

# Table of Contents

UKI-4.0 ® is a bidirectional universal 'Communication Data Bridge' .....	14
Use / Application .....	15
Plugins for Devices / Interfaces and Data Exchanges .....	15
Configuration .....	16
Requirements .....	16
UKI-4.0 <b>for Windows</b> .....	18
System Requirements .....	18
Supported Operating Systems .....	18
Hardware Requirements .....	18
Back-end Database Requirements .....	18
UKI-4.0 Setup and First Start .....	19
UKI-4.0 Setup .....	19
Install and Uninstall Plugins .....	19
Updating UKI-4.0 .....	19
First Start .....	19
UKI-4.0 Project Settings .....	21
Basic Settings .....	22
Web Server Settings .....	23
MySQL/MariaDB/MSSQL Settings .....	24
Installing UKI-4.0 as a Service .....	24
License Management .....	25
Machine Code .....	25
UKI-4.0 <b>for Linux</b> .....	26
System Requirements .....	26
Supported Operating Systems .....	26
Hardware Requirements .....	26
Back-end Database Requirements .....	26
Unsupported Features .....	26
Installing UKI-4.0 and First Start .....	27
Installing UKI-4.0 for Linux .....	27
First Start .....	27
Run UKI-4.0 as a Service .....	28
Install UKI-4.0 as a Service .....	28
Status of the UKI-4.0 Service .....	30
(Re-)start, stop and uninstall the UKI-4.0 Service .....	31
License Management .....	32
Machine Code .....	32
<b>Siemens IOT2050 Image</b> .....	33
Content of the image .....	33
Writing the image .....	33
Putting into service .....	34
Automatic Hostname .....	34
Accessing the UKI-4.0 web configuration .....	34
Access via SSH .....	34
<b>Starting the UKI-4.0 Application Configuration</b> .....	35
<b>General</b> .....	35
(1) UKI-4.0 Toolbar .....	36
Licence Dialog .....	36
About .....	37
(2) Working Area .....	37

(3) Title Bar .....	37
(4) Toolbar .....	38
(5) Menu .....	39
(6) Node Tree .....	40
(7) Data Area .....	41
<b>Nodes</b> .....	41
What are Nodes? .....	41
Folder Nodes .....	41
Datapoint Nodes .....	42
Link Nodes .....	42
Directory Nodes .....	42
<b>Node Properties</b> .....	42
Common Properties .....	42
Node Types .....	42
Datapoint Nodes .....	43
Value Types .....	44
Node View .....	44
Node Structure .....	46
Create a Node .....	47
Add a Folder Node .....	47
Add a Directory Node .....	47
Add a Datapoint Node .....	48
Link Nodes .....	49
Node Access .....	49
Add / Delete Node Access to a User .....	50
Add / Remove via the Node View .....	50
Add / Remove via the User Group .....	50
<b>User</b> .....	51
Add Users .....	51
Edit Users .....	52
<b>User Group</b> .....	52
Add User Group .....	53
Edit User Group .....	53
Add User to User Group .....	53
<b>Plugins</b> .....	54
Device Plugins .....	54
Exchange Plugins .....	54
Interface Plugins .....	55
<b>Activation</b> .....	56
State Machine .....	56
Diagnostics .....	56
Location .....	56
Classification .....	57
Device Plugins .....	57
Exchange Plugins .....	57
Interface Plugins .....	57
<b>Configuration</b> .....	57
Using an Application .....	57
Using a Configuration File .....	58
Location .....	58
Structure .....	58
Entity Attributes .....	58
Synchronization .....	59

- Device Model** ..... 60
- Configuration** ..... 60
  - Using UKI-4.0 ® ..... 60
  - Using an Application ..... 60
- Location ..... 60
  - Using a Configuration File ..... 61
- Location ..... 61
- Structure ..... 61
- Example Configuration File ..... 62
  - Configuration** ..... 63
    - Channel ..... 63
- Settings ..... 63
- Adding a Channel ..... 64
  - Variables ..... 64
- Path ..... 65
- Examples ..... 65
  - Diagnostics** ..... 66
    - Channel ..... 66
    - Variables ..... 67
    - Log file ..... 68
  - Entities** ..... 68
    - Device ..... 68
    - Channel ..... 68
- Folders & Files** ..... 68
- Folders ..... 68
- Files ..... 69
  - About Versions** ..... 69
- This Document ..... 69
- Plugin ..... 69
- Assembly ..... 69
  - Configuration** ..... 70
    - Channel ..... 70
- Settings ..... 70
- Adding a Channel ..... 71
  - Parameters ..... 71
- