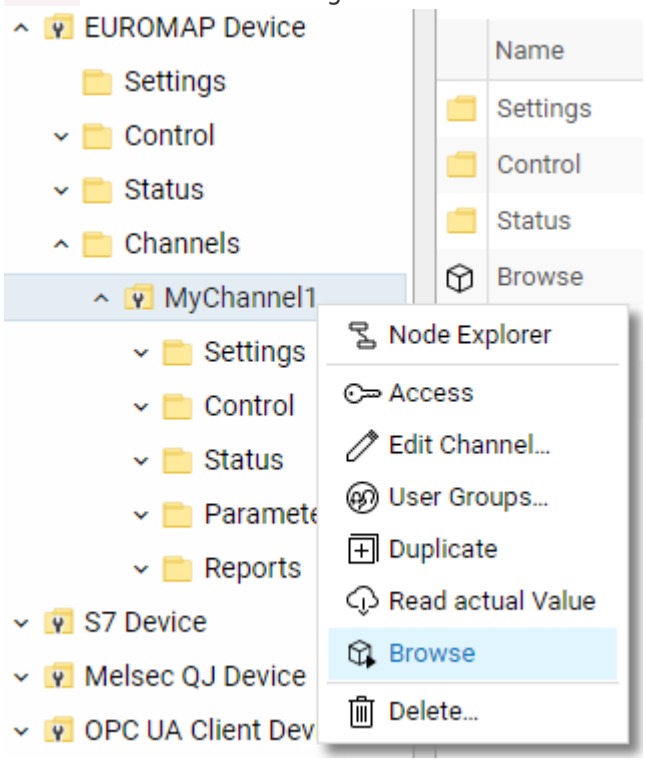
Sie können auch Folder-Nodes erstellen, um Datapoint-Nodes zu gruppieren.

Path

Die `Path`-Property des Nodes wird verwendet, um den Parameternamen in der Maschine festzulegen.

Browse

Sie können die Browse-Methode aufrufen, um die Maschine zu browsen und automatisch die resultierenden Parameter im `Parameters`-Node zu erstellen. Das Browsen wurd durch die Ausführung eines `GETID`-Kommandos durchgeführt.



Lesen/Schreiben

Wenn Parameter-Nodes gelesen werden (durch einen synchronen Lesevorgang oder durch eine Subscription), führt das Plugin ein `REPORT`-Kommando aus, das einen einzelnen Sample der Parameter aufzeichnet, und liest dann die resultierende Repotdatei aus. Die ausgeführte Report-Jobdatei sieht ähnlich aus wie diese:

```
JOB CbxJ-VCT11NK7SJ7 RESPONSE "CBXJQSQI.RSP";
REPORT CbxR-5A2NH4ETRCA "RP3VK4QQ.log" START IMMEDIATE STOP NEVER PARAMETERS @P1, @P2;
```

Wenn Parameter-Nodes geschrieben werden, führt das Plugin ein `SET`-Kommando aus, das die Parameter

auf die angegebenen Werte setzt. Das ausgeführte Set-Kommando sieht ähnlich aus wie dieses:

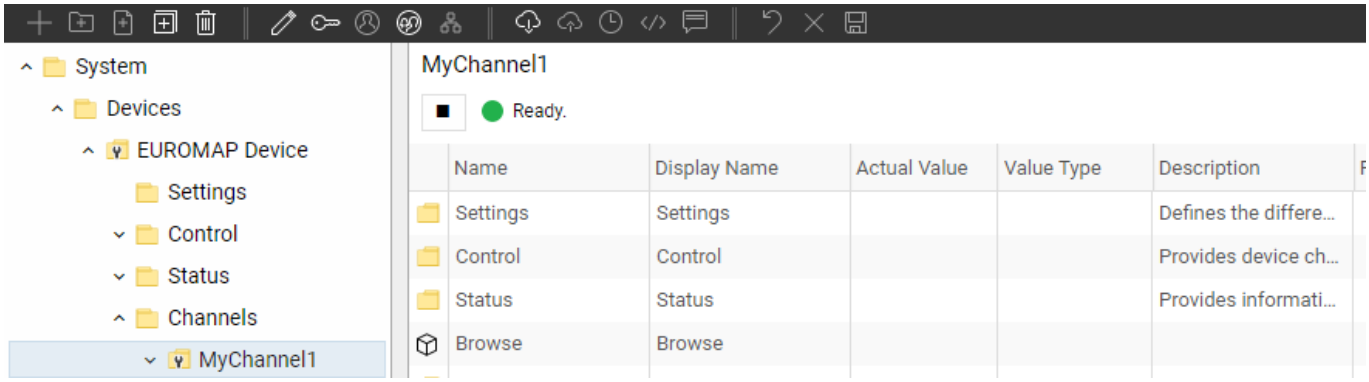
```
JOB CbxJ-LRC9LR6NF9A RESPONSE "CBXJ6RR7.RSP";
SET @P1 "Abcd";
SET @P2 123.45;
```

Fehlerdiagnose

Das EUROMAP Device Plugin liefert je nach zu untersuchender Schicht verschiedene Statusinformationen. Generell werden die kanalbasierten Diagnoseinformationen durch Prüfen, ob der Zugriff auf das angegebene Session-Directory erlaubt ist, produziert. Die parameterbasierten Diagnoseinformationen werden während des Lese-/Schreibzugriffs auf die verschiedenen Parameter produziert.

Kanal

Um den Status des EUROMAP-Kanals zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf das folgende Bild:



Das obige Bild zeigt das Bedienfeld des EUROMAP-Kanals, das alle statusrelevanten Informationen anzeigt. Das Bedienfeld aktualisiert automatisch seine Statusinformation, wenn ein neuer Status verfügbar ist.

Statuskreis

Farbe	Bedeutung
	Der Kanal ist gestoppt. Klicken Sie den ►-Button, um ihn zu starten.
	Der Kanal startet oder stoppt gerade oder wartet auf den Verbindungsaufbau.
	Der Kanal ist bereit für Lese-/Schreiboperationen. Sie ihn können durch Klick auf den ■-Button stoppen.
	Der Kanal läuft, aber die Verbindung ist momentan fehlerhaft. Bitte überprüfen Sie den Statustext für weitere Informationen.

Parameter

Um den Status der verschiedenen Parameter zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf die in UKI-4.0 angezeigte `Status`-Eigenschaft der Spalte. Benutzen Sie den Button „Read actual Value“, um die Werte von der Maschine auszulesen und das Ergebnis in den Parametern zu speichern.

	Name	Display Name	Actual Value	Value Type	Description	Path	Status
	@010X1	@010X1		String		@010X1	Bad: Unable to send an euromap request at session layer: T...
	@010X2	@010X2		Double		@010X2	Bad: Unable to send an euromap request at session layer: T...

Logdatei

Alle kanalbezogenen Statusinformationen werden auch in die kanalspezifische Logdatei im `[LoggingFolder]` protokolliert. Jede Logdatei wird nach dem Namensschema `EUROMAP Device.<ChannelName>.log` benannt.

Der Inhalt einer solchen Logdatei kann wie folgt aussehen:

```
...
2018-04-11 11:32:37.0 +2: [Error] Error (Severity=High): Code=[-1], Text=[The operation has
timed-out.], Details=[]
...
```

Entities

Wie jedes Device Plugin erweitert das EUROMAP Device Plugin das UKI-4.0 Device Modell.

Device

Der Device Typ `EuromapDevice` des Plugins definiert auch den `EuromapDeviceChannel` und erweitert somit die grundlegenden UKI-4.0 `Device` und UKI-4.0 `DeviceChannel` Entities. Während das `EuromapDevice` nur eine Konkretisierung des UKI-4.0 `Device` darstellt, erweitert der `EuromapDeviceChannel` den UKI-4.0 `DeviceChannel` mit den EUROMAP Parameter Entities.

Channel

Der Kanal wird von einem Channel Worker behandelt, der Dateien über das Session-Verzeichnis mit einer Maschine austauscht, um Jobs zu starten und die von der Maschine erzeugten Antwortdateien einzulesen.

Der Worker liest standardmäßig keine Werte. Wenn ein Anwender oder Plugin in UKI-4.0 einen synchronen Lesevorgang der `Channel`-Parameter anfordert (z.B. mit der „Read actual value“-Funktion in der UKI-4.0 Webkonfiguration), liest der Channel Worker diese aus der dazugehörigen Maschine und schreibt diese in die entsprechenden UKI-4.0 -Nodes.

Ähnlich schreibt der Channel Worker auch die Werte in die Maschine, wenn ein Client oder Plugin Werte in die `Channel`-Parameter schreibt.

Damit ein EUROMAP-Parameter regelmäßig gelesen wird, können Sie in der Webkonfiguration bei dem Node „History Options“ auf `Yes` stellen (was eine interne Subscription erstellt), oder Sie können zum Beispiel einen OPC UA Client verwenden, der mit dem OPC UA Server Plugin verbunden ist und damit eine Subscription für die EUROMAP-Parameternodes erstellen. In diesen Fällen liest der Channel Worker in regelmäßigen Intervallen die Parameter von der Maschine und schreibt den neuen Wert nach einer Wertänderung automatisch in den entsprechenden UKI-4.0 -Node.

Ordner & Dateien

Ordner

Name	Pfad	Zweck / Verwendung
AssemblyFolder	<UKI-4.0 InstallDir>/plugins/EuromapDevicePlugin/	Beinhaltet die Plugin-Assemblydatei.
ConfigFolder	<UKI-4.0 ProjectDir>/plugins/EuromapDevicePlugin/	Beinhaltet die Plugin-Konfigurationsdatei.
LoggingFolder	<UKI-4.0 ProjectDir>/log/	Beinhaltet die Plugin-Logdateien.

Dateien

Typ	Pfad	Zweck / Verwendung
Assembly	[AssemblyFolder]/UKI-4.0 .EuromapDevicePlugin.dll	Die Plugin-Assembly Datei.
Logging	[LoggingFolder]/EUROMAP Device.<ChannelName>.log	Die Logdatei.

Versionsinformation

Dieses Dokument

Datum	2020-12-07
Version	1.0

Plugin

Name	EUROMAP Device Plugin
Node	/System/Devices/EUROMAP Device
Version	1.0.0

Assembly

Name	UKI-4.0 .EuromapDevicePlugin.dll
Datum	2020-12-07
Version	1.0.0.0

H1 Device Plugin

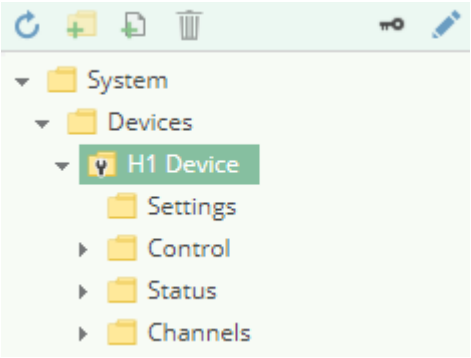
Das H1 Device Plugin ermöglicht das Lesen und Schreiben von Daten von SIMATIC S5 SPS-Geräten unter Verwendung des SINEC-H1-Protokolls basierend auf ISO/MAC (Industrial Ethernet).

Voraussetzungen

- Dieses Plugin funktioniert nur unter **Windows**.
- Um dieses Plugin verwenden zu können, muss WinPcap 4.1.3 installiert sein.

Konfiguration

Die gesamte H1 Device Plugin-Konfiguration befindet sich unter dem Nodepfad `/System/Devices/H1 Device`.



Channel

Ein H1 Device Channel repräsentiert die Verbindung zu einer H1-SPS.

Settings

Network Adapter

Die NIC (Netzwerkkarte), die für die Netzwerkkommunikation verwendet werden soll.

MAC Address

Die MAC-Adresse des Remote-Geräts.

Read SSAP/DSAP

Die SSAP-(Quell-SAP)- und DSAP-(Ziel-SAP)-Werte, die beim Lesen von Daten verwendet werden sollen. Es werden maximal 8 Zeichen verwendet.

Write SSAP/DSAP

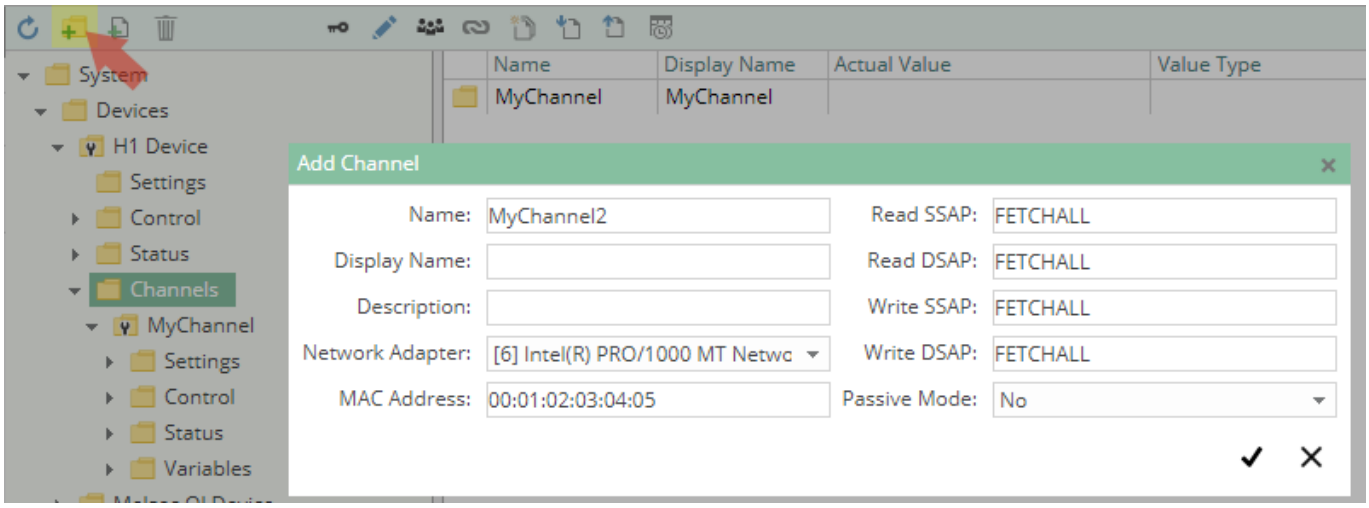
Die SSAP-(Quell-SAP)- und DSAP-(Ziel-SAP)-Werte, die beim Schreiben von Daten verwendet werden sollen. Es werden maximal 8 Zeichen verwendet.

Passive Mode

Gibt an, welcher Teilnehmer die Verbindung initiiert:

- `No`: Die lokale Maschine initiiert die Verbindung.
- `Yes`: Die Remote-Maschine initiiert die Verbindung.

Hinzufügen eines Channels



Um einen neuen H1-Channel zu erstellen, gehen Sie wie folgt vor:

1. Fügen Sie einen Folder Node unter dem Node `H1 Device/Channels` hinzu, oder machen Sie einen Rechtsklick auf den `H1 Device/Channels`-Node und wählen Sie `Add Channel` aus.
2. Tragen Sie im `Add Channel`-Dialog die Settings für die H1-Verbindung ein.
3. Nachdem Sie „Save“ geklickt haben, wird die Channel-Node erstellt.
4. Sie können den Kanal starten, indem Sie die Channel-Node auswählen und den Startbutton klicken.

Variablen

Unter dem `Variables`-Node können Sie Datenpunktnodes erstellen, die von der SPS gelesen und in diese geschrieben werden können.
Die `Value Type`-Eigenschaft muss entsprechend zu dem zugehörigen Operandentyp gesetzt werden.
Derzeit werden folgende Typen unterstützt:

UKI-4.0 **Type**	**Operand**
`Boolean` oder `Boolean-Array`	`X`
`Byte`/`SByte` oder `Byte-Array`/`SByte-Array`	`L` (Links=High Byte) or `R` (Rechts=Low Byte)
`Int16`/`UInt16` oder `Int16-Array`/`UInt16-Array`	`W`
`Int32`/`UInt32` oder `Int32-Array`/`UInt32-Array`	`D`
`Float` oder `Float-Array`	`D`
`TimeSpan` oder `TimeSpan-Array`	`W`

Beachten Sie: 32-Bit-Werte (`Float`, `Int32`) werden als zwei separate (16-Bit)-Wörter gelesen bzw. geschrieben. Wenn der Wert in der SPS sich während des Lesens ändert, kann dies dazu führen, dass ein inkonsistenter Wert gelesen wird, wenn ein Wort bereits den neuen Wert hat, aber das andere Wort noch den alten Wert.

Beachten Sie: Bit-Werte (`Boolean`) werden als (16-Bit)-Wort gelesen bzw. geschrieben. Wenn ein solcher Boolean-Wert geschrieben wird, wird zuerst das Wort von der SPS gelesen, dann die entsprechenden Bits geändert, und dann wird das Wort in die SPS zurückgeschrieben. Wenn ein anderer Teilnehmer gleichzeitig andere Bits dieses Worts ändert, kann dies dazu führen, dass diese Änderungen verloren gehen.

Path

Über die `Path`-Eigenschaft des Nodes wird die Adresse und optional der Typ und/oder (für Arrays) die Länge der Daten festgelegt. Derzeit wird nur `Data Block` (`DB`) als Datenbereich unterstützt.

```
DB<DB number>.D<Operand> <Offset>
DB<DB number>.D<Operand> <Offset>, <Length>
DB<DB number>.D<Operand> <Offset>, <Type>[<Length>]
DB<DB number>.DX <Offset>.<Bit>, <Length>
```

Platzhalter	Beschreibung
`<DB number>`	Die Nummer des Datenblocks.
`<Operand>`	Einer der Operanden, wie sie in der obigen tabelle angegeben sind.
`<Offset>`	Der **Wort-(16-Bit)-Offset**, ab dem auf die Daten zugegriffen werden soll.
`<Bit>`	Bei der Verwendung von `Boolean`/`Boolean-Array` kann die Bitnummer des Worts von 0 bis 15 angegeben werden.
`<Length>`	Die Länge des Arrays.
`<Type>`	Bei der Verwendung von Integer-Typen kann optional ein **BCD**-Typ (Binary Coded Decimal) angegeben werden, um die Werte unter Verwendung der BCD-Kodierung abzulegen. Es kann einer der folgenden Werte angegeben werden: • `BCD`: Der Wert wird BCD-kodiert.

Beispiele

Value Type	Path	Meaning
`Int16`	`DB1000.DW 20`	Im Datenblock 1000, greife auf das Wort bei Offset 20 zu.
`Int16-Array`	`DB1000.DW 30, 10`	Im Datenblock 1000, beginne bei Offset 30 und lies/schreibe 10 Wörter (exklusiver Ende-Offset: 40).
`Int32`	`DB1000.DD 40, BCD[4]`	Im Datenblock 1000, beginne bei Offset 40 und lies/schreibe vier 32-Bit-Werte (exklusiver Ende-Offset: 48) und kodiere/dekodiere die Werte als BCD.
`Boolean-Array`	`DB1000.DX 50.12, 24`	Im Datenblock 1000, greife auf die Bits ab Offset 50.12 bis (exklusives) Offset 52.4 zu (die Bits werden als drei 16-Bit-Wörter gelesen/geschrieben).
`Byte-Array`	`DB1000.DR 60, 2`	Im Datenblock 1000, greife auf die Bytes bei Offset 60 (Low Byte) und 61 (High Byte) zu.

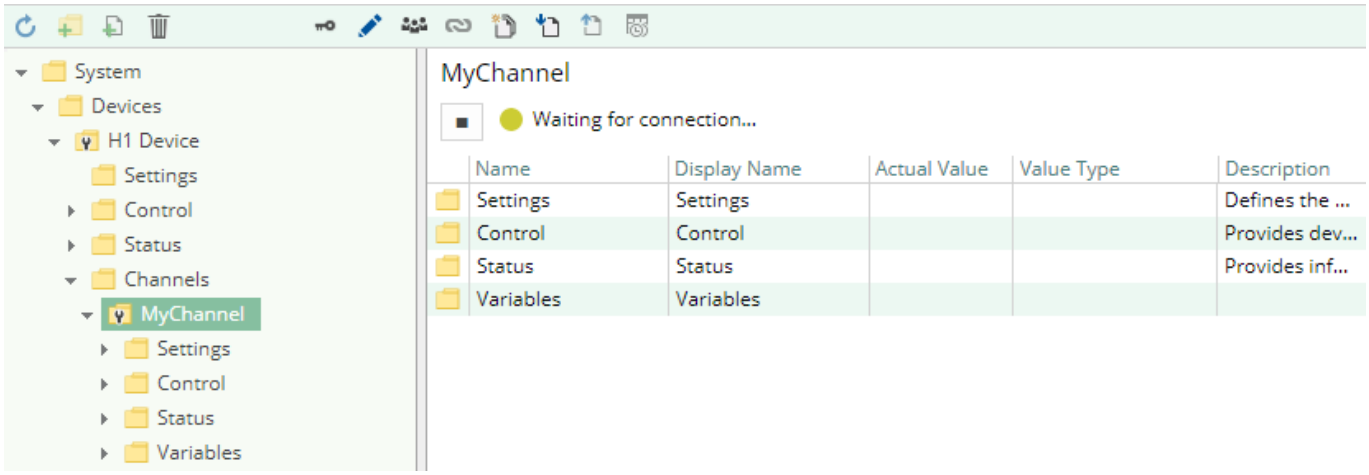
	Name	Display Name	Actual Value	Value Type	Description	Path	Status
	MyInt	MyInt		Int16		DB1000.DW 20	---
	MyIntArray	MyIntArray		Int16-Array		DB1000.DW 30, 10	---
	MyDInt (BCD)	MyDInt (BCD)		Int32		DB1000.DD 40, BCD	---
	MyBooleanArray	MyBooleanArray		Boolean-Array		DB1000.DX 50.12, 24	---
	MyBytes	MyBytes		Byte-Array		DB1000.DR 60, 2	---

Fehlerdiagnose

Das H1 Device Plugin liefert je nach zu untersuchender Schicht verschiedene Statusinformationen. Generell werden die kanalbasierten Diagnoseinformationen durch den Verbindungsstatus des Channels zur SPS produziert. Die variablenbasierten Diagnoseinformationen werden während des Lese-/Schreibzugriffs auf die verschiedenen Variablen produziert.

Kanal

Um den Status des H1-Kanals zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf das folgende Bild:



Das obige Bild zeigt das Bedienfeld des H1-Kanals, das alle statusrelevanten Informationen anzeigt. Das Bedienfeld aktualisiert automatisch seine Statusinformation, wenn ein neuer Status verfügbar ist.

Statuskreis

Farbe	Bedeutung
	Der Kanal ist gestoppt. Klicken Sie den ►-Button, um ihn zu starten.
	Der Kanal startet oder stoppt gerade oder wartet auf den Verbindungsaufbau.
	Der Kanal ist bereit für Lese-/Schreiboperationen. Sie können ihn durch Klick auf den ■-Button stoppen.
	Der Kanal läuft, aber die Verbindung ist momentan fehlerhaft. Bitte überprüfen Sie den Statustext für weitere Informationen.

Variablen

Um den Status der verschiedenen Variablen zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf die in UKI-4.0 angezeigte `Status`-Eigenschaft der Spalte. Benutzen Sie den Button „Read actual Value“, um die Werte von der SPS auszulesen und das Ergebnis in den Variablen zu speichern.

Name	Display Name	Actual Value	Value Type	Description	Path	Status
MyInt	MyInt		Int16		DB1000.DW 20	Bad: The operation has timed-out.
MyIntArray	MyIntArray		Int16-Array		DB1000.DW 30, 10	Bad: The operation has timed-out.
MyDInt (BCD)	MyDInt (BCD)		Int32		DB1000.DD 40, BCD	Bad: The operation has timed-out.
MyBooleanArray	MyBooleanArray		Boolean-Array		DB1000.DX 50.12, 24	Bad: The operation has timed-out.
MyBytes	MyBytes		Byte-Array		DB1000.DR 60, 2	Bad: The operation has timed-out.

Logdatei

Alle kanalbezogenen Statusinformationen werden auch in die kanalspezifische Logdatei im `[LoggingFolder]` protokolliert. Jede Logdatei wird nach dem Namensschema `H1 Device.<ChannelName>.log` benannt.

Der Inhalt einer solchen Logdatei kann wie folgt aussehen:

```
...
2018-04-11 11:32:37.0 +2: [Error] Error (Severity=High): Code=[-1], Text=[The operation has
timed-out.], Details=[]
...
```

Entities

Wie jedes Device Plugin erweitert das H1 Device Plugin das UKI-4.0 Device Modell.

Device

Der Device Typ `H1Device` des Plugins definiert auch den `H1DeviceChannel` und erweitert somit die grundlegenden UKI-4.0 `Device` und UKI-4.0 `DeviceChannel` Entities. Während das `H1Device` nur eine Konkretisierung des UKI-4.0 `Device` darstellt, erweitert der `H1DeviceChannel` den UKI-4.0 `DeviceChannel` mit den H1 Variable Entities.

Channel

Der Kanal wird von einem Channel Worker behandelt, der eine Netzwerkverbindung zur SPS herstellt.

Der Worker liest standardmäßig keine Werte. Wenn ein Anwender oder Plugin in UKI-4.0 einen synchronen Lesevorgang der `Channel` Variablen anfordert (z.B. mit der „Read actual value“-Funktion in der UKI-4.0 Webkonfiguration), liest der Channel Worker diese aus der SPS und schreibt diese in die entsprechenden UKI-4.0 Nodes.

Ähnlich schreibt der Channel Worker auch die Werte in die SPS, wenn ein Client oder Plugin Werte in die `Channel`-Variablen schreibt.

Damit eine Allen-Bradley-Variable regelmäßig gelesen wird, können Sie in der Webkonfiguration bei dem Node „History Options“ auf `Yes` stellen (was eine interne Subscription erstellt), oder Sie können zum Beispiel einen OPC UA Client verwenden, der mit dem OPC UA Server Plugin verbunden ist und damit eine Subscription für die Allen-Bradley Variablennodes erstellen. In diesen Fällen liest der Channel Worker in regelmäßigen Intervallen die Variablen von der SPS und schreibt den neuen Wert nach einer Wertänderung automatisch in die entsprechende UKI-4.0 Node.

Ordner & Dateien

Ordner

Name	Pfad	Zweck / Verwendung
AssemblyFolder	<UKI-4.0 `InstallDir>/plugins/H1DevicePlugin/`	Beinhaltet die Plugin-Assemblydatei.
ConfigFolder	<UKI-4.0 `ProjectDir>/plugins/H1DevicePlugin/`	Beinhaltet die Plugin-Konfigurationsdatei.
LoggingFolder	<UKI-4.0 `ProjectDir>/log/`	Beinhaltet die Plugin-Logdateien.

Dateien

Typ	Pfad	Zweck / Verwendung
Assembly	`[AssemblyFolder]/`UKI-4.0 `.H1DevicePlugin.dll`	Die Plugin-Assembly Datei.

Typ	Pfad	Zweck / Verwendung
Logging	`[LoggingFolder]/H1 Device.<ChannelName>.log`	Die Logdatei.

Versionsinformation

Dieses Dokument

Datum	`2019-11-11`
Version	`1.0`

Plugin

Name	`H1 Device Plugin`
Node	`/System/Devices/H1 Device`
Version	`1.0.0`

Assembly

Name	UKI-4.0 `.H1DevicePlugin.dll`
Datum	`2020-01-07`
Version	`1.0.0.0`

Melsec QJ Device Plugin

Das Melsec QJ Device Plugin stellt eine Verbindung zwischen UKI-4.0 und physischen SPSen der Mitsubishi Melsec-Q-Serie und Melsec-A-Serie über Ethernet (TCP/IP) zur Verfügung.

Voraussetzungen

- Dieses Plugin funktioniert nur unter **Windows**.

Was tut das Plugin?

Das Melsec QJ Device Plugin verbindet und wartet Verbindungen zu einem oder mehreren Melsec-Q- und Melsec-A-SPS-Geräten via TCP/IP.
Jede SPS kann über einen separaten Kanal (`Channel`) verbunden werden.
Für jeden `Channel` wird die Verbindung definiert durch:

- IP-Adresse
- Port
- Device-Typ (Q-Serie, A-Serie).

Die in der Melsec-SPS verfügbaren Variablen können für jeden Kanal separat konfiguriert werden. Das Plugin synchronisiert die Variablenwerte zwischen der SPS und UKI-4.0 ®.

Funktionen

- Lese- und Schreibzugriff auf Melsec-Datentypen (Bit, Word, DWord, Real).
- Automatisiertes Verbindungshandling, auch Auto-Reconnect.

Unterstützte Geräte

- Mitsubishi Melsec-Q-Serie
- Mitsubishi Melsec-A-Serie

Zweck & Anwendung

Zweck

Die angeschlossenen Geräte können einfach mit UKI-4.0 ® gesteuert werden. Durch die Verknüpfung der SPS-Speicher mit den in UKI-4.0 ® definierten Nodes kann die SPS direkt mit vielen anderen Nodes, Geräten, Diensten usw., die in UKI-4.0 ® gewartet werden, interagieren. Auch andere UKI-4.0 ® Teilnehmer können mit den durch das Melsec Device Plugin bereitgestellten SPS-Geräten interagieren.

Anwendung

- Dynamische Generierung von Fertigungsdaten basierend auf verschiedenen Bedingungen und Daten, die von Maschinen, Benutzern, Aufträgen, Zuständen, Diensten usw. generiert werden
- Zusätzliche Sicherheit durch die Überwachung der gesamten Anlage, einschließlich der

betriebsinternen Interaktionen
- Zentrale Steuerung und Durchflussüberwachung zur Früherkennung von möglichen Störungen
- Verbesserung des Anlagenleitstands, indem die von den Geräten erzeugten Produktionsdaten überwacht und aufgezeichnet werden

Installation

Dieses Plugin ist Bestandteil des UKI-4.0 ® Setups. Bitte konsultieren Sie UKI-4.0 ® Setup und erster Start für weitere Informationen darüber, wie dieses Plugin installiert und deinstalliert werden kann.

Anforderungen

- Standardanforderungen von UKI-4.0 ®
- aktivierte ausgehende Verbindungen via TCP-/IP

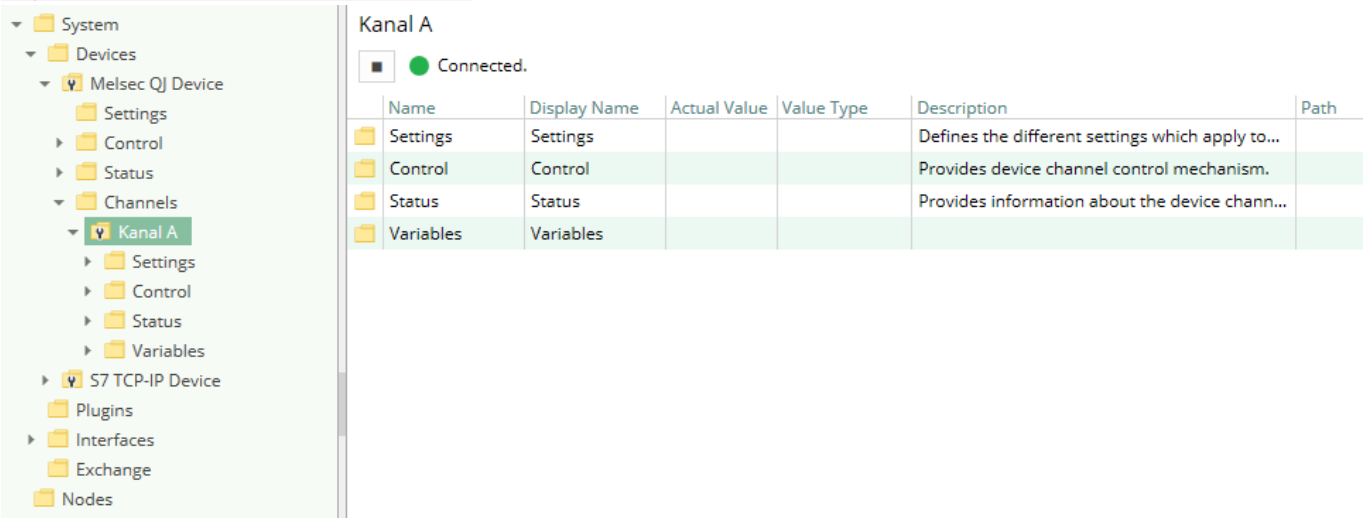
Melsec-SPS-Einstellungen

Bitte folgen Sie dieser Anleitung, um die Melsec-SPS für den Zugriff durch das Melsec QJ Device Plugin zu konfigurieren: Melsec-SPS-Einstellungen

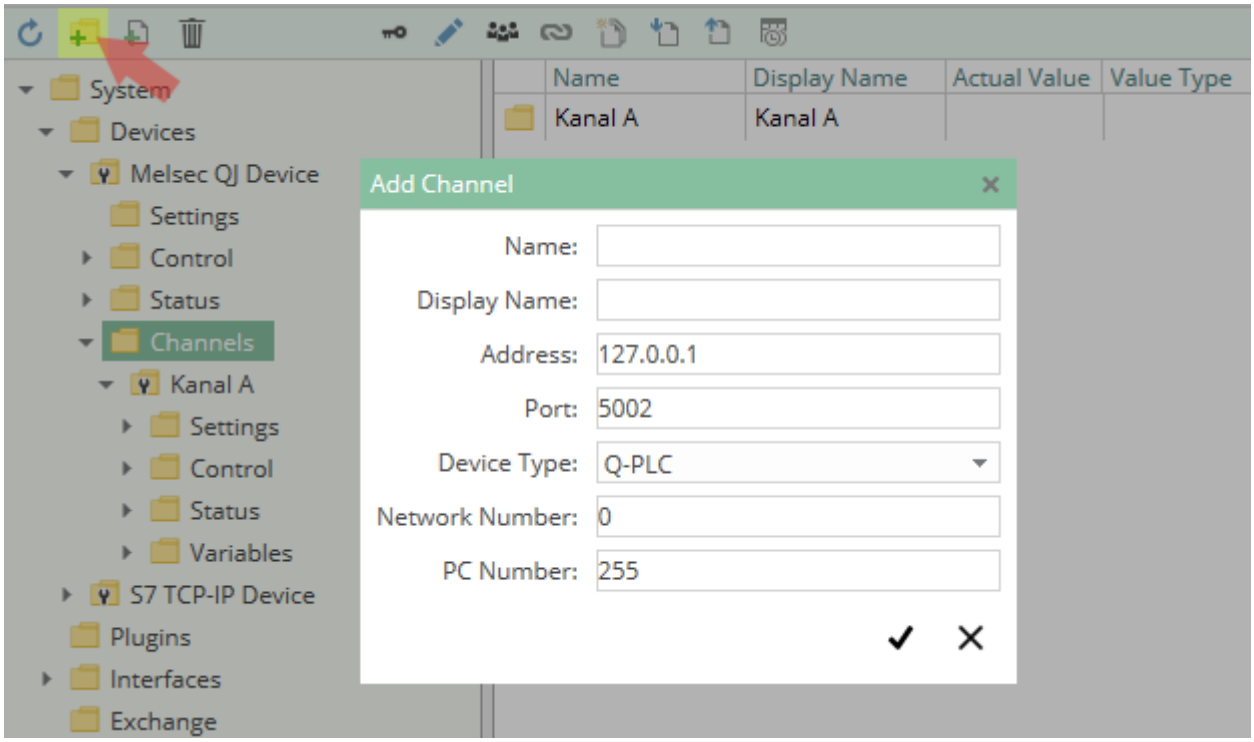
Konfiguration

Übersicht

Die gesamte Melsec QJ Device Plugin-Konfiguration befindet sich unter dem Nodepfad `/System/Device/Melsec QJ Device`.



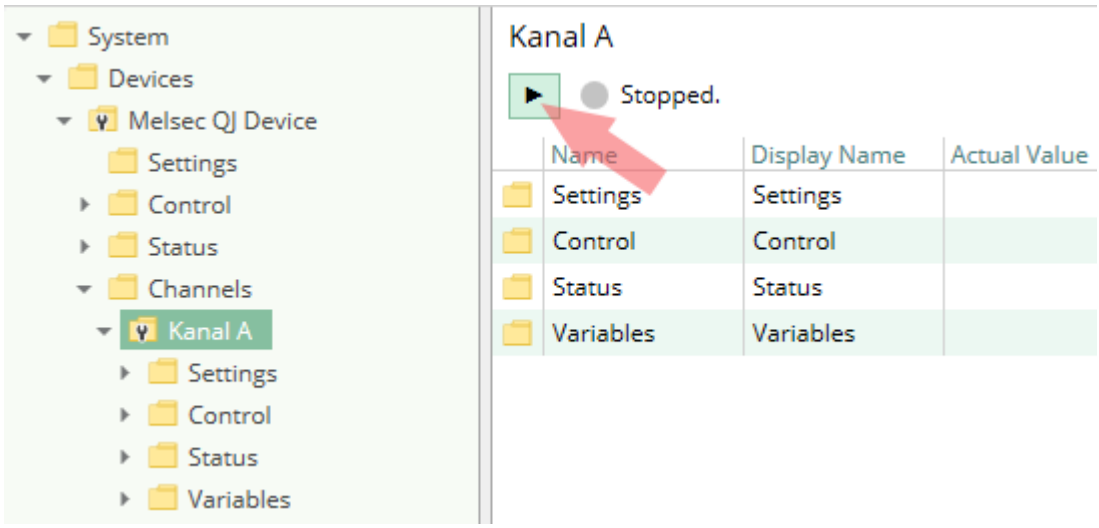
Der Nodebaum im oberen Bild zeigt den Standardnodebaum des Melsec QJ Device Plugins. Um eine oder mehrere Melsec-QJ-Kanäle aufzusetzen, fügen Sie einen Folder Node unter dem Node `Melsec QJ Device/Channels` hinzu, oder machen Sie einen Rechtsklick auf den `Melsec QJ Device/Channels`-Node und wählen Sie `Add Channel` aus.



Kanalspezifische Einstellungen

Name	Typ	Beschreibung
Address	String	Die IP-Adresse der Melsec-SPS, zu der die Verbindung hergestellt werden soll.
Port	Integer	Der Port, zu dem die Verbindung hergestellt werden soll. Standardwert: `5002`
Device Type	Enum	Der Typ der Melsec-SPS, zu dem die Verbindung hersgestellt werden soll. Gültige Werte: `Q-PLC` (Standard) für Q-Serie, `A-PLC` für A-Serie.
Network Number	Integer	Die Netzwerknummer der Melsec-SPS. Normalerweise ist dies `0`.
PC Number	Integer	Die PC-Nummer für die Melsec-SPS. Normalerweise ist dies `255` (für 0xFF).

Nachdem Sie „Save“ geklickt haben, wird die Kanalnode erstellt. Sie können den Kanal starten, indem Sie die Kanalnode auswählen und den Startbutton klicken:



Variablen

Unter dem `Variables`-Node können Sie Datenpunktnodes erstellen, die mit der SPS verbunden werden. Die `Value Type`-Eigenschaft muss dabei auf den Variablentyp festgelegt werden (`Boolean`, `Int16`, `UInt16`, `Int32`, `UInt32`, `Double`, oder die entsprechenden Arraytypen).

Die `Path`-Eigenschaft des Nodes muss dazu die SPS-Adresse enthalten (Datenbereich und Startnummer, diese ist je nach Datenbereich dezimal oder hexadezimal).

Für den bitweisen Zugriff auf Datenbereiche, die keinen Bitzugriff unterstützen, muss zusätzlich die (dezimale) Bitnummer (0-15) nach dem Punkt angegeben werden, z.B. `D 100.2`.
Bei Verwendung eines Arraytyps (z.B. `Int16-Array`) muss im Path zusätzlich nach dem Komma die Arraylänge angegeben werden.

Beispiele:

Value Type	Path	Erklärung
`Int16`	`D 100`	liest 2-Byte-Wort an der Wortadresse 100
`Boolean`	`D 100.15`	liest Bit 15 an der Wortadresse 100
`Boolean`	`M 123`	direkter Bitzugriff auf Bitnummer 123
`Boolean`	`W AF.10`	liest Bit 10 an der Wortadresse AF_H (175)
`Double`	`D 200`	liest 4-Byte-Real-Wert an der Wortadresse 200
`Int32-Array`	`D 220, 10`	liest 10 4-Byte-Werte ab der Wortadresse 220
`Boolean-Array`	`D 100.15, 3`	liest die Bits 100.15, 101.0, 101.1

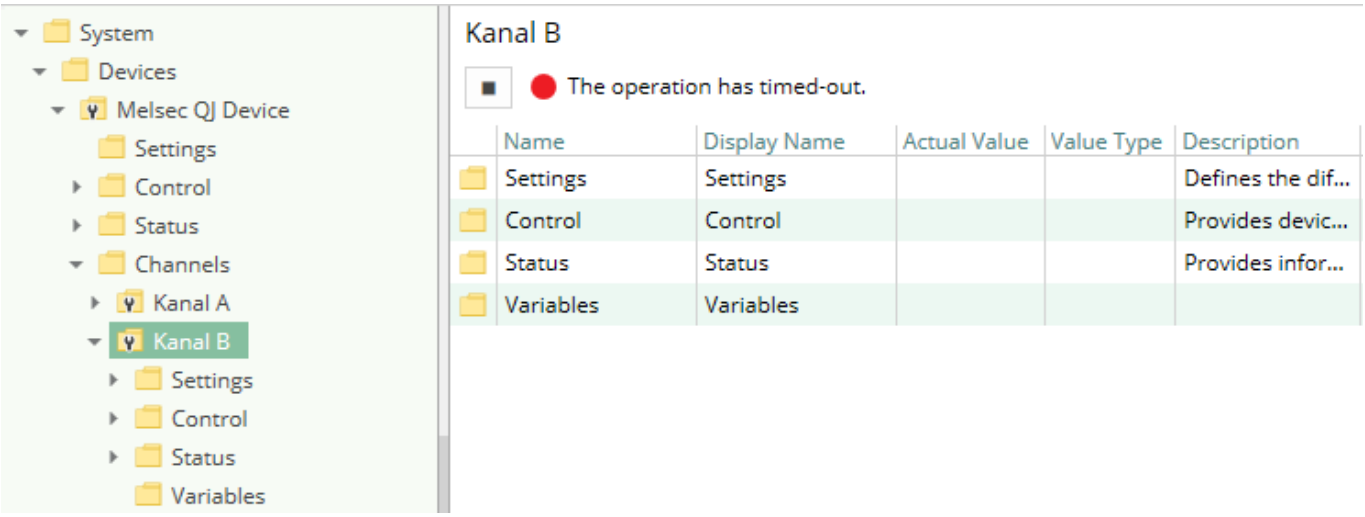
Name	Display Name	Actual Value	Value Type	Description	Path	Status
DataWord 10	DataWord 10	1490	Int16		D 10	Good
DataDWord 10	DataDWord 10	363333074	Int32		D 10	Good
DataWord 11	DataWord 11	5544	Int16		D 11	Good
DataBit 10.0	DataBit 10.0	False	Boolean		D 10	Good
DataBit 10.1	DataBit 10.1	True	Boolean		D 10.1	Good
DataBit 10.2	DataBit 10.2	False	Boolean		D 10.2	Good
DataBit 10.8	DataBit 10.8	True	Boolean		D 10.8	Good
MerkerWord 14	MerkerWord 14	64	Int16		M 14	Good
MerkerWord 15	MerkerWord 15	0	Int16		M 15	Good
MerkerWord 16	MerkerWord 16	0	Int16		M 16	Good

Fehlerdiagnose

Das Melsec QJ Device Plugin liefert je nach zu untersuchender Schicht verschiedene Statusinformationen. Generell werden die kanalbasierten Diagnoseinformationen durch den Verbindungsstatus des Channels zur SPS produziert. Die variablenbasierten Diagnoseinformationen werden während des Lese-/Schreibzugriffs auf die verschiedenen Variablen produziert.

Kanal

Um den Status von verschiedenen SPS-Kanälen zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf das folgende Bild:



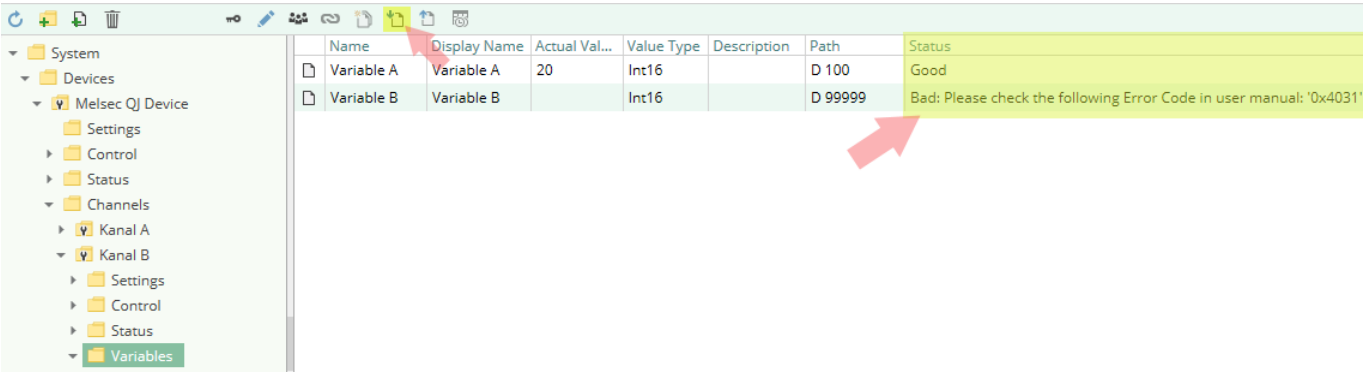
Das obige Bild zeigt das Bedienfeld des SPS-Kanals, das alle statusrelevanten Informationen anzeigt. Das Bedienfeld aktualisiert automatisch seine Statusinformation, wenn ein neuer Status verfügbar ist.

Statuskreis

Farbe	Bedeutung
	Der Kanal ist gestoppt. Klicken Sie den ►-Button, um ihn zu starten.
	Der Kanal startet oder stoppt gerade oder wartet auf den Verbindungsaufbau.
	Der Kanal läuft und es wurde erfolgreich eine Verbindung hergestellt. Sie können ihn durch Klick auf den ■-Button stoppen.
	Der Kanal läuft, aber die Verbindung ist momentan fehlerhaft. Bitte überprüfen Sie den Statustext für weitere Informationen.

Variablen

Um den Status der verschiedenen Variablen zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf die in UKI-4.0 ® angezeigte `Status`-Eigenschaft der Spalte. Benutzen Sie den Button „Read actual Value“, um die Werte aus der SPS auszulesen und das Ergebnis in den Variablen zu speichern.



Logdatei

Alle kanalbezogenen Statusinformationen werden auch in die kanalspezifische Logdatei im `[LoggingFolder]` protokolliert. Jede Logdatei wird nach dem Namensschema `Melsec QJ Device.<ChannelName>.log` benannt. Der Inhalt einer solchen Logdatei kann wie folgt aussehen:

```
...
2018-04-11 11:32:37.0 +2: [Error] Error (Severity=High): Code=[-1], Text=[The operation has
timed-out.], Details=[]
...
```

Entities

Wie jedes Device Plugin erweitert das Melsec QJ Device Plugin das grundlegende UKI-4.0 Device-Modell.

Device

Der Devicetyp `MelsecDevice` des Plugins definiert auch den `MelsecDeviceChannel` und erweitert somit die grundlegenden UKI-4.0 `Device`- und UKI-4.0 `DeviceChannel`-Entities. Während das `MelsecDevice` lediglich eine Konkretisierung des UKI-4.0 `Device` repräsentiert, erweitert der `MelsecDeviceChannel` den UKI-4.0 `ExchangeChannel` mit SQL Tabellenentities.

Channel

Jeder Channel wird von einem Channel Worker behandelt, der eine physische Verbindung zur SPS herstellt. Zum Zweck der Fehlerdiagnose untersucht der Worker die SPS-Verbindung alle 10 Sekunden, um den Statuscode des Channels und die Beschreibung zu aktualisieren, damit Verbindungsausfälle aufgespürt werden.

Standardmäßig liest der Worker keine Werte. Wenn ein Client oder Plugin einen synchronen Lesevorgang des Channels anfordert, liest der Channel Worker die Variablen in UKI-4.0 (z.B. unter Verwendung der UKI-4.0 Webkonfigurations-Funktion „Read actual value“) aus der SPS und schreibt sie dann in die entsprechenden UKI-4.0 Nodes.

Ähnlich schreibt der Channel Worker die Werte in die SPS, die ein Client oder Plugin in die Variablen des Channels schreibt.

Um eine SPS-Variable stetig gelesen zu bekommen, müssen Sie den Node in der Webkonfiguration bearbeiten und „History Options“ auf `Yes` stellen (was eine interne Subscription erstellt), oder Sie können z.B. einen OPC UA Client verbunden mit dem OPC UA Server Interface Plugin benutzen und eine Subscription für die Melsec-Variablennodes anlegen. In diesem Fall liest der Channel Worker die Variablen in einem gleichmäßigen Intervall aus der SPS und, falls der Wert einer der Variablen sich verändert hat, schreibt den neuen Wert in den entsprechenden UKI-4.0 ® Node.

Variable

Jede Melsec-Variable kann anhand eines SPS-Adressoperanden und der Startadresse auf den SPS-Speicher zugreifen. Die Interpretation hängt jedoch von der `Value Type`-Auswahl ab. Unterstützte Variablenformate sind Skalar- und Arraytypen.

Ordner & Dateien

Ordner

Inhalt	Pfad	Zweck / Verwendung
AssemblyFolder	<UKI-4.0 `InstallDir>/plugins/MelsecDevicePlugin/`	Beinhaltet die Plugin-Assemblydatei.
ConfigFolder	<UKI-4.0 `DataDir>/plugins/MelsecDevicePlugin/`	Beinhaltet die Plugin-Konfigurationsdatei.
LoggingFolder	<UKI-4.0 `DataDir>/log/`	Beinhaltet die Plugin-Logdateien.

Dateien

Typ	Pfad	Zweck / Verwendung
Assembly	`[AssemblyFolder]/`UKI-4.0 `.MelsecDevicePlugin.dll`	Die Plugin-Assemblydatei.
DLL	`[AssemblyFolder]/native/windows-x86/`UKI-4.0 `-melsecqjdll.dll`	Die native Plugin-DLL-Datei für Windows (x86).

Typ	Pfad	Zweck / Verwendung
DLL	`[AssemblyFolder]/native/windows-x64/`UKI-4.0 `-melsecqjdll.dll`	Die native Plugin-DLL-Datei für Windows (x64).
Logging	`[LoggingFolder]/Melsec QJ Device.<ChannelName>.log`	Die Logdatei.

Versionsinformation

Dieses Dokument

Datum	`2018-04-13`
Version	`1.0`

Plugin

Name	`Melsec QJ Device Plugin`
Node	`/System/Device/Melsec QJ Device`
Version	`1.0.0`

Assembly

Name	UKI-4.0 `.MelsecDevicePlugin.dll`
Datum	`2018-04-13`
Version	`1.0.0.0`

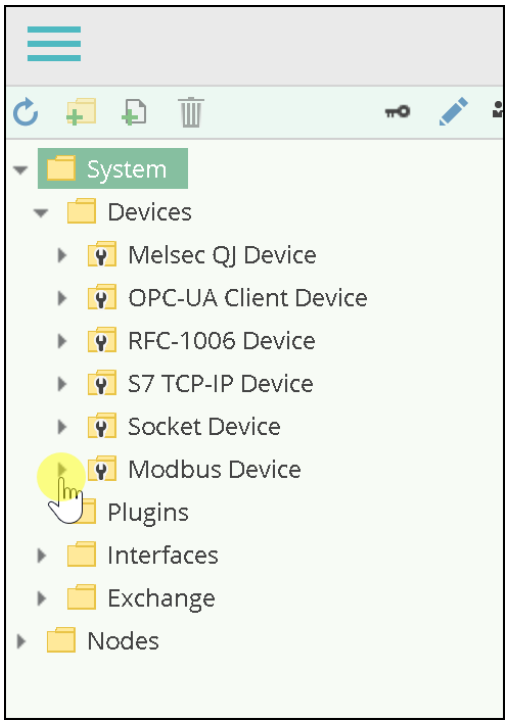
Modbus Device Plugin

Das Modbus Device Plugin ermöglicht das Lesen und Schreiben von Daten via Modbus TCP.

In der aktuellen Version wird der Betrieb als Modbus TCP Client unterstützt.

Konfiguration

Die gesamte Modbus Device Plugin-Konfiguration befindet sich unter dem Nodepfad `/System/Devices/Modbus Device`.



Channel

Ein Modbus Device Channel repräsentiert die Verbindung zu einem Modbus TCP Server.

Settings

IP Address

IP-Adresse des Modbus TCP Servers

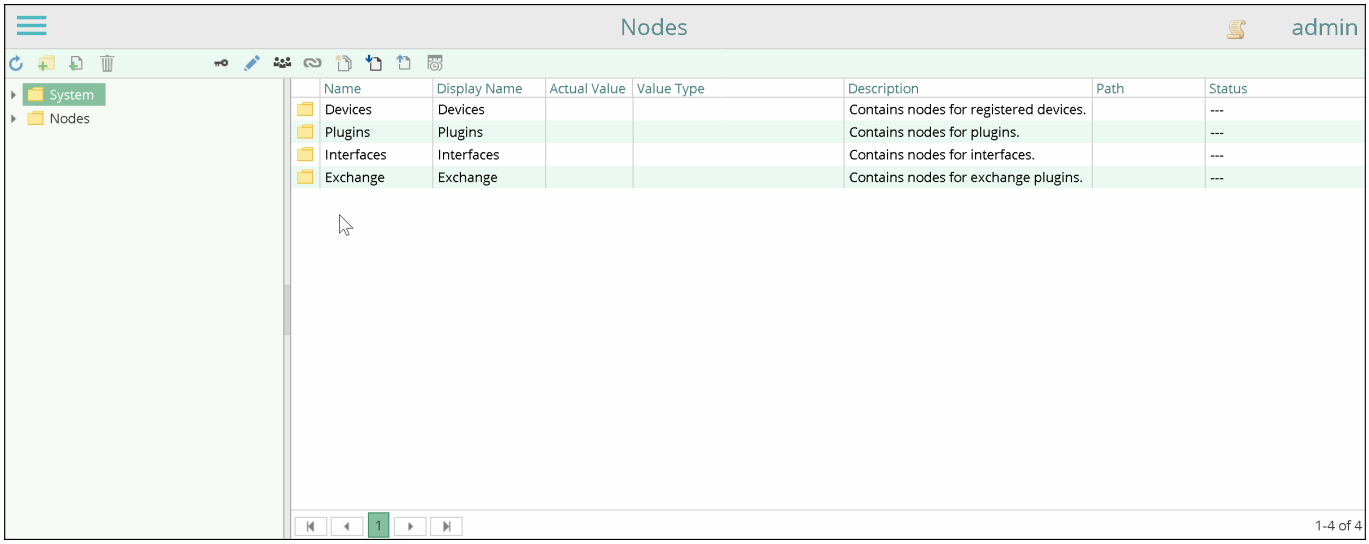
Port

TCP-Port des Modbus TCP Servers. Der Standard-Port für Modbus ist `502`.

Addressing Mode

Adressierungsmodus für das zu verbindende Gerät. Der Standard ist `1-based`.
Wenn im Datenblatt des Modbus Geräts die niedrigste Adresse `1` ist, sollte dieser Modus verwendet werden. Beginnt die Adressierung bei `0`, sollte der Modus `0-based` gesetzt werden.

Hinzufügen eines Channels



Um einen neuen Modbus-Channel zu erstellen, gehen Sie wie folgt vor:

1. Fügen Sie einen Folder Node unter dem Node `Modbus Device/Channels` hinzu, oder machen Sie einen Rechtsklick auf den `Modbus Device/Channels`-Node und wählen Sie `Add Channel` aus.
2. Tragen Sie im `Add Channel`-Dialog die Settings für die Modbus TCP Verbindung ein.
3. Nachdem Sie „Save“ geklickt haben, wird die Channel-Node erstellt.
4. Sie können den Kanal starten, indem Sie die Channel-Node auswählen und den Startbutton klicken.

Variablen

Unter dem `Variables`-Node können Sie Datenpunktnodes erstellen, die über Modbus gelesen und geschrieben werden.
Die `Value Type`-Eigenschaft muss dabei auf den entsprechenden Variablentyp festgelegt werden (`Boolean`, `Byte`, `Int16`, `UInt16`, `Int32`, `UInt32` und die zugehörigen Array-Typen).

Path

Über die `Path`-Eigenschaft des Nodes wird die Modbus-Adresse, der Datentyp im Modbus Server und die Länge der Daten (bei Arrays) definiert:

```
<UnitID>.<Entity><StartAddress>.<Offset>,<Length>,<DataType>
```

Platzhalter	Beschreibung	Mögliche Werte (Bedeutung)	Optional (Standard Wert)
<UnitID>	Unit ID des Modbus Slaves	0 - 255	Ja (`0`)
<Entity>	Objekttyp im Modbus Slave	`c` - Coil `di` - Discrete Input `hr` - Holding Register `ir` - Input Register	Ja (`c`)

Platzhalter	Beschreibung	Mögliche Werte (Bedeutung)	Optional (Standard Wert)
<StartAddress>	Startadresse der Daten	0 - 9999 Die führende Ziffer einer Modbusadresse definiert den Objekttypen, auf den zugegriffen wird. Dieser wird durch `<Entity>` festgelegt, während die führende Ziffer bei der Startadresse weggelassen wird.	Nein
<Offset>	Bei Zugriff auf Bits in Registern: Bit-Offset im 16-Bit Register Offset in der Einheit des definierten Datentypens (wenn `<DataType>` != `bit`)	0 - 15 (bei Zugriff auf Bits) Abhängig von den gültigen Adressen im Modbus Server	Ja (`0`)

<DataType>	Datentyp im Modbus Slave	`bit` = `bool`
		`byte`
		`word` = `uint16`
		`dword` = `uint32`
			Ja
			(`bit` wenn <DataArea> == `c` oder `di` bzw. ein <Offset> angegeben ist
			`word` wenn <DataArea> == `hr` oder `ir`)

Zuordnung von Modbus Function codes

Function code	Beschreibung	Entsprechung
0x01	Lese Coils	Lese Variabel des Typs `Boolean` oder `Boolean-Array` mit Objekttyp `Coil`
0x02	Lese Discrete Inputs	Lese Variabel des Typs `Boolean` oder `Boolean-Array` mit Objekttyp `Discrete Input`
0x03	Lese Holding Register	Lese Variabel des Typs `UInt16` oder `UInt16-Array` mit Objekttyp `Holding Register`
0x04	Lese Input Register	Lese Variabel des Typs `UInt16` oder `UInt16-Array` mit Objekttyp `Input Register`
0x05	Schreibe einzelne Coil	Schreibe Variabel des Typs `Boolean` mit Objekttyp `Coil`
0x06	Schreibe einzelnes Holding Register	Schreibe Variabel des Typs `UInt16` mit Objekttyp `Holding Register`
0x0F	Schreibe mehrere Coils	Schreibe Variabel des Typs `Boolean-Array` mit Objekttyp `Coil`
0x10	Schreibe mehrere Holding Register	Schreibe Variabel des Typs `UInt16-Array` mit Objekttyp `Holding Register`

Beispiele

Value Type	Path	Erklärung
`Boolean`	`16` oder `0.c16,1,bit`	Bit an Coil Adresse 16 in Unit 0
`Boolean`	`1.di0` oder `1.di0,1,bit`	Bit an Discrete Input Adresse 0 in Unit 1
`UInt16`	`1.hr20` oder `1.hr20,1,word`	16-Bit Wert an Holding Register Adresse 20 in Unit 1
`UInt16`	`hr20,4` oder `0.hr20,4,word`	Array aus 4 16-Bit Werten an Holding Register Adresse 20 in Unit 0
`UInt32`	`ir24,uint32` oder `0.ir24,1,dword`	32-Bit Wert an Input Register Adresse 24 in Unit 0
`Bool`	`ir24.1` oder `0.ir24.1,1,bit`	Bit an Bitoffset 1 an Input Register Adresse 24 in Unit 0

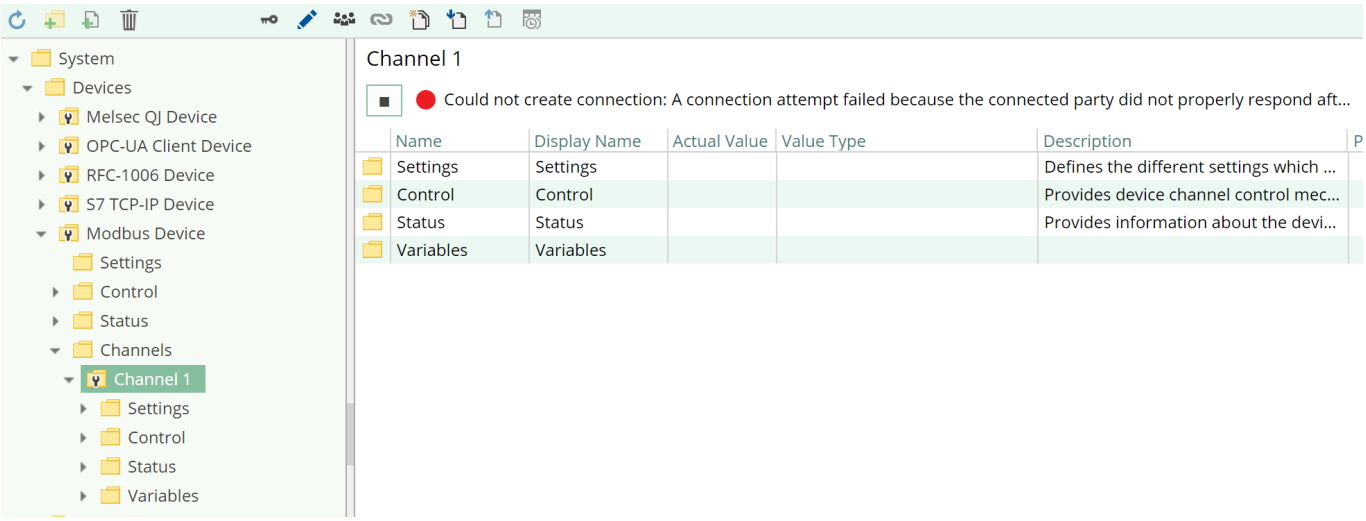
Name	Display Name	Actual Value	Value Type	Description	Path	Status
DO0	DO0	True	Boolean		16	Good
DO1	DO1	True	Boolean		c17	Good
DO2	DO2	True	Boolean		0.c18	Good
DO3	DO3	False	Boolean		0.c19,bool	Good
AI0	AI0	4095	UInt16		ir0	Good
AI1	AI1	4095	UInt16		0.ir1	Good
AI2	AI2	4095	UInt16		0.ir2,word	Good
AI3	AI3	4095	UInt16		0.ir3,uint16	Good
AO0	AO0	123999	UInt32		hr8,dword	Good
AO1	AO1	123039	UInt32		0.hr10,dword	Good
AO2 + AO3	AO2 + AO3	[1040596, 1203]	UInt32-Array		0.hr12,2,dword	Good

Fehlerdiagnose

Das Modbus Device Plugin liefert je nach zu untersuchender Schicht verschiedene Statusinformationen. Generell werden die kanalbasierten Diagnoseinformationen durch den Verbindungsstatus des Channels zum Modbus produziert. Die variablenbasierten Diagnoseinformationen werden während des Lese-/Schreibzugriffs auf die verschiedenen Variablen produziert.

Kanal

Um den Status des Modbus-Kanals zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf das folgende Bild:



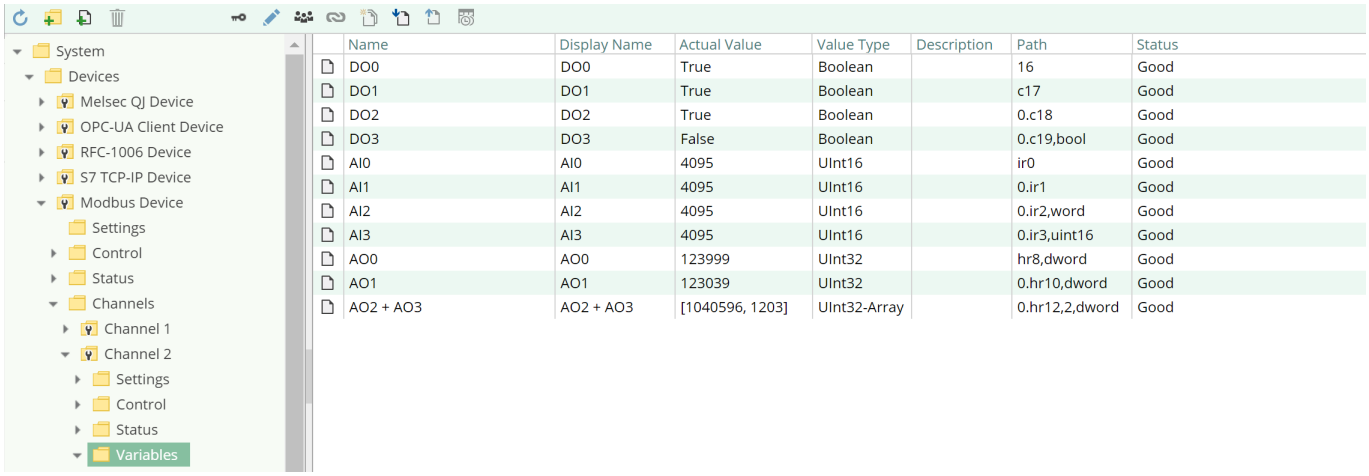
Das obige Bild zeigt das Bedienfeld des Modbus-Kanals, das alle statusrelevanten Informationen anzeigt. Das Bedienfeld aktualisiert automatisch seine Statusinformation, wenn ein neuer Status verfügbar ist.

Statuskreis

Farbe	Bedeutung
⚪	Der Kanal ist gestoppt. Klicken Sie den ▶-Button, um ihn zu starten.
🟡	Der Kanal startet oder stoppt gerade oder wartet auf den Verbindungsaufbau.
🟢	Der Kanal läuft und es wurde erfolgreich eine Verbindung hergestellt. Sie ihn können durch Klick auf den ■-Button stoppen.
🔴	Der Kanal läuft, aber die Verbindung ist momentan fehlerhaft. Bitte überprüfen Sie den Statustext für weitere Informationen.

Variablen

Um den Status der verschiedenen Variablen zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf die in UKI-4.0 ® angezeigte `Status`-Eigenschaft der Spalte. Benutzen Sie den Button „Read actual Value“, um die Werte über Modbus auszulesen und das Ergebnis in den Variablen zu speichern.



Name	Display Name	Actual Value	Value Type	Description	Path	Status
DO0	DO0	True	Boolean		16	Good
DO1	DO1	True	Boolean		c17	Good
DO2	DO2	True	Boolean		0.c18	Good
DO3	DO3	False	Boolean		0.c19,bool	Good
AI0	AI0	4095	UInt16		ir0	Good
AI1	AI1	4095	UInt16		0.ir1	Good
AI2	AI2	4095	UInt16		0.ir2,word	Good
AI3	AI3	4095	UInt16		0.ir3,uint16	Good
AO0	AO0	123999	UInt32		hr8,dword	Good
AO1	AO1	123039	UInt32		0.hr10,dword	Good
AO2 + AO3	AO2 + AO3	[1040596, 1203]	UInt32-Array		0.hr12,2,dword	Good

Logdatei

Alle kanalbezogenen Statusinformationen werden auch in die kanalspezifische Logdatei im `[LoggingFolder]` protokolliert. Jede Logdatei wird nach dem Namensschema `Modbus Device.<ChannelName>.log` benannt.

Der Inhalt einer solchen Logdatei kann wie folgt aussehen:

```
...
2018-04-11 11:32:37.0 +2: [Error] Error (Severity=High): Code=[-1], Text=[The operation has
timed-out.], Details=[]
...
```

Entities

Wie jedes Device Plugin erweitert das Modbus Device Plugin das UKI-4.0 Device Modell.

Device

Der Device Typ `ModbusDevice` des Plugins definiert auch den `ModbusDeviceChannel` und erweitert somit die grundlegenden UKI-4.0 `Device` und UKI-4.0 `DeviceChannel` Entities. Während das `ModbusDevice` nur eine Konkretisierung des UKI-4.0 `Device` darstellt, erweitert der `ModbusDeviceChannel` den UKI-4.0 `DeviceChannel` mit den Modbus Variable Entities.

Channel

Der Kanal wird von einem Channel Worker behandelt, der eine TCP Socket Verbindung zum lokalen Server herstellt. Zur Fehlerdiagnosezwecken überprüft der Worker automatisch alle 10 Sekunden den Status der TCP Socket Verbindung, um den `Channel` Statuscode und seine Beschreibung zu aktualisieren und Verbindungsfehler zu erkennen.

Der Worker liest standardmäßig keine Werte. Wenn ein Anwender oder Plugin in UKI-4.0 ® einen synchronen Lesevorgang der `Channel` Variablen anfordert (z.B. mit der „Read actual value“-Funktion in der UKI-4.0 ® Webkonfiguration), liest der Channel Worker diese vom Modbus Server und schreibt diese in die entsprechenden UKI-4.0 Nodes.

Ähnlich schreibt der Channel Worker auch die Werte in den Modbus Server, wenn ein Client oder Plugin Werte in die `Channel` Variablen schreibt.

Damit eine Modbus Variable regelmäßig gelesen wird, können Sie in der Webkonfiguration bei dem Node „History Options“ auf `Yes` stellen (was eine interne Subscription erstellt), oder Sie können zum Beispiel einen OPC UA Client verwenden, der mit dem OPC UA Serverplugin verbunden ist und damit eine Subscription für die Modbus Variablennodes erstellen. In diesen Fällen liest der Channel Worker in regelmäßigen Intervallen die Variablen vom Modbus und schreibt den neuen Wert nach einer Wertänderung automatisch in die entsprechende UKI-4.0 ® Node.

Variable

Der Modbus-Datentyp bestimmt, auf welche Modbus-Speicherbereich zugegriffen wird. Dabei werden die Variablenformate Skalar und Array unterstützt.

Ordner & Dateien

Ordner

Name	Pfad	Zweck / Verwendung
AssemblyFolder	`<`UKI-4.0 `InstallDir>/plugins/ModbusDevicePlugin/`	Beinhaltet die Plugin Assembly Datei.
ConfigFolder	`<`UKI-4.0 `DataDir>/plugins/ModbusDevicePlugin/`	Beinhaltet die Plugin Konfigurationsdatei.
LoggingFolder	`<`UKI-4.0 `DataDir>/log/`	Beinhaltet die Plugin Log Dateien.

Dateien

Typ	Pfad	Zweck / Verwendung
Assembly	`[AssemblyFolder]/`UKI-4.0 `.ModbusDevicePlugin.dll`	Die Plugin Assembly Datei.
Logging	`[LoggingFolder]/Modbus Device.<ChannelName>.log`	Die Log Datei.

Versionsinformation

Dieses Dokument

Datum	`2018-11-23`
Version	`1.0`

Plugin

Name	`Modbus Device Plugin`
Node	`/System/Devices/Modbus Device`
Version	`1.0.0`

Assembly

Name	UKI-4.0 `.ModbusDevicePlugin.dll`
Datum	`2018-11-23`
Version	`1.0.0.0`

OPC UA Client Device Plugin

Allgemein

Das OPC UA Client Device Plugin bietet eine Verbindung zwischen UKI-4.0 und einem OPC UA Server.

Was tut das Plugin?

Das OPC UA Client Device Plugin baut Verbindungen zu einem oder mehreren OPC Servers auf und hält diese aufrecht. Jeder OPC UA Client kann über einen separaten Kanal verbunden werden.

Funktionen

- Browsen von OPC UA Servern/OPC Classic Servern, um automatisch verfügbare Variablen in UKI-4.0 zu erstellen
- Lesen und Schreiben von OPC UA/OPC Classic Data Variables
- Abonnieren von OPC UA Data Variablen/OPC Classic Variablen, wenn die entsprechenden Nodes in UKI-4.0 abonniert werden

Unterstützte Server

- OPC UA Server, die das `opc.tcp`- oder `http`-Protokoll verwenden
- OPC Classic (COM) Server (spezifiziert durch die ProgID und ClassID) (nur unter Windows x86 und x64)

Zweck & Anwendung

Die verbundenen OPC UA Server können durch die Benutzung von UKI-4.0 einfach gesteuert werden. Indem die Nodes des OPC UA Servers mit den in UKI-4.0 definierten Nodes verbunden werden, kann der OPC Node direct mit vielen anderen Nodes, Geräten, Services etc. interagieren, die in UKI-4.0 gewartet werden.
Auch andere UKI-4.0 -Teilnehmer können mit dem durch das OPC UA Device Plugin verbundenen OPC UA Server interagieren.

Installation

Dieses Plugin ist Bestandteil des UKI-4.0 Setups. Bitte konsultieren Sie UKI-4.0 Setup und erster Start für weitere Informationen darüber, wie dieses Plugin installiert und deinstalliert werden kann.

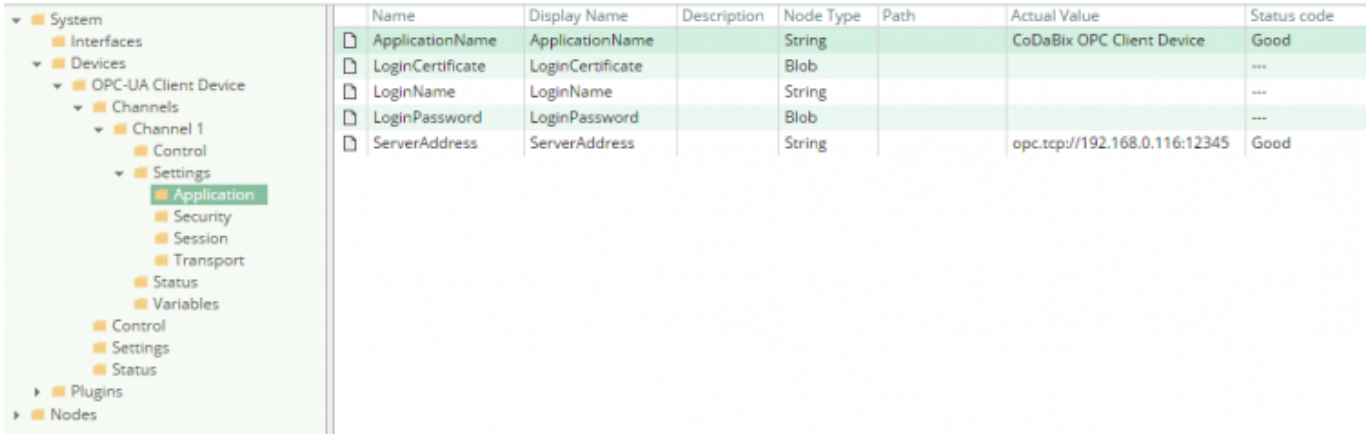
Anforderungen

- Standardanforderungen von UKI-4.0
- Aktivierte ausgehende Verbindung via TCP/IP über den spezifizierten Port

Konfiguration

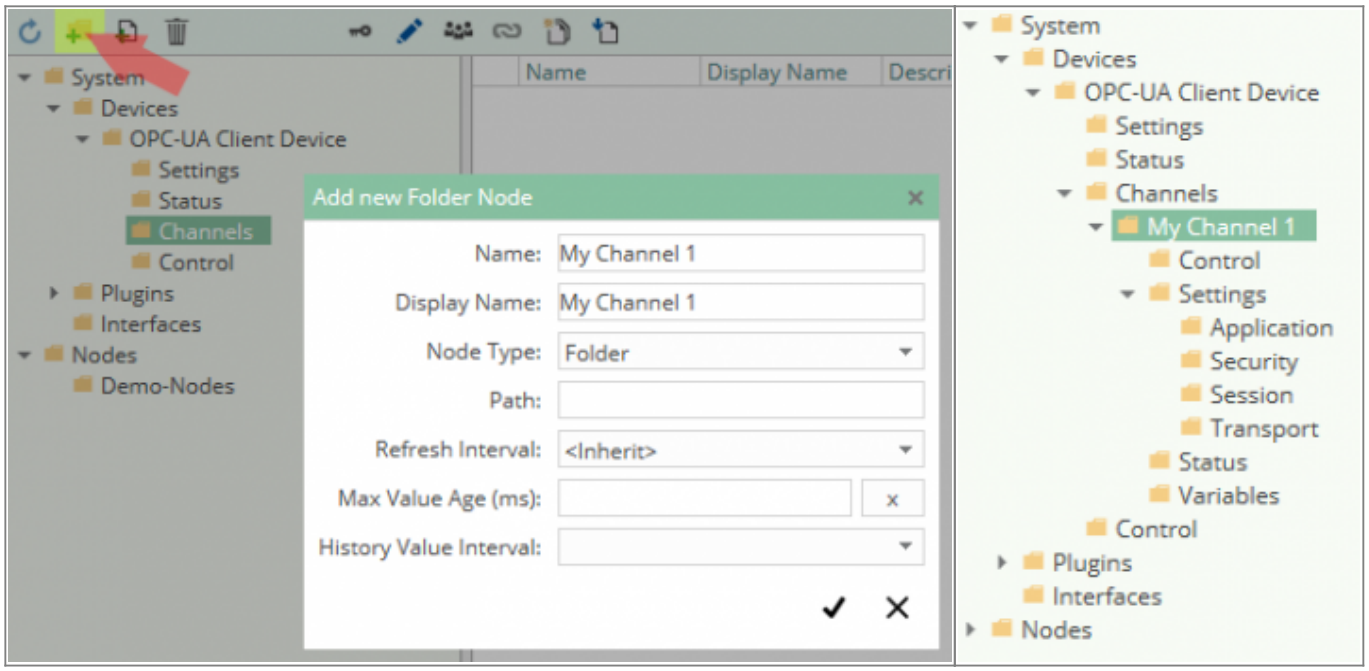
Überblick

Die gesamte OPC UA Client Device Pluginkonfiguration finden Sie unter dem Nodepfad `/System/Devices/OPC UA Client Device`. Diese Wurzelnode des Device Plugins ermöglicht die vollständige Konfigutarion des OPC UA Client Device Plugins.



Benutzung

Der Nodebaum im oberen Bild zeigt den Standardnodebaum des OPC UA Client Device Plugin. Um einen oder mehrere OPC UA Client Device Kanäle zu einzurichten, fügen Sie einen Folder Node unter dem Node `OPC UA Client Device/Channels` hinzu (linkes Bild). Danach erscheint der Standardnodebaumfür einen Kanal (rechts Bild).



Nun können die Einstellungen für den spezifizierten Kanal verändert werden. Auch können neue Variablen unter dem Order `OPC UA Client Device/Channels/<Channel>/Variables` erstellt werden. Die Verbindung zum OPC Node wird durch das Benutzen der `Path` Eigenschaft des neuen Variablennodes genehmigt.

Die `Path` Eigenschaft muss eine gültig formatierte OPC NodeId sein, zum Beispiel `2:Main-PLC/Office 1 - Lights/Front`.

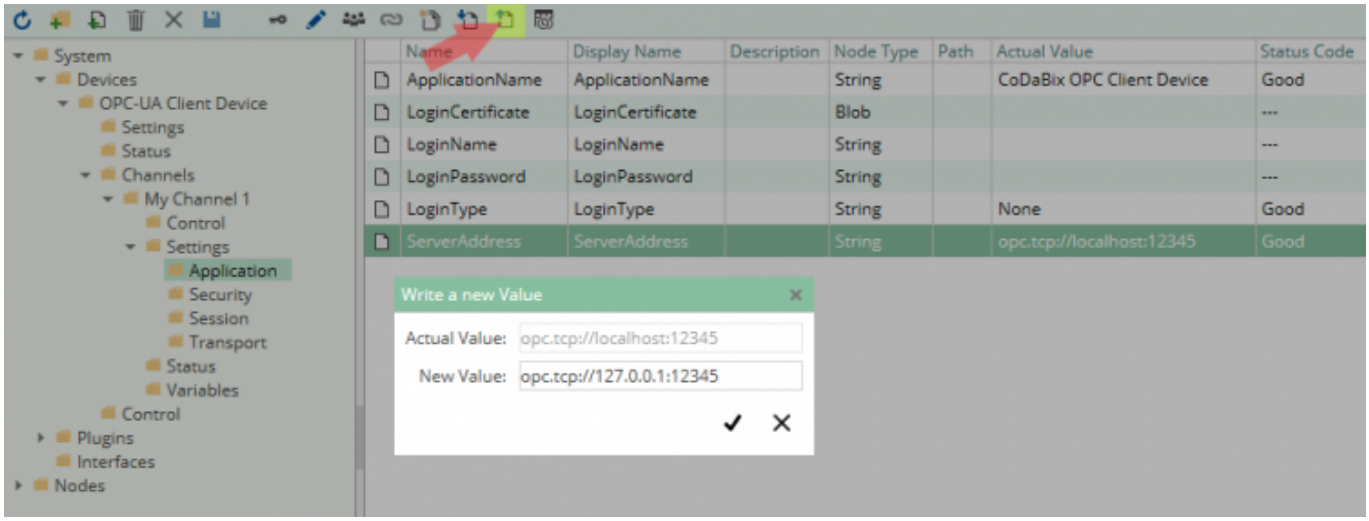
Jede Änderung im `Settings` Ordner wird angewandt, sobald der Kanal neu gestartet wird.

Jede Änderung im `Variables` Ordner führt automatisch eine Rekonfiguration des OPC Clients im spezifizierten Kanal durch.

Einstellungen

Einstellungen ändern

- Wählen Sie die zu ändernde `Settings` Eigenschaft aus. (z.B. „Application/ServerAddress“)
- Klicken Sie auf „Write a new Value“. (siehe Beispiel im unteren Screenshot)
- Geben Sie den neuen Wert ins Eingabefeld ein und speichern Sie die Änderungen.



Übersicht

Name	Typ	Standardwert	Beschreibung
Application			
Application Name	String	UKI-	Mit diesem Namen stellt sich der OPC UA Client dem OPC UA Server vor.
Server Address	String	opc.tcp://localhost:12345	Die URL zum OPC UA Server. Mögliche Protokolle sind: `opc.tcp://`, `http://`, `https://`, `opc.com://`. Um unter Windows (x86 oder x64) auf OPC Classic (COM)-Server zuzugreifen, können Sie das Format `opc.com://<hostname>(:<port>)/<progId>/<classId>` verwenden. Die optionale Portnummer wird von dem intern erstellten Wrapperserver verwendet. Falls die Portnummer nicht angegeben ist, wird sie in einem Bereich zwischen 48000 und 48999 aus dem `<classId>`-Wert generiert. Beispiel für OPC Classic: `opc.com://localhost:4711/OPCManager.DA.XML-DA.Server.DA/{E4EBF7FA-CCAC-4125-A611-EAC4981C00EA}` **Bitte beachten Sie:** Einige OPC Classic (COM)-Server unterstützen nur **x86- (32-Bit)-Clients**. Daher empfehlen wir, die **x86**-Version statt der **x64**-Version von UKI- zu installieren, wenn Sie auf solche Server zugreifen möchten.
Login Type	Enum	None	Definiert, welche Authentifizierungsart verwendet wird. Gültige Werte: `Anonymous`, `Certificate`, `UserPassword`
Login Certificate	Blob		Das Zertifikat des OPC UA Servers, das für die Authentifizierung benutzt wird. Kann als Zertifikatsdatei hochgeladen werden. (Muss den Zertifikatsschlüssel enthalten.)
Login Name	String		Der Benutzer des OPC UA Servers, der für die Authentifizierung benutzt wird.
Login Password	String		Das Passwort des Benutzers des OPC UA Servers.
Session			
Disconnect Timeout	Int32	10000	t.b.a.
Reconnect Timeout	Int32	10000	t.b.a.
Session Timeout	Int32	60000	t.b.a.
Use Break Detection	Boolean	True	Stellt automatisch fest, wenn die Verbindung zum OPC UA Server unterbrochen wird. Dies wird für die automatische Wiederverbindung benutzt, falls die Verbindung zum Server abgebrochen ist.. Gültige Werte: `True`, `False`
Security			
Policy Algorithm	String	Auto	Gültige Werte: `Auto`, `Basic128Rsa15`, `Basic256`, `Basic256Sha256`, `Custom`, `Https`, `None`
Policy Level	Int32	0	t.b.a.
Policy Mode	String	None	Gültige Werte: `None`, `Sign`, `SignAndEncrypt`

Name	Typ	Standardwert	Beschreibung
Use Domain Checks	Boolean	False	Zeigt an, ob der OPC UA Client den OPC UA Server auf ein vertrauenswürdiges Zertifikat prüft oder nicht. Dies ist eine Sicherheitsfunktion, um z.b. Man-in-the-middle Attacken zu verhindern. Gültige Werte: `True`, `False` **Achtung** Ist diese Option auf `true` eingestellt und der OPC UA Server hat kein vertrauenswürdiges X.509-Zertifikat (z.B. ein selbstsigniertes Zertifikat), wird die Verbindung abgelehnt.
Use Secure Endpoint	Boolean	True	Zeigt an, ob der OPC UA Client die Sicherheit des Endpunkts überprüfen soll oder nicht. Manche OPC UA Server könnten diese Option nicht unterstützen. Gültige Werte: `True`, `False`
Certificate	Blob	<Blob>	Das Client-Zertifikat. Mit diesem Zertifikat stellt sich der OPC UA Client dem OPC UA Server vor. Standardmäßig wird ein neues Zertifikat erstellt.
Transport			
Channel Lifetime	Int32	600000	t.b.a.
Max Array Length	Int32	65535	t.b.a.
Max Buffer Size	Int32	65535	t.b.a.
Max Byte String Length	Int32	65535	t.b.a.
Max Message Size	Int32	1048576	t.b.a.
Max String Length	Int32	65535	t.b.a.
Operation Timeout	Int32	60000	t.b.a.
Security Token Lifetime	Int32	3600000	t.b.a.

Variablen

Jeder Node unter `System/Devices/OPC UA Client Device/Channels/My Channel 1/Variables` kann vom OPC Server des spezifizierten Kanals über die `Path` Eigenschaft mit einem OPC Node verbunden werden.

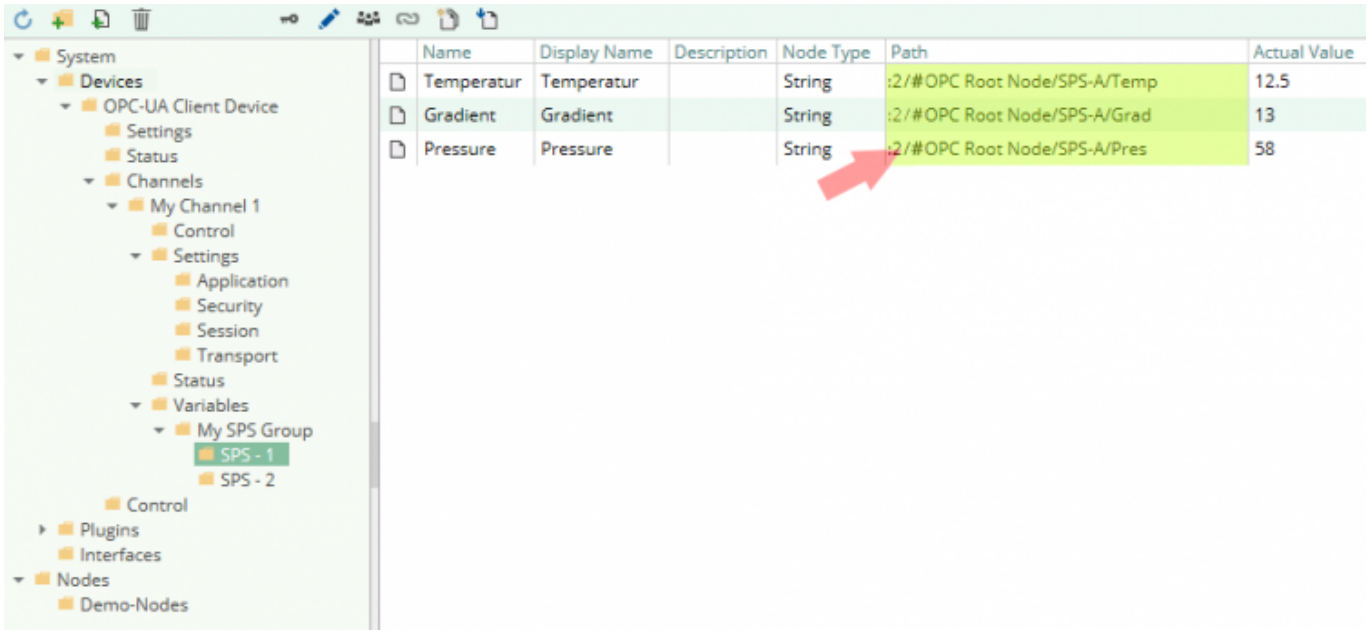
Um einen UKI-4.0 -Node mit einem OPC UA Node zu verbinden, muss die `Path` Eigenschaft als eine OPC UA NodeID formatiert sein.

OPC NodeId

- Beispiel NodeId: `ns=2;s=Machine_1/IsActive`

Die spezifizierte NodeId ist in zwei Bestandteile unterteilt: Der Index des Namensraums und der Identifier der OPC UA Node. Das Schema eines String-Identifiers hängt vom benutzten OPC UA Server ab.

- Namespace Index: `ns=<Number>` z.B. `ns=2`
- Trennzeichen: `;`
- Identifier (String): `s=<ParentNode1>/<ParentNode2>/…/<ParentNodeX>/<NodeName>`

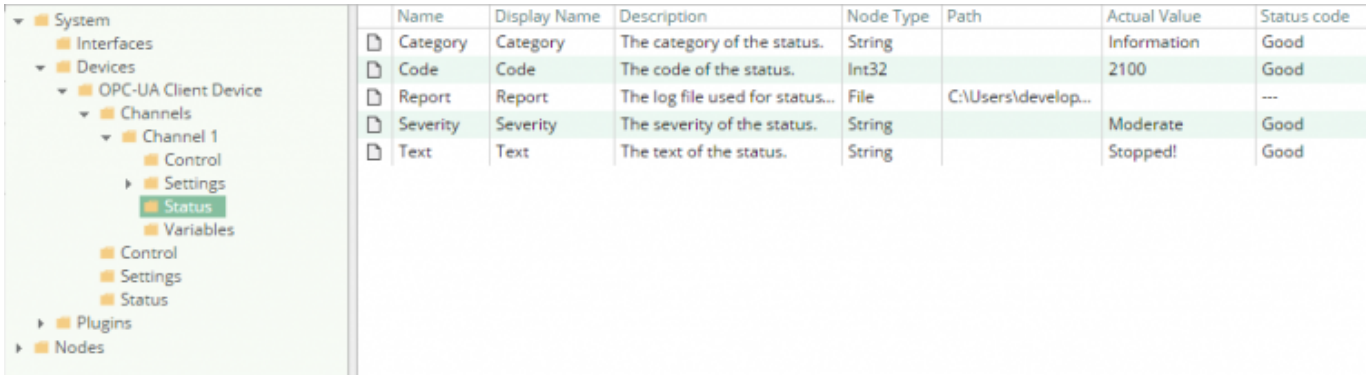


Fehlerdiagnose

Das OPC UA Client Device Plugin stellt je nach zu untersuchender Schicht verschiedene Statusinformationen bereit. Generell wird die kanalbasierte Diagnoseinformation vom Verbindungsstatus des Kanals zum OPC UA Server produziert. Die variablenbasierte Diagnoseinformation wird während des Lese- / Schreibzugriffs der verschiedenen Variablen oder direkt vom OPC UA Server produziert.

Channel

Um den Status von verschiedenen Gerätekanälen zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf folgendes Bild::



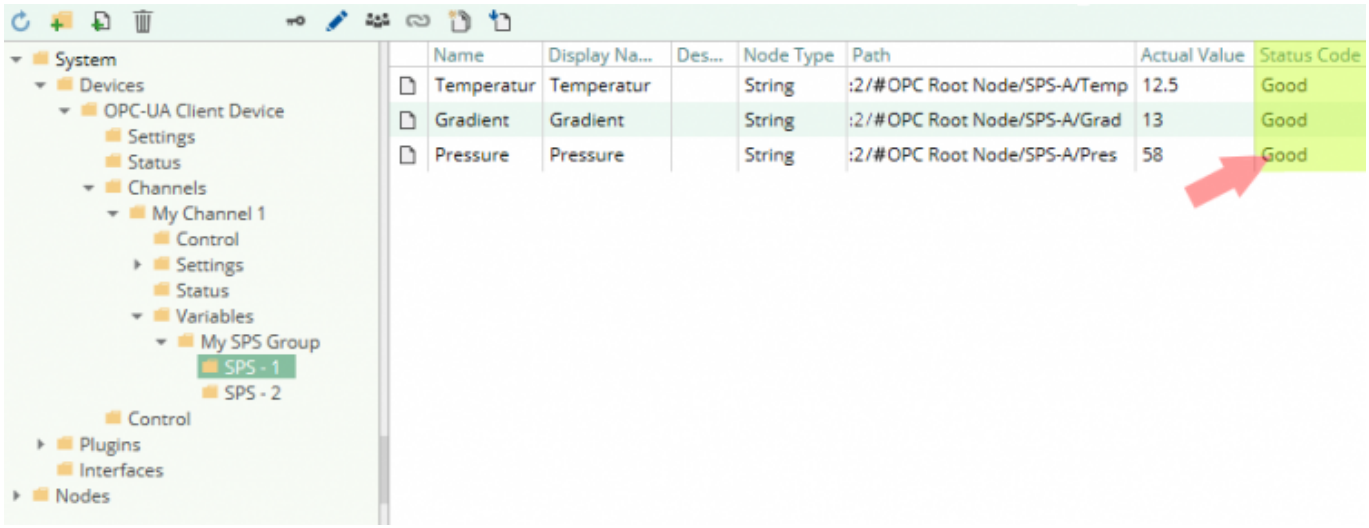
Das obige Bild zeigt den `Status` Node des Kanals, der alle statusrelevanten Informationen abbildet. Die folgenden Datenpunktnodes werden benutzt, um den Kommunikationsstatus zwischen UKI-4.0 und dem OPC UA Server zu erhalten.

Node	Beschreibung
`Category`	Unterteilt den Status in `Information`, `Warning` und `Error` und zeigt somit die generelle Art der Statusinformation an.
`Code`	Definiert den numerischen Ausdruck / Identifier des Status.
`Severity`	Stuft die Statusinfrmation ein in `Low`, `Moderate`, `High` und `Critical` und zeigt somit die Dringlichkeit eines Einschreitens an.
`Text`	Beschreibt die durch die `Code` Eigenschaft identifizierte Statusinformation.

Variablen

Um den Status der verschiedenen OPC UA Client Variablen zu überwachen und zu diagnostizieren, werfen

Sie einen Blick auf die `Status`-Eigenschaft der in UKI-4.0 angezeigten Variable.
Falls die `Status`-Spalte den Wert `Bad` anzeigt, ist in den meisten Fällen der adressierte Datenbereich nicht erreichbar.



Logdatei

Alle Device Channel-relevanten Statusinformationen werden auch in die kanalspezifische Logdatei protokolliert, die sich im `[LoggingFolder]` befindet. Jede Logdatei wird nach dem folgenden Namensschema benannt: `OPC UA Client Device.<ChannelName>.log`. Der Inhalt einer solchen Logdatei kann wie folgt aussehen:

```
...
[15:31:46 05.09.2016] - Information (Severity=Moderate): Code=[10012], Text=[Creating Client
| opc.tcp://192.168.0.116:12345/]
...
```

In Verwendung des Beispielkanals wäre der Name der Logdatei: `OPC UA Client Device.Channel 1.log`

Status Codes

Die folgende Tabelle zeigt die verschiendenen möglichen Statusinformationen:

Code	Kategori	Informationsart
-22000 to -22999	Error	Fehler mit Ausnahme
-21000 to -21999	Error	Interner Fehler
-12000 to -12999	Warning	Warnung mit Ausnahme
-11000 to -11999	Warning	Interne Warnung
10000 to 10999	Information	Information
20000 to 20999	Information	Debug

Entities

Wie jedes Device Plugin erweitert das OPC UA Client Device Plugin das grundlegende UKI-4.0 Device Modell.

Device

Der Devicetyp `OpcClientDevice` des Plugins definiert auch den `OpcClientDeviceChannel` und erweitert somit die grundlegenden UKI-4.0 `Device` und UKI-4.0 `DeviceChannel` Entities. Während das `OpcClientDevice` nur eine Konkretisierung des UKI-4.0 `Device` repräsentiert, erweiter der `OpcClientDeviceChannel` den UKI-4.0 `DeviceChannel` mit den OPC UA Client Variable Entities.

Channel

Jeder Kanal wird von einem Channel Worker behandelt, der eine physische Verbindung zum OPC UA Server herstellt.

Standardmäßig liest der Worker keine Werte. Wenn ein Client oder Plugin ein synchrones Lesen der Variablen des Kanals in UKI-4.0 anfordert (z.B. durch das Benutzen der Funktion „Read actual value“ der UKI-4.0 Webkonfiguration), liest der Channel Worker diese aus dem zugrundeliegenden OPC UA Server und schreibt sie dann in die entsprechenden UKI-4.0 Nodes.

Ähnlich schreibt der Channel Worker die Werte in den zugrundeliegenden OPC UA Server, wenn ein Client oder Plugin Werte in die Variablen des Kanals schreibt.

Um eine OPC UA Client Variable regelmäig gelesen zu bekommen, können Sie den Node in der Webkonfiguration bearbeiten und „History Options“ auf `Yes` setzen (was intern eine Subscription erstellt). In diesem Fall abboniert der Channel Worker die Variablen vom OPC UA Server (der OPC Client bekommt bei einer Wertänderung den neuen Wert automatisch fom OPC Server) und wenn eine dr Variablen sich verändert hat, schreibt er den neuen Wert in den entsprechenden UKI-4.0 Node.

Ordner & Dateien

Ordner

Name	Pfad	Zweck / Verwendung
AssemblyFolder	<UKI-4.0 `InstallDir>/plugins/OpcUaClientDevicePlugin/`	Beinhaltet die Plugin Assembly Datei.
ConfigFolder	<UKI-4.0 `DataDir>/plugins/OpcUaClientDevicePlugin/`	Beinhaltet die Plugin Konfigurationsdatei.
LoggingFolder	<UKI-4.0 `DataDir>/log/`	Beinhaltet die Plugin Log Dateien.

Dateien

Typ	Pfad	Zweck / Verwendung
Assembly	`[AssemblyFolder]/`UKI-4.0 `.OpcUaClientDevicePlugin.dll`	Die Plugin Assembly Datei.
Logging	`[LoggingFolder]/OPC UA Client Device.<ChannelName>.log`	Die Log Datei.

Versionsinformation

Dieses Dokument

Datum	`2021-01-25`
Version	`1.3`

Plugin

Name	`OPC UA Client Device Plugin`
Node	`/System/Devices/OPC UA Client Device`
Version	`1.10.0`

Assembly

Name	UKI-4.0 `.OpcUaClientDevicePlugin.dll`
Datum	`2021-01-25`
Version	`1.10.0.0`

S7 Device Plugin

Das S7 Device Plugin ermöglicht das Lesen und Schreiben von Daten von physischen SIMATIC S7-Geräten über TCP/IP.

Folgende Gerätetypen werden unterstützt:

- S7-1500 (siehe SPS-Einstellungen)
- S7-1200 (siehe SPS-Einstellungen)
- S7-300
- S7-400
- WinAC
- S7-SoftPLC
- LOGO! (siehe SPS-Einstellungen)
- S7-200
- SIMATIC S5 über S5-LAN
- S7-LAN
- VIPA-S7 und jede andere S7-TCP/IP kompatible SPS

Funktionen

- Optimierte Lese- und Schreibzugriffe durch bestmögliche Ausnutzung der Paketgröße.
- Automatisiertes Verbindungshandling, auch auto-reconnect.
- Zugriff auf den SPS-Speicher mit benutzerdefinierten Typen.
- Nutzung der vorhandenen SPS-Projekte, um Kanäle und Variablen einzurichten. Folgende Projektformate werden unterstützt:
  - STEP7 Projekt Dateien (*.s7p)
  - IP-S7 Projekt Dateien (*.ips7)
  - S7 Watch Projekt Dateien (*.wproj)
  - CSV-to-S7 Projekt Dateien (*.ini)

Zweck & Anwendung

Zweck

Die angeschlossenen Geräte können einfach mit UKI-4.0 ® gesteuert werden. Durch die Verknüpfung der SPS-Speicher mit den in UKI-4.0 ® definierten Nodes kann die SPS direkt mit vielen anderen Nodes, Geräten, Diensten usw., die in UKI-4.0 ® gewartet werden, interagieren. Auch andere UKI-4.0 ® Teilnehmer können mit den durch das S7 Device Plugin bereitgestellten SPS-Geräten interagieren.

Anwendung

- Dynamische Generierung von Fertigungsdaten basierend auf verschiedenen Bedingungen und Daten, die von Maschinen, Benutzern, Aufträgen, Zuständen, Diensten usw. generiert werden
- Zusätzliche Sicherheit durch die Überwachung der gesamten Anlage, einschließlich der betriebsinternen Interaktionen
- Zentrale Steuerung und Durchflussüberwachung zur Früherkennung von möglichen Störungen
- Verbesserung des Anlagenleitstands, indem die von den Geräten erzeugten Produktionsdaten überwacht und aufgezeichnet werden

Installation

Dieses Plugin ist Bestandteil des UKI-4.0 ® Setups. Bitte konsultieren Sie UKI-4.0 ® Setup und erster Start für weitere Informationen darüber, wie dieses Plugin installiert und deinstalliert werden kann.

Anforderungen

- Standardanforderungen von UKI-4.0
- aktivierte ausgehende Verbindung via TCP-/IP-Verbindung über Port 102

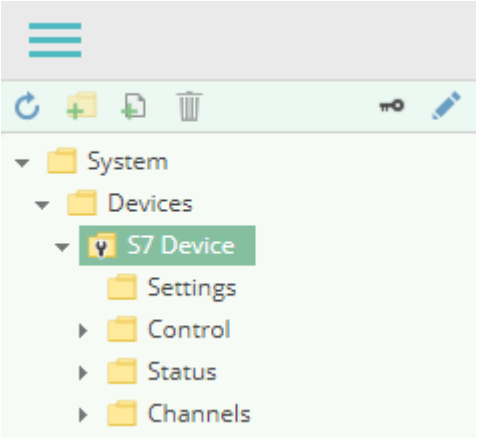
SPS-Einstellungen

Bitte beachten Sie, dass für den Zugriff auf die S7-1200, S7-1500 sowie LOGO! bei den Datenbausteinattributen der **optimierte Datenbausteinzugriff deaktiviert** sein muss.
Eine Anleitung hierzu finden Sie unter: Optimierten Datenbausteinzugriff deaktivieren.

Konfiguration

UKI-4.0UKI-4.0 UKI-4.0UKI-4.0 verwenden

Die gesamte S7 Device Plugin-Konfiguration befindet sich unter dem Nodepfad `/System/Devices/S7 Device`.



Channel

Ein S7 Device Channel repräsentiert die Verbindung zu einer S7-SPS.

Settings

Address
Die IP-Adresse oder der Hostname der S7-SPS.

Rack

Dir Rack-Nummer der SPS (wird ab S7-1200 ignoriert).

Slot

Die Slot-Nummer der SPS (wird ab S7-1200 ignoriert).

Device Type

Der Gerätetyp der SIMATIC S7-SPS. Folgende Gerätetypen werden unterstützt:

- LOGO!
- S7-200
- S7-300
- S7-400
- S7-1200
- S7-1500

Channel Type

Der Typ des Kanals, der benutzt wird, um mit der SPS zu kommunizieren.

DateTime Interpretation

Gibt an, wie die Zeitzone von gelesenen und geschriebenen DateTime-Werten interpretiert werden soll (werden die Datums-/Zeitwerte in der SPS als UTC-Zeit oder als lokale Zeit abgelegt).

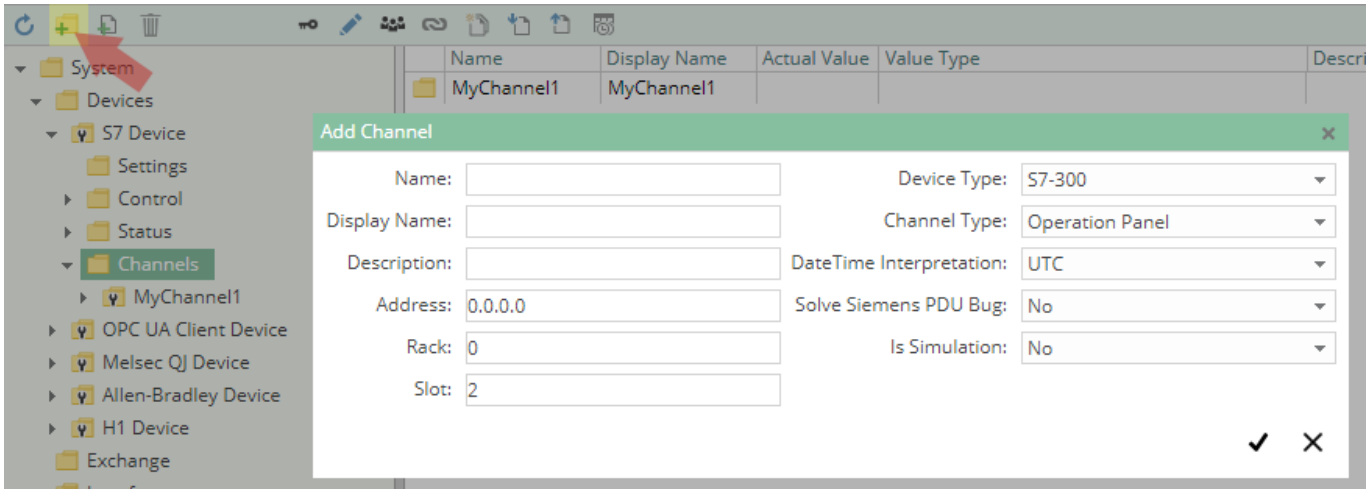
Solve Siemens PDU Bug

Legt fest, ob eine reduzierte PDU-Size verwendet werden soll, um ein Problem in einigen S7-400-SPSen zu umgehen, welche sonst in zufälliger Weise große Datenpakete abschneiden könnten, was sich in Fehlern wie „specified data area doesn't exist“ äußern würde.

Is Simulation

Legt fest, ob dieses Device im Simulationsmodus läuft, d.h. dass keine Verbindung zu einer physischen SPS hergestellt wird.

Hinzufügen eines Channels



Um einen neuen S7-Channel zu erstellen, gehen Sie wie folgt vor:

1. Fügen Sie einen Folder Node unter dem Node `S7 Device/Channels` hinzu, oder machen Sie einen Rechtsklick auf den `S7 Device/Channels`-Node und wählen Sie `Add Channel` aus.
2. Tragen Sie im `Add Channel`-Dialog die Settings für die S7-Verbindung ein.
3. Nachdem Sie „Save“ geklickt haben, wird die Channel-Node erstellt.

4. Sie können den Kanal starten, indem Sie die Channel-Node auswählen und den Startbutton klicken.

Variablen

Unter dem `Variables`-Node können Sie Datenpunktnodes erstellen, die aus der S7 gelesen und in diese geschrieben werden. Zusätzlich dazu können Sie Variablen auch z.B. aus einem **STEP7**-Projekt importieren.
Die `Value Type`-Eigenschaft muss dabei auf den zugehörigen Datentyp festgelegt werden. Aktuell werden folgenden Typen im S7 Device Plugin unterstützt:

S7 Device Type	**SPS-Typ**	**Bevorzugter** UKI-4.0 **Value Type**	**Beschreibung**
`Bool`	`BOOL`	`Boolean` or `Boolean-Array`	Eine Variable vom SPS-Typ BOOL.
`Byte`	`BYTE`	`Byte` or `Byte-Array`	Eine Variable vom SPS-Typ BYTE.
`Char`	`CHAR`	`String` or `String-Array`	Eine Variable vom SPS-Typ CHAR.
`Int`	`INT`	`Int16` or `Int16-Array`	Eine Variable vom SPS-Typ INT. Repräsentiert einen vorzeichenbehafteten 16-Bit Integer.
`Word`	`WORD`	`UInt16` or `UInt16-Array`	Eine Variable vom SPS-Typ WORD. Repräsentiert einen vorzeichenlosen 16-Bit Integer.
`DInt`	`DINT`	`Int32` or `Int32-Array`	Eine Variable vom SPS-Typ DINT. Repräsentiert einen vorzeichenbehafteten 32-Bit Integer.
`DWord`	`DWORD`	`UInt32` or `UInt32-Array`	Eine Variable vom SPS-Typ DWORD. Repräsentiert einen vorzeichenlosen 32-Bit Integer.
`LInt`	`LINT`	`Int64` or `Int64-Array`	Eine Variable vom SPS-Typ LINT. Repräsentiert einen vorzeichenbehafteten 64-Bit Integer.
`LWord`	`LWORD`	`UInt64` or `UInt64-Array`	Eine Variable vom SPS-Typ LWORD. Repräsentiert einen vorzeichenlosen 64-Bit Integer.
`Real`	`REAL`	`Single` or `Single-Array`	Eine Variable vom SPS-Typ REAL. Repräsentiert eine Gleitkommazahl mit einfacher Genauigkeit.
`Double`	`REAL`	`Double` or `Double-Array`	Eine Variable vom SPS-Typ REAL. Repräsentiert eine Gleitkommazahl mit doppelter Genauigkeit.
`LReal`	`LREAL`	`Double` or `Double-Array`	Eine Variable vom SPS-Typ LREAL. Repräsentiert eine Gleitkommazahl mit doppelter Genauigkeit.
`Date`	`DATE`	`DateTime` or `DateTime-Array`	Eine Variable vom SPS-Typ DATE.
`Time`	`TIME`	`TimeSpan` or `TimeSpan-Array`	Eine Variable vom SPS-Typ TIME.
`TimeOfDay`	`TIME_OF_DAY`	`TimeSpan` or `TimeSpan-Array`	Eine Variable vom SPS-Typ TIME_OF_DAY.
`S5Time`	`S5TIME`	`TimeSpan` or `TimeSpan-Array`	Eine Variable vom SPS-Typ S5TIME.
`DateTime`	`DATE_AND_TIME`	`DateTime` or `DateTime-Array`	Eine Variable vom SPS-Typ DATE_AND_TIME.
`String`	`STRING`	`String`	Eine Variable vom SPS-Typ STRING.
`S5String`	`BYTE`	`String`	Eine Variable vom SPS-Typ BYTE. Eine feste Anzahl von Bytes wird als String interpretiert.

Path

Über die `Path`-Eigenschaft des Nodes wird die Adresse, optional eine Typangabe, sowie (bei Arrays oder Strings) die Länge der Daten definiert:

```
<Address>
<Address>, <Length>
<Address>, <Type>
<Address>, <Type>[<Length>]
```

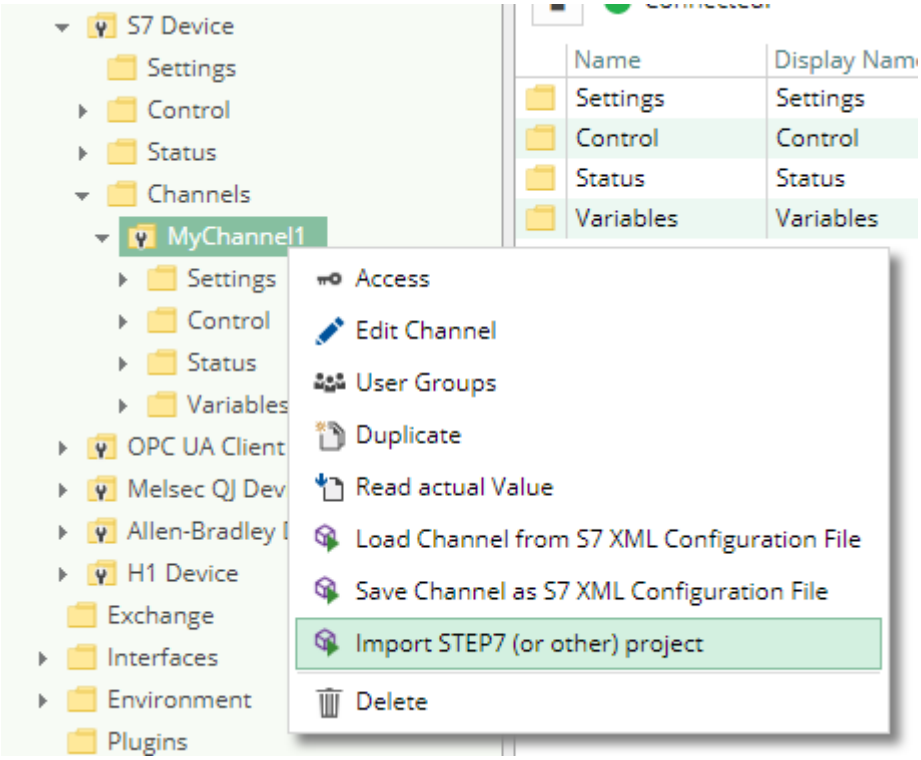
Platzhalter	Beschreibung
`<Address>`	Die Adresse der Daten. Beispiel: `DB10.DBW 16`
`<Length>`	Falls die Variable ein Array oder String ist, geben Sie hier die Länge des Arrays/String an. Diese wird nur beim Lesen verwendet; beim Schreiben wird die Länge aus dem zu schreibenden/String Array ermittelt.
`<Type>`	Falls angegeben, gibt den S7 Device Type an (siehe obige Tabelle), der den standardmäßig abgeleiteten Typen aus dem UKI-4.0 Value Type überschreibt. Beachten Sie: Beim Verwenden des UKI-4.0 Value Type `Double` wird daraus nicht automatisch ein S7 Device Type abgeleitet; Sie müssen explizit `Double` angeben.

Beispiele

UKI-4.0 **Value Type**	**Path**	**Erklärung**
`Boolean`	`DB10.DBX 3.2`	Einzelnes Bit (`BOOL`) bei Adresse `DB10.DBX 3.2`
`Boolean-Array`	`DB10.DBX 3.4, 18`	Bit-(`BOOL`)-Array von Adresse `DB10.DBX 3.4` bis `DB10.DBX 5.6` (exklusiv)
`Int32`	`DB10.DBD 12, DInt`	Einzelnes DInt (`DINT`) bei Adresse `DB10.DBW 12`
`Double-Array`	`DB10.DBD 20, Double[5]`	Double-Array (`REAL`) von Adresse `DB10.DBD 20` bis `DB10.DBD 40` (exklusiv)
`String`	`DB10.DBB 40, 100`	String (`STRING`) von Addresse `DB10.DBB 40` bis `DB10.DBB 142` (exklusiv)
`String`	`DB10.DBB 40, S5String[100]`	S5String (`BYTE`) von Addresse `DB10.DBB 40` bis `DB10.DBB 140` (exklusiv)

Import/Export

Das S7 Device Plugins unterstützt den Import und Export eines Channels als **S7-XML-Konfigurationsdatei** (siehe nächster Abschnitt). Zusätzlich können Sie in einen Channel ein **STEP7**-Projekt (`.s7p`) importieren, das in eine ZIP-Datei gepackt wurde. Die Import- und Export-Optionen werden beim Rechtsklick auf einen Channel angezeigt:



Um eine Konfigurationsdatei bzw. ein STEP7-Projekt als neuen Channel zu importieren, legen Sie zuerst einen neuen Channel (mit Standardsettings) an, und rechtsklicken Sie dann auf ihn, um den Import-Dialog anzuzeigen.

S7-XML-Konfigurationsdatei verwenden

Struktur

Das S7 Device Plugin definiert die Wurzel seines Elementbaums durch das `PluginSettings` Element wie in Plugin Konfiguration - Verwenden einer Konfigurationsdatei beschrieben, und setzt dessen XML-Struktur anhand des `Channel` -Elements fort.

Channel Element

Das `Channel` Element dient als Container für die Elemente `Settings` und `Variables`. Ein Channel identifiziert eine SPS-Verbindung, zu welcher die Kanaleinstellungen gehören. Diese Informationen werden dann vom S7 Device Plugin benutzt, um die SPS mit UKI-4.0 ® zuverbinden.

Jedes `Channel` Element stellt folgende Attribute bereit:

	Verpflichtend	Typ	Zweck
`Identifier`	nein	GUID	Eindeutiger generischer Identifier der Entity, die mit dem Kanal in Verbindung steht. *Dies ist ein allgemeines Entity Attribut, für mehr Information über dessen Nutzung siehe Plugin Konfiguration - Verwenden einer Konfigurationsdatei:* Das Entity Identifier Attribut.
`ChangeType`	nein	ChangeType	Der Zustand der Entity Konfiguration, die dazu verwendet wird, um den Kanal zu repräsentieren. *Dies ist ein allgemeines Entity Attribut, für mehr Informationen über dessen Nutzung siehe Plugin Konfiguration - Verwenden einer Konfigurationsdatei:* Das Entities ChangeType Attribut.
`Name`	**ja**	String	Eindeutiger Name (innerhalb des `Channels` Elements) des Kanals.

Das `Channel` Element kann wie folgt aussehen:

```
<Channel>
  <Settings />
  <Variables />
</Channel>
```

Settings Element

Das `Settings` Element legt die Attribute zur Erstellung des Channels fest. Diese Attribute konfigurieren die Verbindungsparameter des Kanals.

Das `Settings` Element kann wie folgt aussehen:

```
<Settings Address="192.168.0.80"
          Rack="0"
          Slot="2"
          ChannelType="OperationPanel"
          DeviceType="S7400" />
```

Das `Settings` Element stellt folgende Attribute bereit:

	Verpflichtend	Typ	Zweck
`Address`	**ja**	String	IP Adresse der SPS zur Verbindung.
`Rack`	nein	Int32	Rack Nummer der SPS (wird ab S7-1200 ignoriert).
`Slot`	nein	Int32	Slot Nummer der SPS (wird ab S7-1200 ignoriert).
`ChannelType`	nein	S7 Device Channel Type	Der Typ des kanals, der benutzt wird, um mit der SPS zu kommunizieren.
`DeviceType`	nein	S7 Device Type	SPS Gerätetyp.

S7 Device Kanaltyp

Folgende Werte sind als Attribute dieses Typs gültig:

Wert	Beschreibung
`„OperationPanel“`	um sich via OP Kanal mit dem Gerät zu verbinden.
`„ProgrammerDevice“`	um sich via PG Kanalmit dem Gerät zu verbinden.
`„Other“`	um sich über anderen Kanäle mit dem Gerät zu verbinden.

S7 Gerätetyp

Folgende Werte sind als Attribute dieses Typs gültig:

Wert	Beschreibung
`„Logo“`	SIEMENS LOGO!
`„S7200“`	SIMATIC S7-200
`„S7300“`	SIMATIC S7-300
`„S7400“`	SIMATIC S7-400
`„S71200“`	SIMATIC S7-1200
`„S71500“`	SIMATIC S7-1500

Variables Element

Das `Variables` Element dient als Container für ein oder mehrere `Variable` Elemente. Dieses Element wartet alle Variablen, die mit dem Channel in Verbindung stehen.

Das `Variables` Element kann wie folgt aussehen:

```
<Variables>
  <!-- 0-n Variable elements -->
</Variables>
```

Variable Element

Das `Variable` Element dient als Container für das Element `Variables`. Eine Variable identifiziert einen addressierbaren Bereich im Speicher der SPS oder eine Reihe von aufeinanderfolgenden Variablen identifiziert mehrere adressierbare Bereiche im Speicher der SPS. Diese Informationen werden dann vom S7 Device Plugin benutzt, um den SPS-Speicher mit UKI-4.0 ® Nodes zu verbinden.

Jedes `Variable` Element stellt folgende Attribute bereit:

	Verpflichtend	Type	Zweck
`Identifier`	nein	GUID	Der generische eindeutige Identifier der Entity, die mit der Variable in Verbindung steht. *Dies ist ein allgemeines Entity Attribut, für mehr Informationen zu dessen Nutzung siehe Plugin Konfiguration - Verwenden einer Konfigurationsdatei:* Das Entities Identifier Attribut.
`ChangeType`	nein	ChangeType	Der Zustand der Entity Konfiguration, die dazu verwendet wird, um die Variable zu repräsentieren. *Dies ist ein allgemeines Entity Attribut, für mehr Informationen zu dessen Nutzung siehe Plugin Konfiguration - Verwenden einer Konfigurationsdatei:* Das Entities ChangeType Attribut.
`Name`	**ja**	String	Eindeutiger Variablenname innerhalb des `Variables` Elements.
`Description`	nein	String	Beschreibung der Benutzung und des Zwecks des adressierten Speicherbereichs.
`Address`	**ja**	PLC Address	Operand, der benutzt werden soll, um den Speicher der SPS zu adressieren (hier nicht unterstützt: Type=Object„).
`Type`	**ja**	PLC Variable Type	Der Datentyp, der im adressierten Speicher abgelegt ist und wie er interprätiert werden muss.
`Length`	nein	Int32	Länge einer Array oder String Variable (nur unterstützt im Type=„String“ und numerischen Typen). Ist dieses Attribut definiert, definiert die Variable einen Arraywert (falls unterstützt), andernfalls einen Skalarwert.

SPS Variablentyp

Folgende Werte sind als Attribute dieses Typs gültig:

Wert	Beschreibung
`„Bool“`	Eine Variable vom SPS-Typ BOOL.
`„Byte“`	Eine Variable vom SPS-Typ BYTE.
`„Char“`	Eine Variable vom SPS-Typ CHAR.
`„Int“`	Eine Variable vom SPS-Typ INT. Repräsentiert einen vorzeichenbehafteten 16 bit Integer.
`„Word“`	Eine Variable vom SPS-Typ WORD. Repräsentiert einen vorzeichenlosen 16 bit Integer.
`„DInt“`	Eine Variable vom SPS-Typ DINT. Repräsentiert einen vorzeichenbehafteten 32 bit Integer.
`„DWord“`	Eine Variable vom SPS-Typ DWORD. Repräsentiert einen vorzeichenlosen 32 bit Integer.
`„Real“`	Eine Variable vom SPS-Typ REAL. Repräsentiert eine Gleitkommazahl mit einfacher Genauigkeit.
`„Double“`	Eine Variable vom SPS-Typ REAL. Repräsentiert eine Gleitkommazahl mit doppelter Genauigkeit.
`„Date“`	Eine Variable vom SPS-Typ DATE.
`„Time“`	Eine Variable vom SPS-Typ TIME.
`„TimeOfDay“`	Eine Variable vom SPS-Typ TIME_OF_DAY.
`„S5Time“`	Eine Variable vom SPS-Typ S5TIME.
`„DateTime“`	Eine Variable vom SPS-Typ DATE_AND_TIME.
`„String“`	Eine Variable vom SPS-Typ STRING.

Wert	Beschreibung
„S5String“	Eine Variable vom SPS-Typ BYTE. Eine feste Anzahl von Bytes wird als STRING interpretiert.

SPS Variablenadresse

Die folgenden Kombinationen aus Operand und Datentyp sind gültig, um eine gültige SPS Variablenadresse zu konstruieren:

Operands

Operand	Siemens, DE	IEC
Eingang	E	I
Ausgang	A	Q
Merker	M	M
Peripherie	P	P
Zähler	Z	C
Datenbaustein	DB	DB
Timer	T	T

Datentypen

Datentyp	Operand	Bits	Bereich	Beschreibung
BOOL	X	1	0 bis 1	Einzelnes bit, das true (1) or false (0) repräsentiert.
BYTE	B	8	0 bis 255	Ein vorzeichenloser 8-bit Integer.
WORD	W	16	0 bis 65.535	Ein vorzeichenloser 16-bit Integer (Word).
DWORD	D	32	0 bis 2^{32} -1	Ein vorzeichenloser 32-bit Integer (Double Word).
CHAR	B	8	A+00 bis A+ff	Ein ASCII-Code als vorzeichenlose 8-bit Zeichenfolge.
INT	W	16	-32.768 bis 32.767	Ein vorzeichenbehafteter 16-bit Integer.
DINT	D	32	-2^{31} bis 2^{31}-1	Ein vorzeichenbehafteter 32-bit Integer (Double Word).
REAL	D	32	+-1.5e-45 bis +-3.4e38	Eine IEEE754 als 32-bit Gleichkommazahl mit einfacher Genauigkeit.
S5TIME	W	16	00.00:00:00.100 bis 00.02:46:30.000	Eine binär codierte dezimale (BCD) Zahl. die eine Zeitspannein Millisekunden repräsentiert.
TIME	D	32	00.00:00:00.000 bis 24.20:31:23.647	Ein vorzeichenbehafteter 16-bit Integer, der eine Zeitspanne in Millisekunden repräsentiert.
TIME_OF_DAY	D	32	00.00:00:00.000 bis 00.23:59:59.999	Ein vorzeichenloser 16-bit Integer, der eine Zeitspanne in Millisekunden repräsentiert.
DATE	W	16	01.01.1990 bis 31.12.2168	Ein vorzeichenloser 16-bit Integer, der ein Datum in Tagen repräsentiert.
DATE_AND_TIME	D	64	00:00:00.000 01.01.1990 bis 23:59:59.999 31.12.2089	Eine binär codierte dezimale (BCD) Zahl, die Datum und Zeit repräsentiert.

Beispiele

Beispiel	Datentyp	Siemens	IEC
Eingang Byte 1, Bit 0	BOOL	E 1.0	I 1.0
Ausgang Byte 1, Bit 7	BOOL	A 1.7	Q 1.7
Merker Byte 10, Bit 1	BOOL	M 10.1	M 10.1
Datenbaustein 1, Byte 1, Bit 0	BOOL	DB1.DBX 1.0	DB1.DBX 1.0
Eingang Byte 1	BYTE	EB 1	IB 1
Ausgang Byte 10	BYTE	AB 10	QB 10

Beispiel	Datentyp	Siemens	IEC
Merker Byte 100	BYTE	MB 100	MB 100
Peripherie Eingang Byte 0	BYTE	PEB 0	PIB 0
Peripherie Ausgang Byte 1	BYTE	PAB 1	PQB 1
Datenbaustein 1, Byte 1	BYTE	DB1.DBB 1	DB1.DBB 1

Das `Variable` Element kann wie folgt aussehen:

```
<Variable Name="Active Rotations"
          Type="DInt"
          Address="DB101.DBD 0" />
```

Das `Variable` Element mit dem `Type` Attribut und Attributwert „Object“ kann wie folgt aussehen:

```
<Variable Name="Mill Job No. 1" Description="Identifies the mill job no. 1" Type="Object">
  <Variables>
    <Variable Name="Rotations" Type="DInt" Address="DB200.DBD 0" />
    <Variable Name="Use Colling" Type="Bool" Address="DB200.DBX 4.0" />
    <Variable Name="Use Fan" Type="Bool" Address="DB200.DBX 4.1" />
    <Variable Name="Point 1" Type="Object">
      <Variables>
        <Variable Name="X" Description="The x portion of the drill." Type="Int"
Address="DB200.DBW 5" />
        <Variable Name="Y" Description="The y portion of the drill." Type="Int"
Address="DB200.DBW 7" />
      </Variables>
    </Variable>
    <Variable Name="Point 2" Type="Object">
      <Variables>
        <Variable Name="X" Description="The x portion of the drill." Type="Int"
Address="DB200.DBW 9" />
        <Variable Name="Y" Description="The y portion of the drill." Type="Int"
Address="DB200.DBW 11" />
      </Variables>
    </Variable>
    <Variable Name="Point 3" Type="Object">
      <Variables>
        <Variable Name="X" Description="The x portion of the drill." Type="Int"
Address="DB200.DBW 13" />
        <Variable Name="Y" Description="The y portion of the drill." Type="Int"
Address="DB200.DBW 15" />
      </Variables>
    </Variable>
  </Variables>
</Variable>
```

Verwendung

Es wird empfohlen, einen professionellen XML-Editor zum manuellen Editieren der Konfigurationsdatei zu verwenden. Um den Vorteil der XML-Schemadefinition (am Ende der Dokumentation erwähnt) zu nutzen, müssen Sie wie folgt auf das `xsi:noNamespaceSchemaLocation` Attribut im Wurzelelement des Dokuments `PluginSettings` verweisen (die XSD-Datei muss neben der XML-Datei abgelegt werden):

```
<?xml version="1.0" encoding="utf-8" ?>
<PluginSettings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:noNamespaceSchemaLocation="UKI-4.0 .S7DevicePlugin.Settings.xsd">
  <!-- child elements -->
</PluginSettings>
```

Unabhängig davon, ob die Konfigurationsdatei manuell oder automatisiert bearbeitet / erstellt wird, muss der oben dokumentierte Elementbaum vollständig sein, um eine gültige, wohlgeformte und benutzbare Konfigurationsdatei zu erzeugen.

Synchronisation

Eine S7-XML-Konfigurationsdatei wird einem S7 Device Channel in UKI-4.0 zur **automatischen Synchronisierung** zugeordnet, wenn die folgende Bedingung erfüllt ist:

- Eine Datei mit dem Namen UKI-4.0 `.S7DevicePlugin.<ChannelName>.Settings.xml` existiert im ConfigFolder (`<`UKI-4.0 `ProjectDir>/plugins/S7DevicePlugin`) wenn das S7-Plugin gestartet wird (d.h. wenn UKI-4.0 gestartet wird) oder wenn in UKI-4.0 ein neuer Channel erzeugt wird.

In diesem Fall wird die Konfigurationsdatei, wenn sie sich ändert, automatisch nach UKI-4.0 synchronisiert. Umgekehrt werden bei Änderungen in UKI-4.0 (oder in den Channel-Settings) diese in die Konfigurationsdatei synchronisiert.

Unabhängig von der automatische Synchronisation können Sie auch eine manuelle Synchronisation durch den Rechtsklick auf einen S7 Device Channel in UKI-4.0 und das Wählen des entsprechenden Import/Export-Menüeintrages durchführen.

Beispiel Konfigurationsdatei

UKI-4.0 .S7DevicePlugin.Settings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<PluginSettings>
  <S7Device>
    <Channels>
      <Channel Name="Line Controller">
        <Settings Address="192.168.0.80" ChannelType="OperationPanel"
DeviceType="S7400" />
        <Variables>
          <Variable Name="Active Job Name" Type="String" Address="DB100.DBB 0"
Length="64" />
          <Variable Name="Active Job Number" Type="String" Address="DB100.DBB 80"
Length="8" />
          <Variable Name="Active Rotations" Type="DInt" Address="DB101.DBD 0" />
          <Variable Name="Mill Job No. 1" Description="Identifies the mill job no. 1"
Type="Object">
            <Variables>
              <Variable Name="Rotations" Type="DInt" Address="DB200.DBD 0" />
              <Variable Name="Use Colling" Type="Bool" Address="DB200.DBX 4.0" />
              <Variable Name="Use Fan" Type="Bool" Address="DB200.DBX 4.1" />
              <Variable Name="Point 1" Type="Object">
                <Variables>
                  <Variable Name="X" Description="The x portion of the drill."
Type="Int" Address="DB200.DBW 5" />
                  <Variable Name="Y" Description="The y portion of the drill."
Type="Int" Address="DB200.DBW 7" />
                </Variables>
              </Variable>
              <Variable Name="Point 2" Type="Object">
                <Variables>
                  <Variable Name="X" Description="The x portion of the drill."
Type="Int" Address="DB200.DBW 9" />
                  <Variable Name="Y" Description="The y portion of the drill."
Type="Int" Address="DB200.DBW 11" />
                </Variables>
              </Variable>
              <Variable Name="Point 3" Type="Object">
                <Variables>
                  <Variable Name="X" Description="The x portion of the drill."
Type="Int" Address="DB200.DBW 13" />
                  <Variable Name="Y" Description="The y portion of the drill."
Type="Int" Address="DB200.DBW 15" />
                </Variables>
              </Variable>
            </Variables>
          </Variable>
        </Variables>
      </Channel>
    </Channels>
  </S7Device>
</PluginSettings>
```

Beispiel Konfigurationsschema-Datei

UKI-4.0 .S7DevicePlugin.Settings.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="UKI-4.0.S7DevicePlugin.Settings"
```

```
          elementFormDefault="qualified"
          xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="Guid">
    <xs:restriction base="xs:string">
      <xs:pattern value="([0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-
[0-9a-fA-F]{12})|(\{[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-
fA-F]{12}\})"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="PlcAddress">
    <xs:restriction base="xs:string">
      <xs:pattern value="^[ \t]*((DB[ \t]*([\d]+)[ \t]*\.[
\t]*(DB))|(((Z|C)|DB|M|(E|I)|L|(A|Q)|(PE|PI)|(PA|PQ)|T)))[ \t]*((X|B|W|D))?[
\t]*([\d]+)(\.([\d]+))?\b" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="SettingsChangeTypeEnumType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="None" />
      <xs:enumeration value="Created" />
      <xs:enumeration value="Updated" />
      <xs:enumeration value="Deleted" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="VariableEnumType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Bool" />
      <xs:enumeration value="Byte" />
      <xs:enumeration value="Char" />
      <xs:enumeration value="Int" />
      <xs:enumeration value="Word" />
      <xs:enumeration value="DInt" />
      <xs:enumeration value="DWord" />
      <xs:enumeration value="Real" />
      <xs:enumeration value="Double" />
      <xs:enumeration value="Date" />
      <xs:enumeration value="Time" />
      <xs:enumeration value="TimeOfDay" />
      <xs:enumeration value="S5Time" />
      <xs:enumeration value="DateTime" />
      <xs:enumeration value="String" />
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="VariableType">
    <xs:sequence>
      <xs:element name="Variables" type="VariablesType" minOccurs="0" maxOccurs="1" />
    </xs:sequence>

    <xs:attribute name="Identifier" type="Guid" use="required" />
    <xs:attribute name="Name" type="xs:string" use="required" />
    <xs:attribute name="Description" type="xs:string" use="optional" />
    <xs:attribute name="Address" type="PlcAddress" use="optional" />
    <xs:attribute name="Type" type="VariableEnumType" use="required" />
    <xs:attribute name="Length" type="xs:integer" use="optional" default="-1" />
    <xs:attribute name="ChangeType" type="SettingsChangeTypeEnumType" use="optional"
```

```
/>
  </xs:complexType>

  <xs:complexType name="VariablesType">
    <xs:sequence>
      <xs:element name="Variable" type="VariableType" minOccurs="0"
maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="ChannelDeviceEnumType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Logo" />
      <xs:enumeration value="S7200" />
      <xs:enumeration value="S7300" />
      <xs:enumeration value="S7400" />
      <xs:enumeration value="S71200" />
      <xs:enumeration value="S71500" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="ChannelEnumType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="OperationPanel" />
      <xs:enumeration value="ProgrammerDevice" />
      <xs:enumeration value="Other" />
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="ChannelSettingsType">
    <xs:attribute name="Address" type="xs:string" use="required" />
    <xs:attribute name="Rack" type="xs:integer" use="optional" default="0" />
    <xs:attribute name="Slot" type="xs:integer" use="optional" default="2" />
    <xs:attribute name="ChannelType" type="ChannelEnumType" use="optional"
default="OperationPanel" />
    <xs:attribute name="DeviceType" type="ChannelDeviceEnumType" use="optional"
default="S7300" />
    <xs:attribute name="ChangeType" type="SettingsChangeTypeEnumType" use="optional"
/>
  </xs:complexType>

  <xs:complexType name="ChannelType">
    <xs:sequence>
      <xs:element name="Settings" type="ChannelSettingsType" minOccurs="0"
maxOccurs="1" />
      <xs:element name="Variables" type="VariablesType" minOccurs="0" maxOccurs="1" />
    </xs:sequence>

    <xs:attribute name="Identifier" type="Guid" use="required" />
    <xs:attribute name="Name" type="xs:string" use="required" />
    <xs:attribute name="ChangeType" type="SettingsChangeTypeEnumType" use="optional"
/>
  </xs:complexType>

  <xs:complexType name="ChannelsType">
    <xs:sequence>
      <xs:element name="Channel" type="ChannelType" minOccurs="0"
maxOccurs="unbounded" />
    </xs:sequence>
```

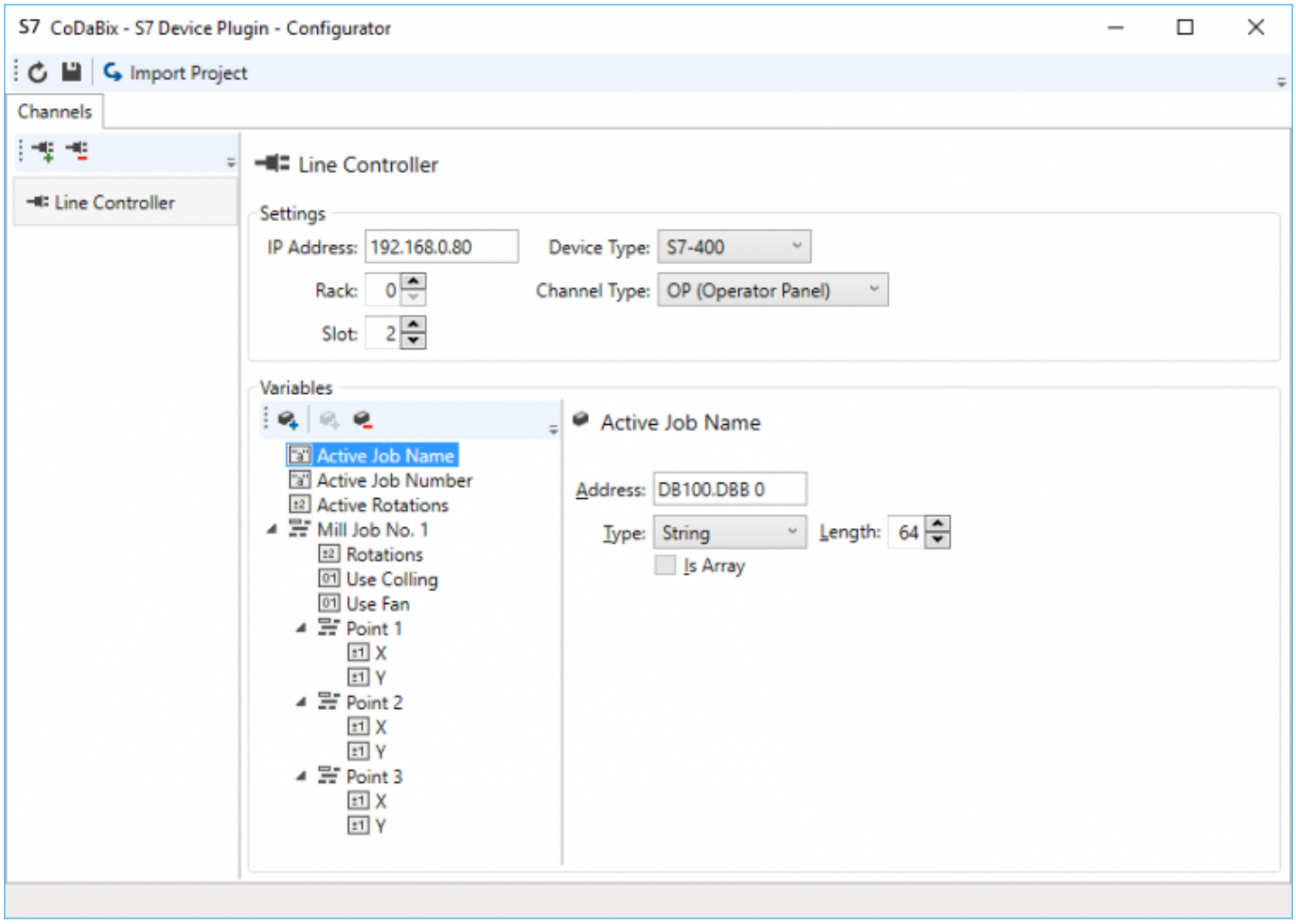
```
  </xs:complexType>

  <xs:complexType name="DeviceType">
    <xs:sequence>
      <xs:element name="Channels" type="ChannelsType" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="PluginSettingsType">
    <xs:sequence>
      <xs:element name="S7Device" type="DeviceType" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>

  <xs:element name="PluginSettings" type="PluginSettingsType" />
</xs:schema>
```

Anwendung verwenden (nur unter Windows)

Beachten Sie: Diese Konfigurationsanwendung ermöglicht es, S7-XML-Konfigurationsdateien (`.xml`) zu laden und zu speichern, die in UKI-4.0 bei einem S7 Device Channel über das Kontextmenü importiert und exportiert werden können, oder die einem UKI-4.0 S7 Device Channel zur automatischen Synchronisierung zugeordnet sind (siehe Abschnitt **S7-XML-Konfigurationsdatei verwenden**).

Die Konfigurationsanwendung befindet sich im Ordner `<`UKI-4.0 `InstallDir>` und wird durch einen Doppelklick auf die Datei mit dem Namen UKI-4.0 `.S7DevicePlugin.Configurator.exe` gestartet.

Überblick



Verwendung

- Einen neuen `Channel` in der linken Liste hinzufügen, indem Sie auf den oberen Plus-Button klicken.
  - Namen vergeben, indem Sie im rechten Ausschnitt den Kanalnamen klicken.
  - Die `Channel Settings` einstellen wie die `IP Adresse` (mindestens erforderlich).
  - Optional den passenden `Device Type` und den zu benutzenden `Channel Type` auswählen.
  - Optional den `Rack` und `Slot` einstellen (falls nicht gleich benutzerdefiniert).
- In der zweiten Liste eine neue `Variable` zum `Channel` hinzufügen, indem Sie auf den oberen ersten Plus-Button klicken.
  - Mit Klick auf den Namen im rechten Ausschnitt ändern Sie den Variablennamen.
  - Optional in den freien Bereich unterhalb des Variablennamens eine Beschreibung der Variable hinzufügen.
  - Den SPS-Adressoperanden festlegen, der benutzt werden soll, um auf den SPS-Speicher für die Variable zuzugreifen.
  - SPS-Variablentyp in der Combo auswählen und im Falle einer String oder Array (Option `Is Array` auswählen) Variable die passende Länge eingeben.
  - Fügen Sie optional zusätzliche Variablen hinzu, falls die vorher hinzugefügte Variable vom Typ „Object“ ist. Dies machen Sie. indem Sie die Object Variable im linken Baum auswählen und dann den zweiten Plus-Button klicken.
- Zum Löschen einer Variablen wählen Sie diese im linken Baum aus und klicken Sie den Minus-Button.
- Zum Löschen eines Channels wählen Sie diesen in der linken Liste aus und klicken Sie den Minus-Button.
- Bei Klick auf den `Import Project`-Button ist es möglich, folgende Projektformate zu importieren:
  - STEP7 Projektdateien (*.s7p)

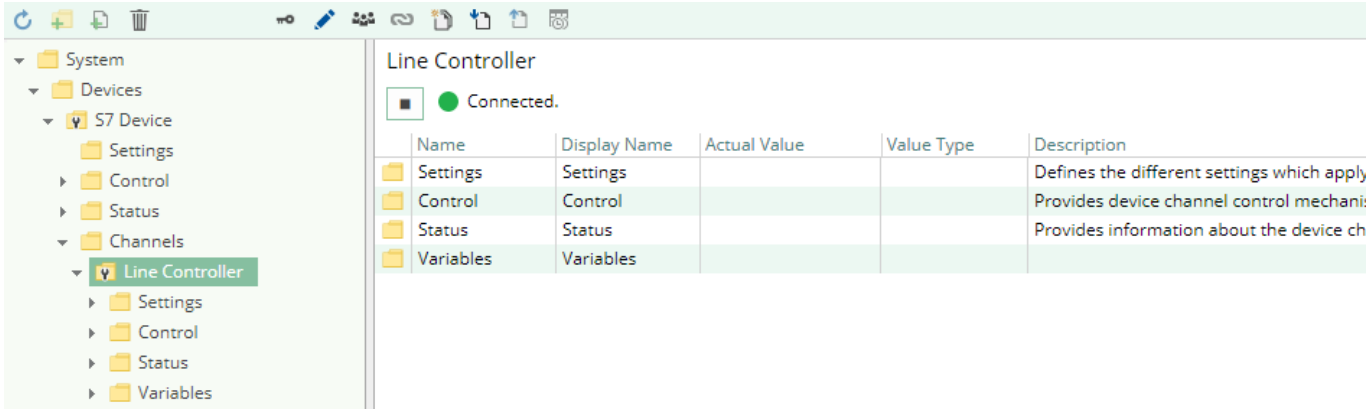
- IP-S7 Projektdateien (*.ips7)
- S7 Watch Projektdateien (*.wproj)
- CSV-to-S7 Projektdateien (*.ini)

Fehlerdiagnose

Das S7 Device Plugin stellt je nach zu inspizierender Schicht verschiedene Statusinformationen zur Verfügung. Generell wird die `Channel`-basierte Diagnoseinformationen vom Verbindungsstatus vom kanal zur S7 SPS. Die `Variable`-basierte Diagnoseinformation wird während des Lese- und Schreibzugriffes verschiedener Variablen generiert.

Kanal

Um den Status des S7–Kanals zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf das folgende Bild:



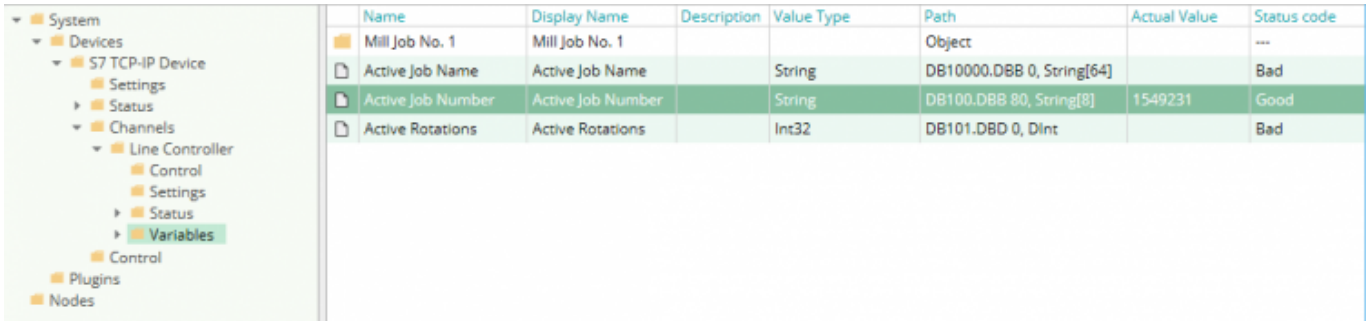
Das obige Bild zeigt das Bedienfeld des S7-Kanals, das alle statusrelevanten Informationen anzeigt. Das Bedienfeld aktualisiert automatisch seine Statusinformation, wenn ein neuer Status verfügbar ist.

Statuskreis

Farbe	Bedeutung
	Der Kanal ist gestoppt. Klicken Sie den ►-Button, um ihn zu starten.
	Der Kanal startet oder stoppt gerade oder wartet auf den Verbindungsaufbau.
	Der Kanal läuft und es wurde erfolgreich eine Verbindung hergestellt. Sie können ihn durch Klick auf den ■-Button stoppen.
	Der Kanal läuft, aber die Verbindung ist momentan fehlerhaft. Bitte überprüfen Sie den Statustext für weitere Informationen.

Variablen

Zur Überwachung und Diagnose der verschiedenen S7 Variablen werfen Sie einen Blick auf die `Status Code`-Eigenschaft der Variable, die in UKI-4.0 angezeigt wird.
Falls die `Status Code`-Spalte den Wert `Bad` anzeigt, ist in den meisten Fällen der adressierte Datenbereich nicht erreichbar.



Logdatei

Alle Device `Channel`-bezogenen Statusinformationen werden auch in der kanalspezifischen Logdatei im `[LoggingFolder]` protokolliert. Jede Logdatei wird im Namensschema `S7 Device.<Channel>.log` benannt. Der Inhalt einer solchen Logdatei kann wie folgt aussehen:

```
...
[15:31:46 05.09.2016] - Error (Severity=High): Code=[-6], Text=[The specified CPU could not be found.]
...
```

Beim Benutzen des Beispielkanals wäre der Name der Logdatei: `S7 Device.Line Controller.log`.

Status Codes

Die folgende Tabelle zeigt die verschiedenen Statusinformationen, die möglich sind:

Code	Kategorie	Schweregrad	Text	Bedeutung
-88	Error	High	The operation has been canceled.	Der Vorgang wurde abgebrochen.
-34	Error	High	The requested PLC block could not be found.	Der angefragte SPS-Bereich wurde nicht gefunden.
-33	Error	High	The PLC does not support bit based operations.	Die SPS unterstützt keine Bit-basierten Operationen.
-32	Error	High	The PLC supplied truncated data.	Die SPS hat unvollständige Daten gesendet.
-31	Error	High	The requested PLC or PC type of data can not be transformed from one into the other.	Der angeforderten SPS- oder PC-Datentyp kann nicht vom einen zum anderen umgewandelt werden.
-30	Error	High	The requested PLC or PC type of data is not available.	Der angeforderte SPS- oder PC-Datentyp ist nicht verfügbar.
-21	Error	High	A connection to the device has already been established.	Es wurde bereits eine Verbindung zum Gerät aufgebaut.
-20	Error	High	The size of the supplied buffer is lower than the amount of data available.	Die Größe des übergebenen Puffers ist geringer als die Menge der verfügbaren Daten.
-11	Error	High	The type or format of data supplied is not supported.	Datentyp oder -format wird nicht unterstützt.
-10	Error	High	The supplied access mode is not supported or unknown.	Der übergebene Zugriffsmodus wird nicht unterstützt oder ist unbekannt.
-9	Error	High	The operation failed upon a value range error.	Vorgang aufgrund eines Wertebereichfehlers fehlgeschlagen.
-8	Error	High	Failed to allocate required memory.	Angeforderter Speicher konnte nicht zugeteilt werden.

Code	Kategorie	Schweregrad	Text	Bedeutung
-7	Error	High	The operation failed upon a socket error.	Vorgang aufgrund eines Socket Fehlers fehlgeschlagen.
-6	Error	High	The specified CPU could not be found.	Die angegebene CPU konnte nicht gefunden werden.
-5	Error	High	A general error occurred.	Ein allgemeiner Fehler ist aufgetreten.
-2	Error	High	The necessary memory could not be allocated (out of memory).	Die erforderliche Speicher konnte nicht allokiert werden (nicht genügend Arbeitsspeicher).
-1	Error	High	The operation has timed out.	Der Vorgang hat zu lange gedauert.
0	Information	Moderate	The operation completed without any kind of error.	Der Vorgang wurde ohne Fehler beendet.
1	Information	Moderate	The operation completed successfully.	Der Vorgang wurde erfolgreich abgeschlossen.
2	Information	Moderate	The addressed data area does not exist.	Der adressierte Datenbereich existiert nicht.

Entities

Wie jedes Device Plugin erweitert das S7 Device Plugin das UKI-4.0 Device Modell.

Device

Der Device Typ `S7Device` des Plugins definiert auch den `S7DeviceChannel` und erweitert somit die grundlegenden UKI-4.0 `Device` und UKI-4.0 `DeviceChannel` Entities. Während das `S7Device` nur eine Konkretisierung des UKI-4.0 `Device` darstellt, erweitert der `S7DeviceChannel` den UKI-4.0 `DeviceChannel` mit dem S7 Variable Entities.

Channel

Jeder Kanal wird von einem Channel Worker behandelt, der eine physische Verbindung zur S7 SPS herstellt. Zur Fehlerdiagnosezwecken liest der Worker automatisch alle 10 Sekunden die SPS-Adresse „MB 0“, um den `Channel` Statuscode und seine Beschreibung zu aktualisieren und Verbindungsfehler zu erkennen.

Der Worker liest standardmäßig keine Werte. Wenn ein Anwender oder Plugin in UKI-4.0 einen synchronen Lesevorgang der `Channel` Variablen anfordert (z.B. mit der „Read actual value“-Funktion in der UKI-4.0 Webkonfiguration), liest der Channel Worker diese aus dem zugrundeliegenden S7 Device und schreibt diese in die entsprechenden UKI-4.0 ® Nodes.

Ähnlich schreibt der Channel Worker auch die Werte in die zugrundeliegende S7, wenn ein Client oder Plugin Werte in die `Channel` Variablen schreibt.

Damit eine S7 Variable regelmäßig gelesen wird, können Sie in der Webkonfiguration bei dem Node „History Options“ auf `Yes` stellen (was eine interne Subscription erstellt), oder Sie können zum Beispiel einen OPC UA Client verwenden, der mit dem OPC UA Serverplugin verbunden ist und damit eine Subscription für die S7 Variablennodes erstellen. In diesen Fällen liest der Channel Worker in regelmäßigen Intervallen die Variablen aus der S7 und schreibt den neuen Wert nach einer Wertänderung automatisch in die entsprechende UKI-4.0 ® Node.

Variable

Jede S7 Variable kann anhand des gleichen SPS-Adressoperanden auf denselben SPS-Speicher zugreifen. Seine Interpretation hängt jedoch von der SPS-Datentyp Auswahl ab. Unterstützte Variablenformate sind Skalar, Array und Objektvariablen. Variablen vom Typ `Object` können weitere Variablen besitzen und sind als `UserDefinedTypes` (UDT in STEP7) definiert.

Ordner & Dateien

Ordner

Name	Pfad	Zweck / Verwendung
AssemblyFolder	<UKI-4.0 InstallDir>/plugins/S7DevicePlugin/	Beinhaltet die Plugin Assembly Datei.
ConfigFolder	<UKI-4.0 DataDir>/plugins/S7DevicePlugin/	Beinhaltet die Plugin Konfigurationsdatei.
LoggingFolder	<UKI-4.0 DataDir>/log/	Beinhaltet die Plugin Log Dateien.

Dateien

Typ	Pfad	Zweck / Verwendung
Assembly	[AssemblyFolder]/UKI-4.0 .S7DevicePlugin.dll	Die Plugin Assembly Datei.
Config App	<UKI-4.0 InstallDir>/UKI-4.0 .S7DevicePlugin.Configurator.exe	Die Plugin Konfigurationsdatei.
Config File	[ConfigFolder]/UKI-4.0 .S7DevicePlugin.<ChannelName>.Settings.xml	Die optionale Channel-Konfigurationsdatei.
Logging	[LoggingFolder]/S7 Device.<ChannelName>.log	Die Log Datei.

Versionsinformation

Dieses Dokument

Datum	2020-21-0
Version	1.3

Plugin

Name	S7 Device Plugin
Node	/System/Devices/S7 Device
Version	1.2.0

Assembly

Name	UKI-4.0 .S7DevicePlugin.dll
Datum	2020-12-01
Version	1.2.0.0

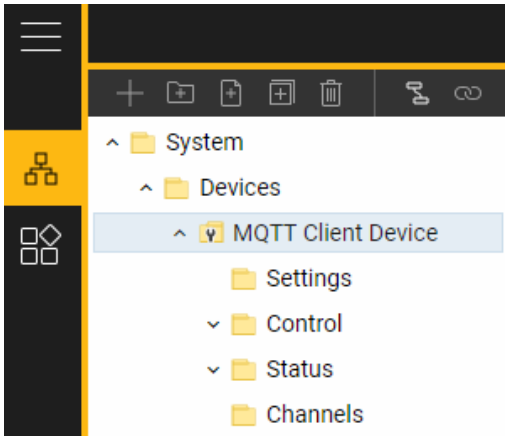
MQTT Client Device Plugin

Das MQTT Client Device Plugin ermöglicht das Veröffentlichen und Abonnieren (engl. Publish/Subscribe, kurz Pub/Sub) von Werten anhand von `Topics` in Geräten, die über einen **MQTT Broker** verfügen.

Beachten Sie: Das MQTT Client Device Plugin ist aktuell noch im **experimentellen Zustand** und ist deshalb nur nach expliziter Auswahl im Windows Installer verfügbar und für den produktiven Einsatz nicht vorgesehen.

Konfiguration

Die gesamte MQTT Client Device Plugin-Konfiguration befindet sich unter dem Nodepfad `/System/Devices/MQTT Client Device`.



Channel

Ein MQTT Client Device Channel repräsentiert die Verbindung zu einen MQTT Broker.

Settings

Client Id

Ein frei wählbarer Bezeichner der es erlaubt im Broker den Client zu identifizieren.

Server Address

Die URL zum MQTT Broker, stets mit Schema 'mqtt.tcp' gefolgt vom Hostname oder IP Adresse des Hosts und schließlich der zu verwendende Port z.B. `mqtt.tcp://mymqttbroker:1883`.

Login Type

Definiert, welche Authentifizierungsart verwendet wird.

Login Name

Der Benutzer des MQTT Brokers, der für die Authentifizierung benutzt wird.

Login Password

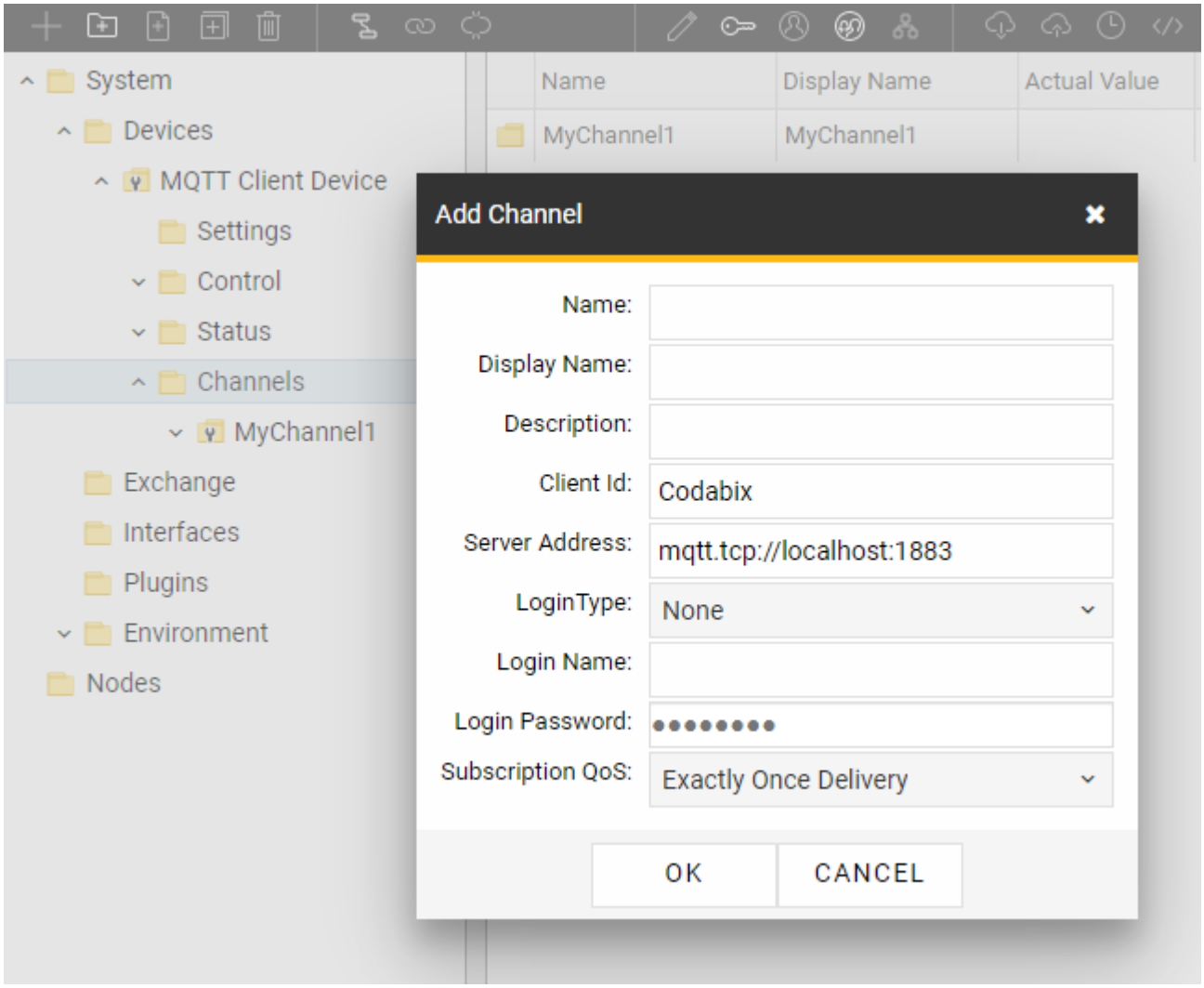
Das Passwort des Benutzers des MQTT Brokers.

Subscription QoS

Die `Qualität des Services` (engl. Quality of Service, kurz QoS) der verwendeten Subscription zur Aktualisierung der Wert der Topics. Der Wert bestimmt die Zustellgarantie für die Aktualisierungs-Nachricht vom Broker zum Client. Die folgenden Werte sind möglich:

- `At Most Once Delivery`: Die Nachricht wird nie, einmal oder mehrmals zugestellt.
- `At Least Once Delivery`: Die Nachricht wird mindestens einmal (oder öfter) zugestellt.
- `Exactly Once Delivery`: Die Nachricht wird exakt einmal zugestellt (Standardwert).

Hinzufügen eines Channels



Um einen neuen MQTT-Channel zu erstellen, gehen Sie wie folgt vor:

1. Fügen Sie einen Folder Node unter dem Node `MQTT Client Device/Channels` hinzu, oder machen Sie einen Rechtsklick auf den `MQTT Client Device/Channels`-Node und wählen Sie `Add Channel` aus.
2. Tragen Sie im `Add Channel`-Dialog die Settings für die MQTT-Verbindung ein.
3. Nachdem Sie „Save“ geklickt haben, wird die Channel-Node erstellt.
4. Sie können den Kanal starten, indem Sie die Channel-Node auswählen und den Startbutton klicken.

Topics

Unter dem `Topics`-Node können Sie Datenpunktnodes erstellen, die auf dem MQTT Broker abonniert beziehungsweise an die Werte veröffentlicht werden sollen.

Unterstützte Node-Werttypen:

- `String` (je nach Broker als JSON)

Sie können auch Folder-Nodes erstellen, um Datapoint-Nodes zu gruppieren.

Path

Die `Path`-Property des Nodes wird verwendet, um den Namen des `Topics` im Broker festzulegen.

Abonnieren

Ein einmal erstellter Topic-Node ändert seinen Wert durch ein aktives Abonnement (engl. Subscription). Hierdurch übermittelt der MQTT-Broker jede Änderung des Topics an den Node, der wiederum diesen Wert als aktuellen Node-Wert übernimmt.

Veröffentlichen

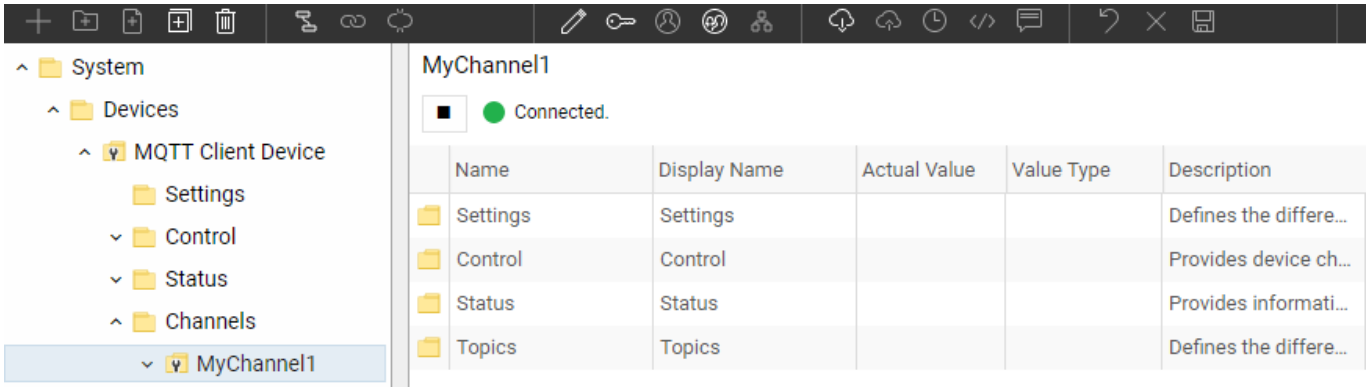
Wenn Topic-Nodes geschrieben werden, führt das Plugin eine Veröffentlichung (engl. Publish) des Wertes durch.

Fehlerdiagnose

Das MQTT Client Device Plugin liefert je nach zu untersuchender Schicht verschiedene Statusinformationen. Generell werden die kanalbasierten Diagnoseinformationen beim Verbindungsauf-/abbau, produziert. Die Topic-basierten Diagnoseinformationen werden während eines aktiven Abonnements beziehungsweise nach einen Schreibzugriff auf den verschiedenen Topics produziert.

Kanal

Um den Status des MQTT-Kanals zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf das folgende Bild:



Das obige Bild zeigt das Bedienfeld des MQTT-Kanals, das alle statusrelevanten Informationen anzeigt. Das Bedienfeld aktualisiert automatisch seine Statusinformation, wenn ein neuer Status verfügbar ist.

Statuskreis

Farbe	Bedeutung
	Der Kanal ist gestoppt. Klicken Sie den ►-Button, um ihn zu starten.
	Der Kanal startet oder stoppt gerade oder wartet auf den Verbindungsaufbau.
	Der Kanal ist bereit für Lese-/Schreiboperationen. Sie ihn können durch Klick auf den ■-Button stoppen.
	Der Kanal läuft, aber die Verbindung ist momentan fehlerhaft. Bitte überprüfen Sie den Statustext für weitere Informationen.

Topic

Um den Status der verschiedenen Topics zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf die in UKI-4.0 angezeigte `Status`-Eigenschaft der Spalte. Aktivieren Sie historische Werte oder stellen Sie ein aktives Abonnement auf den Topics sicher, um die vom Broker bereitgestellten Werte und das Ergebnis in den Topics zu speichern.

	Name	Display Name	Actual Value	Value Type	Description	Path	Status
	A	A	abc	String		a/1	Bad: Operation timed-out.
	B	B	456	String		b/2	Bad: Operation timed-out.

Logdatei

Alle kanalbezogenen Statusinformationen werden auch in die kanalspezifische Logdatei im `[LoggingFolder]` protokolliert. Jede Logdatei wird nach dem Namensschema `MQTT Client Device.<ChannelName>.log` benannt.

Der Inhalt einer solchen Logdatei kann wie folgt aussehen:

```
...
2018-04-11 11:32:37.0 +2: [Error] Error (Severity=High): Code=[-1], Text=[The operation has
timed-out.], Details=[]
...
```

Entities

Wie jedes Device Plugin erweitert das MQTT Client Device Plugin das UKI-4.0 Device Modell.

Device

Der Device Typ `MqttClientDevice` des Plugins definiert auch den `MqttClientDeviceChannel` und erweitert somit die grundlegenden UKI-4.0 `Device` und UKI-4.0 `DeviceChannel` Entities. Während das `MqttClientDevice` nur eine Konkretisierung des UKI-4.0 `Device` darstellt, erweitert der `MqttClientDeviceChannel` den UKI-4.0 `DeviceChannel` mit den MQTT Topic Entities.

Channel

Jeder Kanal definiert seinen eigenen ein- und ausgehenden Datenfluss, um die Werte der Topics entsprechend dem Pub/Sub-Modell mit dem MQTT Broker auszutauschen.

Der für den lesenden Zugriff verantwortliche Datenfluss liefert standardmäßig keine Werte. Auch nicht, wenn ein Anwender oder Plugin in UKI-4.0 einen synchronen Lesevorgang der `Channel`-Topics anfordert

(z.B. mit der „Read actual value“-Funktion in der UKI-4.0 Webkonfiguration). Da, im Sinne des Pub/Sub-Modells, neue Werte vom MQTT Broker an den MQTT Client gesendet werden, ist ein explizites Lesen nicht möglich. Wird ein Wert durch den eingehenden Datenfluss empfangen (mittels Subscription), wird dieser in die entsprechenden UKI-4.0 -Nodes geschrieben.

Der ausgehende Datenfluss wiederum schreibt die Werte in den Topic des Brokers (mittels Publish), wenn ein Client oder Plugin Werte in die `Channel`-Topics schreibt.

Damit der Wert eines MQTT Topics auf dem aktuellen Stand bleibt, können Sie in der Webkonfiguration bei dem Node „History Options“ auf `Yes` stellen (was eine interne Subscription erstellt), oder Sie können zum Beispiel einen OPC UA Client verwenden, der mit dem OPC UA Server Plugin verbunden ist und damit eine Subscription für die MQTT-Topicnodes erstellen. In diesen Fällen verwendet der Channel den eingehenden Datenfluss dazu, um über eine Subscription zum Broker, neue Werte zu Empfangen und schreibt diese bei Wertänderung automatisch in den entsprechenden UKI-4.0 -Node.

Ordner & Dateien

Ordner

Name	Pfad	Zweck / Verwendung
AssemblyFolder	<UKI-4.0 `InstallDir>/plugins/MqttClientDevicePlugin/`	Beinhaltet die Plugin-Assemblydatei.
ConfigFolder	<UKI-4.0 `ProjectDir>/plugins/MqttClientDevicePlugin/`	Beinhaltet die Plugin-Konfigurationsdatei.
LoggingFolder	<UKI-4.0 `ProjectDir>/log/`	Beinhaltet die Plugin-Logdateien.

Dateien

Typ	Pfad	Zweck / Verwendung
Assembly	`[AssemblyFolder]/`UKI-4.0 `.MqttClientDevicePlugin.dll`	Die Plugin-Assembly Datei.
Logging	`[LoggingFolder]/MQTT Client Device.<ChannelName>.log`	Die Logdatei.

Versionsinformation

Dieses Dokument

Datum	`2021-02-05`
Version	`1.0`

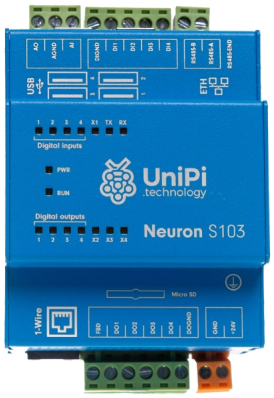
Plugin

Name	`MQTT Client Device Plugin`
Node	`/System/Devices/MQTT Client Device`
Version	`1.0.0`

Assembly

Name	UKI-4.0 `.MqttClientDevicePlugin.dll`
Datum	`2021-02-05`
Version	`1.10.0.0`

UniPi Device Plugin



Das UniPi Device Plugin ist für den Einsatz auf einem UniPi Gerät konzipiert.

Es ermöglicht die Konfiguration des Geräts und das Lesen und Schreiben von Daten.

Unterstützte Geräte

- UniPi Neuron Serie
- UniPi Neuron Extension Serie

Installation

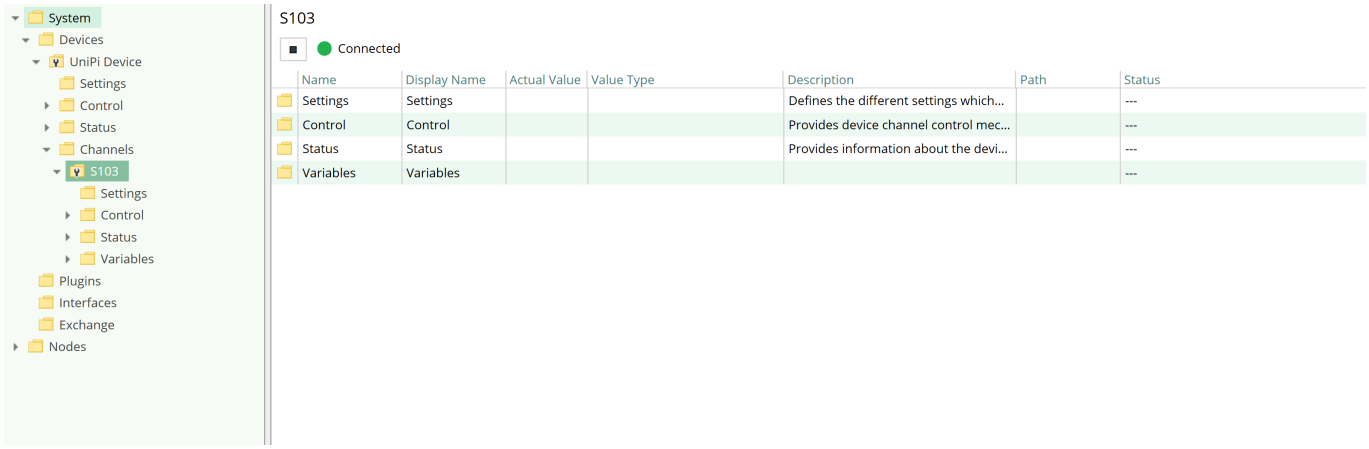
Dieses Plugin ist Bestandteil des UKI-4.0 ® für Raspberry Pi Setups. Bitte konsultieren Sie UKI-4.0 ® Setup und erster Start für weitere Informationen darüber, wie dieses Plugin installiert und deinstalliert werden kann.

Anforderungen

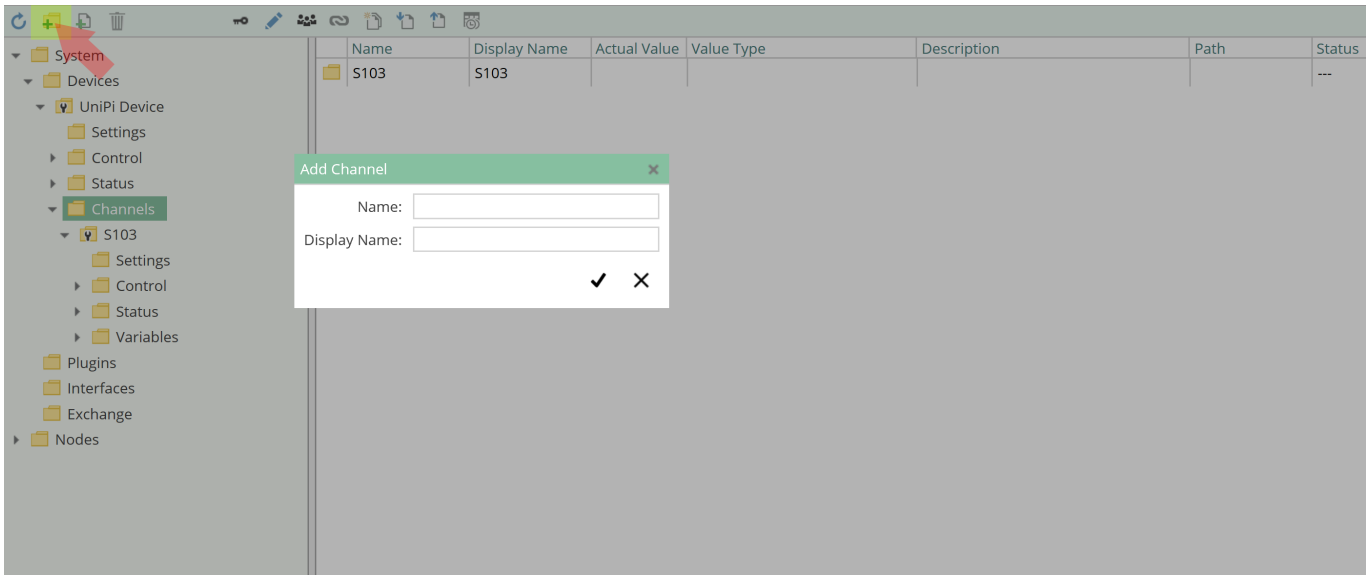
- UniPi Neuron Gerät
- SD Card mit aktuellem UniPian Neuron OS Image (Download)
- Standardanforderungen von UKI-4.0 ®

Konfiguration

Die gesamte UniPi Device Plugin-Konfiguration befindet sich unter dem Nodepfad `/System/Device/UniPi Device`.

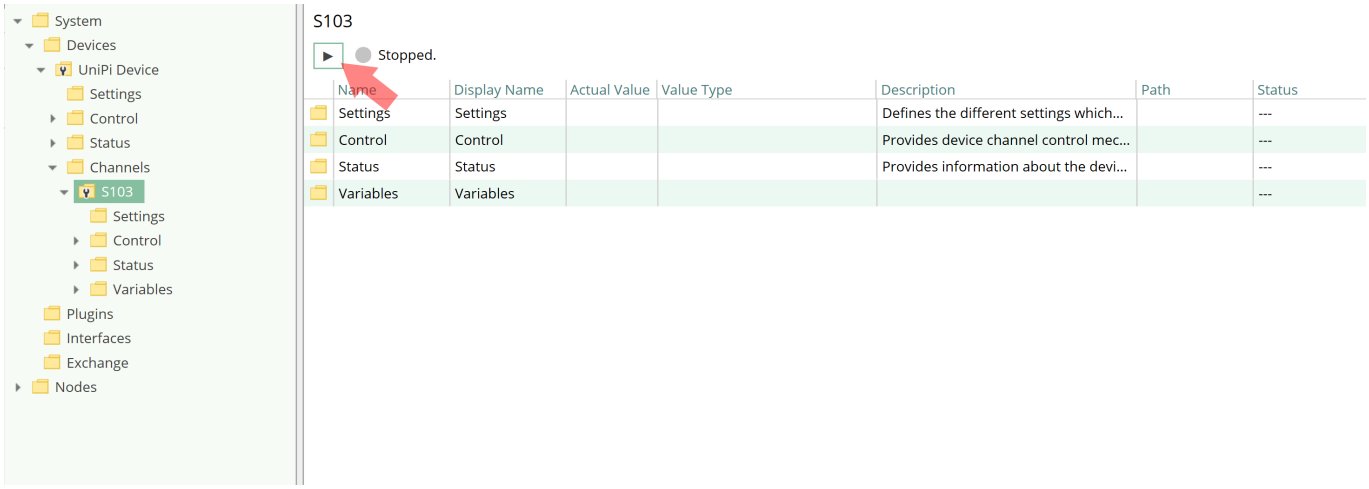


Der Nodebaum im oberen Bild zeigt den Standardnodebaum des UniPi Device Plugins. Um einen neuen UniPi-Kanal aufzusetzen, fügen Sie einen Folder Node unter dem Node `UniPi Device/Channels` hinzu, oder machen Sie einen Rechtsklick auf den `UniPi Device/Channels`-Node und wählen Sie `Add Channel` aus.



Channel

Für den Kanal sind keine weiteren Einstellungen notwendig. Nachdem Sie „Save“ geklickt haben, wird die Channel-Node erstellt. Sie können den Channel starten, indem Sie diese Node auswählen und den Startbutton klicken:



Variablen

Um die Einrichtung der UniPi-Variablen zu vereinfachen, stellen wir für das `UniPi-Modell S103` eine Standard Channel-Konfiguration bereit, in der bereits die wichtigsten Variablen des UniPi Geräts definiert sind. Diese Konfigurationsdatei lässt sich über das Kontextmenü eines UniPi-Channels importieren.

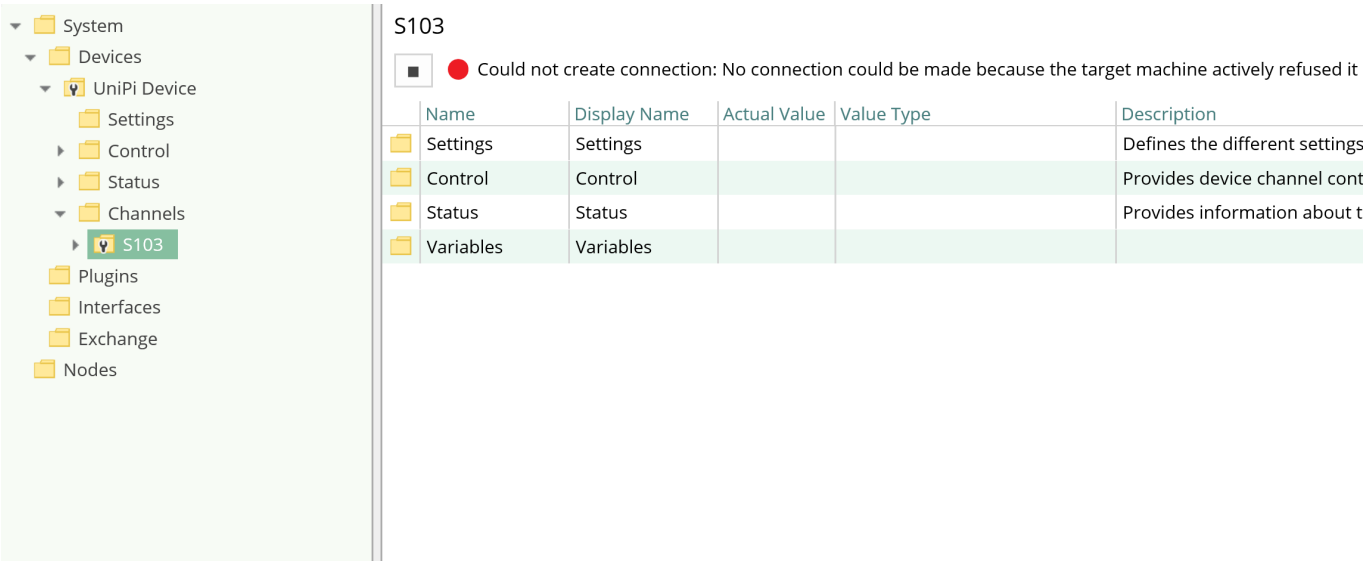
Der Zugriff auf die Daten des UniPi Geräts erfolgt über einen lokalen Modbus TCP Server. Somit werden Variablen analog zum Modbus Device Plugin definiert, im Abschnitt Variablen finden Sie die entsprechende Dokumentation.

Fehlerdiagnose

Das UniPi Device Plugin liefert je nach zu untersuchender Schicht verschiedene Statusinformationen. Generell werden die kanalbasierten Diagnoseinformationen durch den Verbindungsstatus des Channels zum UniPi produziert. Die variablenbasierten Diagnoseinformationen werden während des Lese-/Schreibzugriffs auf die verschiedenen Variablen produziert.

Kanal

Um den Status des UniPi-Kanals zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf das folgende Bild:



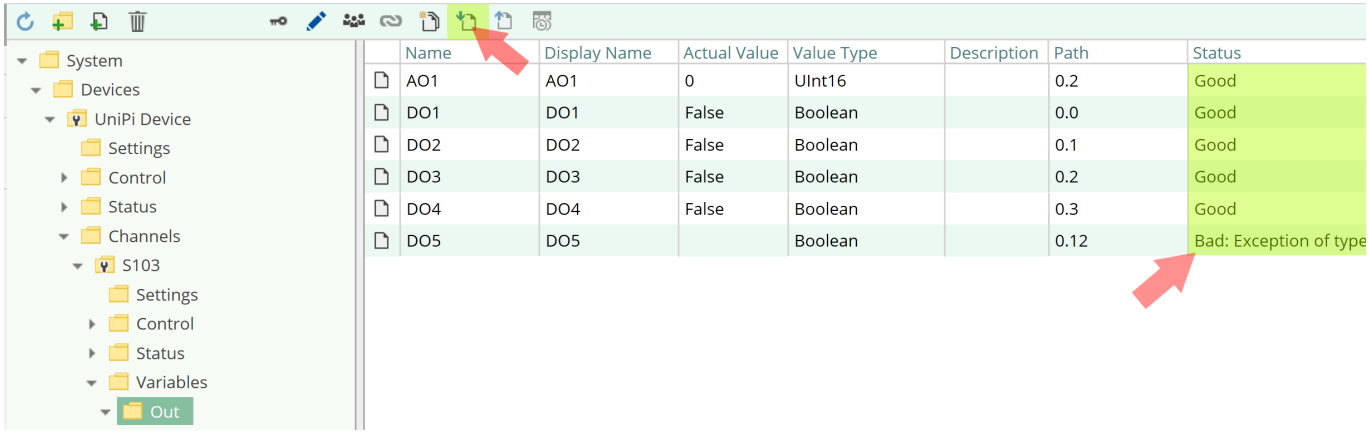
Das obige Bild zeigt das Bedienfeld des UniPi-Kanals, das alle statusrelevanten Informationen anzeigt. Das Bedienfeld aktualisiert automatisch seine Statusinformation, wenn ein neuer Status verfügbar ist.

Statuskreis

Farbe	Bedeutung
⚪	Der Kanal ist gestoppt. Klicken Sie den ►-Button, um ihn zu starten.
🟡	Der Kanal startet oder stoppt gerade oder wartet auf den Verbindungsaufbau.
🟢	Der Kanal läuft und es wurde erfolgreich eine Verbindung hergestellt. Sie können ihn durch Klick auf den ■-Button stoppen.
🔴	Der Kanal läuft, aber die Verbindung ist momentan fehlerhaft. Bitte überprüfen Sie den Statustext für weitere Informationen.

Variablen

Um den Status der verschiedenen Variablen zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf die in UKI-4.0 ® angezeigte `Status`-Eigenschaft der Spalte. Benutzen Sie den Button „Read actual Value“, um die Werte aus des UniPi auszulesen und das Ergebnis in den Variablen zu speichern.



Logdatei

Alle kanalbezogenen Statusinformationen werden auch in die kanalspezifische Logdatei im `[LoggingFolder]` protokolliert. Jede Logdatei wird nach dem Namensschema `UniPi Device.<ChannelName>.log` benannt. Der Inhalt einer solchen Logdatei kann wie folgt aussehen:

```
...
2018-04-11 11:32:37.0 +2: [Error] Error (Severity=High): Code=[-1], Text=[The operation has
timed-out.], Details=[]
...
```

Entities

Wie jedes Device Plugin erweitert das UniPi Device Plugin das UKI-4.0 Device Modell.

Device

Der Device Typ `UniPiDevice` des Plugins definiert auch den `UniPiDeviceChannel` und erweitert somit die grundlegenden UKI-4.0 `Device` und UKI-4.0 `DeviceChannel` Entities. Während das `UniPiDevice` nur eine Konkretisierung des UKI-4.0 `Device` darstellt, erweitert der `UniPiDeviceChannel` den UKI-4.0 `DeviceChannel` mit den UniPi Variable Entities.

Channel

Der Kanal wird von einem Channel Worker behandelt, der eine TCP Socket Verbindung zum lokalen Server herstellt. Zur Fehlerdiagnosezwecken überprüft der Worker automatisch alle 10 Sekunden den Status der TCP Socket Verbindung, um den `Channel` Statuscode und seine Beschreibung zu aktualisieren und Verbindungsfehler zu erkennen.

Der Worker liest standardmäßig keine Werte. Wenn ein Anwender oder Plugin in UKI-4.0 ® einen synchronen Lesevorgang der `Channel` Variablen anfordert (z.B. mit der „Read actual value“-Funktion in der UKI-4.0 Webkonfiguration), liest der Channel Worker diese vom UniPi und schreibt diese in die

entsprechenden UKI-4.0 Nodes.

Ähnlich schreibt der Channel Worker auch die Werte in den UniPi, wenn ein Client oder Plugin Werte in die `Channel` Variablen schreibt.

Damit eine UniPi Variable regelmäßig gelesen wird, können Sie in der Webkonfiguration bei dem Node „History Options“ auf `Yes` stellen (was eine interne Subscription erstellt), oder Sie können zum Beispiel einen OPC UA Client verwenden, der mit dem OPC UA Serverplugin verbunden ist und damit eine Subscription für die UniPi Variablennodes erstellen. In diesen Fällen liest der Channel Worker in regelmäßigen Intervallen die Variablen vom UniPi und schreibt den neuen Wert nach einer Wertänderung automatisch in die entsprechende UKI-4.0 ® Node.

Variable

Unter dem `Variables`-Node können Sie Datenpunktnodes erstellen, die vom UniPi-Gerät gelesen und geschrieben werden.
Die `Value Type`-Eigenschaft muss dabei auf den entsprechenden Variablentyp festgelegt werden (`Boolean`, `Byte`, `Int16`, `UInt16`, `Int32`, `UInt32` und die zugehörigen Array-Typen).

Migration von Version 1.0.0 auf 1.0.1

Ab der Version 1.0.1 gibt es die Möglichkeit andere Datentypen aus den Modbus-Entities zu lesen, als sie über Modbus bereitgestellt werden (`Boolean` in Coils / `UInt16` in Registern). Um diese Funktion abzubilden, wird die Definition des Modbus-Entität einer Variable nicht mehr wie in Version 1.0.0 über den Variablentypen festgelegt, sondern durch den Pfad der Variable definiert. Dies führt zu Inkompatibilität bei Werten, die aus Modbus-Registern gelesen werden.

Die Migration kann durch Import der Standard Channel-Konfiguration im bestehenden UniPi-Channel durchgeführt werden. Sollte dies nicht möglich sein, da dort Änderungen vorgenommen wurden, kann die Migration auch manuell mit Hilfe der Dokumentation des Modbus Device Plugin möglich.

Ordner & Dateien

Ordner

Name	Pfad	Zweck / Verwendung
AssemblyFolder	<UKI-4.0 `InstallDir>/plugins/UniPiDevicePlugin/`	Beinhaltet die Plugin Assembly Datei.
ConfigFolder	`<`UKI-4.0 `DataDir>/plugins/UniPiDevicePlugin/`	Beinhaltet die Plugin Konfigurationsdatei.
LoggingFolder	`<`UKI-4.0 `DataDir>/log/`	Beinhaltet die Plugin Log Dateien.

Dateien

Typ	Pfad	Zweck / Verwendung
Assembly	`[AssemblyFolder]/`UKI-4.0 `.UniPIDevicePlugin.dll`	Die Plugin Assembly Datei.
Logging	`[LoggingFolder]/UniPi Device.<ChannelName>.log`	Die Log Datei.

Versionsinformation

Dieses Dokument

Datum	2019-01-14
Version	2.0

Plugin

Name	UniPi Device Plugin
Node	/System/Devices/UniPi Device
Version	1.0.1

Assembly

Name	UKI-4.0 .UniPiDevicePlugin.dll
Datum	2018-05-28
Version	1.0.1.0

Exchange Plugins

Alle Exchange Plugins nutzen das UKI-4.0 ® Exchange Modell. Jede zur Verfügung gestellte Storage Engine wird über den UKI-4.0 ® Exchange Manager registriert und verwaltet.
Solche Storage Engines können sein:

- Database Management Systeme wie
  - Microsoft SQL Server
  - MySQL
  - Oracle
- Datenbankdateien wie
  - Excel
  - CSV (siehe CSV Exchange Plugin)

Exchange Modell

Das UKI-4.0 ® Exchange Modell erweitert das grundlegende UKI-4.0 ® Entity Modell um für den Exchange typische Entities. Dabei definiert eine Exchange Entity die untergeordneten Entities zur Steuerung (engl. control), für Einstellungen (engl. settings), für den Status des Plugins und die diversen Kanäle (engl. channels), über die das Plugin die unterstützten Storage Engines an UKI-4.0 ® anbindet.

Konfiguration

Jedes mit UKI-4.0 ® ausgelieferte Exchange Plugin lässt sich direkt und ausschließlich in der UKI-4.0 ® Host Anwendung konfigurieren. Verfügt ein Plugin über Konfigurationsparameter, dann können diese in der entsprechenden Exchange Entity modifiziert werden.

UKI-4.0UKI-4.0 UKI-4.0UKI-4.0 ® verwenden

Die gesamte Konfiguration aller Exchange Plugins finden Sie unter dem Nodepfad `/System/Exchange`. Dieser Wurzelnode der Exchange Plugins ermöglicht die vollständige Konfiguration der Exchange Plugins, vorausgesetzt, dass eines der aktiv verwendeten Exchange Plugins eigene Exchange Entities zur Konfiguration bereitstellt.

CSV Exchange Plugin

Allgemein

Das CSV Exchange Plugin stellt einen Datenaustauschmechanismus zwischen UKI-4.0 ® und CSV (comma-separated values) -basierten Dateiformaten bereit.

Was tut das Plugin?

Das CSV Exchange Plugin definiert eine Speicherstruktur für CSV-Dateien. Innerhalb dieser Struktur können ein oder mehrere Datensets definiert werden. Jedes Datenset repräsentiert eine Verbindung zwischen den Spalten im CSV-Speicher und den Nodes in UKI-4.0 ®. Über diese Verbindung liest und schreibt das Plugin die UKI-4.0 ® Nodewerte als Datenset in die CSV-Datei und andersherum.

Funktionen

- Überwachung von Dateiveränderungen
- Dateien nach Beendigung verwerfen
- Dateien nach Beendigung löschen
- Benutzerdefinierte CSV Separatorenkonfiguration
- Relatives Nodeverbindung in DataSet
- Zugriff auf Dateien im lokalen Dateisystem oder über ein SSH-basiertes Transferprotokoll (SCP oder SFTP)

Unterstützte Inhalte

- Microsoft Excel CSV Separator (Standard)
- Benutzerdefinierte CSV Separatoren

Zweck & Anwendung

Zweck

Die konfigurierten CSV-Dateien können einfach zum Lesen und Schreiben von Daten in und aus UKI-4.0 ® verwendet werden. Dies erlaubt einfache Datenimport-/-exportszenarios unter Verwendung von kommaseparierten Werten (comma-separated values = CSV). Auch können andere UKI-4.0 ® Teilnehmer durch das CSV Exchange Plugin in einer CSV-Datei gespeicherte Informationen abliefern und abholen.

Anwendung

- Automatische Datensynchronisation zwischen UKI-4.0 ® und Fremd(unter)systemen zur Datenübertragung
- Datenimport zur Archivierung von Prozessprotokollen
- Datenexport zur nachverarbeitung von UKI-4.0 ®-produierten und -gespeicherten Informationen

Installation

Dieses Plugin ist Bestandteil des UKI-4.0 ® Setups. Bitte konsultieren Sie UKI-4.0 ® Setup und erster Start für weitere Informationen darüber, wie dieses Plugin installiert und deinstalliert werden kann.

Anforderungen

- Standardanforderungen von UKI-4.0 ®
- Lese-/Schreibzugriff auf die konfigurierten CSV-Dateien

Konfiguration

Konfigurationsdatei verwenden

Struktur

Das CSV Exchange Plugin definiert die Wurzel seines Elementenbaums über das `PluginSettings`-Element, wie in Plugin Konfiguration - Verwenden einer Konfigurationsdatei beschrieben, und setzt seinen XML-Baum durch Benutzen des `Servers` Elements fort.

Servers-Element

Das `Servers`-Element dient als Container für ein oder mehrere `Server`-Elemente. Dieses Element verwaltet alle Dateiserver, die dem Exchange Plugin zugeordnet sind.

Das `Servers`-Element kann wie folgt aussehen:

```
<Servers>
  <!-- 0-n Server elements -->
</Servers>
```

Server-Element

Das `Server`-Element dient als Container für das Element `File` sowie für ein oder mehrere `Trigger`-Elemente, falls das `SyncMode`-Attribut auf `SystemToFile` gesetzt ist. Ein Server identifiziert eine CSV-Datei, zu der die Serverkonfiguration gehört. Die Konfiguration beinhaltet auch die Aktionen, die vor, während und nach dem Synchronisationsprozess ausgeführt werden müssen. Diese Information wird daher vom CSV Exchange Plugin genutzt, um CSV-Daten mit UKI-4.0 ® auszutauschen.

Jedes `Server`-Element stellt folgende Liste an Attributen zur Verfügung:

	Verpflichtend	Typ	Zweck
`IsEnabled`	nein	Boolean	Zeigt den Wert `true`, wenn der Datenaustausch für den Dateiserver aktiv ist; andernfalls zeigt es den Wert `false` (Standard: `true`).
`SyncMode`	nein	CSV Sync Mode	Die Richtung, in die die Synchronisierung durchgeführt werden soll.
Zusätzliche Attribute für `SyncMode="FileToSystem"`			
`SyncTrigger`	nein	CSV Sync Trigger	Die Auslösebedingung, die die Synchronisierung einleitet.

	Verpflichtend	Typ	Zweck
`BeforeSyncAction`	nein	CSV Sync Action	Die Aktion, die ausgeführt werden muss, bevor der Synchronisationsprozess beginnt.
`AfterSyncAction`	nein	CSV Sync Action	Die Aktion, die ausgeführt werden muss, nachdem der Synchronisationsprozess geendet hat.
`BeforeSyncMoveFileTo` **(falls** `BeforeSyncAction="MoveFile"`**)**	ja	Node Query Expression	Gibt den neuen vollqualifizierten Datei- oder Verzeichnispfad an, in den die Datei verschoben/umbenannt werden soll. Wenn Sie einen Verzeichnispfad angeben (der mit `/` oder `\` endet), wird der ursprüngliche Dateiname beibehalten, wenn die Datei in das angegebene Verzeichnis verschoben wird. Falls in diesem Fall bereits eine Datei mit dem gleichen Namen existiert, wird an den Dateinamen ein Suffix wie `(1)`, `(2)` usw. angehängt.
`AfterSyncMoveFileTo` **(falls** `AfterSyncAction="MoveFile"`**)**	ja	Node Query Expression	Siehe oben.

CSV Sync Mode

Die folgenden Werte sind gültig für Attribute dieses Typs:

Wert	Beschreibung
`"FileToSystem"`	Die CSV-Datei wird mit den gebundenen UKI-4.0 ® Nodes synchronisiert.
`"SystemToFile"`	Die gebundenen UKI-4.0 ® Nodes werden mit der CSV-Datei synchronisiert. In diesem Fall können Sie über ein oder mehrere `Trigger`-Elemente Trigger definieren, die angeben, wann die Daten in UKI-4.0 gesammelt und in die CSV-Datei geschrieben werden.

CSV Sync Trigger

Die folgenden Werte sind gültig für Attribute dieses Typs:

Wert	Beschreibung
`"FileChanged"`	Der Synchronisationsprozess soll angestoßen werden, sobald die Datei verändert wird. Das bedeutet, dass die Synchronisation jedes Mal dann aufgerufen wird, wenn das Datum sich ändert, an dem die Datei zuletzt geschrieben wurde.

CSV Sync Action

Die folgenden Werte sind gültig für Attribute dieses Typs:

Wert	Beschreibung
`"KeepFile"`	Die Datei soll vor/nach der Synchronisierung behalten werden.
`"TruncateFile"`	Der Inhalt der Datei soll vor/nach der Synchronisierung entfernt werden.
`"DeleteFile"`	Die Datei soll vor/nach der Synchronisierung gelöscht werden.
`"MoveFile"`	Die Datei soll vor/nach der Synchronisierung verschoben oder umbenannt werden. Für diese Aktion muss das `BeforeSyncMoveFileTo`- bzw. `AfterSyncMoveFileTo`-Attribut angegeben werden (siehe oben).

Das `Server`-Element kann wie folgt aussehen:

```
<Server SyncMode="FileToSystem"
        SyncTrigger="FileChanged"
        AfterSyncAction="DeleteFile">
  <!-- 0-n Trigger elements -->
  <!-- File element -->
</Server>
```

Trigger-Element

Das `Trigger`-Element dient (falls `SyncMode="SystemToFile"`) zur Definition eines Triggers, der angibt, wann die Daten von UKI-4.0 ® über einen synchronen Lesevorgang gesammelt und dann in die CSV-Datei geschrieben werden sollen, und stellt folgende Attribute zur Verfügung:

	Verpflichtend	Typ	Zweck
`Type`	**ja**	String	Gibt den Typ des Triggers an. `"ValueChange"`: Der Trigger löst aus, wenn ein neuer Wert in einen bestimmten Node geschrieben wird und der neue Wert sich vom bisherigen Wert des Nodes unterscheidet. `"Edge"`: Der Trigger löst aus, wenn ein bestimmer Wert (angegeben durch das `EdgeValue`-Attribut) in einen bestimmten Node geschrieben wurde (während der vorherige Nodewert ein anderer war). Optional kann ein Wert festgelegt werden, der nach dem synchronen Lesevorgang in den Node zurückgeschrieben werden soll. `"Interval"`: Der Trigger löst in einem regemäßigen Intervall aus, das durch das `interval`-Attribut spezifiziert wird.
`Node` **(falls `Type="Edge"` oder `Type="ValueChange"`)**	**ja**	String	Gibt den vollqualifizierten Pfad zur Node an, die überwacht werden soll.
`EdgeValue` **(falls `Type="Edge"`)**	**ja**	String	Der Wert, auf den der Trigger überprüft. Der Trigger löst aus, wenn dieser Wert auf einen Node geschrieben wird und der bisherige Nodewert ein anderer war.
`ChangeBackValue` **(falls `Type="Edge"`)**	nein	String	Falls angegeben, schreibt das CSV Exchange Plugin diesen Wert in den Node, nachdem der Trigger ausgelöst hat und der Datensatz über einen synchronen Lesevorgang eingesammelt wurde.
`ChangeBackNode` **(falls `Type="Edge"`)**	nein	String	Falls angegeben und falls `ChangeBackValue` angegeben ist, schreibt das CSV Exchange Plugin den Wert in den Node, der durch dieses Attribut angegeben ist, anstatt in den Node, der durch das `Node`-Attribut angegeben ist.
`Interval` **(falls `Type="Interval"`**	**ja**	Int32	Gibt das Trigger-Intervall in Millisekunden an. Beispielsweise bedeutet ein Wert von 2000, dass der Trigger alle zwei Sekunden auslöst.

Das `Trigger`-Element kann wie folgt aussehen:

```
<Trigger Type="Edge" Node="/Nodes/Line1/TriggerNode" EdgeValue="1" ChangeBackValue="0" />
```

File-Element

Das `File`-Element dient als Container für das Element `Bindings` und definiert die Attribute, um die für den Austausch benötigte Datei aufzusetzen. Diese Attribute configurieren die Datei und ihr zum Synchronisieren benutztes Format.

Das `File`-Element stellt folgende Liste an Attributen zur Verfügung:

	Verpflichtend	Typ	Zweck
Path	**ja**	Node Query Expression	Der vollqualifizierte zugreifbare Pfad zur zu synchronisierenden CSV-Datei. Bei Verwendung des SyncMode `SystemToFile` werden die in der Node Query Expression angegebenen Nodes in den synchronen Lesevorgang der Column-Nodes mit eingeschlossen, und dann beim Schreiben der Datei ausgewertet. Bei Verwendung des SyncMode `FileToSystem` werden die angegebenen Nodes beim Starten des Channels einmalig ausgewertet, und zusätzlich können Sie einen Filter wie `XY*.csv` verwenden, wodurch das Plugin alle Dateien, die dem Filter entsprechen, überwacht und verarbeitet. **Beachten Sie:** Für Dateien, auf die über das lokale Dateisystem zugegriffen wird, unterliegt dieser Pfad den Beschränkungen der **File Access Security**, die in den UKI-4.0 ® Einstellungen definiert wurden.
Separator	nein	Char	Der CSV Separator in der Datei, der zum Trennen der Werte voneinander benutzt wird (Standard: `;`).
HasHeader	nein	Bool	Gibt an, ob die erste Zeile in der CSV-Datei Spaltenüberschriften enthält. Gültige Werte: `True`, `False` Bei Angabe von `True` bedeutet dies für den SyncMode `FileToSystem`, dass die erste Zeile der CSV-Datei bei der Synchronisierung übersprungen wird. Für den SyncMode `SystemToFile` bedeutet dies, dass bei der Synchronisierung bei einer leeren CSV-Datei zuerst eine Header-Zeile mit einer Überschrift für jede Spalte geschrieben wird, die in den `<Binding>`-Elementen über das `HeaderName`-Attribut festgelegt werden kann.
Type	nein	String	Legt den Typ des Dateiübertragungsmechanismus fest. `File` (Standard): Auf die Datei wird über das lokale Dateisystem zugegriffen. `Scp`: Auf die Datei wird über einen SSH-Server (SCP-Protokoll) zugegriffen. `Sftp`: Auf die Datei wird über einen SSH-Server (SCP-Protokoll) zugegriffen.
Zusätzliche Attribute, wenn **`Type="Scp"`** oder **`Type="Sftp"`**:			
Hostname	ja	String	Der Hostname des SSH-Servers.
Username	ja	String	Der Benutzername für den SSH-Server.
Password	ja	String	Das **verschlüsselte Passwort** für den SSH-Server. Um das verschlüsselte Passwort zu erhalten, öffnen Sie bitte UKI-4.0 und klicken in der Webkonfiguration auf **Password Security**. Dort können Sie das Originalpasswort eingeben und dieses verschlüsseln, sodass es in dieses Attribut eingefügt werden kann.

Das `File`-Element kann wie folgt aussehen:

```
<File Path="MachineSetup.csv">
  <!-- Bindings element -->
</File>
```

oder beim Benutzen von SCP:

```
<File Path="/directory/MachineSetup.csv" Type="Scp" Hostname="192.168.0.20" Username="user1"
Password="password1">
  <!-- Bindings element -->
</File>
```

Bindings-Element

Das `Bindings`-Element agiert als Container für ein oder mehrere `Binding`-Elemente. Dieses Element verwaltet alle Bindungen (`Bindings`), die der Datei zugeordnet sind.

Das `Bindings`-Element stellt folgende Liste an Attributen zur Verfügung:

	Verpflichtend	Typ	Zweck
`BaseNode`	nein	String	Der Nodepfad, der als Basis für relative Nodepfade der folgenden Node-Bindings verwendet wird.

Das `Bindings`-Element kann wie folgt aussehen:

```
<Bindings BaseNode="/Nodes/Line 1/Tools/CuttingTool">
  <!-- 0-n Binding elements -->
</Bindings>
```

Binding-Element

Das `Binding`-Element bindet eine Spalte in der CSV-Datei an einen UKI-4.0 ® Node. Diese Information wird daher vom CSV Exchange Plugin genutzt, um die kommagetrennten Werte (comma-separated values = CSV) mit UKI-4.0 ® Nodes zu verbinden.

Jedes `Binding`-Element stellt folgende Liste an Attributen zur Verfügung:

	Verpflichtend	Typ	Zweck
`ColumnIndex`	**ja**	Int32	Der null-basierte Index (Index der ersten Spalte ist null) der zu bindenden CSV Spalte.
`Node`	**ja** (falls `SystemValueType` nicht angegeben)	String	Der (relative, falls `BaseNode` benutzt wird) Nodepfad des zu bindenden Nodes.
`ValueType` (falls `Node` angegeben)	nein	Value Type	Falls angegeben, erzwingt die Interpretation des Spaltenwerts als angegebener Typ. Sie können die Formatierung über das `ValueFormat`-Attribut festlegen.
`ValueFormat` (falls `ValueType` angegeben)	nein	String	Gibt die Formatierung für den Wert an.
`SystemValueType`	**ja** (falls `Node` nicht angegeben)	System Value Type	Falls angegeben, wird statt des Werts eines Nodes ein Systemwert für diese Spalte benutzt. Siehe die untere Tabelle für die möglichen Systemwert-Typen.
`SystemValueFormat` (falls `SystemValueType` angegeben)	nein	String	Gibt die Formatierung des Systemwerts an (z.B. Datumsformat).
`HeaderName`	nein	Node Query Expression	Wenn `HasHeader` im `<File>`-Element auf `True` gesetzt ist, gibt dies die Überschrift der Spalte an. Falls nicht angegeben, wird ein Standardname wie „Column1“ verwendet.

Value Type

Die folgenden Werte sind gültig für Attribute dieses Typs:

Wert	Beschreibung
`"DateTime"`	Der Wert der Spalte soll als DateTime-Wert interpretiert werden. Sie können die Formatierung über eine .NET Datums-/Uhrzeit-Formatzeichenfolge (standard oder benutzerdefiniert) festlegen, z.B. `yyyy-MM-dd HH:mm:ss`.
`"TimeSpan"`	Der Wert der Spalte soll als TimeSpan-Wert interpretiert werden. Sie können die Formatierung über eine .NET TimeSpan-Formatzeichenfolge (standard oder benutzerdefiniert) festlegen, z.B. `dd\.hh\:mm\:ss`.

System Value Type

Die folgenden Werte sind gültig für Attribute dieses Typs:

Wert	Beschreibung
`"TriggerTimestamp"` (falls `SyncMode="SystemToFile"`)	In die Spalte wird der Zeitstempel geschrieben, wann der Trigger ausgelöst hat. Sie können die Formatierung über eine .NET Datums-/Uhrzeit-Formatzeichenfolge (standard oder benutzerdefiniert) festlegen, z.B. `yyyy-MM-dd HH:mm:ss`.
`"CreationTimestamp"` (falls `SyncMode="FileToSystem"`)	Der Zeitstempel, der in der Spalte steht, wird als CreationTimestamp für die Nodewerte verwendet, die in UKI-4.0 geschrieben werden sollen. Sie können die Formatierung über eine .NET Datums-/Uhrzeit-Formatzeichenfolge (standard oder benutzerdefiniert) festlegen, z.B. `yyyy-MM-dd HH:mm:ss`.

Das `Binding`-Element kann wie folgt aussehen:

```
<Binding ColumnIndex="0" Node="Depth"/>
<Binding ColumnIndex="1" Node="Last Refresh" ValueType="DateTime" ValueFormat="yyyyMMdd-HHmmss"/>
<Binding ColumnIndex="2" SystemValueType="TriggerTimestamp" SystemValueFormat="yyyyMMdd-HHmmss"/>
```

Node Query Expression

Bestimmte Attribute (z.B. Dateipfad oder Spaltenüberschrift) ermöglichen es, einen Nodepfad anzugeben, was bedeutet, dass dieser Teil durch den Nodewert ersetzt wird. Eine Query Expression hat die Form `${<Nodepfad>|<ID>|<Guid>}` (der Nodepfad selbst darf kein `}`-Zeichen enthalten). Wenn Sie ein Dollarzeichen (`$`) direkt verwenden möchten, müssen Sie dieses als zwei Dollarzeichen (`$$`) escapen.

Wenn eine Expression ausgewertet wird, wird ein synchroner Lesevorgang initiiert. Dies ermöglicht Ihnen, den Wert in einem Script, welches einen UKI-4.0 `.NodeReader` für die entsprechende Node registriert, bereitzustellen.

Beispiele (angenommen Node `/Nodes/A` hat Wert „First“, Node `/Nodes/B` hat Wert „Second“):

Node Query Expression	Resultierende Ausgabe/Dateipfad
`C:\File-${/Nodes/A}.csv`	`C:\File-First.csv`
`$$Header_${/Nodes/A}${/Nodes/B}$$`	`$Header_FirstSecond$`

Benutzung

Es wird empfohlen, einen professionellen XML Editor zur manuellen Bearbeitung der Konfigurationsdatei zu benutzen. Um auch vo der am Ende dieses Artikels erwähnten XML-Schemadefinition zu profitieren, müssen Sie sich auf das Schema beziehen, indem Sie das `xsi:noNamespaceSchemaLocation`-Attribut wie folgt in der Dokumentenwurzel `PluginSettings`anwenden (die XSD-Datei muss neben der XML-Datei platziert sein):

```
<?xml version="1.0" encoding="utf-8" ?>
<PluginSettings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:noNamespaceSchemaLocation="UKI-4.0 .CsvExchangePlugin.xsd">
  <!-- child elements -->
</PluginSettings>
```

Unabhänging davon, ob Sie die Konfigurationsdatei manuell oder automatisiert bearbeiten/erstellen, muss der oben dokumentierte Elementenbaum erfüllt sein, um eine gültige, wohlgeformte und benutzbare Konfigurationsdatei zu erzeugen.

Synchronisation

Sobald das Plugin vom UKI-4.0 ® Plugin Manager geladen und gesartet wird, wird seine Konfigurationsdatei vom Plugin mit den entsprechenden UKI-4.0 ® Entities gelesen und synchronisiert.

Falls sich die Konfigurationsdatei ändert, informiert der Plugin Manager das Plugin. Durch diese Benachrichtigung startet das Plugin neu und benutzt die aktuellsten Konfigurationsänderungen beim Hochfahren.

Beispiel Konfigurationsdatei

UKI-4.0 .CsvExchangePlugin.Settings.xml

```
<PluginSettings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="UKI-4.0 .CsvExchangePlugin.Settings.xsd">
  <Servers>
    <Server SyncMode="FileToSystem"
            SyncTrigger="FileChanged"
            AfterSyncAction="MoveFile"
            AfterSyncMoveFileTo="ProcessedFiles\">
      <File Path="SampleFile*.csv">
        <Bindings BaseNode="/Nodes/Line 1/Tools/CuttingTool">
          <Binding ColumnIndex="0" Node="Depth"/>
          <Binding ColumnIndex="1" Node="Speed"/>
          <Binding ColumnIndex="2" Node="Direction/X"/>
          <Binding ColumnIndex="3" Node="Direction/Y"/>
          <Binding ColumnIndex="4" Node="Date" ValueType="DateTime"
ValueFormat="yyyyMMdd-HHmmss"/>
        </Bindings>
      </File>
    </Server>

    <Server SyncMode="SystemToFile">
      <Trigger Type="Edge" Node="/Nodes/Line 2/Feedback/TriggerNode" EdgeValue="1"
ChangeBackValue="0" />
      <File Path="/home/user/${/Nodes/Line2/CsvFileName}"
            Type="Sftp"
            Hostname="192.168.0.20"
            Username="user1"
            Password="encrypted-password"
            HasHeader="True">
        <Bindings BaseNode="/Nodes/Line 2/Feedback">
          <Binding ColumnIndex="0" Node="CurrentDepth" HeaderName="My Column 1"/>
          <Binding ColumnIndex="1" Node="CurrentSpeed" HeaderName="My Column 2"/>
          <Binding ColumnIndex="2" SystemValueType="TriggerTimestamp"
SystemValueFormat="yyyyMMdd-HHmmss" HeaderName="My Column 3"/>
        </Bindings>
      </File>
    </Server>
  </Servers>
</PluginSettings>
```

Beispiel Konfigurationsschema-Datei

UKI-4.0 .CsvExchangePlugin.Settings.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="UKI-4.0 .CsvExchangePlugin.Settings"
           elementFormDefault="qualified"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="SystemValueTypeType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="CreationTimestamp" />
      <xs:enumeration value="TriggerTimestamp" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="ValueTypeType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Integer" />
```

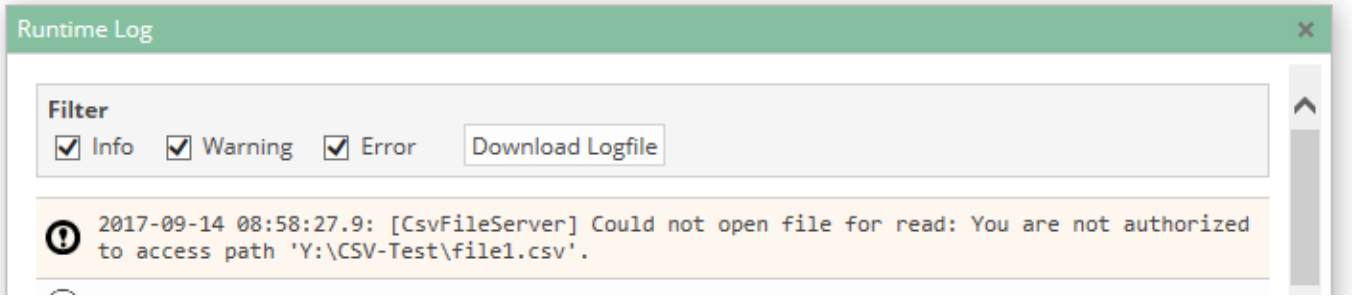
```
      <xs:enumeration value="FloatingPoint" />
      <xs:enumeration value="DateTime" />
      <xs:enumeration value="TimeSpan" />
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="BindingType">
    <xs:attribute name="ColumnIndex" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:integer">
          <xs:minInclusive value="0" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="HeaderName" use="optional">
      <xs:simpleType>
        <xs:restriction base="xs:string" />
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="Node" use="optional">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:minLength value="1" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="SystemValueType" type="SystemValueTypeType" use="optional" />
    <xs:attribute name="SystemValueFormat" type="xs:string" use="optional" />
    <xs:attribute name="ValueType" type="ValueTypeType" use="optional"/>
    <xs:attribute name="ValueFormat" type="xs:string" use="optional" />
  </xs:complexType>

  <xs:complexType name="BindingsType">
    <xs:sequence>
      <xs:element name="Binding" type="BindingType" minOccurs="1"
maxOccurs="unbounded" />
    </xs:sequence>

    <xs:attribute name="BaseNode" type="xs:string" use="optional" />
  </xs:complexType>

  <xs:simpleType name="FileTypeType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="File" />
      <xs:enumeration value="Scp" />
      <xs:enumeration value="Sftp" />
      <xs:enumeration value="file" />
      <xs:enumeration value="scp" />
      <xs:enumeration value="sftp" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="HasHeaderType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="True" />
      <xs:enumeration value="False" />
    </xs:restriction>
  </xs:simpleType>
```

```
  <xs:complexType name="FileType">
    <xs:sequence>
      <xs:element name="Bindings" type="BindingsType" minOccurs="1" maxOccurs="1" />
    </xs:sequence>

    <xs:attribute name="Path" type="xs:string" use="required" />
    <xs:attribute name="Type" type="FileTypeType" use="optional" />
    <xs:attribute name="Hostname" type="xs:string" use="optional" />
    <xs:attribute name="Username" type="xs:string" use="optional" />
    <xs:attribute name="Password" type="xs:string" use="optional" />
    <xs:attribute name="MonitoringInterval" type="xs:integer" use="optional" />

    <xs:attribute name="HasHeader" type="HasHeaderType" use="optional" />

    <xs:attribute name="Separator" use="optional" default=";">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:length value="1" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>

  <xs:simpleType name="TriggerTypeType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Edge" />
      <xs:enumeration value="ValueChange" />
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="TriggerType">
    <xs:attribute name="Type" type="TriggerTypeType" use="required" />
    <xs:attribute name="Node" type="xs:string" use="optional" />
    <xs:attribute name="EdgeValue" type="xs:string" use="optional" />
    <xs:attribute name="ChangeBackValue" type="xs:string" use="optional" />
    <xs:attribute name="ChangeBackNode" type="xs:string" use="optional" />
  </xs:complexType>

  <xs:simpleType name="SyncModeType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="FileToSystem" />
      <xs:enumeration value="SystemToFile" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="SyncTriggerType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="FileChanged" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="SyncActionType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="KeepFile" />
      <xs:enumeration value="TruncateFile" />
      <xs:enumeration value="DeleteFile" />
      <xs:enumeration value="MoveFile" />
    </xs:restriction>
  </xs:simpleType>
```

```
  <xs:complexType name="ServerType" mixed="true">
    <xs:sequence>
      <xs:element name="Trigger" type="TriggerType" minOccurs="0"
maxOccurs="unbounded" />
      <xs:element name="File" type="FileType" minOccurs="1" maxOccurs="1" />
    </xs:sequence>

    <xs:attribute name="IsEnabled" type="xs:boolean" use="optional" />
    <xs:attribute name="SyncMode" type="SyncModeType" use="optional" />
    <xs:attribute name="SyncTrigger" type="SyncTriggerType" use="optional" />
    <xs:attribute name="BeforeSyncAction" type="SyncActionType" use="optional" />
    <xs:attribute name="AfterSyncAction" type="SyncActionType" use="optional" />
    <xs:attribute name="BeforeSyncMoveFileTo" type="xs:string" use="optional" />
    <xs:attribute name="AfterSyncMoveFileTo" type="xs:string" use="optional" />
  </xs:complexType>

  <xs:complexType name="ServersType">
    <xs:sequence>
      <xs:element name="Server" type="ServerType" minOccurs="0" maxOccurs="unbounded"
/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="PluginSettingsType">
    <xs:sequence>
      <xs:element name="Servers" type="ServersType" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>

  <xs:element name="PluginSettings" type="PluginSettingsType" />
</xs:schema>
```

Fehlerdiagnose

Das CSV Exchange Plugin loggt Ereignisse ins UKI-4.0 ® Runtime Log, die mit `[CsvFileServerPlugin]` oder `[CsvFileServer]` anfangen. Sie können das Runtime Log in der Webkonfiguration öffnen, um die letzten Logeinträge anzuzeigen:



Entities

Das CSV Exchange Plugin verwendet das UKI-4.0 ® Entity Model nicht, da es über eine XML-Konfigurationsdatei (UKI-4.0 `.CsvExchangePlugin.Settings.xml`) konfiguriert wird und daher keine Entities zur Verfügung stellt.

Ordner & Dateien

Ordner

Name	Pfad	Zweck/Anwendung
AssemblyFolder	<UKI-4.0 InstallDir>/plugins/CsvExchangePlugin/	Beinhaltet die Plugin Assemblydatei.
ConfigFolder	<UKI-4.0 DataDir>/plugins/CsvExchangePlugin/	Beinhaltet die Plugin Konfigurationsdatei.
LoggingFolder	<UKI-4.0 DataDir>/log/	Beinhaltet die Plugin Logdateien.

Dateien

Typ	Pfad	Zweck/Anwendung
Assembly	[AssemblyFolder]/UKI-4.0 .CsvExchangePlugin.dll	Die Plugin Assemblydatei.
Config File	[ConfigFolder]/UKI-4.0 .CsvExchangePlugin.Settings.xml	Die Konfigurationsdatei.

Versionsinformation

Dieses Dokument

Datum	2019-04-17
Version	1.4

Plugin

Name	CSV File Server
Node	/System/Plugins/Exchange/CSV
Version	1.1.0

Assembly

Name	UKI-4.0 .CsvExchangePlugin.dll
Datum	2019-04-17
Version	1.1.0.0

Database Plugin

Allgemein

Das Database Plugin ermöglicht es Ihnen, UKI-4.0 ® Datenpunktnodewerte in externe Datenbanken wie MySQL oder Microsoft SQL Server zu schreiben.

Was tut das Plugin?

Das Plugin lässt Sie:

- Datenbankverbindungen definieren
- Trigger definieren (z.B. Intervalltrigger)
- DataSets definieren, die bestehen aus
    - Datenpunktnodes
    - Systemwerten

Wenn ein Trigger auslöst, sammelt das Plugin Werte für einen „Datensatz“, indem es einen synchronen Lesevorgang der spezifizierten UKI-4.0 ® Datenpunktnodes durchführt (UKI-4.0 ® fordert das zugrundeliegende Gerät auf, tatsächlich seine Variablen zu lesen) und schreibt dann die Werte des Datensatzes in eine Datenbanktabelle.

Um das Plugin zu benutzen, müssen Sie eine Konfigurationsdatei erstellen.

Funktionen

- Nodewerte sammeln und in eine Datenbank schreiben, wenn ein Trigger auslöst
- Unterstützt verschiedene Triggertypen:
    - Intervalltrigger
    - Flankentrigger (löst aus, wenn der Wert eines Nodes sich in einen festgelegten Wert ändert)
    - Wertänderungstrigger (löst aus, wenn sich der Wert einer Node ändert)

Unterstützte Datenbankmotoren

- MySQL 5.5 oder höher
- Microsoft SQL Server 2008 oder höher

Zweck & Anwendung

- Daten von UKI-4.0 ® auf eine externe Datenbank spiegeln
- Daten von Geräten auf Auslöen eines Triggers einsammeln

Installation

Dieses Plugin ist Bestandteil des UKI-4.0 ® Setups. Bitte konsultieren Sie UKI-4.0 ® Setup und erster Start für weitere Informationen darüber, wie dieses Plugin installiert und deinstalliert werden kann.

Anforderungen

- Die Maschine, die UKI-4.0 ® betreibt, muss Zugriff auf eine der unterstützten Datenbank-Engines haben.

Konfiguration

Dieses Plugin kann nur über die **XML-Konfigurationsdatei** wie unten beschrieben konfiguriert werden. Sie müssen die XML-Konfigurationsdatei („UKI-4.0 .DatabasePlugin.Settings.xml“) im Projektverzeichnis (siehe Ordner & Dateien) anlegen. Wird die Datei geändert während UKI-4.0 läuft, startet das Database Plugin automatisch neu und benutzt die neue Konfigurationsdatei.

Struktur

PluginSettings-Element

Jedes UKI-4.0 ® Plugin definiert die Wurzel seines Elementenbaums über das „PluginSettings“-Element wie folgt:

```
<?xml version="1.0" encoding="utf-8" ?>
<PluginSettings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <!-- Channels element -->
</PluginSettings>
```

Channels-Element

Das `Channels`-Element dient als Container für ein oder mehrere `Channel`-Elemente. Das `Channel`-Element definiert einer Gruppe von Datenbankverbindungen, Triggern und Datensätzen. Für jeden Kanal gibt es einen Worker-Thread, der einen Trigger durch das Sammeln von Nodewerten und anschließendes Schreiben dieser Werte in spezifizierte Datenbanken verarbeitet.

```
<Channels>
  <Channel id="ch1" active="true">
    <!-- DbConnections element -->
    <!-- Triggers element -->
    <!-- DataSets element -->
  </Channel>

  <!-- More <Channel> elements... -->
</Channels>
```

Jedes `Channel`-Element stellt folgende Attribute zur Verfügung:

Attribut	Verpflichtend	Typ	Zweck
`id`	**ja**	String	Identifiziert diesen Kanal eindeutig. Auf diesen Wert kann z.B. in einem der `SystemValue`-Elemente verwiesen werden.
`active`	nein	Boolean	Bestimmt, ob dieser Kanal tatsächlich betrieben wird. Wenn `"false"`, ist der Kanal nicht gestartet, andernfalls schon.

DbConnections-Element

Das `DbConnections`-Element dient als Container für ein oder mehrere `DbConnection`-Elemente.

Das `DbConnection`-Element definiert eine physische Verbindung zu einer Datenbank, inklusive des Tabellennamens. Dies bestimmt, in welche Datenbanktabelle geschrieben wird.

```
<DbConnections>
  <DbConnection id="con1" type="MSSQL"
                hostname="192.168.0.1" port="1234"
                username="myuser" password="pw"
                database="MyDB" table="MyLogTable" />

  <!-- More <DbConnection> elements... -->
</DbConnections>
```

Jedes `DbConnection`-Element stellt folgende Attribute zur Verfügung:

Attribut	Verpflichtend	Type	Purpose
`id`	**yes**	String	Identifiziert diese Datenbankverbindung eindeutig. Dies wird in den `DataSet`-Elementen benutzt, um auf diese Verbindung zu verweisen.
`type`	**yes**	Enumeration	Spezifiziert den Typ der Datenbank-Engine. `"MSSQL"`: Microsoft SQL Server `"MySQL"`: MySQL Server
`hostname`	**yes**	String	Der Hostname, der DNS-Name oder die IP-Adresse des Datenbankservers.
`port`	**yes**	Integer	Der TCP-Port des Datenbankservers. Bei der Verwendung von `MSSQL` können Sie `0` als Port angeben, in welchem Fall eine Verbindung zur standardmäßigen SQL-Server-instanz hergestellt werden soll.
`username`	**yes**	String	Der Benutzername, der zum Herstellen der Datenbankverbindung benutzt wird.
`password`	**yes**	String	Das **verschlüsselte Passwort** für den Benutzernamen. Um das verschlüsselte Passwort zu erhalten, öffnen Sie bitte UKI-4.0 und klicken in der Webkonfiguration auf **Password Security**. Dort können Sie das Originalpasswort eingeben und dieses verschlüsseln, sodass es in dieses Attribut eingefügt werden kann.
`database`	**yes**	String	Der Datenbank-/Schemaname.
`table`	**yes**	String	Der Name der Datenbanktabelle, in die die Werte geschrieben werden sollen.
`connectTimeout`	no	Integer	Der Timeout in Sekunden, der beim Verbinden mit der Datenbank verwendet werden soll. Falls nicht spezifiziert, ist der benutzte Wert 30.
`commandTimeout`	no	Integer	Der Timeout in Sekunden, der beim Ausführen eines Datenbankbefehls verwendet werden soll. Falls nicht spezifiziert, ist der benutzte Wert 60.
`maxPoolSize`	no	Integer	Die maximale Anzahl der gebündelten aktiven Datenbankverbindungen. Falls nicht spezifiziert, ist der benutzte Wert 40.

Beachten Sie: Im Moment schreibt das Database Plugin immer Werte in eine Datenbanktabelle, indem es eine `INSERT`-Anweisung ausführt (bedeutet, dass neue Werte bei jedem Schreibvorgang an die Tabelle angehängt werden).

Triggers-Element

Das `Triggers`-Element dient als Container für ein oder mehrere `Trigger`-Elemente.

Das `Trigger`-Element definiert ein Objekt, das ein Ereignis auslöst. Auf ein Trigger kann in einem DataSet Element verwiesen werden, um festzulegen, wann das Database Plugin Nodewerte einsammeln und in die Datenbank schreiben soll.

```
<Triggers>
  <Trigger id="t1" type="interval" interval="5000" />

  <Trigger id="t2" type="edge" node="/Path/to/TriggerNode" edgeValue="1" changeBackValue="0"
/>

  <Trigger id="t3" type="valueChange" node="/Path/to/TriggerNode" />

  <!-- More Trigger elements... -->
</Triggers>
```

Jedes `Trigger`-Element stellt folgende Attribute zur Verfügung:

Attribut	Verpflichtend	Typ	Zweck
`id`	**ja**	String	Identifiziert eindeutig den Trigger. Dies wird in den `DataSet` Elementen benutzt, um auf diesen Trigger zu verweisen.
`type`	**ja**	Enumeration	Spezifiziert den Typ des Triggers. `"interval"`: Der Trigger wird im eingestellten Intervall regelmäßig ausgelöst. `"edge"`: Der Trigger löst aus, wenn ein bestimmter Wert in den angegebenen Node geschrieben wurde (während der vorherige Node-Wert ein anderer Wert war). `"valueChange"`: Der Trigger löst aus, wenn sich der Wert des angegebenen Nodes ändert.
`interval` (falls `type="interval"`)	**ja**	Integer	Spezifiziert das Triggerintervall in Millisekunden. Beispielsweise bedeutet ein Wert von `2000`, dass der Trigger alle zwei Sekunden auslöst.
`node` (falls `type="edge"` or `type="valueChange"`)	**ja**	String	Spezifiziert den Nodepfad des Nodes, den der Trigger benutzen soll, um Wertänderungen festzustellen.
`edgeValue` (falls `type="edge"`)	**ja**	String	Der Wert, auf den der Trigger überprüft. Der Trigger löst aus, wenn dieser Wert auf einen Node geschrieben wird und der vorherige Nodewert ein anderer war.
`changeBackValue` (falls `type="edge"`)	nein	String	Falls spezifiziert, schreibt das Database Plugin diesen Wert in den Node, nachdem der Trigger ausgelöst hat und der Datensatz über einen synchronen Lesevorgang eingesammelt wurde.
`instantChangeBackValue` (falls `type="edge"`)	nein	String	Falls spezifiziert, schreibt das Database Plugin diesen Wert in den Node, sofort nachdem der Trigger ausgelöst hat (bevor der Datensatz eingesammelt wurde).

DataSets-Element

Das `DataSets`-Element dient als Container für ein oder mehrere `DataSet` Elemente.

Das `DataSet`-Element definiert eine Gruppe von Datenpunktnodes und SystemValues, die zu einem Datensatz eingesammelt werden und dann auf eine oder mehrere Datenbankverbindungen geschrieben werden. In einem DataSet können Sie auf ein oder mehrere `DbConnection`-Elemente (bedeutet, dass ein DataSet auf all diese Datenbankverbindungen geschrieben wird) und einen oder mehrere Trigger (bedeutet, dass die Nodewerte gesammelt und geschrieben werden, wenn einer dieser Trigger auslöst) verweisen.

```
<DataSets>
  <DataSet id="ds1" writeDelay="2000" writeBufSize="10">

    <!-- One or more DbConnection elements referencing a database connection... -->
    <DbConnection id="con1" />

    <!-- One or more Trigger elements referencing a trigger... -->
    <Trigger id="t1" />

    <!-- Nodes element -->
    <!-- SystemValues element--->
    <!-- AfterSyncActions element -->
  </DataSet>

  <!-- More <DataSet> elements... -->
</DataSets>
```

Jedes `DataSet`-Element stellt folgende Attribute zur Verfügung:

Attribut	Verpflichtend	Typ	Zweck
`id`	**ja**	String	Identifiziert dieses DataSet eindeutig.
`writeDelay`	nein	Integer	Spezifiziert die maximale Verzögerung (in ms), nachdem die gepufferten Datensätze auf die Datenbank geschrieben werden.
`writeBufSize`	nein	Integer	Spezifiziert die maximale Anzahl an gepufferten Datensätzen, bis sie auf die Datenbank geschrieben werden.

Für jedes DataSet, nachdem eine Reihe an Werten eingesammelt wurde (ein Datensatz), wird der Datensatz gepuffert, damit effizient gleich mehrere Datensätze in einem geschrieben werden können. Falls entweder das `writeDelay`- oder das `writeBufSize`-Attribut spezifiziert ist (oder beide), werden die gepufferten Datensätze erst in die Datenbank geschrieben, wenn die `writeDelay` Zeit seit dem letzten Schreibvorgang abgelaufen ist oder die Anzahl der gepufferten Datensätze, die durch `writeBufSize`-Attribut bestimmt wird, überschritten wurde. Wenn keines der Attribute spezifiziert ist, werden die Datensätze sofort geschrieben.

Jedes`DbConnection`-Element und `Trigger`-Element stellt folgende Attribute zur Verfügung (im `DataSet`-Element):

Attribut	Verpflichtend	Typ	Zweck
`id`	**ja**	String	Verweist auf eine vorher definierte DbConnection oder einen Trigger.

Nodes-Element

Das `Nodes`-Element dient als Containter for ein oder mehrere `Node`-Elemente.

Das `Node`-Element verweist auf einen UKI-4.0 ® Datenpunknode, entweder einen absoluten oder relativen Nodepfad benutzend. Wenn das `Nodes`-Element einen Pfad in seinem `root`-Attribut spezifiziert, ist der `path` der `Node`-Elemente relativ zum `root`-Pfad der `Nodes`.

Beispiel mit root-Pfad:

```
<Nodes root="/Nodes/Demo-Nodes/">
  <Node path="Temperature" column="ColTemp" />
  <Node path="Pressure" column="ColPressure" />
  <Node path="Pressure" property="timestamp" column="ColPressureTimestamp" />

  <!-- More Node elements... -->
</Nodes>
```

Beispiel ohne root-Pfad:

```
<Nodes>
  <Node path="/Nodes/Demo-Nodes/Temperature" column="ColTemp" />
  <Node path="/Nodes/Demo-Nodes/Pressure" column="ColPressure" />
  <Node path="/Nodes/Demo-Nodes/Pressure" property="timestamp" column="ColPressureTimestamp"
/>

  <!-- More Node elements... -->
</Nodes>
```

Jedes `Nodes`-Element stellt folgende Attribute zur Verfügung:

Attribut	Verpflichtend	Typ	Zweck
`root`	nein	String	Spezifiziert den Pfad zum Parent Node. Falls spezifiziert, ist der Pfad der `Node`-Elemente relativ zu diesem root-Pfad. Andernfalls muss der Pfad der `Node`-Elemente der komplette Pfad sein.

Jedes `Node`-Element stellt folgende Attribute zur Verfügung:

Attribut	Verpflichtend	Typ	Zweck
`path`	**ja**	String	Der relative Pfad von `root` zum Node oder der absolute Pfad, wenn der `root` des `Nodes` Elements nicht spezifiziert ist.
`column`	**ja**	String	Der Name der Datenbanktabellenspalte, in welche der Wert geschrieben werden soll.
`property`	nein (Standardwert: `value`)	String	Die Eigenschaft des zu schreibenden Nodes. `"value"` (default): Der Nodewert. `"timestamp"`: Der Zeitstempel des Nodewerts (UTC). `"timestampLocal"`: Der Zeitstempel des Nodewerts (local time). `"nodeID"`: Der Identifier (lokaler Identifier) des Nodes. `"nodeName"`: Der Name des Nodes. `"nodeDisplayName"`: Der Anzeigename des Nodes. `"nodeUnit"`: Die Einheit des Nodes.

SystemValues

Das `SystemValues`-Element dient als Container für ein oder mehrere `SystemValue`-Elemente.

Das `SystemValue`-Element enthält entweder einen tatsächlichen Wert order verweist auf einen vordefinierten Systemwert, der in die Datenbank geschrieben werden soll.

```
<SystemValues>
  <!-- Example 1: Log the time when the trigger fired to DB column "colTriggerTime" -->
  <SystemValue type="triggerTimestamp" column="colTriggerTime" />

  <!-- Example 2: Log the value "12345" to the column "colLiteralValue": -->
  <SystemValue value="12345" column="colLiteralValue" />
</SystemValues>
```

Jedes `SystemValue`-Element stellt folgende Attribute zur Verfügung:

Attribut	Verpflichtend	Typ	Zweck
`column`	**ja**	String	Der Name der Datenbank-Tabellenspalte, in die der Wert geschrieben werden soll.
`type`	**ja** (falls `value` nicht spezifiziert ist)	Aufzählung	Legt fest, welcher Systemwert benutzt werden soll. `"triggerTimestamp"`: Der Zeitstempel, an dem der Trigger ausgelöst hat (UTC). `"triggerTimestampLocal"`: Der Zeitstempel, an dem der Trigger ausgelöst hat (lokale Zeit). `"channelID"`: Der Identifier des enthaltenden Kanals.
`value`	**ja** (falls `type` nicht spezifiziert ist)	String	Legt einen direkten (tatsächlichen) Wert fest, der geschrieben werden soll.

Beachten Sie: Entweder das `type`- oder das `value`-Attribut muss festgelegt sein, aber nicht beides.

AfterSyncActions-Element

Das `AfterSyncActions`-Element dient als Container für ein oder mehrere `AfterSyncAction`-Elemente.

Das `AfterSyncAction`-Element ermöglicht es, einen Wert auf einen Node zu schreiben, nachdem der Datensatz eingesammelt wurde.

```
<AfterSyncActions>

  <AfterSyncAction type="writeNodeValue" node="/Path/to/Node" value="MyValue" />

</AfterSyncActions>
```

Jedes `AfterSyncAction`-Element stellt folgende Attribute zur Verfügung:

Attribut	Verpflichtend	Typ	Zweck
`type`	**ja**	Aufzählung	Legt den Typ der AfterSyncAction fest. `"writeNodeValue"`: Nachdem der Datensatz eingesammelt wurde, wird der spezifizierte Wert auf den Node geschrieben.
`node`	**ja**	String	Spezifiziert den Nodepfad des Nodes des zu schreibenden Werts fest.
`value`	**ja**	String	Der zu schreibende Wert.

Beispiel Konfigurationsdatei

Nachfolgend finden Sie eine beispielhafte Konfigurationsdatei, die die oben beschriebenen Elemente benutzt:

UKI-4.0 .DatabasePlugin.Settings.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<PluginSettings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Channels>
    <Channel id="ch1" active="true">
      <DbConnections>
        <DbConnection id="con1" type="MSSQL"
                      hostname="192.168.0.1" port="1234"
                      username="myuser" password="pw"
                      database="MyDB" table="MyLogTable" />

      </DbConnections>

      <Triggers>
        <Trigger id="t1" type="interval" interval="5000" />
      </Triggers>

      <DataSets>
        <DataSet id="ds1" writeDelay="2000" writeBufSize="10">

          <DbConnection id="con1" />
          <Trigger id="t1" />

          <Nodes root="/Nodes/Demo-Nodes/">
            <Node path="Temperature" column="ColTemp" />
            <Node path="Pressure" column="ColPressure" />
          </Nodes>

          <AfterSyncActions>
            <AfterSyncAction type="writeNodeValue" node="/Path/to/Node"
value="MyValue" />
          </AfterSyncActions>

        </DataSet>
      </DataSets>
    </Channel>
  </Channels>
</PluginSettings>
```

Fehlerdiagnose

Das Database Plugin schreibt die folgenden Ereignisse in die Logdatei `[LoggingFolder]\logfile.txt` (siehe Pfad der Logdatei):

- Kanäle werden gestartet oder gestoppt (weil UKI-4.0 ® startet oder stoppt oder die Konfigurationsdatei sich geändert hat und das Plugin neustartet).
- Ein Wertesatz wurde erfolgreich in die Datenbank geschrieben (nachdem ein synchroner Lesevorgang durchgeführt wurde).
- Ein Fehler ist aufgetreten, als versucht wurde, einen Wertesatz in die Datenbank zu schreiben.

Entities

Das Database Exchange Plugin verwendet das UKI-4.0 ® Entity Modell nicht, da es über eine XML-

Konfigurationsdatei (UKI-4.0 `.DatabasePlugin.Settings.xml`) konfiguriert wird und daher keine Entities zur Verfügung stellt.

Ordner & Dateien

Ordner

Name	Pfad	Zweck / Anwendung
AssemblyFolder	<UKI-4.0 `InstallDir>/plugins/DatabasePlugin/`	Beinhaltet die Plugin Assemblydatei.
ConfigFolder	<UKI-4.0 `DataDir>/plugins/DatabasePlugin/`	Beinhaltet die Plugin Konfigurationsdatei.
LoggingFolder	<UKI-4.0 `DataDir>/plugins/DatabasePlugin/`	Beinhaltet die Plugin Logdatei.

Files

Typ	Pfad	Zweck / Anwendung
Assembly	`[AssemblyFolder]/`UKI-4.0 `.DatabasePlugin.dll`	Die Plugin Assemblydatei.
Config File	`[ConfigFolder]/`UKI-4.0 `.DatabasePlugin.Settings.xml`	Die Konfigurationsdatei.
Logging	`[LoggingFolder]/logfile.txt`	Die Logdatei.

Versionsinformation

Dieses Dokument

Datum	`2018-06-14`
Version	`1.5`

Plugin

Name	`Database Plugin`
Version	`1.0.10`

Assembly

Name	UKI-4.0 `.DatabasePlugin.dll`
Datum	`2018-06-14`
Version	`1.0.10.0`

SQL Exchange Plugin

Allgemein

Das SQL Exchange Plugin erlaubt es Ihnen, UKI-4.0 ® Datenpunktnodewerte bidirektional mit außenliegenden Datenbanken wie MyQSL, Oracle oder Microsoft SQL Server auszutauschen.

Was tut das Plugin?

Das Plugin lässt Sie:

- Datenbanken (Verbindungen) definieren
- Tabellen browsen oder definieren
- Tabellenspalten browsen oder definieren
- Spalten lesen und schreiben, indem es einen Read oder Write SQL Ausdruck spezifiziert

Funktionen

- Spalten einer Datenbankzeile lesen (ausgewählt von der *Read SQL Expression*)
- Spalten einer Datenbankzeile schreiben, entweder durch Einsetzen einer neuen Zeile oder durch Updaten einer existierenden Zeile (ausgewählt von der *Write SQL Expression*)
- Abbonieren von Spalten einer Datenbankzeile, wenn die entsprechenden UKI-4.0 ® Variablennodes abboniert sind

Unterstützte Datenbankmotoren

- MySQL 5.5 oder höher
- Microsoft SQL Server 2008 oder höher
- Oracle
- SQLite

Zweck & Anwendung

- Eine Tabellenreihe anhängen oder eine existierende Reihe updaten
- Daten aus der neuesten Zeile order einer spezifischen Zeile einer Tabelle lesen
- Subscriptions erstellen, um benachrichtigt zu werden, wenn Daten in der Datenbank verändert werden
- Aufrufen von **Stored Procedures** über Methoden-Nodes in UKI-4.0 (wird derzeit nur für **Oracle**-Datenbanken unterstützt)

Installation

Dieses Plugin ist Bestandteil des UKI-4.0 ® Setups. Bitte konsultieren Sie UKI-4.0 ® Setup und erster Start für weitere Informationen darüber, wie dieses Plugin installiert und deinstalliert werden kann.

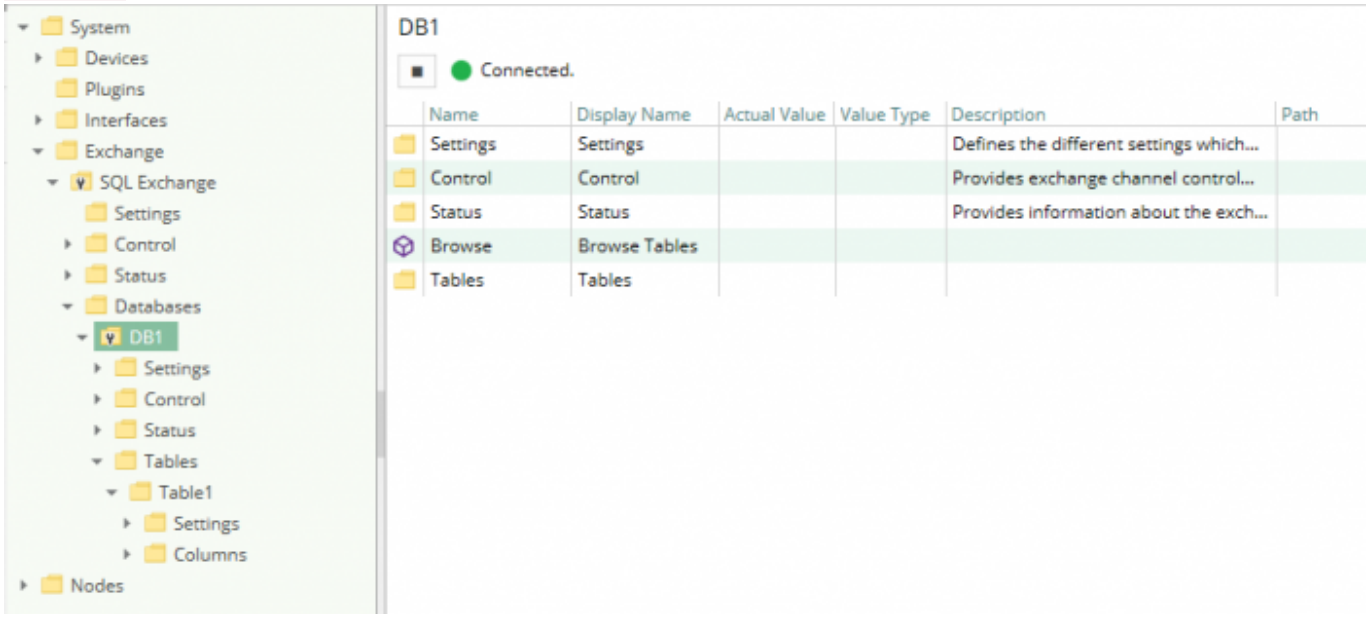
Anforderungen

- Die Maschine, die UKI-4.0 ® betreibt, muss Zugang zu einem der unterstützten Datenbankmotoren haben.

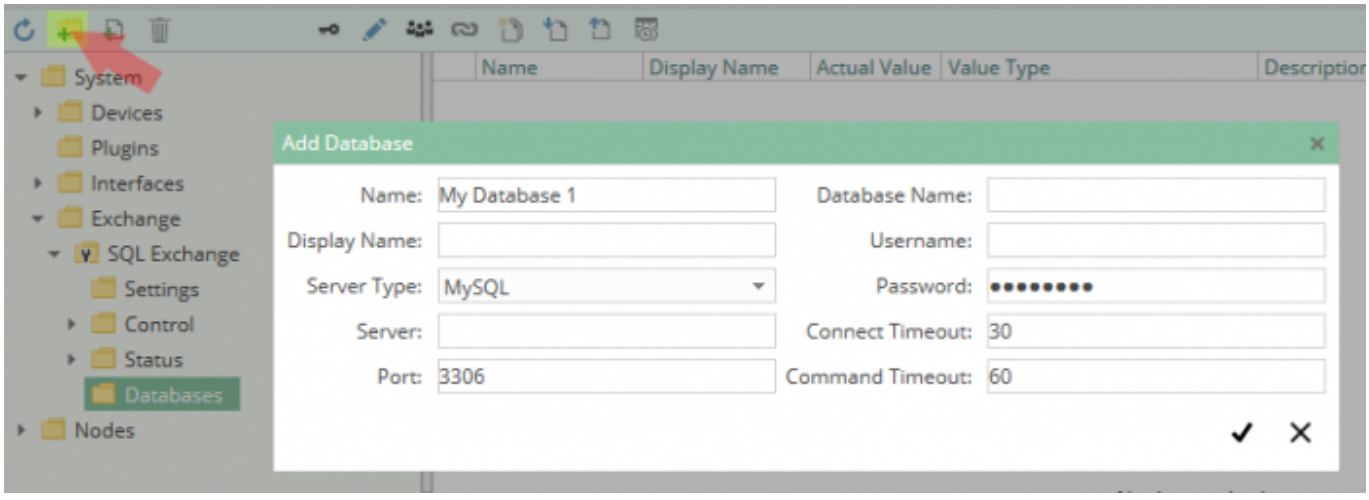
Konfiguration

Übersicht

Die gesamte SQL Exchange Pluginkonfiguration befindet sich unter dem Nodepfad `/System/Exchange/SQL Exchange`.



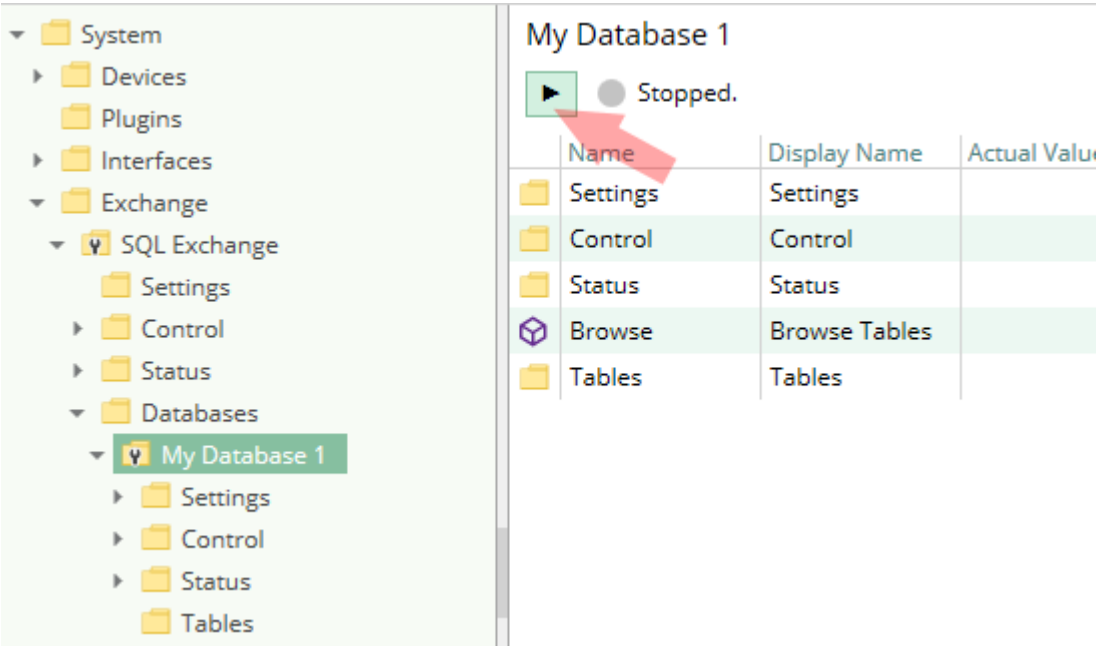
Der Nodebaum im oberen Bild zeigt den Standardnodebaum des SQL Exchange Plugins. Um eine oder mehrere SQL Exchange **Datenbanken** aufzusetzen, fügen Sie einen Folder Node unter dem Node `SQL Exchange/Databases` hinzu, oder machen Sie einen Rechtsklick auf den `SQL Exchange/Databases` Node und wählen Sie `Add Database` aus.



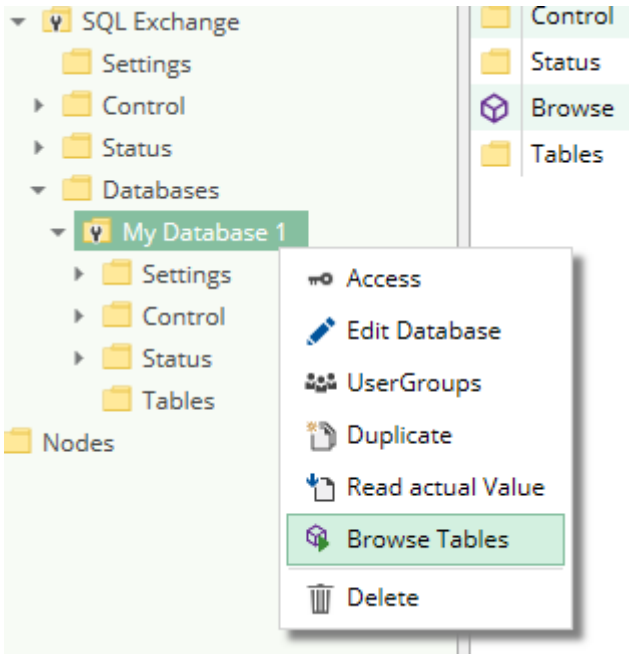
Datenbankspezifische Einstellungen

Name	Typ	Beschreibung
Server Type	Enum	Definiert den Servertyp, zu dem die Datenbankverbindung hergestellt wird. Gültige Werte: `MySQL`, `Oracle`, `MSSQL`, `ODBC`, `SQLite`
Server	String	Der Hostname oder die IP Adresse, zu der die Verbindung hergestellt werden soll. Für **SQLite** ist dies der Dateipfad zur Datenbank; andere Verbindungsproperties wie `Port` werden hierbei ignoriert.
Port	Integer	Der Port, zu dem die Verbindung hergestellt werden soll. Bei der Verwendung von `MSSQL` können Sie `0` als Port angeben, in welchem Fall eine Verbindung zur standardmäßigen SQL-Server-instanz hergestellt werden soll.
Database Name	String	Der Name der Datenbank bzw. des Schemas. Für **ODBC** ist dies der Data Source Name (DSN) und ist das einzige Feld, dass angegeben werden muss. Die DSN kann unter Windows in der App **ODBC-Datenquellen** konfiguriert werden (Benutzer-DSN oder System-DSN). Beachten Sie, dass bei UKI-4.0 (x86) die 32-Bit- und bei UKI-4.0 (x64) die 64-Bit-ODBC-Datenquellen-App verwendet werden muss.
Username	String	Der Benutzername für die Verbindung.
Password	Password	Das Passwort.
Connect Timeout	Integer	Der Timeout in Sekunden, nachdem ein Versuch zu verbinden abgebrochen wird.
Command Timeout	Integer	Der Timeout in Sekunden, nach dem ein anhaltender Befehl abgebrochen wird.

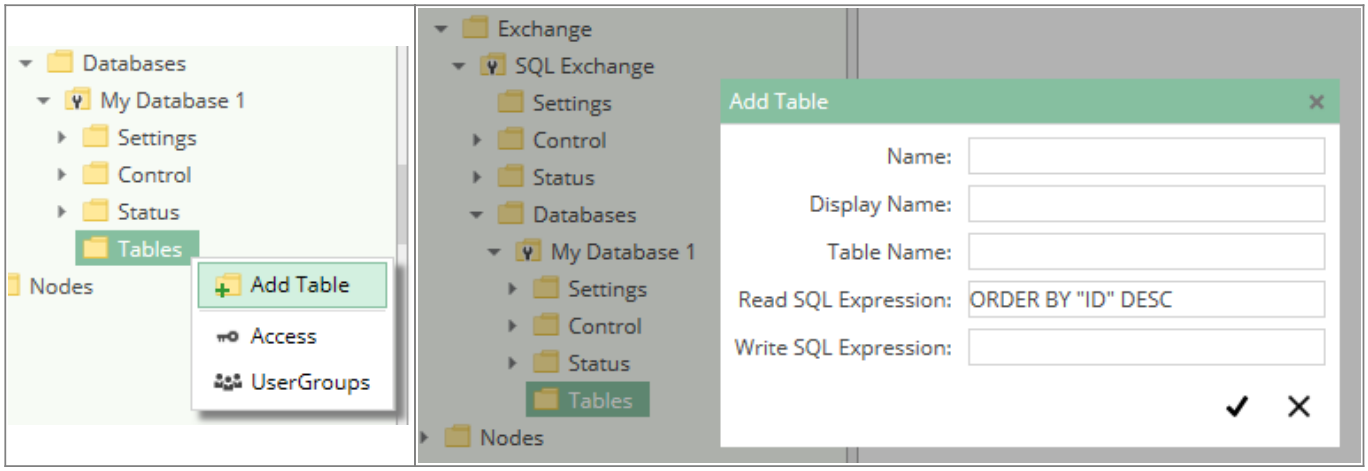
Nachdem Sie „Save“ geklickt haben, wird die Datenbanknode erstellt. Sie können Sie starten, indem Sie die Datenbanknode auswählen und den Startbutton klicken:



Um eine oder mehrere Tabellen (`Tables`) für die Datenbank zu erstellen, können Sie einen Rechtsklick auf die Datenbanknode machen und dann den `Browse Tables` Menüeintrag verwenden:



Alternativ können Sie einen Rechtsklick auf den `Tables` Node machen und `Add Table` auswählen (um eine existierende Tabelle zu bearbeiten, machen Sie einen Rechtsklick auf den `Table` Node und wählen Sie `Edit Table` aus):

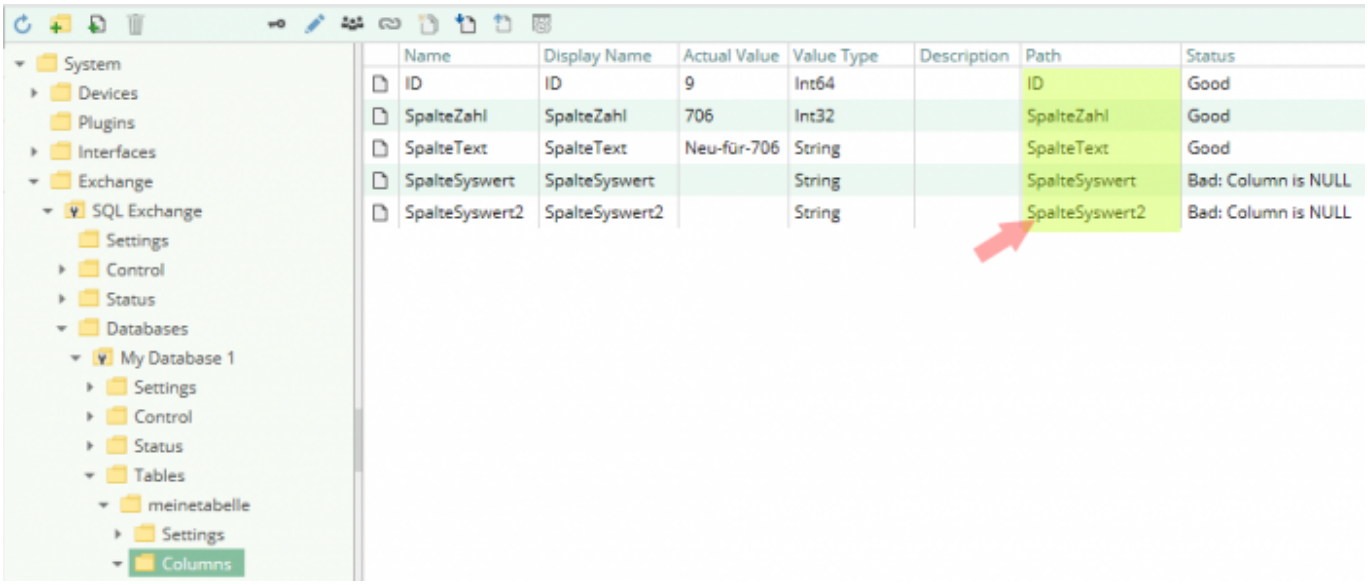


Tabellenspezifische Einstellungen

Name	Typ	Beschreibung
Table Name	String	Der name der Tabelle in der Datenbank.
Read SQL Expression	String	Ein SQL String (z.B. ein `WHERE`-Satz oder ein `ORDER BY`-Satz) der angewandt wird, wenn die Spalten gelesen werden. Die erste Reihe, die von der Datenbank zurückgeliefert wird, wird gelesen. Sie können einen `WHERE`-Satz spezifizieren (z.B. `WHERE "ID" = 123`), um nur eine spezifische Spalte zu lesen. Standard ist `ORDER BY "ID" DESC`, um Reihen absteigend nach ID zu ordnen, damit die Reihe mit der höchsten ID benutzt wird.
Write SQL Expression	String	Ein SQL String (z.B. ein `WHERE`-Satz), der angewandt wird, wenn die Spalten geschrieben werden. Sie können einen `WHERE`-Satz spezifizieren (z.B. `WHERE "ID" = 123`), um eine existierende Reihe zu updaten. Der Standardwert ist ein leerer String, was bedeutet, dass eine neue Reihe jedes Mal dann eingefügt wird, wenn Spaltennodes geschrieben werden.

Columns

Jeder Node innerhalb der `Columns` Node einer `Table` Node kann mithilfe der `Path` Eigenschaft einer Spalte (`Column`) einer Tabelle zugewiesen werden.



Syntax:

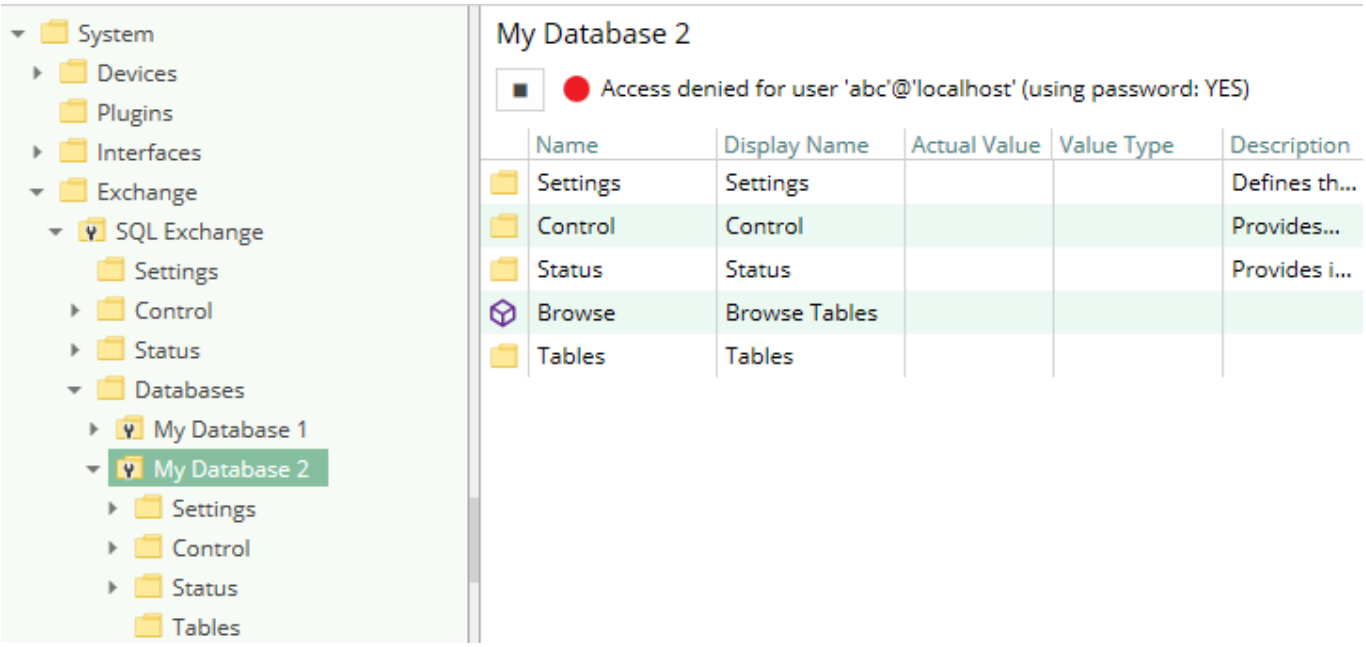
`<ColumnName>`	Nur der Spaltenname ist spezifiziert. Wenn aus ihm gelesen/in ihn geschrieben wird, werden die Read SQL or Write SQL Ausdrücke der Tabelle benutzt. Beispiel: `MyColumn1`
`<ColumnName>;` `<ReadWriteExpression>`	Der Spaltenname ist gemeinsam mit einem Ausdruck spezifiziert, der beim Lesen oder Schreiben aus der/in die Spalte benutzt wird. Beispiel: `MyColumn1; WHERE ID = 123`
`<ColumnName>; <ReadExpression>;` `<WriteExpression>`	Der Spaltenname ist gemeinsam mit einem Ausdruck, der beim Lesen aus der Spalte benutzt wird und einem anderen Ausdruck, der beim Schreiben in die Spalte benutzt wird, spezifiziert. Beispiel: `MyColumn1; WHERE ID = 123; WHERE ID = 456`

Fehlerdiagnose

Das SQL Exchange Plugin liefert je nach zu untersuchender Schicht verschiedene Statusinformationen. Generell werden die kanalbasierten Diagnoseinformationen durch den Verbindungsstatus des Channels zum Datenbankserver produziert. Die variablenbasierten Diagnoseinformationen werden während des Lese- / Schreibzugruffs auf die verschiedenen Spalten produziert.

Datenbank

Um den Status der verschiedenen Datenbankkanäle zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf das folgende Bild:



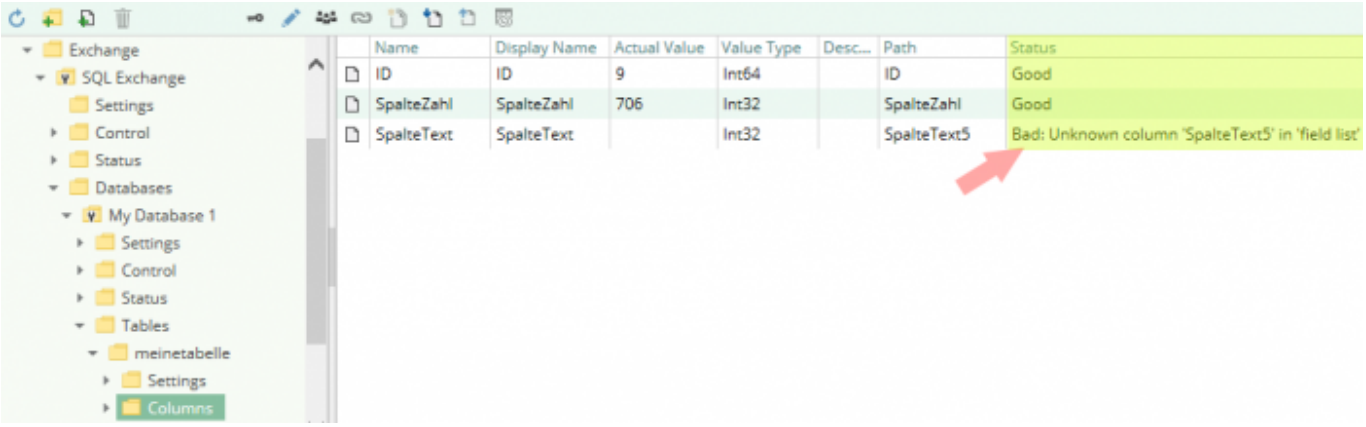
Das obige Bild zeigt das Bedienfeld des Datenbankkanals, das alle statusrelevanten Informationen anzeigt. Das Bedienfeld aktualisiert automatisch seine Statusinformation, wenn ein neuer Status verfügbar ist.

Statuskreis

Farbe	Bedeutung
	Der Datenbankkanal ist gestoppt. Klicken Sie den ► Button, um ihn zu starten.
	Der Datenbankkanal startet oder stoppt gerade oder wartet auf den Verbindungsaufbau.
	Der Datenbankkanal läuft und es wurde erfolgreich eine Verbindung hergestellt. Sie können ihn durch Klick auf den ■ Button stoppen.
	Der Datenbankkanal läuft, aber die Verbindung ist momentan fehlerhaft. Bitte überprüfen Sie den Statustext für weitere Informationen.

Columns

Um den Status der verschiedenen Tabellenspalten zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf die in UKI-4.0 ® angezeigte `Status`-Eigenschaft der Spalte. Benutzen Sie den Button „Read actual Value“, um die Werte aus der Datenbank auszulesen und das Ergebnis in den Spaltennodes zu speichern.



Logdatei

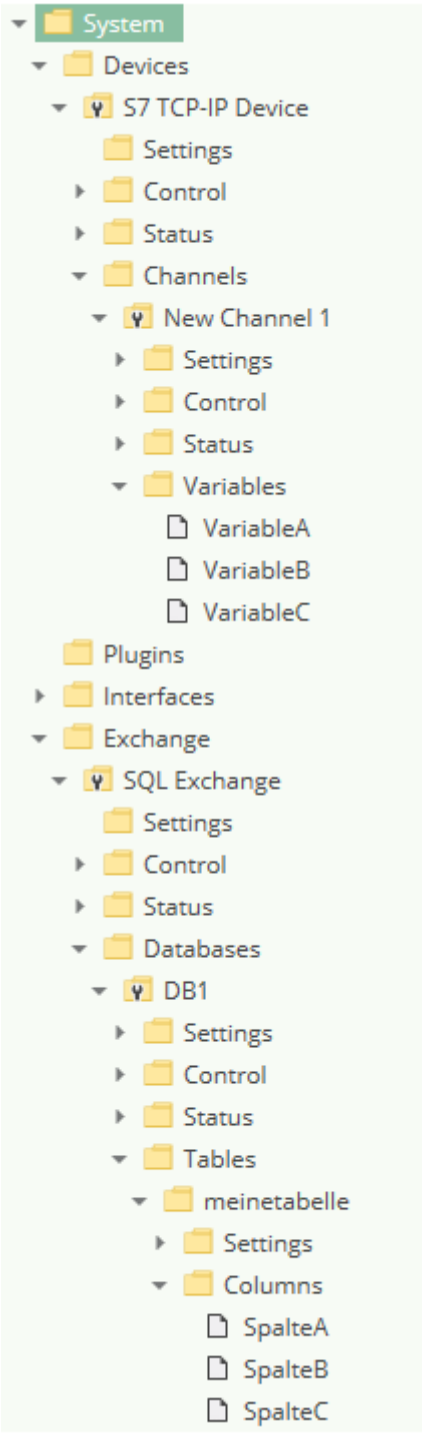
Alle datenbankbezogenen Statusinformationen werden auch in die datenbankspezifische Logdatei im `[LoggingFolder]` protokolliert. Jede Logdatei wird nach dem Namensschema `SQL Exchange.<DatabaseName>.log` benannt. Der Inhalt einer solchen Logdatei kann wie folgt aussehen:

```
...
[14:55:34 25.07.2017] - Error (Severity=High): Code=[-1], Text=[Access denied for user
'abc'@'localhost' (using password: YES)], Details=[]
...
```

Beispiel Variablen in SQL-Datenbank synchronisieren

Unter diesem Beitrag finden Sie ein Beispielscript das zum Synchronisieren von Werten von einer Quelle (z.B. S7 Device Plugin oder OPC UA Client Device) in ein Ziel (z.B. SQL Exchange Plugin) verwendet werden kann.

Das Beispielscript enthält oben eine Liste an Nodes, wo jeweils die Quell-Node und Ziel-Node angegeben ist.
Für das Beispielscript könnten die Nodes im S7-Plugin und im SQL Exchange Plugin so aussehen („VariableA“ aus S7 wird in „SpalteA“ aus SQL geschrieben. „VariableB“ wird in „SpalteB“ geschrieben usw.):



Informationen zum Einrichten des SQL Exchange Plugins für den Zugriff auf eine SQL-Datenbank finden Sie weiter oben beschrieben.

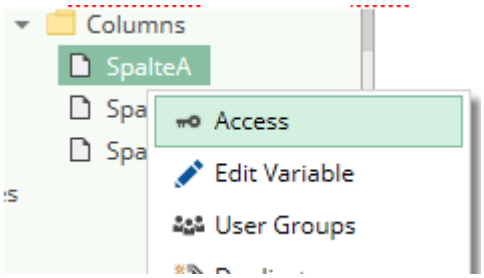
Für Ihr UKI-4.0 müssen Sie dann die Nodeliste im Script (Zeile 6-10) entsprechend anpassen, so dass diese nach diesesm Muster aussieht:

```
    const syncNodePaths: [string, string][] = [
        "/Pfad/zur/QuellNode_1", "/Pfad/zur/ZielNode_1"],
        "/Pfad/zur/QuellNode_2", "/Pfad/zur/ZielNode_2"],
    ];
```

sodass es immer eine Quelle und ein Ziel gibt.

Hier ist die Quelle eine SPS-Variable und das Ziel eine Datenbank-Spalte, dies kann aber auch umgekehrt erfolgen, sodass z.B. aus einer Datenbankzeile Werte gelesen und in die SPS geschrieben werden.

Den kompletten Node-Path einer Node bekommen Sie, wenn Sie diese auswählen und dann rechtsklicken und auf „Access“ gehen:



Trigger

Für das Auslösen der Synchronisierung (d.h. das Übertragen der Werte von den Quell-Nodes in die Ziel-Nodes) gibt es verschiedene Möglichkeiten.

Eine einfache Möglichkeit ist, dieses in einem regelmäßigen Intervall auszulösen. Dies ist in dem Beispielscript aktuell implementiert (siehe Zeile 29-37, „Option A“).
Hier wird in einer Schleife immer 2 Sekunden gewartet und dann durch den Aufruf von „await synchronizeNodesAsync(syncNodes)“ die Synchronisierung gestartet.

Eine weitere Möglichkeit ist, eine separate Trigger-Node zu verwenden und die Synchronisierung nur dann auszulösen, wenn die Triggernode einen bestimmten Wert annimmt (z.B. „1“).
Dies ist im Script als „Option B“ in den Zeilen 40-48 beschrieben. Um diese Option zu verwenden, müssten Sie den Code von Option A auskommentieren und den Code von Option B aktivieren. Dieser Code erstellt eine Subscription, wodurch der Wert in einem bestimmten Intervall (z.B. 500 ms) gelesen wird und bei einer Wertänderung dann das ValueChanged-Event ausgelöst wird.

Hierzu müssen Sie dann den Pfad zur Trigger-Node angeben. Aktuell würde dann immer, wenn sich der Wert der Trigger-Node ändert, die Synchronisierung gestartet werden.

Soll die Synchronisierung nur ausgelöst werden, wenn die Trigger-Node einen bestimmten Wert wie „1“ annimmt, können Sie innerhalb des ValueChanged-EventListeners den aktuellen Wert der Trigger-Node abfragen, z.B.:

```
    let triggerNode = UKI-4.0 .findNode("path/to/TriggerNode", true);
    triggerNode.addValueChangedEventListener(e => {
        if (e.newValue.value == 1)
            runtime.handleAsync(synchronizeNodesAsync(syncNodes));
    });
    // Read the Trigger node value every 500 ms.
    UKI-4.0 .subscribeNodes([triggerNode], 500);
```

Um dieses Script zu verwenden, gehen Sie bitte im UKI-4.0 -Fenster auf den Menüpunkt „Script Interface“ und legen dort ein neues Script an. Wenn Sie auf dieses anschließend rechtsklicken, können Sie im Kontextmenü den Script-Editor öffnen (dies geht auch über einen Klick auf den Button in der Symbolleiste).

Nun löschen Sie bitte den vorhandenen Template-Code und fügen dann den Code aus der Textdatei ein. Wenn das Script dann mit dem aktuellen Code aktiv werden soll, markieren Sie die Checkbox „Go Live“ und klicken dann auf den Speichern-Button.

Beispielscript

SampleScriptSyncNodes.ts

```
runtime.handleAsync(async function () {

    // A list of node pairs with the source node and destination node, which should be
synchronized.
    const sourceNodePath = "/System/Devices/S7 TCP-IP Device/Channels/New Channel
1/Variables/";
    const destNodePath = "/System/Exchange/SQL
Exchange/Channels/DB1/Tables/meinetabelle/Columns/";
    const syncNodePaths: [string, string][] = [
        [sourceNodePath + "VariableA", destNodePath + "SpalteA"],
        [sourceNodePath + "VariableB", destNodePath + "SpalteB"],
        [sourceNodePath + "VariableC", destNodePath + "SpalteC"]
    ];

    // Find the specified nodes.
    const syncNodes: [UKI-4.0.Node, UKI-4.0.Node][] = [];
    for (let syncNodePath of syncNodePaths) {
        let sourceNode = UKI-4.0.findNode(syncNodePath[0], true);
        let destNode = UKI-4.0.findNode(syncNodePath[1], true);

        syncNodes.push([sourceNode, destNode]);
    }


    // Option A:
    // Run a loop where we synchronize the source nodes into the destination nodes
every 2000 ms.
    while (true) {
        // Synchronize the nodes
        await synchronizeNodesAsync(syncNodes);

        // Wait 2000 ms
        await timer.delayAsync(2000);
    }

    // Option B:
    // Synchronize the nodes only when the value of the trigger node has changed.
    // For this, we create a subscription specifying the interval, and then use the
value changed event.
    // let triggerNode = UKI-4.0.findNode("path/to/TriggerNode", true);
    // triggerNode.addValueChangedEventListener(e => {
    //     UKI-4.0.scheduleCallback(() =>
runtime.handleAsync(synchronizeNodesAsync(syncNodes)));
    // }, true);
    // // Read the Trigger node value every 500 ms.
    // UKI-4.0.subscribeNodes([triggerNode], 500);



    async function synchronizeNodesAsync(syncNodes: [UKI-4.0.Node, UKI-4.0.Node][]) {
        // Synchronize the source nodes into the destination nodes.
        let readRequest = syncNodes.map((value => value[0]));
        logger.log("Reading " + readRequest.length + " node values...");
        let values = await UKI-4.0.readNodeValuesAsync(readRequest);

        // Check if every node has a value and a "Good" status, and build the write
request.
        let nodesToWrite: {
            node: UKI-4.0.Node,
```

```
            value: UKI-4.0.NodeValue
        }[] = [];

        for (let i = 0; i < values.length; i++) {
            if (values[i] == null)
                logger.logWarning("Node '" + readRequest[i] + "' doesn't have a value;
please check the device!");
            else if (values[i]!.status.isBad)
                logger.logWarning("Node '" + readRequest[i] + "' has status '" +
values[i]!.status.statusText + "'; please check the device!");

            nodesToWrite.push({
                node: syncNodes[i][1],
                value: values[i] || new UKI-4.0.NodeValue(null)
            });
        }

        // Write the values.
        logger.log("Writing node values...");
        await UKI-4.0.writeNodeValuesAsync(nodesToWrite);
    }

} ());
```

Entities

Wie jedes Exchange Plugin erweitert das SQL Exchange Plugin das grundlegende UKI-4.0 Exchange Modell.

Exchange

Der Exchangetyp `SqlExchange` des Plugins definiert auch den `SqlExchangeChannel` und erweitert somit die grundlegenden UKI-4.0 `Exchange` und UKI-4.0 `ExchangeChannel` Entities. Während der `SqlExchange` lediglich eine Konkretisierung des UKI-4.0 `Exchange` repräsentiert, erweitert der `SqlExchangeChannel` den UKI-4.0 `ExchangeChannel` mit SQL Tabellenentities.

Channel

Jeder Channel wird von einem Channel Worker behandelt, der eine physische Verbindung zur Datenbank herstellt. Zum Zweck der Fehlerdiagnose untersucht der Worker die Datenbankverbindung alle 10 Sekunden, um den Statuscode des `Channel`s und die Beschreibung zu aktualisieren, damit Verbindungsausfälle aufgespürt werden.

Standardmäßig liest der Worker keine Werte. Wenn ein Client oder Plugin einen synchronen Lesevorgang des `Channel`s anfordert, liest der Channel Worker die Variablen in UKI-4.0 (z.B. unter Verwendung der UKI-4.0 Webkonfigurations-Funktion „Read actual value“) aus der Datenbank und schreibt sie dann in die entsprechenden UKI-4.0 Nodes.

Ähnlich schreibt der Channel Worker die Werte in eine Datenbank, die ein Client oder Plugin in die Variablen des Channels schreibt.

Um eine Datenbankvariable stetig gelesen zu bekommen, müssen Sie den Node in der Webkonfiguration bearbeiten und „History Options“ auf `Yes` stellen (was eine interne Subscription erstellt), oder Sie können z.B. einen OPC UA Client verbunden mit einem OPC UA Server Plugin benutzen und eine Subscription für die S7 Variablennodes anlegen. In diesem Fall liest der Channel Worker die Variablen in einem

gleichmäßigen Intervall aus der Datenbank und, falls der Wert einer der Variablen sich verändert hat, schreibt den neuen Wert in den entsprechenden UKI-4.0 ® Node.

Tabelle

Jede Tabellenentity repräsentiert die Informationen, die benötigt werden, um auf eine Datenbanktabelle zuzugreifen. Auf Tabellenlevel können Sie spezifizieren, wie die Zeile ausgewählt wird, aus der die Werte gelesen und in die die Werte geschrieben werden.

Spalte

Jede Spaltenentity repräsentiert eine einzige Datenbanktabellenspalte.

Ordner & Dateien

Ordner

Inhalt	Pfad	Zweck / Verwendung
AssemblyFolder	<UKI-4.0 InstallDir>/plugins/SqlExchangePlugin/	Beinhaltet die Plugin Assembly Datei.
ConfigFolder	<UKI-4.0 DataDir>/plugins/SqlExchangePlugin/	Beinhaltet die Plugin Konfigurationsdatei.
LoggingFolder	<UKI-4.0 DataDir>/log/	Beinhaltet die Plugin Log Dateien.

Dateien

Typ	Pfad	Zweck / Verwendung
Assembly	[AssemblyFolder]/UKI-4.0 .SqlExchangePlugin.dll	Die Plugin Assembly Datei.
Logging	[LoggingFolder]/SQL Exchange.<DatabaseName>.log	Die Log Datei.

Versionsinformation

Dieses Dokument

Datum	2018-06-14
Version	1.1

Plugin

Name	SQL Exchange Plugin
Node	/System/Exchange/SQL Exchange
Version	1.0.5

Assembly

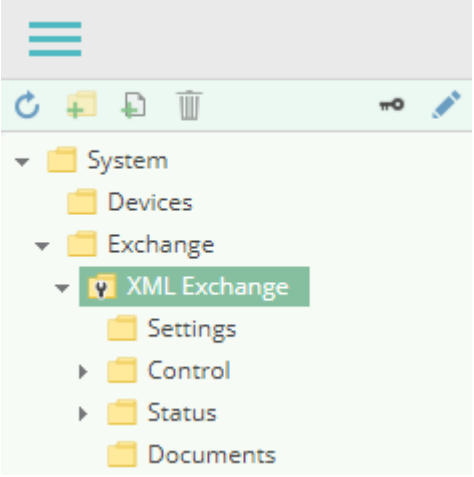
Name	UKI-4.0 .SqlExchangePlugin.dll
Datum	2019-02-04
Version	1.0.5.0

XML Exchange Plugin

Das XML Exchange Plugin ermöglicht das Lesen und Schreiben von Daten (Attribute und Textknoten) in XML-Dateien (mit fester Struktur).

Konfiguration

Die gesamte XML Exchange Plugin-Konfiguration befindet sich unter dem Nodepfad `/System/Exchange/XML Exchange`.



Document

Ein XML Exchange Document (ähnlich eines **Channels** bei Device-Plugins) repräsentiert eine XML-Datei.

Settings

File Path

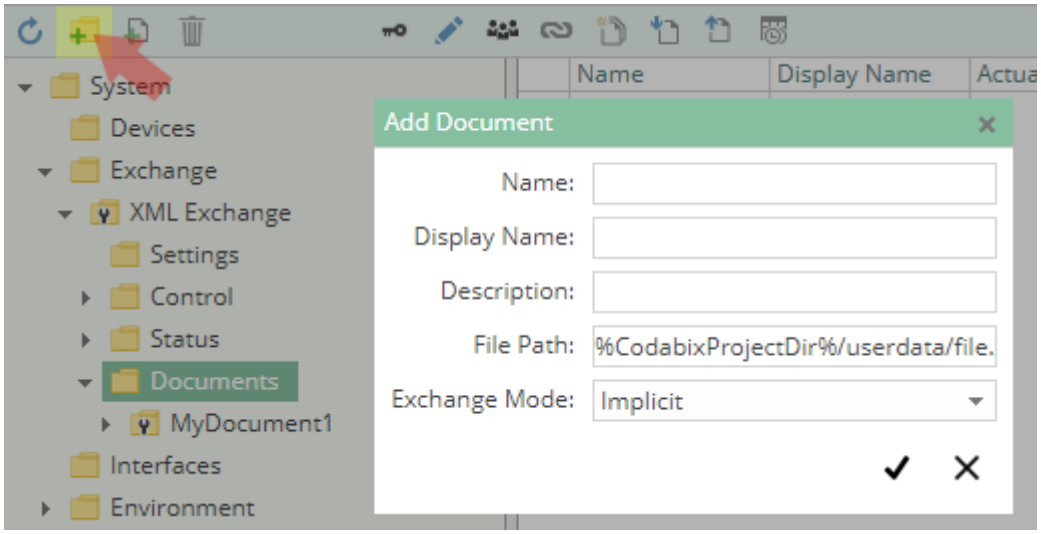
Der Pfad zur XML-Datei im Dateisystem.

Exchange Mode

Gibt an, wie auf die XML-Datei zugegriffen werden soll:

- **Implicit**: Die XML-Datei wird eingelesen, sobald eine oder mehrere Variablen gelesen werden; und die Datei wird geschrieben, sobald eine oder mehrere Variablen geschrieben werden.
- **Explicit**: Beim Lesen oder Schreiben von Variablen findet kein Dateizugriff statt. Die Datei muss explizit gelesen oder geschrieben werden, indem die `Load`- oder `Save`-Methode aufgerufen wird, um den Inhalt der Datei in die Variablen zu übertragen und umgekehrt.

Hinzufügen eines Documents (Channels)



Um ein neues **XML Document** (entspricht einem **Channel** bei Device Plugins) zu erstellen, gehen Sie wie folgt vor:

1. Fügen Sie einen Folder Node unter dem Node `XML Exchange/Documents` hinzu, oder machen Sie einen Rechtsklick auf den `XML Exchange/Documents`-Node und wählen Sie `Add Document` aus.
2. Tragen Sie im `Add Document`-Dialog die Settings für die XML-Datei ein.
3. Nachdem Sie „Save“ geklickt haben, wird die Document-Node erstellt.
4. Sie können den Kanal starten, indem Sie die Document-Node auswählen und den Startbutton klicken.

Variablen

Unter dem `DocumentElement`-Node können Sie Variablen (Nodes) erstellen, die den Aufbau des XML-Dokuments abbilden (d.h. Baumstruktur der UKI-4.0 -Variablen entspricht der Struktur der XML-Knoten):

- Ein Folder-Node-Variable bildet ein **XML-Element** ab, wobei die `Path`-Eigenschaft den Namen des Elements enthält; oder, falls dieser leer ist, der Name der Variablen als Name des XML-Elements verwendet wird.
- Ein Datapoint-Node-Variable bildet ein **XML-Attribut** oder **-Textknoten** ab, das als `String` gelesen und geschrieben werden kann. Die `Path`-Eigenschaft kann folgende Ausdrücke enthalten:
  - `text()`: Die Variable repräsentiert einen **XML-Textknoten**.
  - Beginnt **nicht** mit `/` (z.B.: `abc`): Die Variable repräsentiert ein **XML-Attribut** des übergeordneten XML-Elements mit dem Namen, der angegeben wurde (im Beispiel `"abc"`).
  - Beginnt mit `/` (z.B.: `/*/A/B/text()[2]`): Die Variable repräsentiert eine **XPath-Abfrage**, die als Ergebnis entweder einen Text zurückliefert (dann kann in die Variable nicht geschrieben werden), oder ein Node Set, von dem der erste Knoten verwendet wird (muss ein Text-, Attribut- oder Kommentar-Knoten sein).
    Beachten Sie: Bei Verwendung einer XPath-Abfrage wird die Position der Variable inerhalb des Nodes-Baums nicht berücksichtigt.

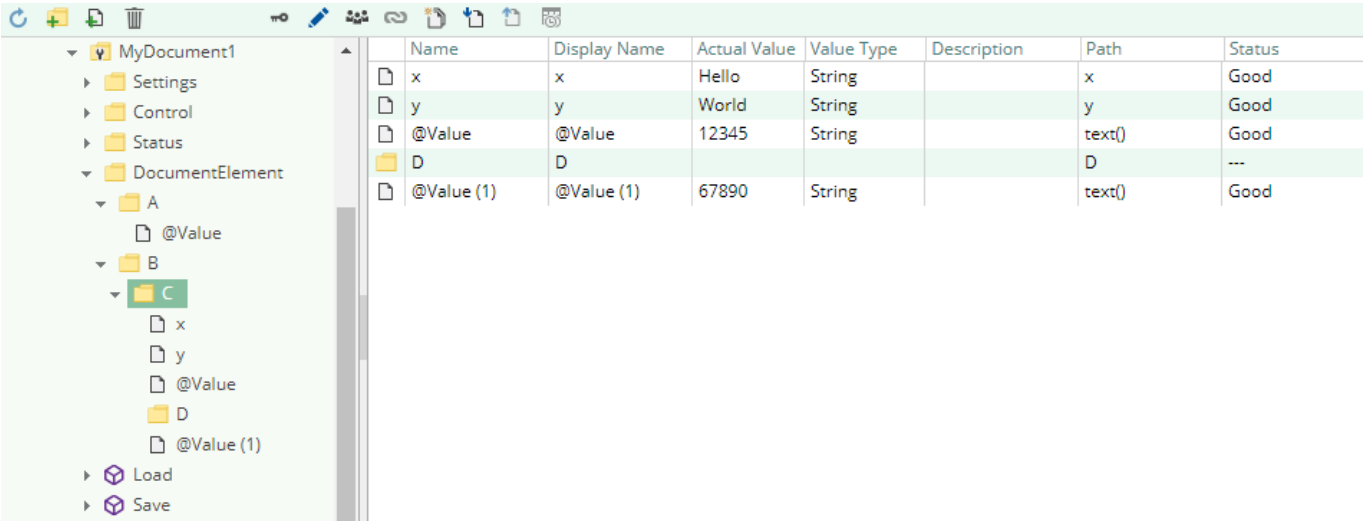
Bei einer vorhandenen XML-Datei können Sie das Document (Channel) browsen, um die UKI-4.0 -Variablen für die XML-Dokumentenstruktur automatisch anzulegen.

Beispiel

example.xml

```
<Root>
  <A>Test</A>
  <B>
    <C x="Hello" y="World">
      12345
      <D />
      67890
    </C>
  </B>
</Root>
```

Nachdem Browsen des Documents wird folgende Nodestruktur in UKI-4.0 angelegt:



Lesen/Schreiben

Sowohl beim Lesen und Schreiben werden UKI-4.0 -Variablen anhand deren Position innerhalb des Nodebaums zu XML-Knoten zugeordnet. Falls beim Schreiben von Werten Variablen in UKI-4.0 existieren, diese aber nicht in der XML-Datei gefunden werden können (gilt beispielsweise auch, wenn die XML-Datei noch nicht existiert), werden die entsprechenden Knoten in der XML-Datei angelegt.

Leseverhalten (abhängig vom eingestellten **Exchange Mode** in den Settings).

- Exchange Mode **„Implicit“**:
  - Synchroner Lesevorgang von einer oder mehreren Variablen:
    - Die XML-Datei wird komplett eingelesen.
    - Die XML-Knoten werden den existierenden UKI-4.0 -Variablen über die Baumstruktur zugeordnet.
    - Es werden nur Werte in diejenigen UKI-4.0 -Variablen geschrieben, die Teil des synchronen Lesevorgangs waren.
      - Falls das Lesen der XML-Datei fehlschlägt, wird ein Wert mit dem Status `Bad` in die Variablen geschrieben.
  - Aufruf der `Load`-Methode:
    - Es wird ein Fehler ausgelöst, da die Methode nur im Modus **Explicit** verwendet werden kann.
- Exchange Mode **„Explicit“**:
  - Synchroner Lesevorgang von einer oder mehreren Variablen:
    - Es findet kein Dateizugriff statt; als Ergebnis des synchronen Lesevorgangs werden die aktuellen Werte der gelesenen Variablen zurückgegeben.

- Aufruf der `Load`-Methode:
  - Die XML-Datei wird komplett eingelesen.
  - Die XML-Knoten werden den UKI-4.0 -Variablen über die Baumstruktur zugeordnet.
  - Es werden in alle UKI-4.0 -Variablen die Werte geschrieben, die aus den zugeordneten XML-Knoten gelesen wurden.
    - Falls das Lesen der XML-Datei fehlschlägt, wird nichts in die Variablen geschrieben, sondern die Methode löst einen Fehler aus.

Schreibverhalten:

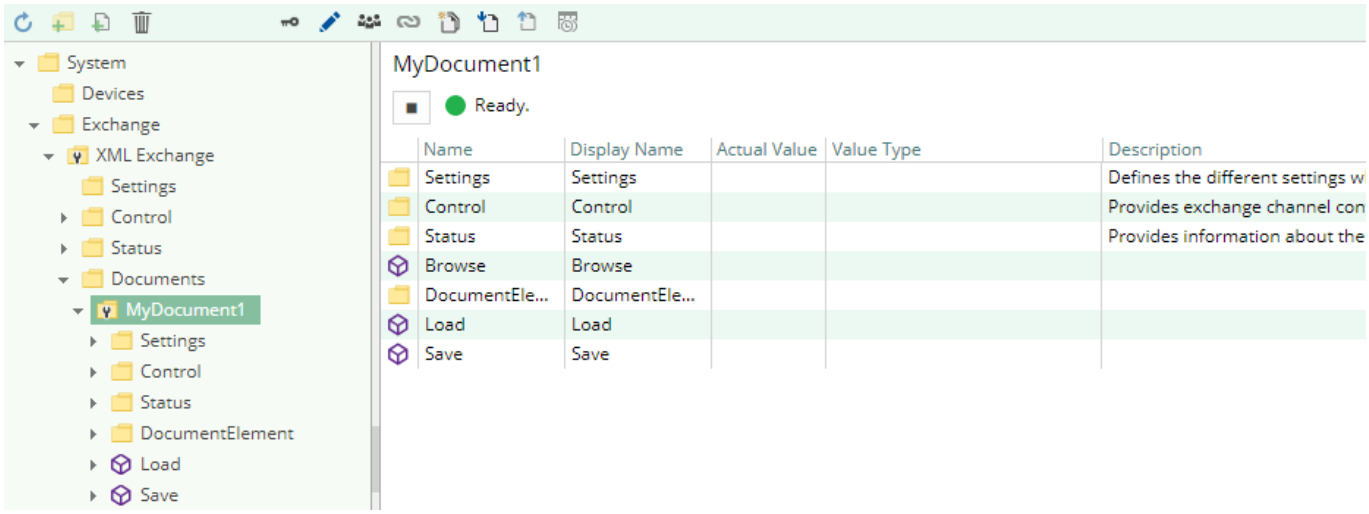
- Exchange Mode **„Implicit“**:
  - Schreiben von Werten in eine oder mehrere Variablen:
    - Die XML-Datei wird komplett eingelesen.
    - Die XML-Knoten werden den existierenden UKI-4.0 -Variablen über die Baumstruktur zugeordnet.
      Falls für eine existierende UKI-4.0 -Variable kein XML-Knoten gefunden wird, wird einer in der XML-Datei erzeugt (auch wenn die Variable nicht Teil des Schreibvorgangs ist), jedoch zunächst ohne Wert (Text).
    - Es werden nur Werte in diejenigen XML-Knoten eingefügt, deren zugeordnete UKI-4.0 -Variablen Teil der Schreibvorgangs waren.
    - Die XML-Datei wird geschrieben.
  - Aufruf der `Save`-Methode:
    - Es wird ein Fehler ausgelöst, da die Methode nur im Modus **Explicit** verwendet werden kann.
- Exchange Mode **„Explicit“**:
  - Schreiben von Werten in eine oder mehrere Variablen:
    - Es findet kein Dateizugriff statt; als Ergebnis des Schreibvorgangs wird der Status `Good` zurückgegeben.
  - Aufruf der `Save`-Methode:
    - Die XML-Datei wird komplett eingelesen.
    - Die XML-Knoten werden den existierenden UKI-4.0 -Variablen über die Baumstruktur zugeordnet.
      Falls für eine existierende UKI-4.0 -Variable kein XML-Knoten gefunden wird, wird einer in der XML-Datei erzeugt, jedoch zunächst ohne Wert (Text).
    - Es werden in alle XML-Knoten die aktuellen Werte eingefügt, die in den UKI-4.0 -Variablen enthalten sind.
    - Die XML-Datei wird geschrieben.

Fehlerdiagnose

Das XML Exchange Plugin liefert je nach zu untersuchender Schicht verschiedene Statusinformationen. Generell werden die dokumentenbasierten Diagnoseinformationen durch den Zugriffsstatus auf die XML-Datei produziert. Die variablenbasierten Diagnoseinformationen werden während des Lese-/Schreibzugriffs auf die verschiedenen Variablen produziert.

Kanal

Um den Status des XML-Dokuments (Kanal) zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf das folgende Bild:



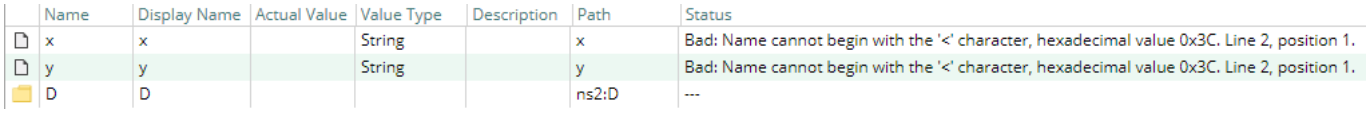
Das obige Bild zeigt das Bedienfeld des XML-Dokuments, das alle statusrelevanten Informationen anzeigt. Das Bedienfeld aktualisiert automatisch seine Statusinformation, wenn ein neuer Status verfügbar ist.

Statuskreis

Farbe	Bedeutung
	Der Kanal ist gestoppt. Klicken Sie den ►-Button, um ihn zu starten.
	Der Kanal startet oder stoppt gerade oder wartet auf den Verbindungsaufbau.
	Der Kanal ist bereit für Lese-/Schreiboperationen. Sie können ihn durch Klick auf den ■-Button stoppen.
	Der Kanal läuft, aber die Verbindung ist momentan fehlerhaft. Bitte überprüfen Sie den Statustext für weitere Informationen.

Variablen

Um den Status der verschiedenen XML-Variablen zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf die in UKI-4.0 angezeigte `Status`-Eigenschaft der Spalte. Falls der Exchange Mode auf **„Implicit“** eingestellt ist, benutzen Sie den Button „Read actual Value“, um die Werte von der XML-Datei auszulesen und das Ergebnis in den Variablen zu speichern. Ansonsten können Sie die `Load`-Methode aufrufen, um zu prüfen, ob die XML-Datei erfolgreich gelesen werden kann.



Logdatei

Alle kanalbezogenen Statusinformationen werden auch in die kanalspezifische Logdatei im `[LoggingFolder]` protokolliert. Jede Logdatei wird nach dem Namensschema `XML Exchange.<ChannelName>.log` benannt.

Der Inhalt einer solchen Logdatei kann wie folgt aussehen:

```
...
2018-04-11 11:32:37.0 +2: [Error] Error (Severity=High): Code=[-1], Text=[The operation has
timed-out.], Details=[]
...
```

Entities

Wie jedes Exchange Plugin erweitert das XML Exchange Plugin das UKI-4.0 Exchange-Modell.

Exchange

Der Exchange-Typ `XmlExchange` des Plugins definiert auch das `XmlExchangeDocument` und erweitert somit die grundlegenden UKI-4.0 `Exchange` und UKI-4.0 `ExchangeChannel` Entities. Während das `XmlExchange` nur eine Konkretisierung des UKI-4.0 `Exchange` darstellt, erweitert das `XmlExchangeDocument` den UKI-4.0 `ExchangeChannel` mit den XML Variable Entities.

Channel

Der Kanal wird von einem Channel Worker behandelt, der über Dateisystemoperationen Zugriff auf die XML-Datei herstellt.

Der Worker liest standardmäßig keine Werte. Wenn der Exchange Mode auf **„Implicit“** gestellt ist und ein Anwender oder Plugin in UKI-4.0 einen synchronen Lesevorgang der `Channel` Variablen anfordert (z.B. mit der „Read actual value“-Funktion in der UKI-4.0 Webkonfiguration), liest der Channel Worker diese aus der XML-Datei und schreibt diese in die entsprechenden UKI-4.0 Nodes.
Wenn der Exchange Mode auf **Explicit** gestellt ist, liest der Channel Worker Werte, sobald die `Load`-Methode des Channels aufgerufen wird.

Ähnlich schreibt der Channel Worker auch die Werte in die Datei, wenn ein Client oder Plugin Werte in die `Channel`-Variablen schreibt („Implicit“) oder wenn die `Save`-Methode des Channels aufgerufen wird („Explicit“).

Damit eine XML-Exchange-Variable im Exchange Mode „Implicit“ regelmäßig gelesen wird, können Sie in der Webkonfiguration bei dem Node „History Options“ auf `Yes` stellen (was eine interne Subscription erstellt), oder Sie können zum Beispiel einen OPC UA Client verwenden, der mit dem OPC UA Server Plugin verbunden ist und damit eine Subscription für die XML-Variablennodes erstellen. In diesen Fällen liest der Channel Worker in regelmäßigen Intervallen die Variablen aus der XML-Datei und schreibt den neuen Wert nach einer Wertänderung automatisch in die entsprechenden UKI-4.0 Nodes.

Ordner & Dateien

Ordner

Name	Pfad	Zweck / Verwendung
AssemblyFolder	<UKI-4.0 `InstallDir>/plugins/XmlExchangePlugin/`	Beinhaltet die Plugin-Assemblydatei.
ConfigFolder	<UKI-4.0 `ProjectDir>/plugins/XmlExchangePlugin/`	Beinhaltet die Plugin-Konfigurationsdatei.
LoggingFolder	<UKI-4.0 `ProjectDir>/log/`	Beinhaltet die Plugin-Logdateien.

Dateien

Typ	Pfad	Zweck / Verwendung
Assembly	`[AssemblyFolder]/`UKI-4.0 `.XmlExchangePlugin.dll`	Die Plugin-Assembly Datei.
Logging	`[LoggingFolder]/XML Exchange.<ChannelName>.log`	Die Logdatei.

Versionsinformation

Dieses Dokument

Datum	`2020-02-06`
Version	`1.0`

Plugin

Name	`XML Exchange Plugin`
Node	`/System/Exchange/XML Exchange`
Version	`1.0.0`

Assembly

Name	UKI-4.0 `.XmlExchangePlugin.dll`
Datum	`2020-02-11`
Version	`1.0.0.0`

Interface Plugins

Alle Interface Plugins verwenden die UKI-4.0 ® API. Jedes zur Verfügung gestellte Interface wird über den UKI-4.0 ® Interface Manager registriert und verwaltet.
Solche Interfaces können sein:

- RESTful Anwendungen (siehe REST Interface Plugin)
- Scripts (siehe Script Interface Plugin)
- andere Untersysteme (siehe OPC UA Server Plugin)

Interface Modell

Die UKI-4.0 ® API wird durch Interfaces erweitert. Jedes Interface erweitert somit die Zugänglichkeit von UKI-4.0 ® für andere Plattformen und Technologien. Zusätzlich besteht die Möglichkeit, dass ein Interface Gebrauch vom UKI-4.0 ® Interface Modell macht. Hierbei erweitert das UKI-4.0 ® Interface Modell das grundlegende UKI-4.0 ® Entity Modell um interfacetypische Entities.

Dabei definiert eine Interface Entity die untergeordneten Entities zur Steuerung (engl. control), für Einstellungen (engl. settings), für den Status des Plugins und die diversen Kanäle (engl. channels), über die das Plugin mit UKI-4.0 ® beziehungsweise anderen Plattformen interagiert.

Konfiguration

Jedes mit UKI-4.0 ® ausgelieferte Interface Plugin lässt sich direkt und ausschließlich in der UKI-4.0 ® Host Anwendung konfigurieren. Verfügt ein Plugin über Konfigurationsparameter, dann können diese in der entsprechenden Interface Entity modifiziert werden.

UKI-4.0UKI-4.0 UKI-4.0UKI-4.0 ® verwenden

Die gesamte Konfiguration aller Interface Plugins finden Sie unter dem Nodepfad `/System/Interfaces`. Dieser Wurzelnode der Interface Plugins ermöglicht die vollständige Konfiguration der Interface Plugins, vorausgesetzt, dass eines der aktiv verwendeten Interface Plugins eigene Interface Entities zur Konfiguration bereitstellt.

OPC UA Server Interface Plugin

Allgemein

Das OPC UA Server Interface Plugin ermöglicht es Ihnen, sich über das standardisierte OPC UA Interface zu UKI-4.0 zu verbinden.

Was tut das Plugin?

Mit diesem Interface können Sie OPC UA Server unter Verwendung des **opc.tcp**-Protokolls erstellen und UKI-4.0 Nodes für OPC UA Clients bereitstellen.

Funktionen

- Browsen der UKI-4.0 ® Nodes
- (Synchrones) Lesen und Schreiben von Werten in Nodes
- Subscriptions zu Nodes erstellen
- Zugriff auf historische Werte von UKI-4.0 ® Nodes über HDA (Historical Data Access)
- Fernzugriff auf Dateien auf dem UKI-4.0 ® Host PC über den OPC UA Dateityp
- Unterstützt die **SignAndEncrypt** Richtlinie für verschlüsselten Transport

Unterstützte OPC Protokolle

- opc.tcp (Binary Message Encoding)
- http (Binary und XML Message Encoding - in einer späteren Version)

Zweck & Anwendung

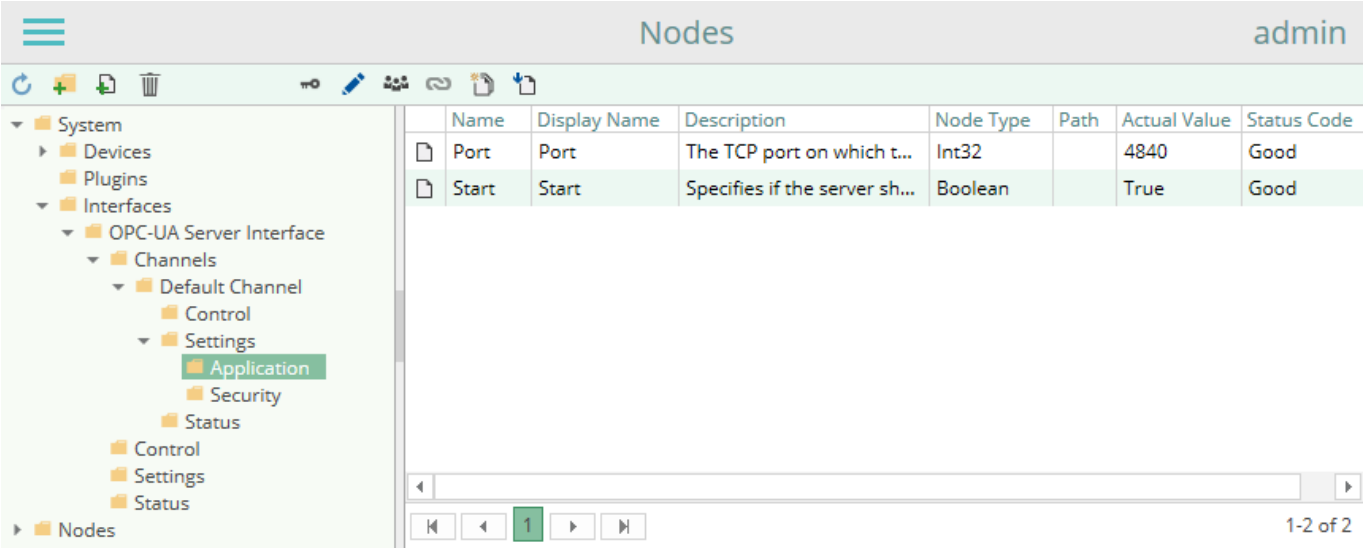
- Zugriff auf S7-Variablen oder andere Nodes über OPC UA

Installation

Dieses Plugin ist Bestandteil des UKI-4.0 ® Setups. Bitte konsultieren Sie UKI-4.0 ® Setup und erster Start für weitere Informationen darüber, wie dieses Plugin installiert und deinstalliert werden kann.

Konfiguration

Das OPC UA Server Interface Plugin kann nur über den Interface Node in UKI-4.0 ® (`/System/Interfaces/OPC UA Server Interface/Channels/Default Channel/Settings`) konfiguriert werden:



Nodes:

Name	Typ	Zweck
Application		
ApplicationCertificate	Blob	Enthält das Annwendungsinstanz-Zertifikat des OPC UA Servers im PKCS12 (.pfx)-Format.
Port	Int32	Gibt den TCP-Port für das `opc.tcp`-Protokoll an.
Security		
AutoAcceptUntrustedCertificates	Boolean	Gibt an, ob der OPC UA Server automatisch unbekannte Client-Anwendungszertifikate akzeptieren soll, wenn der Sign- oder SignAndEncrypt-Mode verwendet wird.
PolicyAlgorithm	String	Gültige Werte: `0`: None (Standard) `1`: Basic128Rsa15
PolicyLevel	Int32	
PolicyMode	String	Gültige Werte: `0`: None (Standard) `1`: Sign `2`: SignAndEncrypt

Um die Einstellungen zu ändern, können Sie einen neuen Nodewert über die UKI-4.0 ® Webkonfiguration durch den „Write a new value“-Button ( ) schreiben.

Um sich zum OPC UA Server zu verbinden, benutzen Sie bitte die folgende URL (ersetzen Sie <Hostname> mit dem Hostnamen bzw. der IP-Adresse Ihres PCs oder mit „localhost“, wenn Sie sich vom lokalen PC aus verbinden) und ersetzen Sie <Port> mit Ihrer Portnummer):

```
opc.tcp://<Hostname>:<Port>/
```

Benutzerkonfiguration

Das OPC UA Server Interface Plugin setzt voraus, dass OPC UA Clients die Username-Authentifizierung verwenden, d.h. sie müssen einen Benutzernamen und ein Passwort verwenden. Der OPC UA Server verwendet dazu die Benutzer, die in UKI-4.0 ® konfiguriert sind (siehe UKI-4.0 ® Webkonfiguration). Jeder Benutzer, der in UKI-4.0 ® konfiguriert ist, kann sich zum OPC UA Server mit dem entsprechenden Benutzernamen und Passwort verbinden.

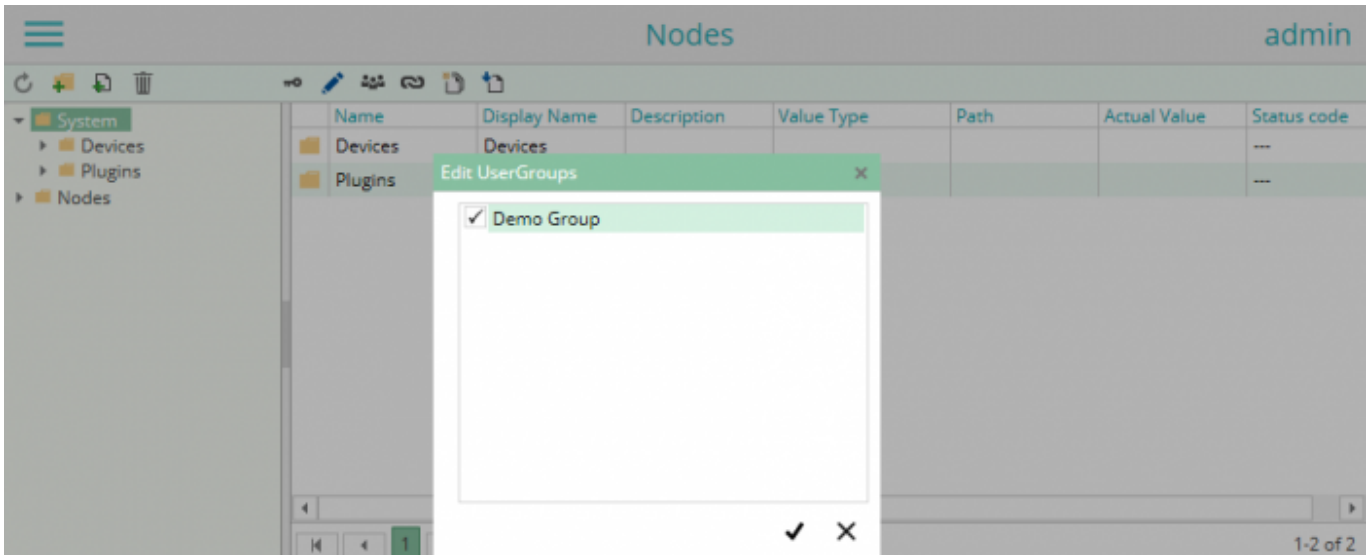
Standardmäßig enthält UKI-4.0 ® einen Demo-Benutzer (Name „demo@user.org“, Passwort „demo“), welcher in der „Demo Group“-Benutzergruppe enthalten ist.

Von sich aus kann ein Benutzer über OPC UA auf kein UKI-4.0 ® Node zugreifen. Um das zu ändern,

müssen Sie den entsprechenden Benutzer zu einer Benutzergruppe („UserGroup“) hinzuzufügen und anschließend den Zugriff für diese Benutzergruppe auf einen Node erlauben. Weitere Informationen finden Sie unter UKI-4.0 ® Webkonfiguration.

Um beispielsweise dem Demo-Benutzer (demo@user.org) den Zugriff auf den „System“-Node zu erlauben:

1. Öffnen Sie die UKI-4.0 ® Webkonfiguration.
2. Wählen Sie den Menüpunkt „Nodes“.
3. Wählen Sie den „System“-Node aus.
4. Klicken Sie auf das „UserGroups“-Icon ( ).
5. Im Dialogfenster „Edit UserGroups“ wählen Sie die „Demo Group“ aus



Zugriff auf den OPC UA Server

Nachdem Sie die Benutzer konfiguriert haben, können Sie auf den OPC UA Server z.B. mit unserem Tool **OPC Watch** zugreifen. Hierzu verwenden Sie bitte folgende Konfiguration:

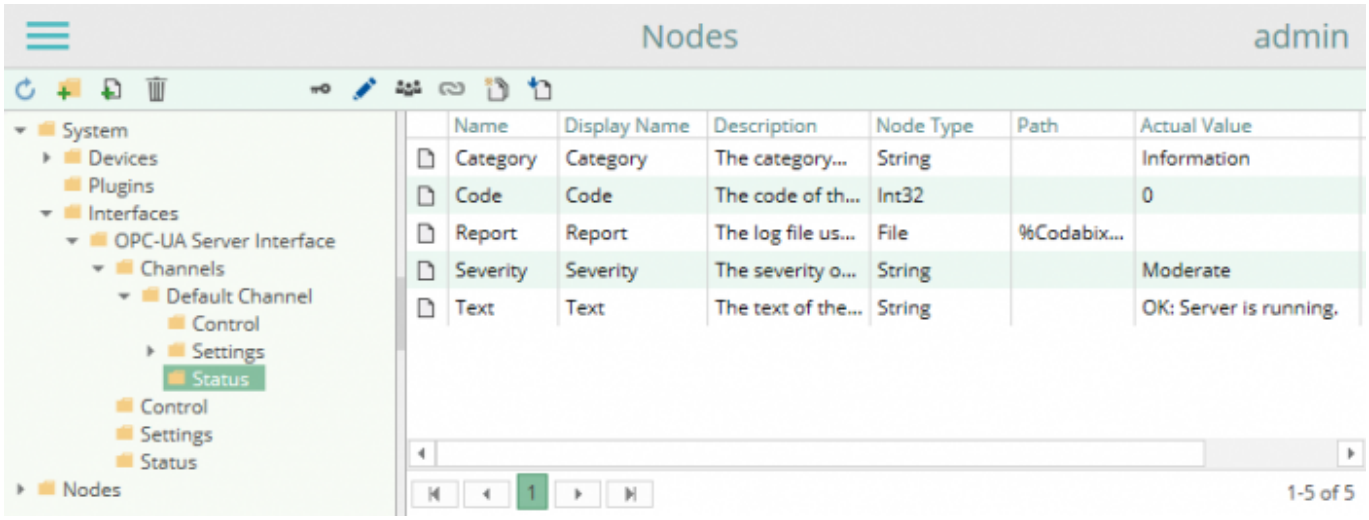
- ServerAddress: `opc.tcp://localhost:4840/`
- UserIdentity / Username: `demo@user.org`
- UserIdentity / Password: `demo`

OPC Namespace: Die UKI-4.0 ® Nodes werden über die OPC UA Namespace URI

"UKI-4.0 ://opc.server/" abgebildet. Standardmäßig hat dieser Namensraum den Namespace Index „2“.

Fehlerdiagnose

Das OPC UA Server Interface Plugin stellt Informationen zur Fehlerdiagnose im Status Node des Kanals bereit.



Node	Beschreibung
Code	Definiert den numerischen Ausdruck / Identifier des Status. Ein negativer Wert bedeutet, dass der Server nicht gestartet werden konnte, wohingegen 0 oder ein positiver Wert bedeuten, dass der Server erfolgreich gestartet ist.
Category	Kategorisiert den Status in Information, Warning und Error und stellt daher die allgemeine Statusinformation dar.
Severity	Unterteilt die Statusinformation in Low, Moderate, High und Critical und zeigt so die Dringlichkeit eines Einschreitens an.
Text	Beschreibt die durch die Code Eigenschaft identifizierte Statusinformation.

Entities

Als Interface Plugin erweitert das OPC UA Server Interface Plugin das grundlegende UKI-4.0 ® Interface Modell.

Interface

Der Interfacetyp OpcServerInterface des Plugins definiert auch den OpcServerInterfaceChannel und erweitert somit die grundlegenden UKI-4.0 Interface und UKI-4.0 InterfaceChannel Entities. Während das OpcServerInterface lediglich eine Konkretisierung des UKI-4.0 Interface repräsentiert, erweitert der OpcServerInterfaceChannel den UKI-4.0 InterfaceChannel.

Channel

Jeder Channel repräsentiert eine OPC UA Serverinstanz, die konfiguriert, gestartet und gestoppt werden kann.

Ordner & Dateien

Ordner

Name	Pfad	Zweck / Verwendung
Assembly	<UKI-4.0 InstallDir>/plugins/OpcUaServerInterfacePlugin/	Beinhaltet die Plugin Assembly Datei.
Config	<UKI-4.0 DataDir>/plugins/OpcUaServerInterfacePlugin/	Beinhaltet die Plugin Konfigurationsdatei.

Dateien

Typ	Pfad	Zweck / Verwendung
Assembly	[AssemblyFolder]/UKI-4.0 .OpcUaServerInterfacePlugin.dll	Die Plugin Assembly Datei.

Versionsinformation

Dieses Dokument

Datum	2016-12-21
Version	1.3

Plugin

Name	OPC UA Server
Node	/System/Interfaces/OPC-UA Server Interface
Version	1.0.9

Assembly

Name	UKI-4.0 .OpcUaServerInterfacePlugin.dll
Datum	2018-02-01
Version	1.0.9.0

REST Interface Plugin

Allgemein

Das REST Interface Plugin erlaubt den Zugriff auf die Nodes von UKI-4.0 ® über HTTP Anfrage formatiert als JSON Objekt.

Was tut das Plugin?

- Nodes lesen (Werte, optional historische Werte, Kinder und Eigenschaften) von UKI-4.0 ®.
- Einen neuen Istwert auf einen spezifizierten Node schreiben.
- Einen neuen Node anlegen.

z.B. liest die folgende HTTP Anfrage den Nodestruktur Baum von UKI-4.0 ®:

UKI-4.0 -Request.js

```
var oIE = WScript.CreateObject("InternetExplorer.Application");
oIE.navigate("about:blank");
var token = oIE.Document.Script.prompt("Enter Token",
"1:QqjkQAtk4hyWRnbTt1+dZdGFCg3QE+nS");

var request;
try {request = new XMLHttpRequest();}
catch (error) {
   try {request = new ActiveXObject("Msxml2.XMLHTTP");}
   catch (error) {
      request = new ActiveXObject("Microsoft.XMLHTTP");
   }
}

request.open("POST", "http://localhost:8181/api/json", false);
request.send('{"tk": "' + token + '", "browse": { "na": "" } }');

WScript.Echo(request.responseText);
```

Funktionen

Unterstützte Aktionen für Nodes:

Aktion	Beschreibung
Lesen	Liest den spezifizierten Node.
Browsen	Liest den spezifizierten Node mitsamt Childnodes.
Schreiben	Schreibt einen neuen Istwert in den spezifizierten Node.

Unterstützte Clients

- HTTP Clients, die HTTP/1.1 sowie das JSON-Format unterstützen

Zweck & Anwendung

- Einfache Verbingung zu anderen Systemen
- Einfacher Lese- und Schreibzugriff auf die Nodes von UKI-4.0 ® durch das benutzen einer beliebigen Programmiersprache ohne zusätzliche Bibliotheken
- Verbinden von UKI-4.0 ® zu existierender RESTable Software (z.B.. SCADA Systeme)

Dokumentation des RESTful API

Hier finden Sie die Dokumentation der REST API des Plugins:

- Dokumentation der RESTful API

Installation

Das REST Interface Plugin ist ein fester Bestandteil von UKI-4.0 ® und muss nicht installiert werden.

Konfiguration

Das REST Interface Plugin bietet zur Zeit keine Konfigurationsmöglichkeiten an.

Fehlerdiagnose

Das REST Interface Plugin stellt einen Statuscode für Funktionen zu Diagnosezwecken zur Verfügung - siehe Anfrage & Antwort der REST API Documentation.

Entities

Das REST Interface Plugin verwendet das UKI-4.0 ® Entity Modell nicht und stellt daher keine Entities zur Verfügung.

Ordner & Dateien

Das REST Interface Plugin ist ein fester Bestandteil von UKI-4.0 ® und hat deshalb keine zusätzlichen Dateien oder Ordner.

Versionsinformation

Dieses Dokument

Datum	`2017-07-12`
Version	`1.2`

Plugin

Name	`REST Interface Plugin`
Version	Entspricht der UKI-4.0 ® Version

Assembly

Das REST Interface Plugin ist ein fester Bestandteil von UKI-4.0 ® und hat deshalb keine separate Assembly.

REST API Dokumentation

Die Schnittstelle muss eine HTTP Anfrage mit einem JSON Objekt (Kodierung: UTF-8) im Anfragekörper sein.

Beispiel:

```
POST /api/json HTTP/1.1
Content-Type: application/json; charset=utf-8
Host: localhost:8181
Content-Length: 79

{
   "tk": "1:QqjkQAtk4hyWRnbTt1+dZdGFCg3QE+nS",
   "browse": { "na": "" }
}
```

Zugriff

Der Zugriff auf das REST API kann durch die Benutzung des folgenden Links gewährt werden:

```
http(s)://<ip/domain>:<port>/api/json
```

z.B.

```
http://localhost:8181/api/json
```

Siehe Konfiguration des Ports

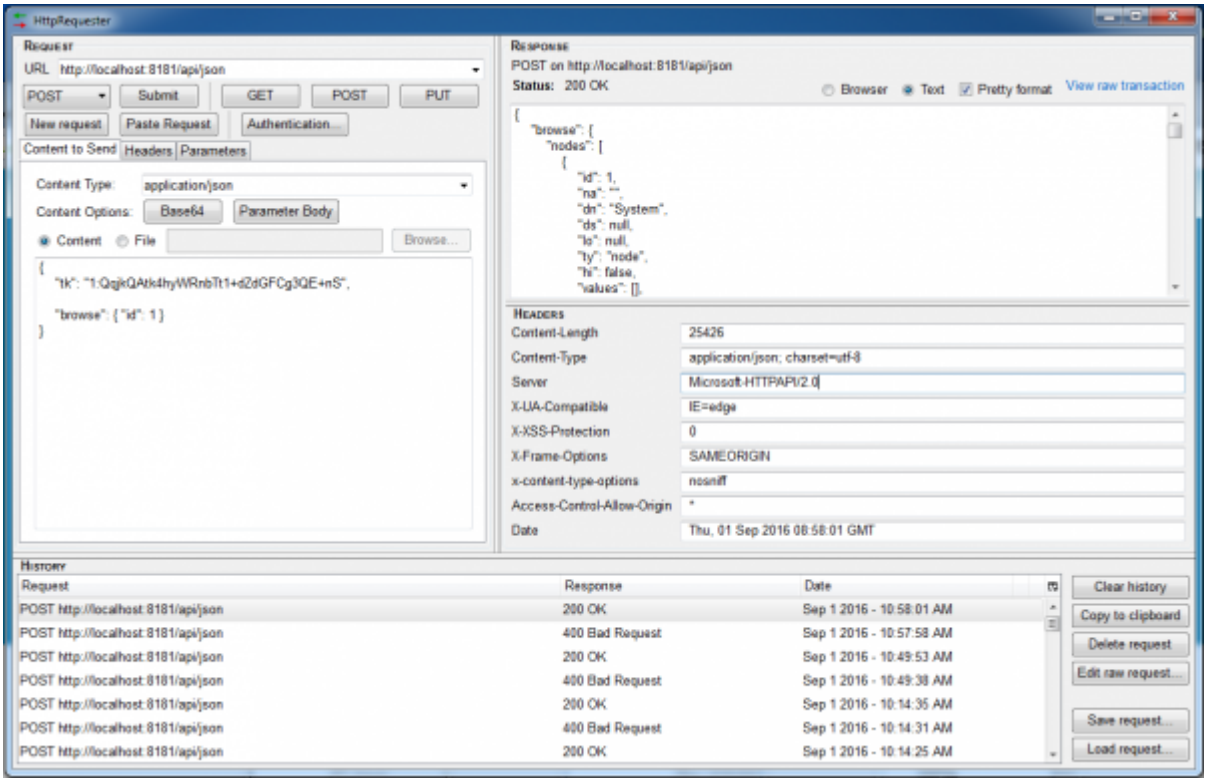
Anfrage & Antwort

Spezifikationen der Anfrage

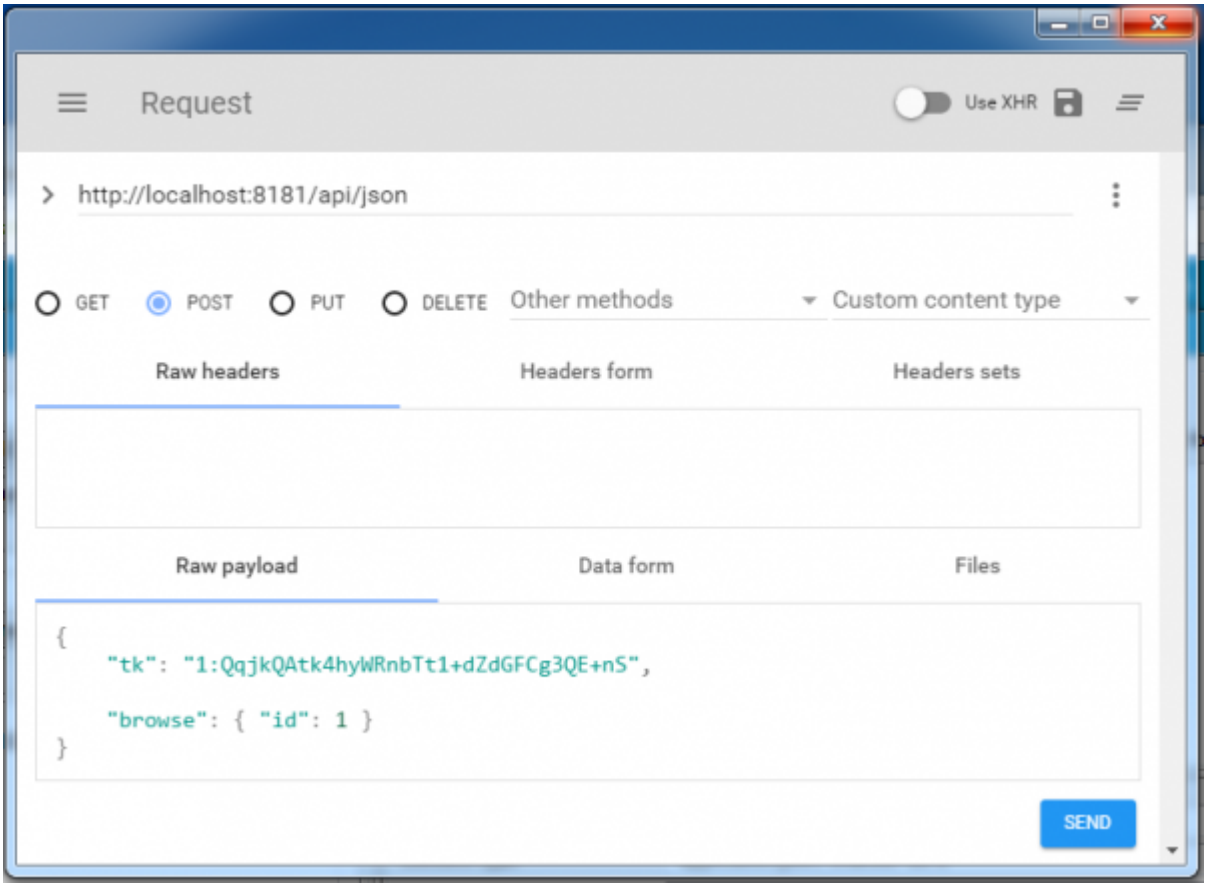
Methode	POST
Inhaltstyp	`application/json`
Kodierung	UTF-8
Inhalt	JSON Objekt

Tools für eine Beispielanfrage

- HTTP Anfrage Script
- HTTP Anfragetool für Firefox



- HTTP Anfragetool für Google Chrome



Allgemeine Anfrage

	Verpflichtend	Typ	Zweck
`tk`	**ja** (falls `username` und `password` nicht spezifiziert sind)	String	Der **Token** des Nodes, der authorisiert werden soll. Von diesem Node werden alle Kinder relativ adressiert. Wie man den Token findet, sehen Sie unter Nodezugriff z.B. wird der Token des Nodes `System/Plugins` festgelegt, dann ist `"na":"OPC UA Server"` der relative Pfad des Nodes `/System/Plugins/OPC UA Server`.
`username`	**ja** (falls `tk` nicht spezifiziert ist)	String	Die Emailadresse oder Telefonnummer des zu authentifizierenden Benutzers.
`password`	**ja** (falls `tk` nicht spezifiziert ist)	String	Das Passwort des Benutzers.
`get`	nein	Objekt	Nodes lesen
`browse`	nein	Objekt	Nodes mit allen Kindern lesen (rekursiv)
`set`	nein	Objekt	Nodes schreiben
`create`	nein	Objekt	Nodes erstellen
`update`	nein	Objekt	Nodes updaten
`delete`	nein	Objekt	Nodes löschen

Beispielanfrage (Token benutzen):

```
{
   "tk": "961:oseIqCm8WOOnPEE5g0tt1TZz2wbL/qUq",
   "get": {
      "na": "Temperature"
   }
}
```

Beispielanfrage (Benutzername-Passwort benutzen):

```
{
   "username": "demo@user.org",
   "password": "demo",
   "get": {
      "na": "/Demo-Nodes/Temperature"
   }
}
```

Anfrageframe:

```
{
   "tk": /* Token */,
   "username": /* Username (falls 'tk' nicht angegeben ist) */,
   "password": /* Passwort (falls 'tk' nicht angegeben ist) */,

   "get":    { /* "Read"-Elemente */ },
   "browse": { /* "Browse"-Elemente */ },
   "set":    { /* "Set"-Elemente */ },
   "create": { /* "Create"-Elemente */ },
   "update": { /* "Update"-Elemente */ },
   "delete": { /* "Delete"-Elemente */ }
}
```

Allgemeine Antwort

	Verpflichtend	Typ	Zweck
`get`	nein	Objekt	Lesen & Browsen Antwort
`browse`	nein	Objekt	Lesen /& Browsen Antwort
`set`	nein	Objekt	Schreiben Antwort
`create`	nein	Objekt	Erstellen Antwort
`update`	nein	Objekt	Updaten Antwort
`delete`	nein	Objekt	Löschen Antwort
`res`	**ja**	Objekt	Ergebnis der Abfrage

Mögliche `res`-Ergebnisse:

	Verpflichtend	Typ	Zweck
`value`	**ja**	Zahl	Ergebniscodes: >= 0: OK - Alles ist gut! < 0: Es ist ein Fehler aufgetreten!
`reason`	nein	String	Grund für den Fehler

Beispielantwort:


```
{
   "get": {
      "nodes": [
         {
            "id": 963,
            "na": "Temperature",
            "dn": "Temperature (°C)",
            "ds": "",
            "lo": null,
            "ty": "double",
            "hi": false,
            "min": -20,
            "max": 90,
            "unit": "°C",
            "values": [
               {
                  "va": 2,
                  "ts": 1472738754519,
                  "st": 0
               }
            ]
         }
      ],
      "res": {
         "value": 0
      }
   }
}
```


Antwortframe:

```
{
   "get":    { /* "Get"-Elemente */ },
   "browse": { /* "Browse"-Elemente */ },
   "set":    { /* "Set"-Elemente */ },
   "create": { /* "Create"-Elemente */ },
   "update": { /* "Update"-Elemente */ },
   "delete": { /* "Delete"-Elemente */ },
   "res":    { /* "Result"-Elements */ },
}
```

Lesen / Browsen

Anfrage

Die „Get“ Funktion ermöglicht es, einen oder mehrere (historische) Werte zu lesen. „Browsen“ is wie „Get“, erlaubt aber das Erhalten des Istwerts des Nodes, daher gibt es rekursiv alle Childnodes mitsamt deren Istwerten wider.

	Verpflichtend	**Typ**	**Zweck**
`id`	**ja** (falls `na` nicht spezifiziert ist)	Zahl	Lokaler Identifier („Id“) des Nodes.
`na`	**ja** (falls `id` nicht spezifiziert ist)	String	Name des Nodes. Falls ein Token zur Authentifizierung benutzt wird, ist dies der relative Pfas vom authentifizierten Node zum Zielnode (falls leer ist der authentifizierte Node selbst der Zielnode). Andernfalls (für Benutzername-Passwort-Authentifizierung) ist es der absolute Pfad zum Zielnode.
`count` (falls es kein `ttl` gibt)	nein	Zahl	Die Anzahl an historischen Werten (max. 1000). Standard ist `1`, was bedeutet, dass der Istwert (statt historischen Werten) wird zurückgegeben.
`from` (falls es kein `ttl` gibt)	nein	Zahl	Falls spezifiziert, der Startzeitstempel, von dem aus historische Werte zurückgeholt werden sollen.
`to` (falls es kein `ttl` gibt)	nein	Zahl	Falls spezifiziert, der Endzeitstempel, bis zu dem historische Werte zurückgeholt werden sollen.
`ttl` (falls es kein `count` gibt)	nein	Zahl oder `null`	Time To Live. Wenn dieser Parameter nicht existiert, gibt die Funktion direkt den aktuellen („gecachten“) Wert des Nodes zurück. Wenn der Parameter existiert, wird auf dem Node ein synchroner Lesevorgang durchgeführt, um vom zugrundeliegenden Gerät zu lesen. Ein Wert von `null` bedeutet, dass das standardmäßige Maximalwertalter („Max Value Age“) des Nodes verwendet werden soll. Ansonsten ist der Wert das Alter in Millisekunden, das der aktuelle Nodewert haben kann, um direkt zurückgegeben zu werden. Ist der Nodewert älter, wird er synchron vom Gerät gelesen. Um einen synchronen Lesevorgang unabhängig vom eingestellten Max Value Age des Nodes zu erzwingen, geben Sie `0` als Wert an.

Beispiel Leseanfrage (Token benutzen):

```
{
   "tk": "961:oseIqCm8WOOnPEE5g0tt1TZz2wbL/qUq", /* Authentication for "/Nodes/Demo-Nodes/"
*/
   "get": [{
      "na":    "Temperature", /* Reads Node: "/Nodes/Demo-Nodes/Temperatur" */
   },
   {
      "id":    964,          /* Reads Node: "/Nodes/Demo-Nodes/Pressure" */
      "count": 5             /* Reads the last 5 historical values */
   }]
}
```

Beispiel Leseanfrage (Benutzername-Passwort benutzen):

```
{
   "username": "demo@user.org",
   "password": "demo",

   "get": [{
      "na":    "/Nodes/Demo-Nodes/Temperature", /* must specify the absolute Node Path here
*/
   },
   {
      "id":    964,          /* Reads Node: "/Nodes/Demo-Nodes/Pressure" */
      "count": 5             /* Reads the last 5 historical values */
   }]
}
```

Antwort

- Für jedes angeforderte „Get“- oder „Browse“-Objekt antwortet der Server mit einem Node (in Array), das folgende Eigenschaften hat:

	Verpflichtend	**Typ**	**Zweck**
`res`	**ja**	Objekt	Ergebnis der Abfrage
`id`	**ja**	Zahl	Identifier
`na`	**ja**	String	Name
`dn`	**ja**	String	Anzeigename
`ds`	**ja**	String	Beschreibung
`lo`	**ja**	String	Ort
`ty`	**ja**	String	Typ
`hi`	**ja**	Bool	Hat historische Werte
`min`	**ja**	Zahl	Minimalwert
`max`	**ja**	Zahl	Maximalwert
`values`	**ja**	Objekt Array	
`nodes`	nein (nur bei Browsen)	Objekt Array	Beinhaltet alle Childnodes (rekursiv). Für jedes Kind sind die Eigenschaften die gleichen wie in dieser Tabelle.

Mögliche `res`-Ergebnisse:

	Verpflichend	**Typ**	**Zweck**
`value`	**ja**	Zahl	Ergebniscodes: >= 0: OK - Alles ist gut! < 0: Es ist ein Fehler aufgetreten!

	Verpflichend	Typ	Zweck
reason	nein	String	Grund für den Fehler

Mögliche value-Ergebnisse:

	Verpflichend	Typ	Zweck
va	**ja**	jeder	Wert
ts	**ja**	Zahl	Zeitstempel
ct	**ja**	Zahl	Kategorie
st	**ja**	Zahl	Status
sttext	nein	String	Status-Text (falls gesetzt)

Beispiel Leseantwort:


```
{
   "get": {
      "nodes": [
         {
            "id": 963,
            "na": "Temperature",
            "dn": "Temperature (°C)",
            "ds": "",
            "lo": null,
            "ty": "double",
            "hi": false,
            "min": -20,
            "max": 90,
            "values": [
               {
                  "va": 78,
                  "ts": 1472740618590,
                  "st": 0
               }
            ]
         },
         {
            "id": 964,
            "na": "Pressure",
            "dn": "Pressure",
            "ds": "",
            "lo": "",
            "ty": "double",
            "hi": true,
            "min": 10,
            "max": 110,
            "values": [
               {
                  "va": 86,
                  "ts": 1472740619105,
                  "st": 0
               },
               {
                  "va": 84,
                  "ts": 1472740617887,
                  "st": 0
               },
               {
                  "va": 98,
                  "ts": 1472740616967,
```

```
                "st": 0
            },
            {
                "va": 32,
                "ts": 1472740615136,
                "st": 0
            },
            {
                "va": 31,
                "ts": 1472740612718,
                "st": 0
            }
        ]
    }
  ],
  "res": {
    "value": 0
  }
 }
}
```


Schreiben

Anfrage

Mit der „Set“-Funktion ist es möglich, Werte auf einen spezifizierten Node zu schreiben.

Falls der Zeitstempel nicht festgelegt ist, ist es wichtig, die Anfragen für denselben Node sequenziell zu senden. Bevor also eine weitere Anfrage gesendet wird, muss man auf eine erfogreiche Antwort warten. Anders kann es passieren, dass die Werte in einer anderen Reihenfolge als beabsichtigt sortiert werden. Verschiedene Nodes können selbstverständlich parallel geschrieben werden.

	Verpflichtend	**Typ**	**Zweck**
`id`	**ja** (wenn `na` nicht spezifiziert ist)	Zahl	Identifier
`na`	**ja** (wenn `id` nicht spezifiziert ist)	String	Name
`va`	**ja**	jeder	Neuer Wert
`ts`	nein	Zahl	Zeitstempel
`st`	nein	Zahl	Status

Beispiel Schreibanfrage (Token benutzen):

```
{
    "tk": "961:oseIqCm8WOOnPEE5g0tt1TZz2wbL/qUq", /* Authentication for "/Nodes/Demo-Nodes/"
*/
    "set": [
        {
            "na": "Temperature",  /* Write Value 21 to Node: "/Nodes/Demo-Nodes/Temperatur"
*/
            "va": 21
        },
        {
            "id": 964,            /* Write Value 84 with the Timestamp 02.09.16 07:09   */
            "va": 84,             /* and the Status 0                                   */
            "ts": 1472800198510,  /* to Node: "/Nodes/Demo-Nodes/Pressure"               */
            "st": 0
        }
     ]
}
```

Beispiel Schreibanfrage (Benutzername-Passwort benutzen):

```
{
    "username": "demo@user.org",
    "password": "demo",

    "set": [
        {
            "na": "/Nodes/Demo-Nodes/Temperature",  /* Write Value 21 to Node (specified by
absolute Node Path) */
            "va": 21
        },
        {
            "id": 964,            /* Write Value 84 with the Timestamp 02.09.16 07:09   */
            "va": 84,             /* and the Status 0                                   */
            "ts": 1472800198510,  /* to Node: "/Nodes/Demo-Nodes/Pressure"               */
            "st": 0
        }
    ]
}
```

Antwort

Für die „Set“-Funktion hat das Nodearray lediglich Antwortobjekte für gescheiterte Anfragenobjekte.

	Verpflichtend	Typ	Zweck
`res`	**ja**	Objekt	Ergebnis der Abfrage
`nodes`	**ja**	Objekt Array	Beinhaltet alle Childnodes (rekursiv). Für jedes Kind sind die Eigenschaften die gleichen wie in dieser Tabelle.

Mögliche `res`-Ergebnisse:

	Verpflichtend	Typ	Zweck
`value`	**ja**	Zahl	Ergebniscodes: >= 0: OK - Alles ist gut! < 0: Es ist ein Fehler aufgetreten!
`reason`	no	String	Grund für den Fehler

Beispiel Schreibantwort:

```
{
   "set": {
      "nodes": [],
      "res": {
         "value": 0
      }
   }
}
```

Beispiel Schreibantwort (Fehlerfall):

```
{
   "set": {
      "nodes": [
         {
            "id": 964,
            "va": 84,
            "ts": 4728001980, /* 24.02.1970 17:20 */
            "st": 0,
            "res": {
               "value": -1,
               "reason": "values[0].Timestamp is lower than 01.01.2000 00:00:00 +00:00"
            }
         }
      ],
      "res": {
         "value": -1,
         "reason": "At least one error occured when processing the nodes:
values[0].Timestamp is lower than 01.01.2000 00:00:00 +00:00"
      }
   }
}
```

Erstellen

Anfrage

	Verpflichtend	Typ	Zweck
`pid`	**ja** (falls `pna` nicht spezifiziert ist)	Zahl	Parent Identifier Neuer Node wird als Kind dieses Nodes erstellt.
`pna`	**ja**	String	Parent Name Neuer Node wird als Kind dieses Nodes erstellt. z.B. wenn `pna=„Demo-Nodes“` ist, wird der neue Node in diesem Ordner angelegt. Der neue Node wird `„/Nodes/Demo-Nodes/Name“` sein.
`na`	**ja** (falls `pid` nicht spezifiziert ist)	String	Name
`dn`	nein	String	Anzeigename
`ds`	nein	String	Beschreibung
`lo`	nein	String	Ort
`path`	nein	String	Path (z.B. Adresse bei Device-Variablen)
`ty`	**ja**	String	Typ

	Verpflichtend	Typ	Zweck
hi	nein	Zahl	Hysteresis
min	nein	Zahl	Minimalwert
max	nein	Zahl	Maximalwert

Beispiel Erstellanfrage:

```
{
   "tk": "938:843uqQeSaLAg+7TALdOhGDkLOHFZevZw", /* Authentication for "/Nodes" */
   "create": [{
      "pna": "Demo-Nodes",              /* Creates Node "/Nodes/Demo-Nodes/New-Node-1" */
      "na": "New-Node-1",               /* with Type string                            */
      "ty": "string"
   },
   {
      "pid": 961,                       /* Creates Node "Nodes/Demo-Nodes/New-Node-2"  */
      "na": "New-Node-2",               /* with specified Displayname (dn),            */
      "dn": "Display New-Node-2",       /* Description (ds) and Type double            */
      "ds": "This is the new Node 2",
      "ty": "double"
   }]
}
```

Antwort

	Verpflichtend	Typ	Zweck
res	**ja**	Objekt	Ergebnis der Abfrage
nodes	**ja**	Objekt Array	Beinhaltet alle Childnodes (rekursiv). Für jedes Kind sind die Eigenschaften dieselben wie in dieser Tabelle.

Mögliche res-Ergebnisse:

	Verpflichtend	Typ	Zweck
value	**ja**	Zahl	Ergebniscodes: >= 0: OK - Alles ist gut! < 0: Es ist ein Fehler aufgetragen!
reason	nein	String	Grund für den Fehler

Beispiel Erstellantwort:

```
{
   "tk": "938:843uqQeSaLAg+7TALdOhGDkLOHFZevZw",
   "create": [{
     "pna": "Demo-Nodes",
     "na": "New-Node-1",
     "ty": "string"
   },
   {
     "pid": 961,
     "na": "New-Node-2",
     "dn": "Display New-Node-2",
     "ds": "This is the new Node 2",
     "ty": "double"
   }]
}
```

Beispiel Erstellantwort (Fehlerfall):

```
{
   "create": {
      "nodes": [
         {
            "pna": "Demo-Nodes",
            "na": "Existing-Node",
            "ty": "string",
            "res": {
               "value": -1,
               "reason": "An object with the same name does already exist. Please choose
another name."
            }
         },
         {
            "pid": 961,
            "na": "New-Node-2",
            "dn": "Display New-Node-2",
            "ds": "This is the new Node 2",
            "ty": "qwertz",
            "res": {
               "value": -1,
               "reason": "Could not find the Node Type \"qwertz\"."
            }
         }
      ],
      "res": {
         "value": -1,
         "reason": "At least one error occured when processing the nodes: An object with the
same name does already exist. Please choose another name."
      }
   }
}
```


Updaten

- Alle Eigenschaften eines Nodes modifizieren.
- Diese Funktion ist noch nicht implementiert, wird aber bald eingebaut.

Löschen

- Einen Node entfernen.
- Diese Funktion ist noch nicht implementiert, wird aber bald eingebaut.

Script Interface Plugin

Allgemein

Das Script Interface Plugin ermöglicht es Ihnen, **Scripts** in der Sprache *JavaScript* zu schreiben, um UKI-4.0 zu programmieren. Scripts sind **leichtgewichtige Erweiterungen** von UKI-4.0 .

- Ein schneller und einfacher (aber dennoch mächtiger) Weg, die Funktionalität von UKI-4.0 zu erweitern
- implementiert in **JavaScript**, einer leistungsfähigen Scriptsprache, und **TypeScript**, um Fehler durch Bereitstellung von statischer Typisierung zu vermeiden und eine reichhaltige Entwicklererfahrung zu bieten
- kann verändert und neu gestartet werden, während UKI-4.0 läuft
- gespeichert in der UKI-4.0 Backend-Datenbank - somit ist es an die Daten gebunden, nicht an die Installation
- läuft in einer isolierten Umgebung, hat nur Zugriff auf bestimmte definierte UKI-4.0 -APIs, nicht Betriebssystem-APIs
- Sie müssen keine Entwicklungsumgebung installieren, Projekte anlegen, kompilieren, DLLs kopieren, usw. …
  Stattdessen schreiben Sie einfach ein Script im eingebauten, webbasierten Scripteditor mit **IntelliSense**-Unterstützung

Was tut das Plugin?

- Suchen, Erstellen und Manipulieren von Nodes und Lesen und Schreiben von Nodewerten
- Abonnieren von Events, z.B. wenn ein Nodewert geschrieben wurde
- Berechnungen ausführen, mathematische Funktionen aufrufen, Zufallszahlen generieren
- Einen Intervalltimer erstellen, welcher regelmäßig eine Scriptfunktion aufruft
- Dateien lesen und schreiben
- Eingehende HTTP(S)-Requests des UKI-4.0 Webservers verarbeiten (inkl. WebSocket-Verbindungen)
- HTTP(S)-Anfragen zu externen Servern ausführen

Wenn Sie bereits grundlegende Programmiererfahrung haben, werden Sie sich beim Schreiben von Scripts schnell zurecht finden. JavaScript ist eine der beliebtesten Scriptsprachen und in Kombination mit TypeScript (das statische Typsicherheit bietet) kann ein Script **von einem Einzeiler** (z.B. Anpassung eines Nodewerts, bevor er geschrieben wird) **bis hin zu komplexem Code** mit Namespaces, Klassen, Abhängigkeitsbeziehungen zu anderen Scripts und vielem mehr **skalieren**.

Hinweis: Sie müssen keine TypeScript-Kentnisse besitzen, um Scripts schreiben zu können.

Funktionen

- Hohe Performance durch die Kompilierung von Scriptcode in CIL-Bytecode
- Typsicherheit von Variablen, Eigenschaften und mehr durch TypeScript
- Geschützt gegen versehentliche Endlosschleifen durch das Anwenden eines Timeouts auf die

Scriptausführung

Unterstützte Spezifikationen

- Unterstützung von **TypeScript 4.1** im Scripteditor
- Vollständige Unterstützung für **ECMAScript 5.1** (Syntax und Bibliothek)
- Nahezu vollständige Unterstützung der **ECMAScript 2015+**-Syntax (`let` / `const`, `class`, `for-of`, `async` / `await` etc.) über Downlevel-Kompilierung zu ECMAScript 5.1 durch TypeScript
- Teilweise Unterstützung der **ECMAScript 2015**-Bibliothek (Collections, Typed Arrays, `Promise`)
- Unterstützt **Async Functions** für langanhaltende Operationen über die `async`- / `await`-Schlüsselwörter (erweitert das ereignisgesteuerte, blockierungsfreie JavaScript-Modell)

Zweck & Anwendung

Sie können Scripts benutzen für

- einfache Schaltungen („Wenn ein Knopf gedrückt wird, soll das Licht angeschalten werden und nach 3 Minuten wieder ausgehen“)
- Nachbearbeitung von Werten, die von einem Gerät gelesen oder auf ein Gerät geschrieben werden, z.B. durch das Anstellen von Berechnungen
- die Generierung von komplexen Nodehierarchien / -strukturen
- die Generierung von Demodaten (z.B. alle 2 Sekunden einen zufälligen Wert auf einen Node schreiben)
- Big Data: Sammeln von externen Daten um einen bestimmten Punkt herum, z.B. um das aktuelle Wetter von Berlin in einem Datenpunktnode zu speichern
- das Bereitstellen einer komplexen, benutzerdefinierten Bedingung für Trigger

Script Interface Plugin Development Guide

Hier finden Sie den Development Guide für das Script Interface Plugin:

- Script Interface Plugin Development Guide

Installation

Das Script Interface Plugin ist ein fester Bestandteil von UKI-4.0 und muss nicht installiert werden.

Konfiguration

Das Script Interface Plugin ermöglicht die Konfiguration von Scripts in der UKI-4.0 Webkonfiguration über den Menüpunkt „Script Interface“ - siehe Scripts verwalten im Script Interface Plugin Development Guide.

Fehlerdiagnose

Das Script Interface Plugin ermöglicht die Diagnose von Scripts in der UKI-4.0 Webkonfiguration über den Menüpunkt „Script Interface“ - siehe Scripts verwalten im Script Interface Plugin Development Guide.

Entities

Das Script Interface Plugin verwendet das UKI-4.0 Entity Model nicht, da es das Verwalten von Scripts über die UKI-4.0 Webkonfiguration ermöglicht, und stellt deshalb keine Entities zur Verfügung.

Ordner & Dateien

Das Script Interface Plugin ist ein fester Bestandteil von UKI-4.0 und hat deshalb keine zusätzlichen Dateien oder Ordner.

Versionsinformation

Dieses Document

Datum	`2017-11-14`
Version	`1.8`

Plugin

Name	`Script Interface Plugin`
Version	Entspricht der UKI-4.0 -Version

Assembly

Das Script Interface Plugin ist ein fester Bestandteil von UKI-4.0 und hat deshalb keine separate Assembly.

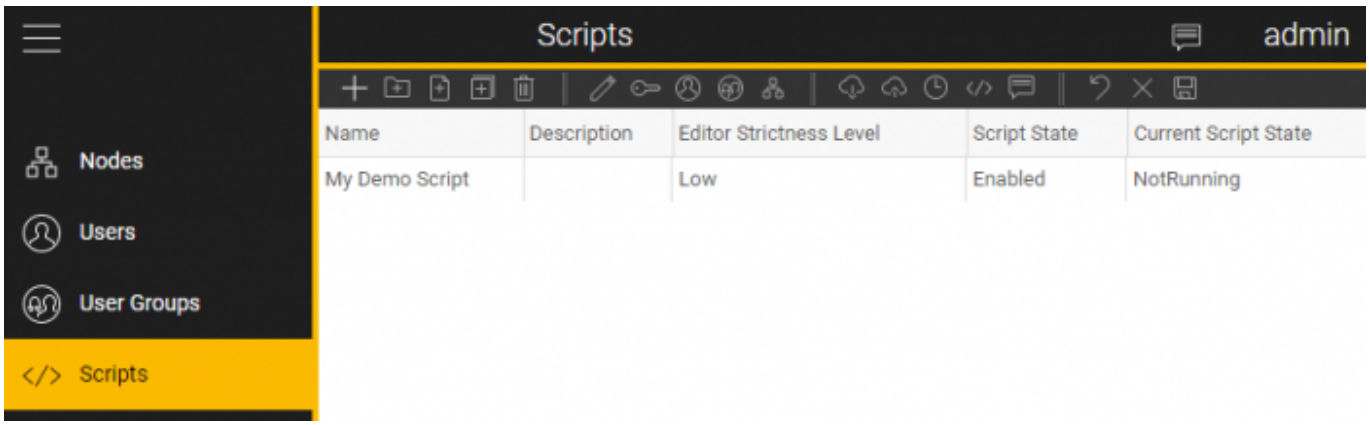
Script Interface Plugin Development Guide

Beispielcode Script

```
// Finde den Node über den Node-Pfad.
const pressureNode = UKI-4.0 .findNode("/Nodes/Injection molding/Pressure", true);

// Erstelle einen Intervalltimer, der unsere Funktion alle 3 Sekunden aufruft.
timer.setInterval(function () {

    // Generiere eine zufällige Zahl zwischen 20 und 150.
    let randomNumber = Math.random() * 130 + 20;

    // Schreibe diese Zahl nun in die Node.
    UKI-4.0 .writeNodeValueAsync(pressureNode, randomNumber);

}, 3000);
```

Scripts verwalten

Die UKI-4.0 Webkonfiguration enthält den Menüpunkt „Script Interface“, mit dem Sie Scripts erstellen, editieren und löschen können. Sie können auch Scripts stoppen, sodass sie nicht weiter ausgeführt werden, ohne sie zu löschen.



Eigenschaft	Beschreibung
Name	Name des Scripts, um es zu identifizieren. Dieser Name wird auch in Stapelüberwachungen benutzt, wenn eine Ausnahme aufgetreten ist.
Description	Hier können Sie eine detailliertere Beschreibung des Scripts eingeben.

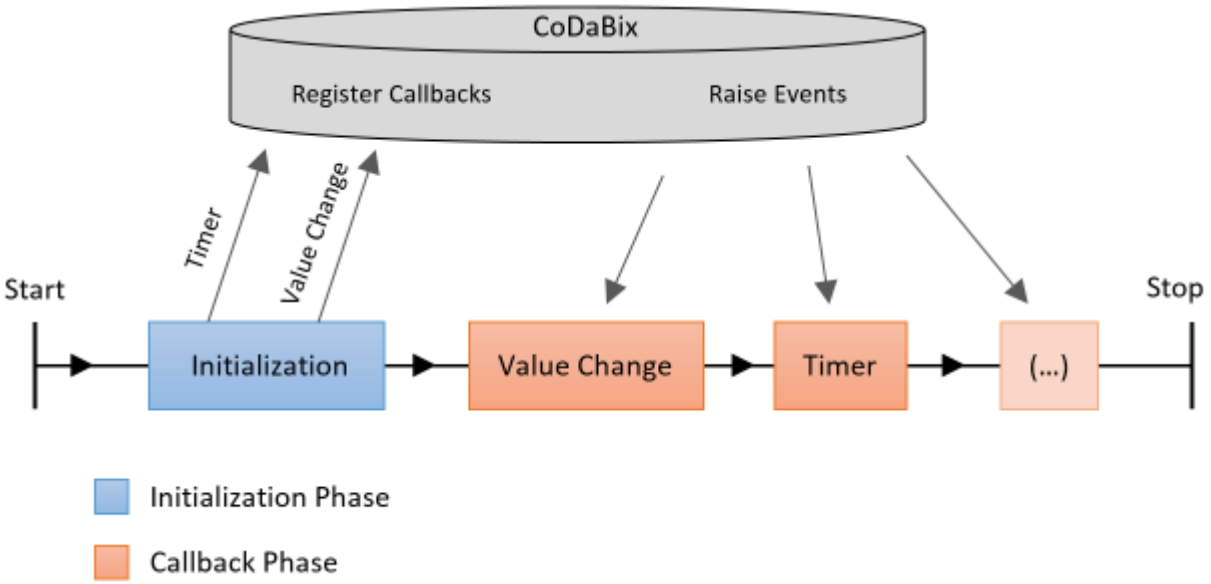
Eigenschaft	Beschreibung
Editor Strictness Level	Bestimmt, wie streng der Editor bestimmten Code handhabt. **Low** (Standard): Der Editor bemängelt die bei den folgenden Optionen beschriebenen Fälle nicht. **Medium**: Der Editor bemängelt implizite `any`-Typen, implizite Returns sowie Fallthrough-Cases in `switch`-Anweisungen. **High**: Zusätzlich zu den Fällen des „Medium“-Levels bemängelt der Editor nicht verwendete lokale Variablen sowie nicht verwendete Funktionsparameter. Beachten Sie: Für Library Scripts (in einer späteren Version verfügbar) ist das Level immer „High“.
Script State	**Enabled**: Das Script soll ausgeführt werden. **Disabled**: Das Script soll gestoppt werden.
CurrentScriptState	Zeigt den aktuellen Zustand des Scripts an. **NotRunning**: Das Script wird nicht ausgeführt, entweder weil es auf „Disabled“ gesetzt ist, oder weil es kürzlich erst erstellt wurde und noch kein Live-Code dafür existiert. **Running**: Das Script wurde gestartet und seitdem ist kein Fehler aufgetreten (das kann auch der Fall sein, wenn das Script vor kurzem wegen einer unbehandelten Ausnahme neugestartet wurde). **Stopped**: Das Script wurde gestartet und ausgeführt, aber es sind keine Event-Listener oder Callbacks mehr aktiv. **StoppedAndScheduledForRestart**: Das Script wurde wegen einer unbehandelten Ausnahme während der Ausführung gestoppt und wird nach einer kurzen Zeit automatisch neugestartet.

Der Standartwert für „Editor Strictness Level“ ist „Low“. Wir empfehlen diese Einstellung, wenn Sie gerade erst mit Scripts und JavaScript anfangen. Wenn Sie ein erfahrener TypeScript Entwickler sind, empfehlen wir die Einstellung „Medium“ oder „High“, sodass der Editor Sie dabei unterstützen kann, eine saubere Codegrundlage zu entwickeln.

Beachten Sie: Es kann bis zu 3 Sekunden dauern, bis eine Änderung (z.B. Starten oder Stoppen eines Scripts) in Kraft tritt und weitere 3 Sekunden, bis der CurrentScriptState und das Script Log aktualisiert werden.

Script-Verarbeitung

Nachdem Sie eine Liveversion (siehe Abschnitt Going Live) eines Scripts erstellt haben, wird diese automatisch gestartet, solange dessen Status auf „Enabled“ gestellt ist. Beim Starten befindet sich das Script in der sogenannten „Initialisierungsphase“. Während dieser Phase kann das Script Callbacks registrieren (z.B. um Ereignisse oder Timer zu behandeln). Wenn einer der Callbacks aufgerufen wird, befindet sich das Script in der sogenannten „Callback-Phase“ (auch in dieser Phase kann es noch weitere Callbacks registrieren).
Das folgende Diagramm veranschaulicht die Phasen eines Scripts:



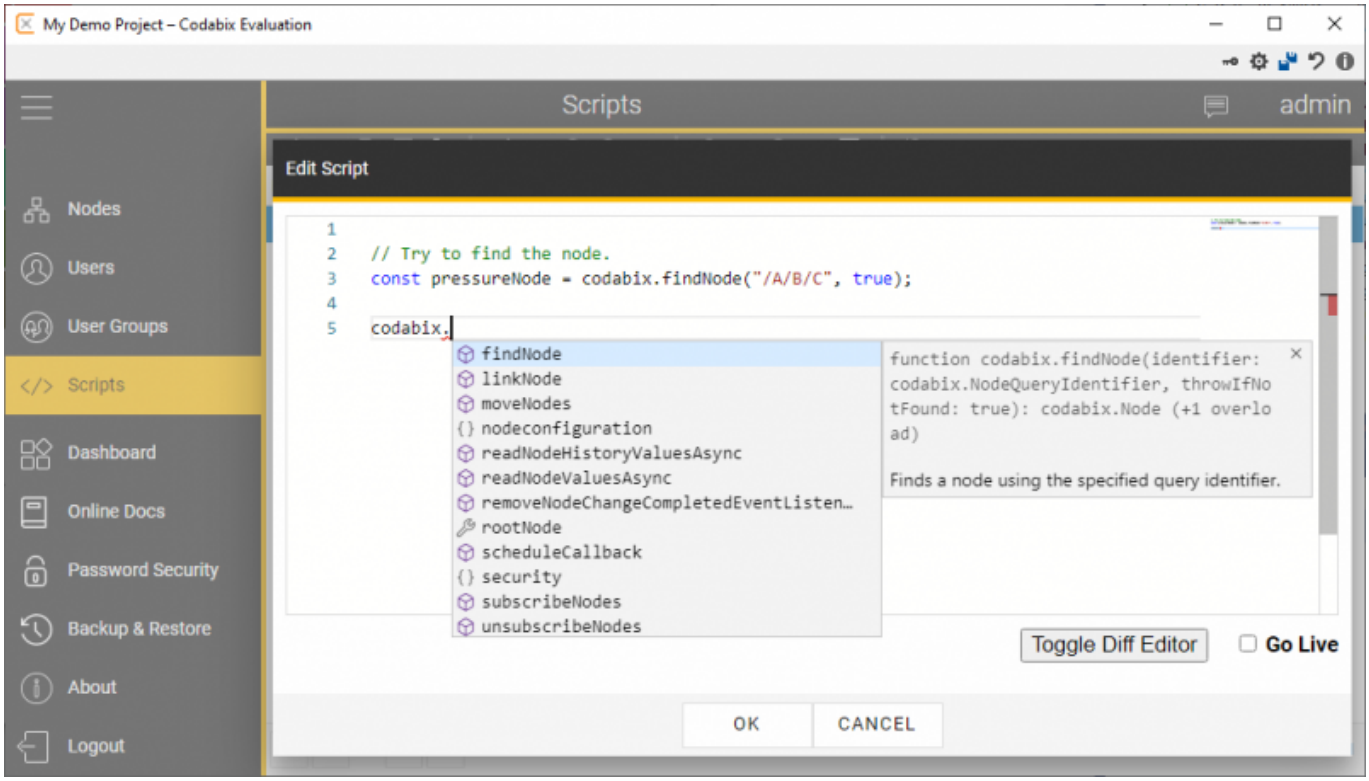
Beachten Sie: Obwohl es im obigen Diagramm nicht dargestellt wird, kann das Script immer noch weitere Callbacks für andere Events registrieren, wenn es sich bereits in der Callback-Phase befindet.

Auf das Script wird ein Timeout von ungefähr **15000 ms** angewandt, damit unbeabsichtigte Endlosschleifen wie `while (true) {}` das System nicht lahmlegen können. Wenn das Script nach einer Zeitüberschreitung noch nicht fertig ist, wird es gestoppt und so behandelt, also ob eine unbehandelte Ausnahme aufgetreten ist.

Sowohl in der Initialisierungs- als auch in der Callback-Phase wird das Script beim Auftreten einer unbehandelten Ausnahme nach einer kurzen Zeit (ca. 3 Sekunden) automatisch neu gestartet.

Der eingebaute Scripteditor

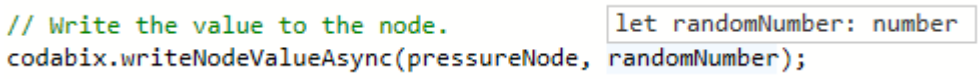
Durch Klick auf das Scripteditor-Icon (</>) öffnet sich der Editor und Sie können Ihren Scriptcode schreiben.



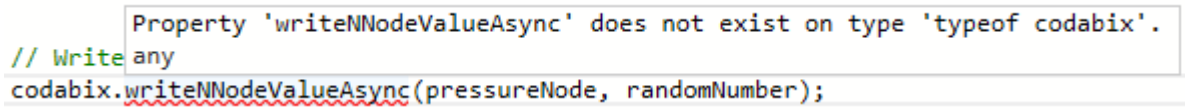
Wenn Sie bereits mit Visual Studio oder VS Code gearbeitet haben, wird Ihnen der Script-Editor bekannt

vorkommen (tatsächlich basiert dieser auf dem Monaco-Editor von VS Code). Der Editor stellt während des Tippens IntelliSense für UKI-4.0 API-Methoden bzw. Interfaces sowie für eingebaute JavaScript-Klassen zur Verfügung, wie Sie im obigen Screenshot sehen können.

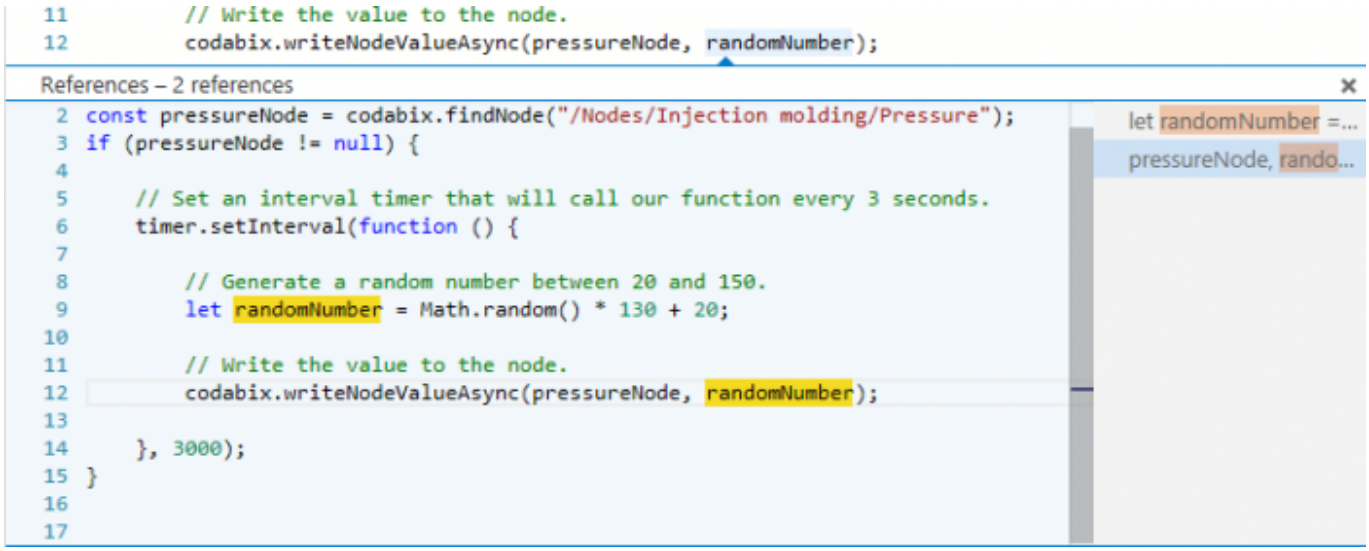
Wenn Sie mit der Maus über Variablen- oder Methodenaufrufe fahren, erscheint ein Tooltip, der den Typ anzeigt:



Wenn das Script einen Fehler enthält, unterringelt der Editor diesen Fehler rot und zeigt den Fehler beim Darüberfahren mit der Maus an:

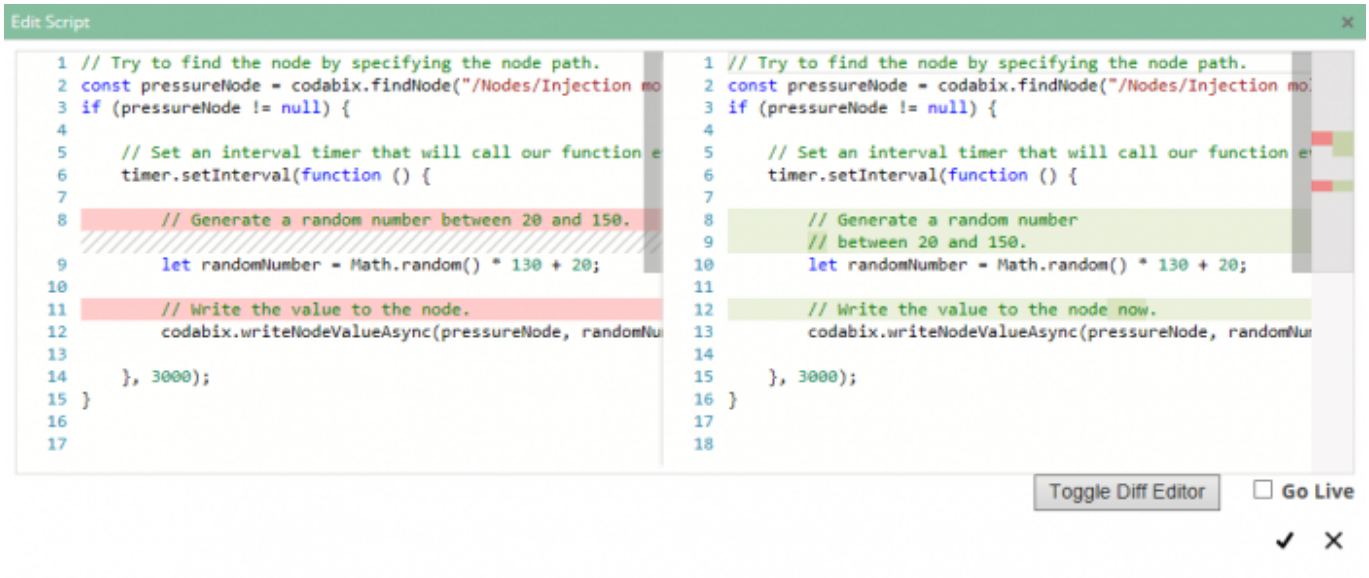


Wenn Sie an einer Stelle im Code rechtsklicken, erscheint ein Kontextmenü mit nützlichen Befehlen. Beispielsweise können Sie alle Referenzen zu einer bestimmten Variable im Code ermitteln (und diese z.B. auch umbenennen):



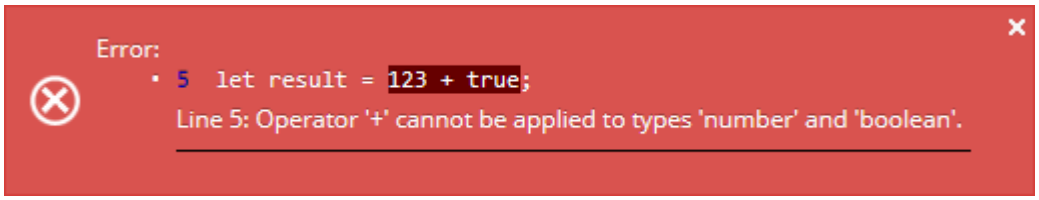
Going Live

Im Script-Editor können Sie den Code für ein Script schreiben und speichern („Entwurf“), ohne dass dieser Entwurf ausgeführt wird. Erst wenn Sie **„Go Live“** auswählen, wird der aktuelle Entwurf als **„Live-Version“** gespeichert und tatsächlich ausgeführt. Dies ermöglicht Ihnen, schrittweise an einem Scriptcode zu arbeiten, ohne die momentan ausgeführte Live-Version zu beeinflussen. Mit dem Button „Toggle Diff Editor“ können Sie zu einem Diff-Editor wechseln, der einen Vergleich der Änderungen zwischen der Live-Version und dem aktuellen Entwurf anzeigt.



Sobald Sie mit der Editierung des Entwurfs fertig sind, wählen Sie die Checkbox **„Go Live“** aus und klicken auf den Speichern-Button. Dadurch wird der aktuelle Entwurf zur Live-Version, sodass dieser tatsächlich ausgeführt wird.

Wenn ihr Script zur Entwurfszeit einen Fehler enthält, während Sie versuchen, dies als Live-Version zu speichern, erscheint eine Dialogbox mit dem Fehler:



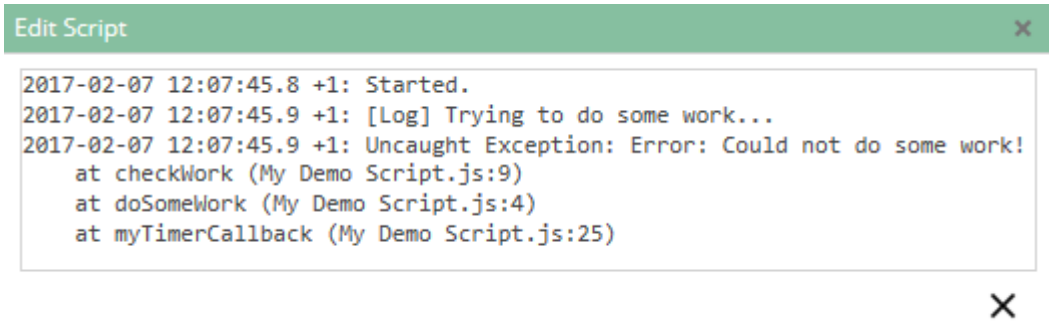
Andernfalls wird der Scripteditor geschlossen und der neue Scriptcode nach ein paar Sekunden ausgeführt.

Nützliche Tastenkombination

- **Strg+F:** Suchen/Ersetzen
- **Strg+F2:** Umbenennen (scriptweit)
- **Umschalt+F12:** Alle Verweise suchen
- **Strg+F12:** Gehe zu Definition
- **Strg+K, Strg+C:** Auswahl auskommentieren
- **Strg+K, Strg+U:** Auskommentierung der Auswahl aufheben

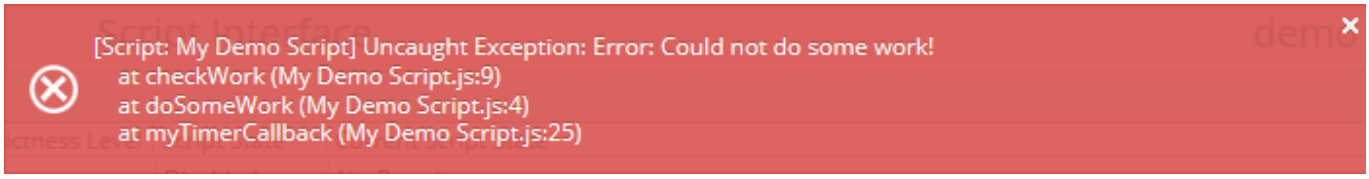
Anzeigen des Scriptprotokolls

Jedem Script ist eine Logdatei zugeordnet. Sobald ein Script gestartet wurde (oder eine unbehandelte Ausnahme aufgetreten ist), wird ein Eintrag in die Logdatei gemacht. Zusätzlich können Sie einen Logeintrag direkt aus dem Scriptcode heraus über den Aufruf `logger.log()` erstellen.
Wenn Sie auf den Script-Log-Button ( ) klicken, erscheint ein Dialogfenster mit dem Inhalt der Logdatei. Falls beispielsweise das Script gestartet wurde, aber eine unbehandelte Ausnahme in einem Callback auftrat, könnte das Log wie folgt aussehen:



Wenn eine Ausnahme auftritt, enthält der Logeintrag eine Stapelnachverfolgung (Stacktrace), in dem die Zeilennummern des Scripts angezeigt werden (nach dem Doppelpunkt), die angeben, an welcher Stelle im Code die entsprechenden Funktionen zum Zeitpunkt des Auftretens der Ausnahme ausgeführt wurden.

Beachten Sie: Wenn eine unbehandelte Ausnahme auftritt, wird diese auch im Runtime Log angezeigt:



Scriptcode schreiben

JavaScript-Grundlagen

Im Folgenden finden Sie eine kurze Zusammenfassung der JavaScript-Grundlagen. Für ein detaillierteres Tutorial besuchen Sie bitte den JavaScript Guide auf MDN.

In einem Script können Sie Variablen, die Werte aufnehmen können, mit `let` und `const` deklarieren (`const` bedeutet, dass die Variable nicht verändert werden kann). Sie können Werte über den „`=`“-Operator zuweisen (wohingegen „`==`“ zum Prüfen auf Gleichheit verwendet wird):

```
let anzahl = 123;
const meinText = "Hallo Welt!";
```

JavaScript unterstützt eine Reihe an grundlegenden Werttypen:

- **`number`**: Ein Number (Gleitkommazahl mit doppelter Genauigkeit) kann sowohl Ganzzahlen als auch Dezimalzahlen speichern. Sie können Numbers für Berechnungen verwenden, z.B.:

  ```
  let ergebnis = (2 + 0.5) * 3;   // 7.5
  ```

- **`boolean`**: Ein Boolean ist entweder `true` (wahr) oder `false` (falsch). Ein Boolean entsteht beispielsweise als Ergebnis eines Vergleichs und kann für Kontrollfluss-Anweisungen wie `if`, `while` usw. verwendet werden.
- **`string`**: Ein String kann aus einer beliebigen Anzahl an Zeichen (Characters) bestehen und wird verwendet, um Text zu speichern. Strings können mit dem „`+`“-Operator zusammengefügt werden:

  ```
  let anzeigeText = "Das Ergebnis von 5+6 ist " + (5+6);   // "Das Ergebnis von 5+6 ist 11"
  ```

- `object`: Ein Objekt speichert Eigenschaften („Properties“), die aus einem Schlüssel (String) und einem Wert (beliebiger Typ) bestehen. So enthält beispielsweise das UKI-4.0 -Objekt Eigenschaften, die auch Methoden sind, wie `findNode`. Auf Objekt-Eigenschaften wird meist über die Punkt-Notation (`.`) zugegriffen (UKI-4.0 `.findNode(…)`, UKI-4.0 `.rootNode`, …).

Sie können Kontrollfluss-Anweisungen verwenden, um Vergleiche durchzuführen:

```
let ergebnis = "Der Wert ist "
if (anzahl > 200) {
    ergebnis += "größer als";
}
else if (anzahl < 200) {
    ergebnis += "kleiner als";
}
else {
    ergebnis += "gleich";
}
ergebnis += " 200.";
```

Sie können Funktionen erstellen, die wiederverwendbaren Code enthalten. Beispielsweise könnten Sie eine Funktion erstellen, die den Durchschnitt aus zwei numerischen Werten berechnet:

```
function durchschnitt(wert1, wert2) {
    return (wert1 + wert2) / 2;
}

// Berechne den Durchschnitt aus 100 und 250 und schreibe ihn ins Script-Log
logger.log("Der Durchschnitt aus 100 und 250 ist " + durchschnitt(100, 250) + ".");
```

Wenn Sie diesen Code ausführen, schreibt dieser in das Script-Log soetwas wie:

```
2016-09-28 14:57:41.7 Z: [Log] Der Durchschnitt aus 100 und 250 ist 175.
```

Script API

Das Script-Interface stellt die folgenden API-Namespaces zur Verfügung, die in einem Script genutzt werden können:

- UKI-4.0 : Enthält die UKI-4.0 -spezifische Funktionalität, z.B. um auf Nodes zuzugreifen und diese zu verändern.
- `timer`: Enthält Methoden um einen Timer zu erstellen, welcher eine von Ihnen im Script definierte Funktion (regelmäßig) nach einer angegebenen Zeit aufruft.
- `logger`: Enthält eine `log`-Methode, mit der Sie in das Script-Log schreiben können.
- `storage`: mit dem Storage-Objekt können Sie Informationen speichern, die auch nach einem Neustart des Scripts erhalten bleiben.
- `io`: Unterstützt I/O-Vorgänge, z.B. Dateizugriff.
- `net`: Unterstützt netzwerkspezifische Vorgänge, z.B. das Registrieren von HTTP-Handlern.
- `runtime`: Enthält Funktionen zum Interagieren mit der Script-Laufzeitumgebung.

Beachten Sie: Der Script-Editor unterstützt **IntelliSense**, sodass Sie die im UKI-4.0 -Namespace vorhandenen Methoden sehen können, indem Sie UKI-4.0 `.` schreiben (beachten Sie den Punkt nach

„UKI-4.0 “). Auch wenn eine Methode ein Objekt (wie eine Node) zurückgibt, können Sie wiederrum einen Punkt eintippen, um zu sehen welche Methoden dieses hat.

Auf UKI-4.0UKI-4.0 UKI-4.0UKI-4.0 zugreifen

Einen Node finden und dessen Wert protokollieren

Nehmen wir an, dass Sie UKI-4.0 mit dem „Demo-Data (Continuous)“-Plugin installiert haben und nun auf die Node **Nodes → Demo-Data → Temperature** zugreifen möchten. Dazu müssen Sie zuerst den Node-Path oder den Identifier der Node abrufen. Öffnen Sie dazu bitte in der UKI-4.0 Webkonfiguration die Node-Ansicht, wählen Sie die entsprechende Node aus und klicken Sie auf das **Access**-Symbol (🔑). Anschließend kopieren Sie bitte den „Absolute Node Path“. Wir übergeben diesen Pfad dann an die UKI-4.0 `.findNode()`-Methode sowie den Parameter `true`, damit die Methode eine Ausnahme wirft, falls die Node nicht gefunden werden konnte:

```
// Finde die "Temperature"-Node und überprüfe, ob die Node
// einen Wert hat.
const temperatureNode = UKI-4.0.findNode("/Nodes/Demo-Nodes/Temperature", true);
if (temperatureNode.value != null) {
    // OK, Node hat einen Wert. Nun protokollieren wir diesen Wert.
    logger.log("Aktuelle Temperatur: " + temperatureNode.value.value);
}
```

Ihr Script-Log könnte dann wie folgt aussehen:

```
2016-09-28 15:08:45.2 Z: Started.
2016-09-28 15:08:45.3 Z: [Log] Aktuelle Temperatur: 71
2016-09-28 15:08:45.3 Z: Stopped.
```

Allerdings wird in diesem Beispiel nur ein einziger Wert protokolliert. Der Grund dafür ist, dass nach dem Starten des Scripts der Code ausgeführt wird, welcher die Node sucht und deren Wert protokolliert, doch danach ist das Script zu Ende.

Wenn wir nun den Wert nicht nur einmal, sondern alle 5 Sekunden protokollieren möchten, können wir dies erreichen, indem wir einen Timer erstellen und diesem eine Funktion mitgeben, welche dieser regelmäßig nach einem Intervall aufruft (Beachten Sie: Für Callbacks verwenden Sie am besten statt `function () {...}` eine **Fat-Arrow-Function**: `() => {...}`).

```
// Finde den "Temperature"-Node.
const temperatureNode = UKI-4.0.findNode("/Nodes/Demo-Nodes/Temperature", true);

// Nun erstellen wir einen Timer, der den Wert des
// Nodes alle 5 Sekunden protokolliert.
const interval = 5000;
timer.setInterval(() => {

    // Wenn der Node einen Wert hat, protokollieren wir diesen.
    if (temperatureNode.value != null) {
        logger.log("Aktuelle Temperatur: " + temperatureNode.value.value);
    }

}, interval);
```

Wenn Sie dieses Script ausführen, könnte Ihr Script-Log wie folgt aussehen:

```
2016-09-28 15:15:42.6 Z: Started.
2016-09-28 15:15:47.6 Z: [Log] Aktuelle Temperatur: 70
2016-09-28 15:15:52.6 Z: [Log] Aktuelle Temperatur: 75
2016-09-28 15:15:57.6 Z: [Log] Aktuelle Temperatur: 63
2016-09-28 15:16:02.6 Z: [Log] Aktuelle Temperatur: 71
```

Nodeereignisse

Das obige Beispiel verwendet einen Timer, der eine Script-Funktion regelmäßig aufruft. Es ist jedoch auch möglich, sich für bestimmte Ereignisse (Events) eines Nodes zu registrieren:

- **ValueChanged:** Wird ausgelöst, wenn ein Wert in den Node geschrieben wurde (`value`-Eigenschaft).
  Beachten Sie: Dieses Event wird auch ausgelöst, wenn der neue Wert gleich dem alten Wert ist. Um festzustellen, ob sich der Wert tatsächlich geändert hat, können Sie die `isValueChanged`-Property des Listener-Arguments prüfen.
- **PropertyChanged:** Wird ausgelöst, wenn sich eine Eigenschaft des Nodes (außer der `value`-Eigenschaft) ändert, wie z.B. `name`, `displayName` usw.
- **ChildrenChanged:** Wird ausgelöst, wenn unterhalb des aktuellen Nodes eine oder mehrere Kindnodes hinzugefügt oder entfernt werden.

Sie können auf Events reagieren, indem Sie einen **Eventlistener** (Callback) zu dem Node hinzufügen, dessen Ereignis Sie behandeln möchten.
Beispiel:

```
// Finde den "Temperature"-Node und füge einen Handler für das "ValueChanged"-Ereignis
hinzu.
const temperatureNode = UKI-4.0 .findNode("/Nodes/Demo-Nodes/Temperature", true);

temperatureNode.addValueChangedEventListener(e => {

    // Protokolliere den alten und den neuen Wert des Nodes.
    logger.log("Alter Wert: " + (e.oldValue && e.oldValue.value)
        + ", Neuer Wert: " + e.newValue.value);

});
```

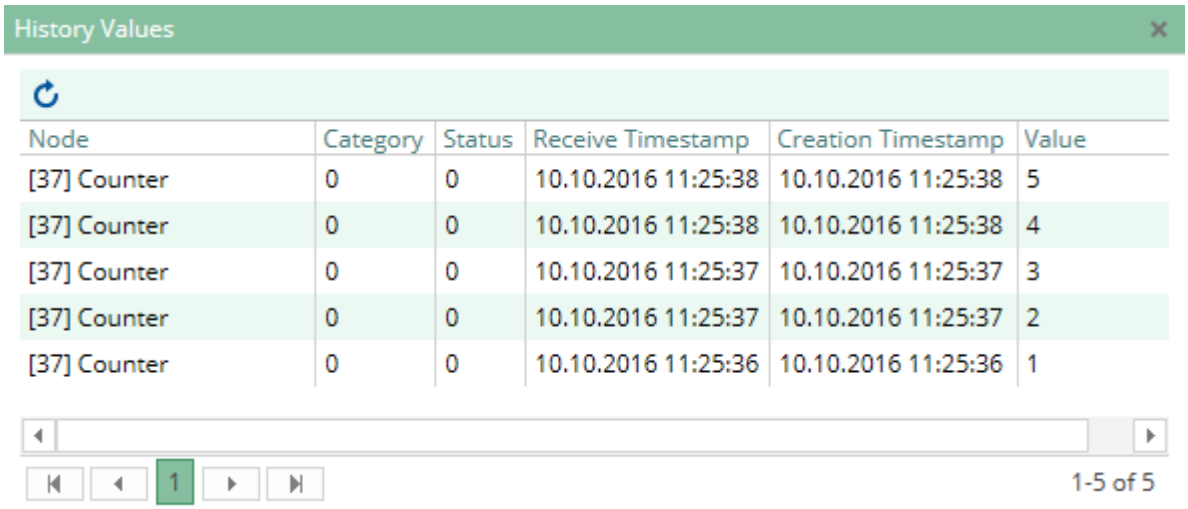
Beachten Sie: Innerhalb eines Node-Eventlisteners können Sie keinen Nodewerte (synchron) lesen oder schreiben oder andere Änderungen an Nodes vornehmen. Wenn Sie dies tun möchten, benutzen Sie bitte UKI-4.0 `.scheduleCallback()`, um einen Callback anzugeben, der so schnell wie möglich nach dem Verlassen des Eventlisteners ausgeführt werden soll.

Werte in einen Node schreiben

Sie können aus einem Script heraus auch Werte in einen Node schreiben. Für das folgende Beispiel wählen Sie bitte in der Node-Ansicht der UKI-4.0 Webkonfiguration den Node **„Nodes“** aus und erstellen einen Datenpunkt-Node mit dem Namen „Zähler“, und wählen für die Option „History Options“ **on Value Change** aus. Anschließend erstellen Sie eine Script mit diesem Code:

```
const zählerNode = UKI-4.0 .findNode("/Nodes/Zähler", true);

// Deklariere eine Zähler-Variable.
let meinZähler = 0;

// Erstelle einen Callback, der den Zähler erhöht und den aktuellen
// Wert in einen Node schreibt, solange bis der Zähler 5 ist.
let timerID = timer.setInterval(() => {
    meinZähler = meinZähler + 1;
    UKI-4.0 .writeNodeValueAsync(zählerNode, meinZähler);

    if (meinZähler == 5)
        timer.clearInterval(timerID);
}, 500);
```

Wechseln Sie nun wieder zur Node-Ansicht, wählen den „Zähler“-Node aus und öffnen Sie die historischen Werte ( ):



Hier können Sie sehen, dass die Werte 1, 2, 3, 4, 5 in die Node in einem Abstand von 0,5 Sekunden geschrieben wurden.

Beachten Sie: Es gibt auch einen einfacheren Weg, diesen Code zu schreiben (ohne Callbacks), und zwar über eine asynchrone Funktion (async function), wie es im nächsten Kapitel gezeigt wird.

Asynchrone Funktionen

Das Beispiel aus dem vorherigen Kapitel erstellt einen Timer, der einen Callback aufruft. Solcher Code kann jedoch schnell unübersichtlich werden, wenn man komplexere Bedingungen hat. **Asynchrone Funktionen** (Async Functions) ermöglichen Ihnen, diesen Code zu vereinfachen, da er wie synchroner Code (ohne Callbacks) aussieht. UKI-4.0 stellt einige asynchrone Funktionen bereit, die Sie daran erkennen können, dass ihr Name auf `Async` endet.
Beispielsweise kann nun der obige Code durch die Verwendung der Funktion `timer.delayAsync` so umgeschrieben werden:

```
const zählerNode = UKI-4.0 .findNode("/Nodes/Zähler", true);

// Schreibe die Werte 1 bis 5 in den Node, und warte dazwischen jeweils 0,5 Sekunden.
for (let i = 1; i <= 5; i++) {
    await UKI-4.0 .writeNodeValueAsync(zählerNode, i);
    await timer.delayAsync(500);
}
```

Beachten Sie die Verwendung des Schlüsselwortes `await`. **Await** (Abwarten) bedeutet soetwas wie „Unterbrich die Ausführung an der aktuellen Stelle solange, bis die asynchrone Operation der Funktion abgeschlossen ist“. Die `delayAsync`-Funktion gibt hier ein `Promise`-Objekt („Versprechen“) zurück, welches nach dem Verstreichen von 0,5 Sekunden aufgelöst wird. Sobald das `Promise` aufgelöst ist, wird die Ausführung fortgesetzt. Andere asynchrone Funktionen geben ebenfalls `Promise`-Objekte zurück, welche ggf. mit Werten aufgelöst werden können.

Wenn Sie `await` weglassen, würde Ihr Code nicht 0,5 Sekunden warten, sondern sofort mit dem nächsten Schleifendurchlauf weitermachen. Bitte beachten Sie aber, dass während des „Abwartens“ einer asynchronen Operation mit `await` anderer Code ausgeführt werden kann; beispielsweise könnte ein Event einer Node ausgelöst werden, während Ihr Code an der `await`-Position wartet.

Im obigen Beispiel wird `await` auch benutzt, um `writeNodeValueAsync` abzuwarten. Das Schreiben von Nodewerten kann einige Zeit dauern, wenn die Node an ein Gerät (wie eine S7) gekoppelt ist. In diesem Fall würde die Ausführung solange warten, bis der Wert tatsächlich in das Gerät geschrieben wurde. Wenn Sie darauf nicht warten möchten, können Sie das `await` hier auch weglassen (wobei Sie in dem Fall den `void`-Operator vor den Funktionsaufruf setzen müssen, um dem Compiler zu signalisieren, dass das zurückgegebene `Promise`-Objekt absichtlich verworfen wurde).

Wenn Sie nun allerdings diesen Code direkt ausprobieren, wird er noch nicht funktionieren, denn damit man `await` benutzen kann, muss die umgebende Funktion mit dem `async`-Schlüsselwort gekennzeichnet sein. Wenn diese umgebende, asynchrone Funktion die „Hauptfunktion“ ist, umschließen Sie diese am besten mit einem Aufruf von `runtime.handleAsync()`, damit unbehandelte Ausnahmen nicht lautlos „verschluckt“ werden:

Async-Template.js

```
runtime.handleAsync(async function () {

    // Ihr Code...

} ());
```

Hier ist ein komplettes Beispiel eines Scripts, das asynchrone Funktionen benutzt:

```
runtime.handleAsync(async function () {

    for (let i = 0; i < 10; i++) {
        logger.log("Durchlauf " + i);
        await timer.delayAsync(1000);
    }

} ());
```

Beachten Sie: Wenn Sie eine asynchrone Funktion als Callback für einen Eventlistener verwenden möchten, umschließen Sie diese am besten ebenfalls in einen `runtime.handleAsync`-Aufruf wie im

folgenden Beispiel, in welchem das Ereignis behandelt wird, wenn die Node einen neuen Wert bekommt:

```
const meineNode = UKI-4.0 .findNode("/Nodes/A", true);

myNode.addValueChangedEventListener(() => runtime.handleAsync((async () => {
    // Führe asynchronen Code aus...
}) ()));
```

Nodewerte mit einem synchronen Lesevorgang lesen

Ein weiteres Beispiel einer asynchronen Funktion ist UKI-4.0 `.readNodeValuesAsync()`. Diese Methode führt einen synchronen Lesevorgang auf dem Gerät aus, und das Lesen von Werten vom Gerät kann einige Zeit dauern. Deshalb sollten Sie `await` verwenden, um die Ausführung an der aktuellen Stelle zu unterbrechen, bis die gelesenen Werte eintreffen:

```
runtime.handleAsync(async function () {

    const node1 = UKI-4.0 .findNode("/Nodes/A", true);
    const node2 = UKI-4.0 .findNode("/Nodes/B", true);

    // Führe einen synchronen Lesevorgang aus, und warte ab, bis die Werte vom Gerät
eintreffen.
    let [nodeValue1, nodeValue2] = await UKI-4.0 .readNodeValuesAsync(node1, node2);

    logger.log("Node1 Wert: " + (nodeValue1 && nodeValue1.value)
        + ", Node2 Wert: " + (nodeValue2 && nodeValue2.value));

} ());
```

Dateizugriff

Die Namespaces **`io.file`**, **`io.directory`** und **`io.path`** enthalten Methoden und Klassen zum Arbeiten mit Dateien und Verzeichnissen. Beispielsweise können Sie Textdateien lesen und schreiben, Dateien kopieren, verschieben oder löschen, sowie alle Dateien in einem Verzeichnis auflisten.
Beachten Sie, dass der Dateizugriff den Einschränkungen der **Access Security** unterliegt, die in den UKI-4.0 Projekteinstellungen definiert wurden.

Die meisten I/O-Operationen sind als **asynchrone Funktionen** implementiert, die ein `Promise`-Objekt zurückgeben. Der Grund dafür ist, dass die I/O-Operationen einige Zeit in Anspruch nehmen können, bis sie abgeschlossen sind (dies hängt auch vom Dateisystem ab). Damit UKI-4.0 währenddessen nicht blockiert ist, werden die I/O-Operationen im Hintergrund ausgeführt. Sie können diese in asynchronen Funktionen über das `await`-Schlüsselwort aufrufen.

Beachten Sie: Unter Windows 10 Version 1511 und älter (sowie Windows Server 2012 R2 und älter), z.B. unter Windows 7, ist die **maximale Pfadlänge** auf **260 Zeichen** (`MAX_PATH`) begrenzt.
Unter Windows 10 Version 1607 und höher (sowie Windows Server 2016 und höher) können Sie längere Pfade benutzen. Dazu müssen Sie jedoch erst die Einstellung „Lange Win32-Pfade aktivieren“ in den Windows-Gruppenrichtlinien aktivieren, siehe Enabling Win32 Long Path Support.

Grundlegende Dateioperationen

Aufzählen der Dateien im UKI-4.0 **Data „log“-Verzeichnis:**

```
runtime.handleAsync(async function () {

    // Rufe den Pfad zum UKI-4.0 "log"-Verzeichnis ab. Dafür benutzen wir die durch UKI-4.0
    // definierte Umgebungsvariable "%UKI-4.0 ProjectDir%".
    // Mit combinePath können Pfad-Elemente unabhängig vom Betriebssystem kombiniert werden.
    const UKI-4.0 LogDir = io.path.combinePath(
            runtime.expandEnvironmentVariables("%UKI-4.0 ProjectDir%"), "log");
    const fileList = await io.directory.getFilesAsync(UKI-4.0 LogDir);

    let result = "";
    for (let file of fileList) {
        result += "File: " + file + "\n";
    }

    logger.log("Files in " + UKI-4.0 LogDir + ":\n" + result);

} ());
```

Das Ergebnis könnte wie folgt aussehen:

Edit Script

```
2017-01-17 13:08:02.7 Z: [Log] Files in C:\Users\developer4\Desktop\NewCodabixData6\log:
File: OPC-UA Server Interface.Default Channel.log
File: S7 TCP-IP Device.New Channel 1.log
```

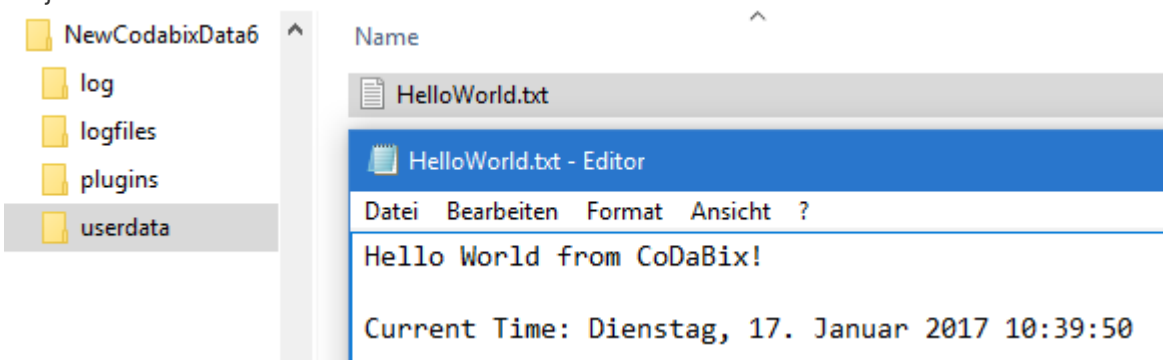
- `io.path.combinePath()`: Kombiniert zwei oder mehr Zeichenfolgen zu einem Pfad, sodass dies unabhängig vom Betriebssystem funktioniert. Beispielsweise ist das Pfad-Trennzeichen unter Windows `\`, während es unter Linux `/` ist. Diese Methode nutzt automatisch das richtige Trennzeichen zum Kombinieren der Zeichenfolgen.
- `io.directory.getFilesAsync()`: Gibt ein `string[]`-Array zurück, welches die Namen der Dateien des angegebenen Verzeichnisses enthält (`io.directory.getDirectoriesAsync()` gibt entsprechend die Namen der Unterverzeichnisse zurück).
- `io.file.copyFileAsync()`: Kopiert eine Datei.
- `io.file.moveFileAsync()`: Verschiebt eine Datei oder benennt eine Datei um.
- `io.file.deleteFileAsync()`: Löscht eine Datei.
- `runtime.expandEnvironmentVariables()`: Ersetzt den Namen jeder Umgebungsvariablen (eingeschlossen in zwei `%`-Zeichen) im angegebenen String durch deren Wert. UKI-4.0 definiert die folgenden Umgebungsvariablen zusätzlich zu denen des Betriebssystems:
  - UKI-4.0 `ProjectDir`: Enthält den Pfad zum aktuell verwendeten UKI-4.0 -Projektverzeichnis.
  - UKI-4.0 `InstallDir`: Enthält den Installationspfad von UKI-4.0 .

Lesen und Schreiben von Textdateien

Eine Textdatei in einem Schritt schreiben:

```
runtime.handleAsync(async function () {

    // Erstelle einen String und schreibe ihn in eine Textdatei (HelloWorld.txt).
    let filePath =
io.path.combinePath(runtime.expandEnvironmentVariables("%UKI-4.0 ProjectDir%"),
            "userdata", "HelloWorld.txt");
    let content = "Hello World from UKI-4.0 !\r\n\r\n" +
            "Current Time: " + new Date().toLocaleString();

    await io.file.writeAllTextAsync(filePath, content);

} ());
```

Dieses Script erstellt die Textdatei `HelloWorld.txt` im `userdata`-Verzeichnis Ihres UKI-4.0 -Projektzeichnis. Die Datei könnte dann so aussehen:



Durch den Aufruf von `io.file.writeAllTextAsync()` wird die Datei ein einem Schritt geschrieben (und mit `io.file.readAllTextAsync()` wird sie in einem Schritt gelesen). Sie können Textdateien jedoch auch zeilenweise lesen oder schreiben, wie im folgenden Beispiel.

Eine Textdatei zeilenweise lesen:

```
runtime.handleAsync(async function () {

    // Wir möchten die aktuelle UKI-4.0 -Runtime-Logdatei auslesen.
    const runtimeLogDir = io.path.combinePath(
            runtime.expandEnvironmentVariables("%UKI-4.0 ProjectDir%"), "log");
    const runtimeLogFiles = await io.directory.getFilesAsync(runtimeLogDir);

    // Die zurückgegebenen Dateien sind aufsteigend nach ihrem Namen sortiert; daher
    // verwenden wir den letzten Eintrag, um an die heutige Logdatei zu kommen.
    const logFile = io.path.combinePath(runtimeLogDir,
            runtimeLogFiles[runtimeLogFiles.length - 1]);

    // Öffne die Datei mit einem Reader und lies die ersten 5 Zeilen.
    let result = "";
    let reader = await io.file.openReaderAsync(logFile);
    try {
        let lines = await reader.readLinesAsync(5);
        for (let i = 0; i < lines.length; i++) {
            // Füge die Zeile an den Result-String an.
            result += "\n[" + (i + 1) + "]: " + lines[i];
        }
    }
    finally {
        // Stelle sicher, dass der Reader geschlossen wird, wenn wir fertig sind.
        await reader.closeAsync();
    }

    // Logge den Result-String.
    logger.log(result);

} ());
```

Dieser Code liest die ersten 5 Zeilen der aktuellen UKI-4.0 -Runtime-Logdatei aus und schreibt diese ins Script-Log:

Edit Script

```
2017-01-17 15:23:36.1 +1: [Log]
[1]: 2017-01-17 09:11:20.7 +01:00: [Info] CoDaBix® Engine starting...
[2]:     • CoDaBix® Version:      0.10.1
[3]:     • OS Version:            Windows 10 Anniversary Update (Version 1607, RS1) [10.0.14393]
[4]:     • .NET Version:          .NET Framework 4.6.2 [4.6.01586]; CLR: v4.0.30319
[5]:     • Runtime Architecture:  AMD64 (64-bit)
2017-01-17 15:23:36.0 +1: Started.
```

HTTP-Handler

Der Namespace `net` bietet Methoden zum Registrieren von **HTTP-Handler**, womit Sie eine Script-Funktion angeben können, die immer dann aufgerufen wird, wenn ein Client einen HTTP-Request an den UKI-4.0 -Webserver sendet. Damit können Sie beispielsweise HTML-Seiten, ähnlich wie bei PHP oder ASP.NET, dynamisch generieren. Außerdem können Sie damit auch **WebSocket-Verbindungen** entgegennehmen (siehe nächstes Kapitel).

Um einen HTTP-Handler zu registrieren, müssen Sie die `net.registerHttpRoute()`-Methode aufrufen, die als Argumente einen Pfad sowie einen Callback erwartet. Der Callback wird dann jedes Mal, wenn ein HTTP-Request für den URL-Pfad `/scripthandlers/<path>` eintrifft, aufgerufen, wobei `<path>` den registrierten Pfad darstellt.

Beachten Sie, dass ein bestimmter Pfad über alle Scripts hinweg zur gleichen Zeit nur einmal registriert

sein kann. Wenn ein anderes Script bereits einen Handler für den gleichen Pfad registriert hat, wirft die Methode eine Ausnahme.
Beachten Sie außerdem, dass der Pfad immer „roh“, also URL-kodiert angegeben werden muss. Wenn der User beispielsweise im Browser als Adresse `/scripthandlers/März` eingeben können soll, müssten Sie als Pfad `M%C3%A4rz` angeben.

Der Callback muss hierbei eine Funktion sein, die ein `Promise`-Objekt zurückgibt (das passiert automatisch, wenn Sie eine **asynchrone Funktion** erstellen). Beim Eintreffen eines HTTP-Requests wird diese Funktion dann aufgerufen, und der HTTP-Request bleibt solange aktiv, bis das Promise-Objekt „fulfilled“ ist (also die asynchrone Funktion zurückkehrt). Der Callback enthält als Parameter ein `net.HttpContext`-Objekt, das die Properties `request` und `response` enthält, um entsprechend auf Objekte für die Anfrage (Request) des Clients und die Antwort (Response) an den Client zuzugreifen.

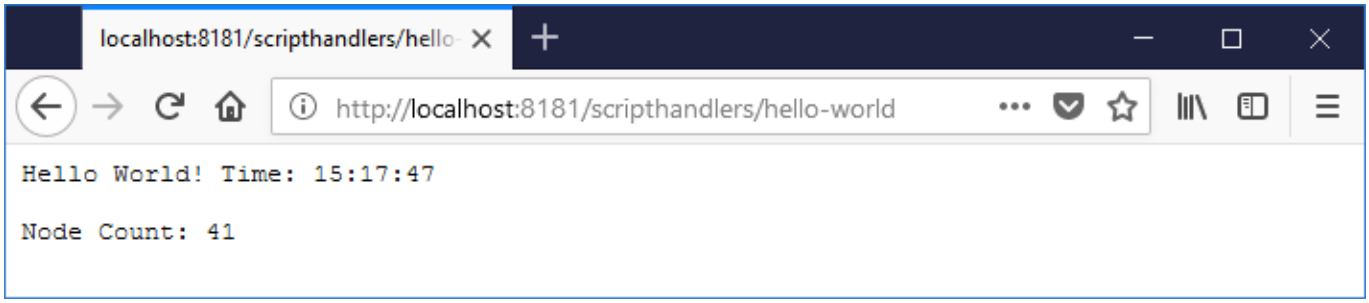
Wenn der Callback eine Ausnahme wirft (oder das zurückgegebene `Promise` rejectet), wird der Request abgebrochen und eine Warnung ins Runtime-Log geschrieben, das Script läuft jedoch weiter.

Textseite dynamisch generieren

Nehmen wir an, Sie möchten die aktuelle Uhrzeit sowie die derzeitige Anzahl an UKI-4.0 -Nodes in einer Textseite ausgeben, wenn der Benutzer im Browser `http://localhost:8181/scripthandlers/hello-world` auf dem lokalen PC eingibt (vorausgesetzt, der lokale Port für den UKI-4.0 -Webserver ist auf den Standardwert 8181 eingestellt). Im folgenden Scriptcode wird dazu ein HTTP-Handler für diesen Pfad registriert, der beim Aufruf einen Text mit den entsprechenden Informationen generiert:

```
runtime.handleAsync(async function () {

    net.registerHttpRoute("hello-world", async ctx => {
        let response = ctx.response;

        // Erstelle einen Text mit dem aktuellen Datum und der Anzahl an UKI-4.0 -Nodes.
        let text = "Hello World! Time: " + new Date().toLocaleTimeString() + "\n\n" +
            "Node Count: " + countNodes(UKI-4.0 .rootNode.children);

        // Setze den Content-Type auf das Textformat, damit der Browser weiß,
        // wie er das Dokument anzeigen soll.
        response.contentType = "text/plain";

        // Schließlich schreiben wir den generierten Text in die Response.
        await response.writeBodyCompleteAsync(text);
    });

    function countNodes(nodes: UKI-4.0 .Node[]): number {
        let count = nodes.length;
        for (let node of nodes) {
            count += countNodes(node.children);
        }
        return count;
    }

}());
```

Wenn Sie diese URL nun im Browser aufrufen, erhalten Sie eine Ausgabe ähnlich wie dieser Screenshot zeigt:



Wenn Sie die Seite (mit F5) aktualisieren, sehen Sie dann immer wieder die aktuelle Uhrzeit angezeigt.

Das Script verwendet die Methode `writeBodyCompleteAsync()` des `response`-Objektes, um den generierten Text an den Browser zu senden (als Textkodierung wird UTF-8 verwendet). Es wird empfohlen, diese Methode statt `writeBodyAsync()` aufzurufen, wenn Sie den kompletten Ausgabetext im Script erstellen können. `writeBodyAsync()` hingegen können Sie verwenden, wenn Sie immer nur kleinere Teile des Response-Texts generieren und sofort an den Client schicken möchten.

HTML-Seite mit Eingabeformular generieren

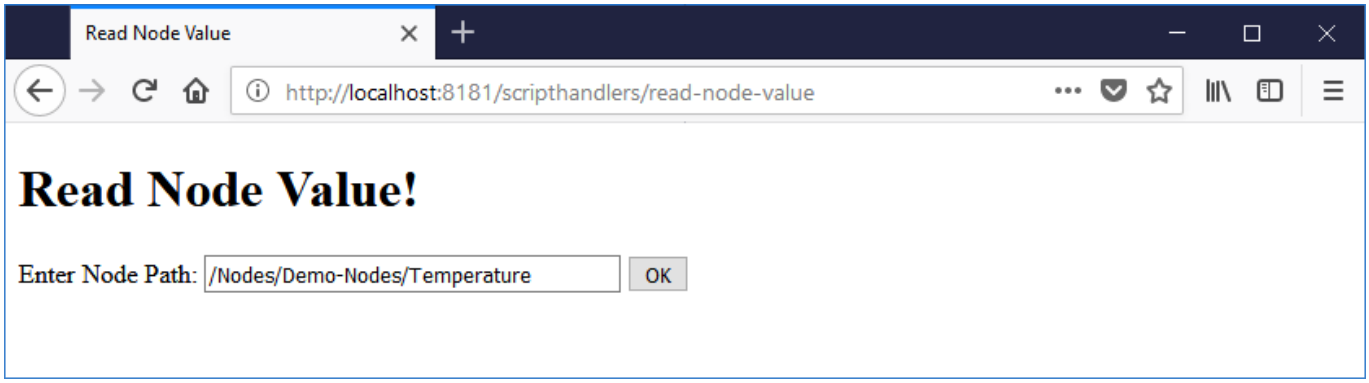
Nehmen wir nun an, Sie möchten eine HTML-Seite erstellen, auf der der Benutzer einen Nodepfad eingeben kann. Wenn er das Formular abschickt, soll der Wert des Nodes in einem synchronen Lesevorgang gelesen und dann ausgegeben werden. Dies wird in folgendem Scriptcode gemacht:

```
runtime.handleAsync(async function () {

    const readNodeHandlerPath = "read-node-value";
    net.registerHttpRoute(readNodeHandlerPath, async ctx => {
        let request = ctx.request;
        let response = ctx.response;

        // Erstelle eine HTML-Seite mit einem Formular zum Eingeben
        // eines Node-Pfads. Wenn dieser eingegeben wurde, suchen wir den
        // Node und starten einen synchronen Lesevorgang.

        // Nachschauen, ob das Formular bereits abgeschickt wurde. Wenn nicht,
        // schreiben wir einen Beispielpfad in die Textbox.
        const inputNodePathName = "nodePath";
        const inputNodePathValue = request.queryString[inputNodePathName];
        let resultMessage = "";

        if (inputNodePathValue != undefined) {
            // Das Formular wurde abgeschickt, daher suchen wir nun den Node heraus.
            let node = UKI-4.0.findNode(inputNodePathValue);
            if (!node) {
                // Wir konnten den Node nicht finden.
                resultMessage = `Error! Node with path '${inputNodePathValue}' was not
found!`;
            }
            else {
                // Synchronen Lesevorgang ausführen... (Dies könnte etwas dauern, je nach
Device)
                // Der HTTP-Request bleibt solange aktiv.
                let resultValue = (await UKI-4.0.readNodeValuesAsync(node))[0];
                if (resultValue == null) // Der Node hat noch keinen Wert
                    resultMessage = `Result: Node does not have a value!`;
                else
                    resultMessage = `Value of Node '${inputNodePathValue}':
${resultValue.value} ${node.unit || ""}`;
```

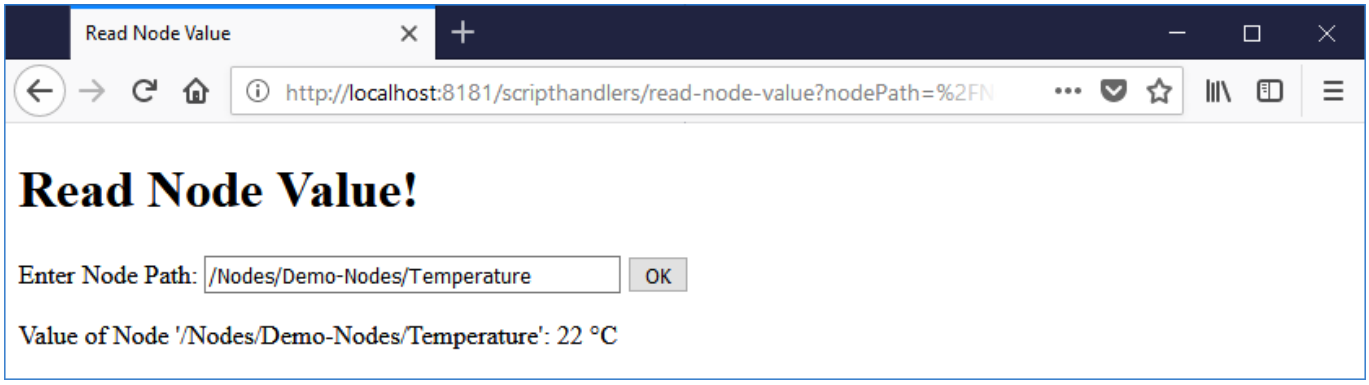
```
            }
        }

        let html = `<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8" />
    <title>Read Node Value</title>
</head>
<body>
    <h1>Read Node Value!</h1>
    <form method="get" action="${xml.encode(readNodeHandlerPath)}">
        Enter Node Path:
        <input type="text" name="${xml.encode(inputNodePathName)}" style="width: 250px;"
            value="${xml.encode(inputNodePathValue == undefined ? "/Nodes/Demo-
Nodes/Temperature" : inputNodePathValue)}" />
        <button>OK</button>
    </form>
    <p>
        ${xml.encode(resultMessage)}
    </p>
</body>
</html>`;

        // Setze den Content-Type auf HTML, und schreibe das generierte HTML in die
Response.
        response.contentType = "text/html";
        await response.writeBodyCompleteAsync(html);
    });

}());
```

Beachten Sie: Das Script nutzt die Methode `io.util.encodeXml()`, um Strings so zu kodieren, dass diese in HTML- oder XML-Text (bzw. Attributwerten) gefahrlos ausgegeben werden können, um damit keine Angriffsfläche für HTML-Injection bzw. Cross-Site-Scripting (XSS) zu bieten. Dies ist besonders dann wichtig, wenn Strings in einer HTML-Seite ausgegeben werden sollen, die vom Benutzer oder von externen Quellen stammen könnten.

Wenn Sie nun die URL `http://localhost:8181/scripthandlers/read-node-value` im Browser aufrufen, erhalten Sie folgendes Formular angezeigt:



Wenn Sie nun auf den „OK“-Button klicken (und Sie das Demo-Data-Plugin installiert haben), wird der aktuelle Wert der Temperature-Demo-Node angezeigt:

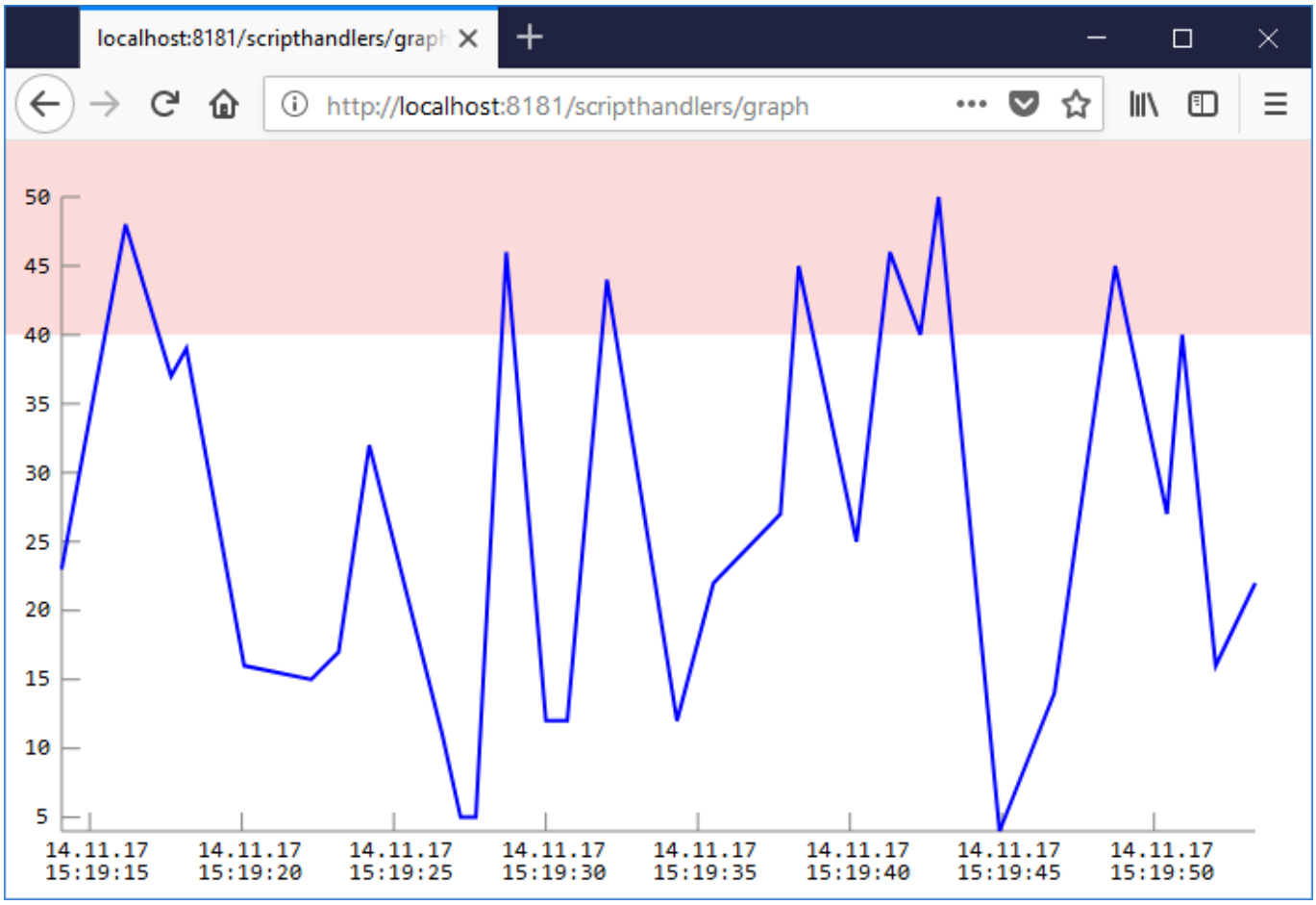


Das Script hat also auf das abgesendete Formular reagiert und den aktuellen Wert des Nodes ausgegeben. Da das Formular die `GET`-Methode benutzt, ist der Parameter in der URL als angehängter Query-String sichtbar: `?nodePath=%2FNodes%2FDemo-Nodes%2FTemperature`

Genauso können Sie nun beispielsweise einen Pfad auf eine S7-Variable, OPC UA Client Variable oder ähnliches eingeben. In diesem Fall wird nicht nur der derzeit gespeicherte Wert der Variablen ausgegeben, sondern durch den Aufruf der UKI-4.0 `.readNodeValuesAsync()`-Methode ein synchroner Lesevorgang gestartet, der den Wert vom Device liest und diesen anschließend in die HTML-Seite ausgibt.

Historische Daten als Diagramm anzeigen

Sie können über den HTTP-Handler auch eine SVG-Grafik generieren, die historische Werte als Diagramm anzeigt (im Beispiel vom Gradient-Demonode):



Dies wird über folgenden Scriptcode ermöglicht:

```
// Utilities for generating a SVG diagram.
namespace SvgLibrary {
```

```
export interface Value {
    date: number,
    value: number
}

type DPoint = [number, number];

const escapeXml = xml.encode;

/**
 * Formats pixel coordinates. A max. precision of 3 should be good enough here.
 * @param x
 */
let pform = (x: number): string => (Math.round(x * 1000) / 1000).toString();

function determineGap(x: number): number {
    let sign = x < 0 ? -1 : 1;
    x = Math.abs(x);
    let tenLog = Math.floor(Math.log10(x));

    if (x > 5 * 10 ** tenLog)
        x = 10 * 10 ** tenLog;
    else if (x > 2 * 10 ** tenLog)
        x = 5 * 10 ** tenLog;
    else if (x > 1 * 10 ** tenLog)
        x = 2 * 10 ** tenLog;
    else
        x = 1 * 10 ** tenLog;

    return x * sign;
}

function formatTwoDigits(values: number[], joinStr: string): string {
    let outStr = "";
    for (let i = 0; i < values.length; i++) {
        if (i > 0)
            outStr += joinStr;
        let str = values[i].toString();
        while (str.length < 2)
            str = "0" + str;
        outStr += str;
    }
    return outStr;
}

export function generateValueChartSvg(values: Value[], width: number, height: number,
    minValue?: number, maxValue?: number, useStairway = false): string {

    // Ensure the array isn't empty.
    if (values.length == 0)
        throw new Error("No history values available.");

    let outStr = "";

    // Determine the maximum and minimum values.
    // Ensure the values are ordered by date.
    values.sort((a, b) => a.date - b.date);

    let minV: number | null = null, maxV: number | null = null;
```

```
        let minD = values[0].date;
        let maxD = values[values.length - 1].date;
        for (let n of values) {
            minV = minV === null ? n.value : Math.min(minV, n.value);
            maxV = maxV === null ? n.value : Math.max(maxV, n.value);
        }
        if (minV == null || maxV == null)
            throw new Error("Could not determine min/max");

        // Ensure that if all values are the same we don't get Infinity/NaN.
        if (maxV - minV < 0.00001 * minV)
            maxV = minV + 0.00001 * minV, minV = minV - 0.00001 * minV;

        const padding = 30;
        let yTop = maxV + (maxV - minV) / (height - 2 * padding) * padding; // 20 pixel
padding
        let yBottom = minV - (maxV - minV) / (height - 2 * padding) * padding;
        let xLeft = minD - (maxD - minD) / (width - 2 * padding) * padding;
        let xRight = maxD + (maxD - minD) / (width - 2 * padding) * padding;

        let convCoords = (coords: DPoint): DPoint =>
            [(coords[0] - xLeft) / (xRight - xLeft) * width,
                (yTop - coords[1]) / (yTop - yBottom) * height];

        // Create the svg and draw the points.
        outStr += `<svg xmlns="http://www.w3.org/2000/svg" version="1.1" baseProfile="full"
width="${width}px" height="${height}px" viewBox="0 0 ${width} ${height}">`;

        let convMax = maxValue == null ? null : convCoords([0, maxValue]);
        let convMin = minValue == null ? null : convCoords([0, minValue]);
        // Draw rects for the min and max values.
        if (convMax != null && convMax[1] >= 0)
            outStr += `<rect x="0" y="0" width="${width}" height="${pform(convMax[1])}"
fill="#fcdada"/>`;
        if (convMin != null && convMin[1] < height)
            outStr += `<rect x="0" y="${pform(convMin[1])}" width="${width}"
height="${pform(height - convMin[1])}" fill="#fcdada"/>`;

        // Draw a line for the x and y axis. We simply draw it at the bottom / left.
        let conv1 = convCoords([minD, minV]);
        let conv2 = convCoords([maxD, minV]);
        outStr += `<line x1="${escapeXml(pform(conv1[0]))}"
y1="${escapeXml(pform(conv1[1]))}" x2="${escapeXml(pform(conv2[0]))}"
y2="${escapeXml(pform(conv2[1]))}" stroke="grey" stroke-width="1"/>`;
        conv1 = convCoords([minD, maxV]);
        conv2 = convCoords([minD, minV]);
        outStr += `<line x1="${escapeXml(pform(conv1[0]))}"
y1="${escapeXml(pform(conv1[1]))}" x2="${escapeXml(pform(conv2[0]))}"
y2="${escapeXml(pform(conv2[1]))}" stroke="grey" stroke-width="1"/>`;

        // Now draw some small lines which indicates the continuous x and y values.
        // We initially use 25 pixel then get the smallest number of 1*10^n, 2*10^n or
5*10^n that is >= our number.
        const fontSize = 12;
        let xLineGap = determineGap(40 / (width - 2 * padding) * (xRight - xLeft));
        let xStart = Math.floor(minD / xLineGap + 1) * xLineGap;
        let yLineGap = determineGap(20 / (height - 2 * padding) * (yTop - yBottom));
        let yStart = Math.floor(minV / yLineGap + 1) * yLineGap;
        for (let x = xStart; x <= maxD; x += xLineGap) {
```

```
            let conv1 = convCoords([x, minV]);
            let conv2 = [conv1[0], conv1[1] - 10];
            outStr += `<line x1="${escapeXml(pform(conv1[0]))}"
y1="${escapeXml(pform(conv1[1]))}" x2="${escapeXml(pform(conv2[0]))}"
y2="${escapeXml(pform(conv2[1]))}" stroke="grey" stroke-width="1"/>`;
            let xDate = new Date(x);
            let textContent1 = formatTwoDigits([xDate.getDate(), xDate.getMonth() + 1,
xDate.getFullYear() % 100], ".");
            let textContent2 = formatTwoDigits([xDate.getHours(), xDate.getMinutes(),
xDate.getSeconds()], ":");
            outStr += `<text x="${escapeXml(pform(conv1[0] - textContent1.length * fontSize
/ 4))}" y="${escapeXml(pform(conv1[1] + 14))}" style="font-family: Consolas, monospace;
font-size: ${fontSize}px;">${escapeXml(textContent1)}</text>`;
            outStr += `<text x="${escapeXml(pform(conv1[0] - textContent2.length * fontSize
/ 4))}" y="${escapeXml(pform(conv1[1] + 14 + fontSize))}" style="font-family: Consolas,
monospace; font-size: ${fontSize}px;">${escapeXml(textContent2)}</text>`;
        }
        for (let y = yStart; y <= maxV; y += yLineGap) {
            let conv1 = convCoords([minD, y]);
            let conv2 = [conv1[0] + 10, conv1[1]];
            outStr += `<line x1="${escapeXml(pform(conv1[0]))}"
y1="${escapeXml(pform(conv1[1]))}" x2="${escapeXml(pform(conv2[0]))}"
y2="${escapeXml(pform(conv2[1]))}" stroke="grey" stroke-width="1"/>`;
            let textContent = y.toString();
            outStr += `<text x="${escapeXml(pform(conv1[0] - 8 - textContent.length *
fontSize / 2))}" y="${escapeXml(pform(conv1[1] + (fontSize / 16 * 10) / 2))}" style="font-
family: Consolas, monospace; font-size: ${fontSize}px;">${escapeXml(textContent)}</text>`;
        }

        // Draw the points.
        let pointList = "";
        let prevPoint: DPoint | null = null;
        for (let i = 0; i < values.length; i++) {
            let convPoint = convCoords([values[i].date, values[i].value]);
            if (useStairway && prevPoint !== null) {
                // Use the previous y coordinate with the current x coordinate.
                pointList += `${pform(convPoint[0])},${pform(prevPoint[1])} `;
            }
            pointList += `${pform(convPoint[0])},${pform(convPoint[1])} `;
            prevPoint = convPoint;
        }
        outStr += `<polyline fill="none" stroke="blue" stroke-width="2"
points="${escapeXml(pointList)}"/>`;
        outStr += "</svg>";
        return outStr;
    }
}

runtime.handleAsync(async function () {
    const gradientNode = UKI-4.0 .findNode("/Nodes/Demo-Nodes/Gradient", true);

    net.registerHttpRoute("graph", async ctx => {
        const response = ctx.response;

        // Lies die letzten 50 historischen Werte des Gradient-Demo-Nodes aus und stelle
        // sie dann in einem SVG-Diagramm dar.
        let historyValues = await UKI-4.0 .readNodeHistoryValuesAsync(gradientNode, null,
null, 30);
        let svgValues: SvgLibrary.Value[] = [];
```

```
        for (let historyValue of historyValues) {
            if (typeof historyValue.value != "number")
                throw new Error("Expected a number value");

            svgValues.push({
                value: historyValue.value as number,
                date: historyValue.timestamp.getTime()
            });
        }

        // Generiere das SVG-Diagramm.
        let resultString: string;
        try {
            resultString = SvgLibrary.generateValueChartSvg(svgValues, 700, 400, undefined,
40);
                // Setze den Content-Type auf SVG.
            response.contentType = "image/svg+xml";
        }
        catch (e) {
            // Gib den Fehler aus.
            resultString = String(e);
            response.statusCode = 500;
            response.contentType = "text/plain";
        }

        await response.writeBodyCompleteAsync(resultString);
    });

}());
```

Erweiterte HTTP-Programmierung

Die bisher gezeigten Beispiele generieren eine HTML-Seite oder SVG-Grafik, die im Browser angezeigt wird, sich danach jedoch nicht mehr verändert. Um dynamische HTML5-Apps zu erstellen, können Sie jedoch z.B. auch eine statische HTML-Seite mit einem JavaScript/TypeScript erstellen, das wiederum mit dem Script auf UKI-4.0 -Seite kommuniziert, beispielsweise über JSON (zum Serialisieren und Deserialisieren von JSON können Sie `JSON.stringify()` und `JSON.parse()` verwenden). Dadurch kann das Script im Browser regelmäßig neue Informationen vom Script in UKI-4.0 abfragen.

Im nächsten Kapitel werden WebSocket-Verbindungen beschrieben, mit denen Ihre HTML-Seite über eine bidirektionale, socketähnliche Verbindung mit dem UKI-4.0 -Script kommunizieren kann.

WebSocket-Verbindungen im HTTP-Handler

Sie können in einem UKI-4.0 -Script auch **serverseitige WebSocket-Verbindungen** betreiben, die z.B. von einem Browser-Script über die dort verfügbare `WebSocket`-Klasse hergestellt werden. WebSocket erlaubt es, in beiden Richtungen (Server an Client sowie Client an Server) jederzeit neue Daten zu senden bzw. empfangen, solange die Verbindung besteht; wohingegen bei HTTP-Requests der Client ständig neue Requests ausführen müsste, um nachzusehen, ob der Server neue Informationen hat („Polling“). So könnten Sie beispielsweise in einem UKI-4.0 -Script einen Eventlistener für ein `ValueChanged`-Event eines Nodes hinzufügen, und immer wenn ein neuer Nodewert geschrieben wurde, diesen per WebSocket-Verbindung an alle verbundene Browser zur Darstellung senden.

Um eine WebSocket-Verbindung entgegenzunehmen, prüfen Sie zuerst mit `ctx.isWebSocketRequest`, ob es sich tatsächlich um einen WebSocket-Request handelt. Falls ja, rufen Sie `ctx.acceptWebSocketAsync()` auf, das ein `net.RawWebSocket`-Objekt zurückgibt. Auf diesem Objekt können Sie nun mit `receiveAsync()`

Messages empfangen und mit `sendAsync()` Messages senden. Das von diesen Methoden zurückgegebene `Promise`-Objekt wird dabei erst „fulfilled“, wenn der Empfangs-/Sendevorgang abgeschlossen ist. Falls ein Fehler beim Empfangen oder Senden auftritt (z.B. weil die Verbindung bereits geschlossen wurde), werfen die Methoden eine Ausnahme (oder rejecten das zurückgegebene `Promise`).

Beachten Sie: Sie können auf einem einzelnen `RawWebSocket`-Objekt gleichzeitig eine Nachricht empfangen und auch senden (es kann also gleichzeitig jeweils eine `receiveAsync()`- als auch eine `sendAsync()`-Operation ausstehend sein).

Für die folgenden WebSocket-Beispiele verwenden wir eine statische HTML-Seite, die eine WebSocket-Verbindung zu UKI-4.0 aufbaut. Speichern Sie daher bitte zuerst folgende Datei UKI-4.0 `Websocket.html` im `webfiles`-Ordner in Ihrem Projektverzeichnis:

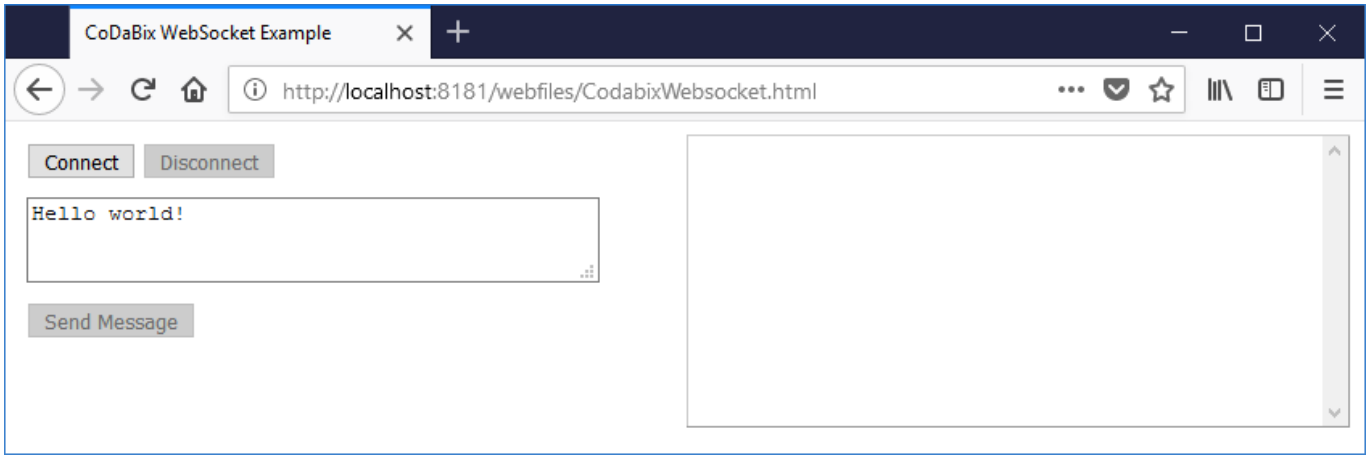
UKI-4.0 Websocket.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>UKI-4.0 WebSocket Example</title>
    <style>
        #connect-container {
            float: left;
            width: 400px
        }

        #connect-container div {
            padding: 5px;
        }

        #console-container {
            float: left;
            margin-left: 15px;
            width: 400px;
        }

        #console {
            border: 1px solid #CCCCCC;
            border-right-color: #999999;
            border-bottom-color: #999999;
            height: 170px;
            overflow-y: scroll;
            padding: 5px;
            width: 100%;
            white-space: pre-wrap;
        }

        #console p {
            padding: 0;
            margin: 0;
        }
    </style>
    <script>
"use strict";
document.addEventListener("DOMContentLoaded", function() {

    var ws = null;
```

```
    var connectButton = document.getElementById('connect');
    var disconnectButton = document.getElementById('disconnect');
    var echoButton = document.getElementById('echo');
    var messageBox = document.getElementById('message');
    var consoleArea = document.getElementById('console');

    function setConnected(connected, fullyConnected) {
        connectButton.disabled = connected;
        disconnectButton.disabled = !connected;
        echoButton.disabled = !(connected && fullyConnected);
    }
    setConnected(false);

    function connect() {
        if (ws != null)
            return;

        if (typeof WebSocket == "undefined")
            alert("WebSocket is not supported by this browser.");

        var wsUrl = (window.location.protocol == "https:" ? "wss:" : "ws:") + '//' +
            window.location.host + "/scripthandlers/websocket";
        var localWs = ws = new WebSocket(wsUrl);
        setConnected(true);
        log("Info: WebSocket connecting...");

        ws.onopen = function () {
            if (localWs != ws)
                return;

            setConnected(true, true);
            log("Info: WebSocket connection opened.");
        };
        ws.onmessage = function (event) {
            if (localWs != ws)
                return;

            log("Received: " + event.data);
        };
        ws.onclose = function (event) {
            if (localWs != ws)
                return;

            setConnected(false);
            log("Info: WebSocket connection closed, Code: " + event.code +
                (event.reason == "" ? "" : ", Reason: " + event.reason));
            ws = null;
        };
    }

    function disconnect() {
        if (ws == null)
            return;

        setConnected(false);
        log("Info: WebSocket connection closed by user.");
        ws.close();
        ws = null;
    }
```

```
    function sendText() {
        if (ws == null)
            return;

        var message = messageBox.value;
        ws.send(message);
        log("Sent: " + message);
    }

    function log(message) {
        var p = document.createElement('p');
        p.style.wordWrap = 'break-word';
        p.appendChild(document.createTextNode(message));
        consoleArea.appendChild(p);
        while (consoleArea.childNodes.length > 25) {
            consoleArea.removeChild(consoleArea.firstChild);
        }
        consoleArea.scrollTop = consoleArea.scrollHeight;
    }

    // Add event listeners to the buttons.
    connectButton.onclick = function() {
        connect();
    };

    disconnectButton.onclick = function() {
        disconnect();
    };

    echoButton.onclick = function() {
        sendText();
    };

}, false);
    </script>
</head>
<body>
<div>
    <div id="connect-container">
        <div>
            <button id="connect">Connect</button>
            <button id="disconnect">Disconnect</button>
        </div>
        <div>
            <textarea id="message" style="width: 350px">Hello world!</textarea>
        </div>
        <div>
            <button id="echo">Send Message</button>
        </div>
    </div>
    <div id="console-container">
        <div id="console"></div>
    </div>
</div>
</body>
</html>
```

Sie sollten diese Seite nun aufrufen können, wenn Sie im Browser http://localhost:8181/webfiles/UKI-4.0 Websocket.html aufrufen (vorausgesetzt, der lokale Port für den UKI-4.0 -Webserver in den UKI-4.0 Projekteinstellungen ist auf den Standardwert 8181 eingestellt und die Option „Serve Static Web Files“ wurde nicht deaktiviert):



Einfaches Echo-WebSocket

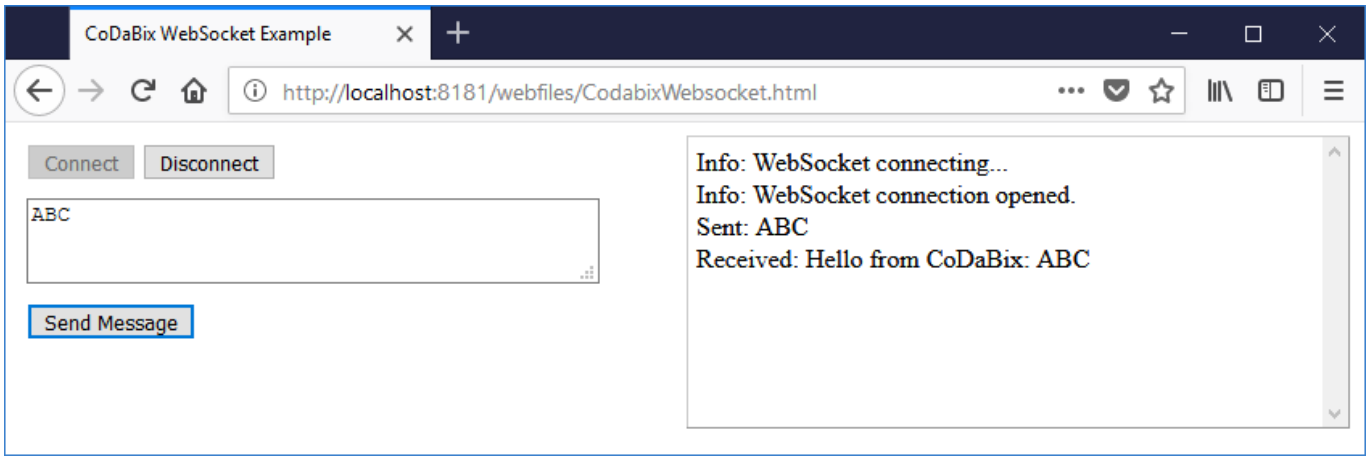
Im einfachsten Fall können Sie Nachrichten von einem WebSocket empfangen, und diese direkt wieder an den Client zurücksenden, wie das folgende UKI-4.0 -Script zeigt:

```
runtime.handleAsync(async function () {

    const maxMessageLength = 10000;

    net.registerHttpRoute("websocket", async ctx => {
        // Check if the client sent a WebSocket request.
        if (!ctx.isWebSocketRequest) {
            ctx.response.statusCode = 400;
        }
        else {
            // Accept the WebSocket request.
            let ws = await ctx.acceptWebSocketAsync();

            // In a loop, receive messages from the client and send them back.
            // Once the client closes the connection, receive() will return null
            // or throw an exception.
            while (true) {
                let message = await ws.receiveAsync(maxMessageLength);
                if (message == null || message.length == maxMessageLength)
                    break; // connection closed or message too large

                logger.log("Received message: " + message);

                // Send the message back to the client.
                await ws.sendAsync("Hello from UKI-4.0 : " + message);
            }
        }
    });

}());
```

Der HTTP-Handler nimmt eine WebSocket-Verbindung entgegen, und liest dann immer wieder Nachrichten vom Client und sendet diese zurück. Dies passiert solange, bis der Client die Verbindung trennt.

In der HTML-Seite können Sie nun mit „Connect“ eine Verbindung herstellen und Nachrichten an UKI-4.0

senden, die dann sofort zurückkommen sollten:



"Chat" mit Hintergrund-Sendeoperationen

Nehmen wir nun an, Sie möchten über WebSocket allen verbundenen Benutzern jeweils sofort den aktuellen Wert des Temperatur-Demonodes senden, sobald dieser in UKI-4.0 geschrieben wird. Zusätzlich sollen die Benutzer untereinander chatten können.

Da die `sendAsync()`-Methode des WebSockets so lange (asynchron) blockiert, bis die Daten vollständig gesendet wurden, können wir diese nicht einfach aufrufen, wenn eine neuer Temperaturwert oder eine Chatnachricht gesendet werden soll, da möglicherweise die vorherige Nachricht noch nicht vollständig an den Client gesendet wurde. Stattdessen verwenden wir eine **Queue**, in der die zu sendenden Nachrichten abgelegt werden. Eine andere Scriptfunktion nimmt dann jeweils immer eine Nachricht aus dieser Queue und sendet diese über das WebSocket. Sobald der Sendevorgang abgeschlossen ist, nimmt sie die nächste Nachricht aus der Queue, oder wartet bis wieder mindestens eine Nachricht in der Queue ist.

Im folgenden Scriptcode erledigt dies die Klasse `WebSocketSender`, die Sie auch für andere Scripts wiederverwenden können.

(Bitte deaktivieren Sie vorher das zuvor beschriebene Echo-WebSocket-Script, falls Sie dieses ausführen, da beide Scripte den gleichen HTTP-Path registrieren würden.)

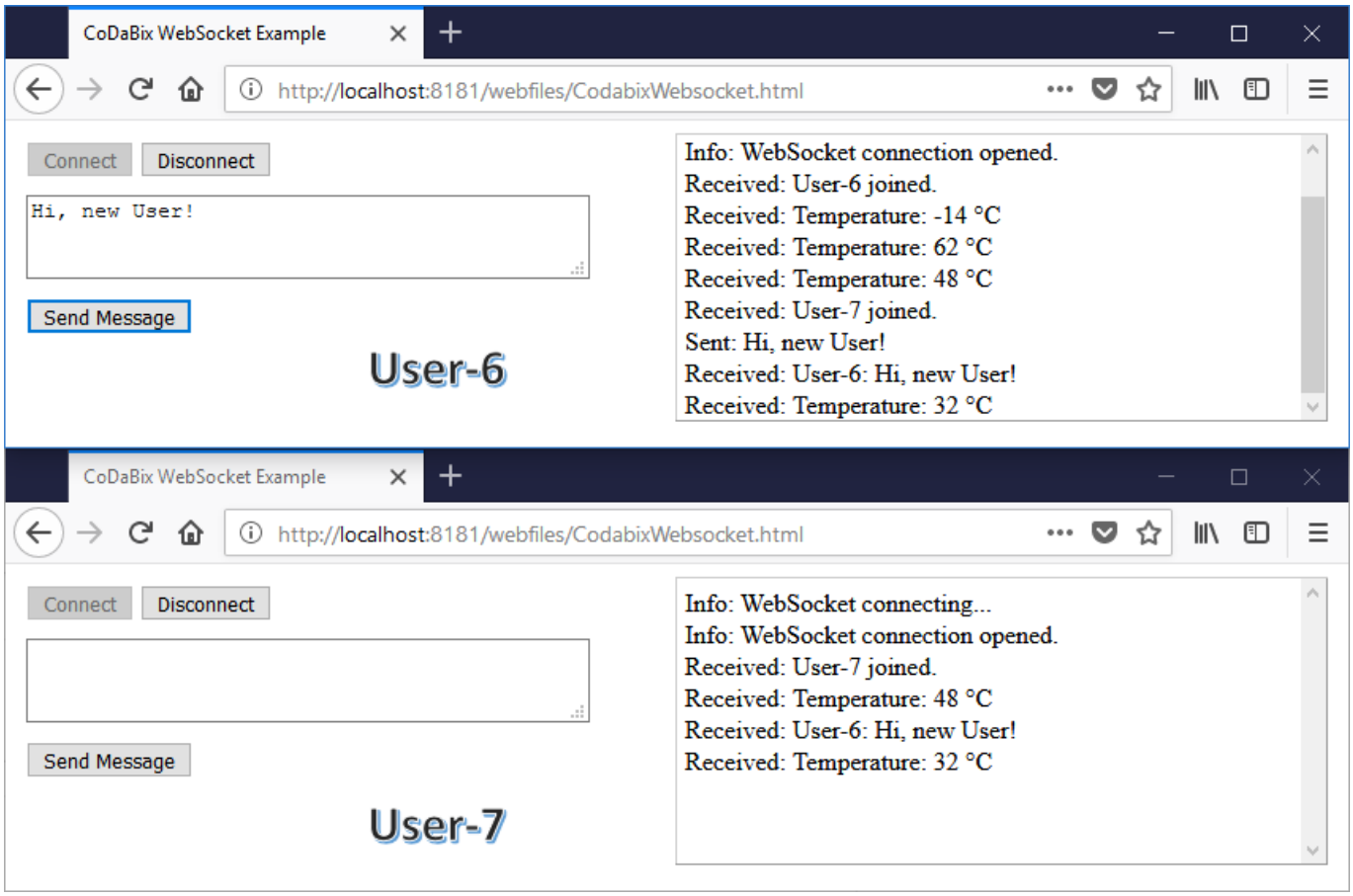
```
runtime.handleAsync(async function () {

    /**
     * A utility class that implements a Send Queue for WebSockets. This allows you to
     * push messages to be sent to the client to the queue without having to wait until
     * the client has received it.
     */
    class WebSocketSender {
        // The queued messages to be sent to the client.
        private sendQueue: string[] = [];
        // The callback to resolve the promise of the sender function.
        private sendResolver: (() => void) | null = null;
        private senderIsExited = false;

        constructor(private ws: net.RawWebSocket) {
            // Start the sender function which will then wait for the next message queue
entry.
            runtime.handleAsync(this.runSenderAsync());
        }

        /**
         * Adds the specified message to the send queue.
```

```
         */
        public send(s: string) {
            // If the sender already exited due to an error, do nothing.
            if (this.senderIsExited)
                return;

            // TODO: Check if the number of buffered messages exceeds a specific limit.
            this.sendQueue.push(s);

            // If the sender waits for the next queue entry, release the sender.
            if (this.sendResolver) {
                this.sendResolver();
                this.sendResolver = null;
            }
        }

        private async runSenderAsync() {
            try {
                // In a loop, we will wait until the next send queue entry arrives.
                while (true) {
                    for (let i = 0; i < this.sendQueue.length; i++) {
                        await this.ws.sendAsync(this.sendQueue[i]);
                    }
                    this.sendQueue = [];

                    // Create a Promise and cache the resolve handler, so that the
                    // send() method can fulfill the Promise later.
                    await new Promise<void>(resolve => this.sendResolver = resolve);
                }
            }
            catch (e) {
                // An error occured, e.g. when the client closed the connection.
                // This means the receive handler should also get an exception when it
                // tries to receive the next message from the client.
                logger.log("Send Error: " + e);
            }
            finally {
                this.senderIsExited = true;
            }
        }
    }


    // The maximum message size which we will receive.
    const maxReceiveLength = 10000;

    interface ChatUser {
        name: string;
        sender: WebSocketSender;
    }

    // The set of currently connected users.
    const chatUsers = new Set<ChatUser>();
    let currentUserId = 0;


    // Register for the ValueChanged event of the "Temperature" node. Every time a
    // new value occurs, we will broadcast it to all connected users.
    const temperatureNode = UKI-4.0.findNode("/Nodes/Demo-Nodes/Temperature", true);
```

```
    const getTemperatureValueMessage = function (value: UKI-4.0.NodeValue | null) {
        return `Temperature: ${value && value.value} ${temperatureNode.unit || ""}`;
    };

    temperatureNode.addValueChangedEventListener(e => {
        // Send the new value to all connected users.
        const message = getTemperatureValueMessage(e.newValue);
        chatUsers.forEach(u => {
            u.sender.send(message);
        });
    });


    // Register the websocket handler.
    net.registerHttpRoute("websocket", async ctx => {
        // Check if the client sent a WebSocket request.
        if (!ctx.isWebSocketRequest) {
            ctx.response.statusCode = 400;
        }
        else {
            // Accept the WebSocket request.
            let ws = await ctx.acceptWebSocketAsync();
            await handleWebSocketAsync(ws);
        }
    });

    // The function that handles the WebSocket connection.
    const handleWebSocketAsync = async function (ws: net.RawWebSocket) {
        // Create a send queue which we use to send messages to the client.
        let sender = new WebSocketSender(ws);

        // Generate a new user name, then notify the existing users
        // that a new user joined.
        let user = {
            name: "User-" + ++currentUserId,
            sender: sender
        };
        chatUsers.add(user);

        try {
            const userJoinedMessage = `${user.name} joined.`;
            chatUsers.forEach(u => {
                u.sender.send(userJoinedMessage);
            });

            // Send the initial temperature value to the new user.
            user.sender.send(getTemperatureValueMessage(temperatureNode.value));

            // Now start to receive the messages from the client.
            while (true) {
                let receivedMessage: string | null;
                try {
                    receivedMessage = await ws.receiveAsync(maxReceiveLength);
                    if (receivedMessage == null || receivedMessage.length >=
maxReceiveLength)
                        break; // connection closed or message too large
                }
                catch (e) {
```

```
                    logger.log("Receive Error: " + e);
                    break;
                }

                // Broadcast the received message.
                const broadcastMessage = `${user.name}: ` + receivedMessage;
                chatUsers.forEach(u => {
                    u.sender.send(broadcastMessage);
                });
            }
        }
        finally {
            // The client has closed the connection, or there was an error when receiving.
            // Therefore, we notify the other users that the current user has left.
            chatUsers.delete(user);

            const userLeftMessage = `${user.name} left.`;
            chatUsers.forEach(u => {
                u.sender.send(userLeftMessage);
            });
        }
    }

}());
```

Sobald sich ein Benutzer verbindet, wird diesem ein Benutzername zugewiesen (z.B. User-6) und eine Nachricht an alle verbundenen Benutzer geschickt. Der neue Benutzer erhält auch den aktuellen Wert des Temperatur-Nodes. Sobald dann immer ein neuer Wert für die Temperatur geschrieben wurde oder ein anderer Benutzer eine Chatnachricht sendet, wird diese an alle verbundenen Benutzer (praktisch ein Broadcast) geschickt:



Senden von HTTP-Requests

Der Namespace `net.httpClient` stellt Methoden zur Verfügung, die es Ihnen ermöglichen, HTTP-Requests an andere Server zu senden. So können Sie beispielsweise externe REST-APIs über JSON-Objekte ansprechen.

Wichtig: Beim Senden von Requests über das Internet (oder über andere potentiell unsichere Netzwerke), stellen Sie bitte sicher, dass Sie falls möglich immer `https:`-URLs verwenden, da `http:`-URLs eine unsichere Verbindung verwenden und deshalb die Authentizität des Servers sowie die Vertraulichkeit und Integrität der Daten nicht gewährleistet ist. Dies gilt vor allem dann, wenn der Request vertrauliche Zugangsdaten enthält.

Beachten Sie: Derzeit können Request- und Response-Bodies nur als Textdaten, nicht als Binärdaten verwendet werden.

Einfache GET-Requests

Der folgende Code führt einen einfachen GET-Request aus und loggt den Response-Body (falls vorhanden):

```
let response = await net.httpClient.sendAsync({
    url: "https://www.UKI-4.0 .com/en/start"
});

if (response.content) {
    logger.log("Result: " + response.content.body);
}
```

Zugreifen auf eine JSON-bsierte REST-API

Wenn Sie auf eine URL zugreifen möchten, die ein JSON-Ergebnis liefert, können Sie `JSON.parse()` verwenden, um den JSON-Response-Body-String in ein JavaScript-Objekt zu konvertieren und auf dieses zuzugreifen.

Das folgende Beispiel greift auf eine externe REST-API zu, um Wetterdaten (Temperatur) in einem regelmäßigen Intervall abzufragen und in die `/Nodes/Temperature`-Node zu schreiben:

```
    let temperatureNode = UKI-4.0.findNode("/Nodes/Temperature", true);

    while (true) {
        let valueToWrite: UKI-4.0.NodeValue;
        try {
            let response = await net.httpClient.sendAsync({
                url: "https://api.openmeteo.com/observations/openmeteo/1001/t2"
            });

            let result = JSON.parse(response.content!.body);
            valueToWrite = new UKI-4.0.NodeValue(result[1]);
        }
        catch (e) {
            logger.logWarning("Could not retrieve weather data: " + e);
            valueToWrite = new UKI-4.0.NodeValue(null, undefined, {
                statusCode: UKI-4.0.NodeValueStatusCodeEnum.Bad,
                statusText: String(e)
            });
        }

        // Write the temperature value.
        await UKI-4.0.writeNodeValueAsync(temperatureNode, valueToWrite);

        // Wait 5 seconds
        await timer.delayAsync(5000);
    }
```

Das folgende Beispiel zeigt, wie ein POST-Request zu der UKI-4.0 -internen REST-API gesendet werden kann (für `<encrypted-password>` müssten Sie das verschlüsselte Benutzerpasswort angeben):

```
    let result = await net.httpClient.sendAsync({
        // Note: For external servers you should "https:" for a secure connection.
        url: "http://localhost:8181/api/json",
        method: "POST",
        content: {
            headers: {
                "content-type": "application/json"
            },
            body: JSON.stringify({
                username: "demo@user.org",
                password: UKI-4.0.security.decryptPassword("<encrypted-password>"),
                browse: {
                    na: "/Nodes"
                }
            })
        }
    });

    let jsonResult = JSON.parse(result.content!.body);
    // TODO: Process JSON result...
```

Empfohlene Vorgehensweisen

Die folgende Liste gibt einige Empfehlungen für performante und saubere Scripts:

- Falls Sie kein Library-Script schreiben, platzieren Sie Ihren gesamten Code innerhalb einer IIFE

(Immediately-Invoked Function Expression) wie folgt:

Blank Template.js

```
runtime.handleAsync(async function () {

    // Ihr Code...

} ());
```

Dies verhindert das „Verschmutzen“ des globalen Gültigkeitsbereichs, und der TypeScript-Compiler kann nicht verwendete lokale Variablen erkennen und beschwert sich nicht über das Verwenden von anonymen Typen in exportierten Variablen.
Wir empfehlen, die Funktion mit dem `async`-Attribut zu markieren, damit Sie asynchrone Funktionen unter Verwendung des `await`-Keywords aufrufen können. In diesem Fall sollten Sie wie im obigen Beispiel das zurückgegebene `Promise`-Objekt an `runtime.handleAsync()` übergeben, um sicherzustellen, dass unbehandelte Ausnahmen nicht lautlos „verschluckt“ werden.

- Verwenden Sie immer `let` oder `const` zum Deklarieren von Variablen, nicht `var`, da letzteres die Variable nicht im Gültigkeitsbereich des lokalen Blocks erzeugt, sondern der gesamten Funktion.
- Anstelle von `arguments`, welches ein „magisches“, arrayähnliches Objekt ist, benutzen Sie Rest Parameters (`...args`), welche eine echte `Array`-Instanz sind und die vorhergehenden Funktionsparameter nicht enthalten.
- Wenn Sie ein Array enumerieren möchten, benutzen Sie nicht `for-in` - verwenden Sie stattdessen `for-of`, da ersteres keine bestimmte Auflistungsreihenfolge garantiert.
- Wenn Sie eine Ausnahme auslösen, verwenden Sie keine primitiven Werte wie bei `throw "Meine Fehlermeldung.";`. Erstellen Sie stattdessen eine Instanz des `Error`-Objekts und verwenden Sie dieses für die `throw`-Anweisung, wie bei `throw new Error("Meine Fehlermeldung");`. Dadurch ist sichergestellt, dass Sie eine Stapelnachverfolgung der Stelle abrufen können, an der die Ausnahme ausgelöst wurde.
- Falls möglich, vermeiden Sie es, Closures in Code zu erstellen, der sehr oft ausgeführt wird (z.B. in einer Schleife), da das Erstellen von Closures in JavaScript teuer ist. Prüfen Sie stattdessen, ob Sie eine Funktion durch das Übergeben von Argumenten wiederverwenden können.
  Angenommen, Sie möchten das ValueChanged-Event eines Nodes behandeln und dort einen neuen Wert schreiben (wofür UKI-4.0 `.scheduleCallback()` verwendet werden muss), vermeiden Sie folgenden Code, der eine jedes Mal, wenn das Event aufgerufen wird, eine Closure erstellt:

```
myNode.addValueChangedEventListener(e => UKI-4.0.scheduleCallback(() => {
    void UKI-4.0.writeNodeValueAsync(otherNode, e.newValue);
}));
```

Erstellen Sie stattdessen einmalig eine Closure und rufen dann im Eventlistener die folgende Variante von UKI-4.0 `.scheduleCallback()` auf, bei der Argumente weitergegeben werden können:

```
const scheduledHandler = (newVal: UKI-4.0.NodeValue) => {
    void UKI-4.0.writeNodeValueAsync(otherNode, newVal);
};
myNode.addValueChangedEventListener(e => UKI-4.0.scheduleCallback(scheduledHandler,
e.newValue));
```

Passwörter sicher ablegen

In einigen Fällen müssen Sie Passwörter im Scriptcode ablegen, um beispielsweise auf externe passwortgeschützte HTTP-Dienste zuzugreifen. Damit ein Passwort nicht im Klartext abgelegt werden muss, können Sie es zuvor einmalig über die UKI-4.0 Webkonfiguration im Menüpunkt „Password Security“ verschlüsseln, und anschließend im Scriptcode am Anfang das verschlüsselte Passwort über einen Aufruf von UKI-4.0 `.security.decryptPassword()` entschlüsseln. Das Passwort wird dadurch mit dem **Password Key** der Backend-Datenbank **verschlüsselt**, der beim Anlegen der Datenbank unter Verwendung eines kryptographisch sicheren Zufallszahlengenerators erstellt wurde.
Die gleiche Vorgehensweise wird auch beim Speichern von Passwörtern in Datenpunktnodes vom Typ „Password“ verwendet, wie es beispielsweise bei Device-Plugins der Fall ist.

Dies bietet Ihnen zusätzlichen Schutz:

- Passwörter werden nicht im Klartext angezeigt, daher können Sie den Scriptcode gefahrlos anzeigen, ohne dass andere Personen das Passwort direkt ablesen können.
- Wenn Sie beim Erstellen eines **Backups** die Option *Include Password Key* deaktivieren, können die Passwörter nicht mehr aus dem Backup extrahiert (entschlüsselt) werden.

Beachten Sie: In den UKI-4.0 Settings kann ein neuer Password Key generiert werden, wodurch Passwörter, die mit dem alten Key verschlüsselt wurden, nicht mehr lesbar sind.

Wenn Sie beipielsweise das Passwort „test“ im Scriptcode verwenden möchten, gehen Sie in der UKI-4.0 Webkonfiguration auf „Password Security“ und geben dort das Passwort ein. Nach einem Klick auf „Encrypt“ wird das verschlüsselte Passwort ausgegeben.

Beispiel:

Das verschlüsselte Passwort können Sie nun kopieren und anschließend in dem Script einbauen, in dem Sie das Password benötigen:

```
// Entschlüssle das Passwort, das wir für den HTTP-Request benötigen.
const myPassword = UKI-4.0 .security.decryptPassword(
    "K8D/VWnVQG45HSFlD1G94RwXzrSxH3ARlzzhHekoAdf+prcwe5y6S4FhPUug1Ycw#");

// ...

function doRequest() {
    let request = {
        user: "myUser",
        password: myPassword
    }
    // TODO: Sende den Request...
}
```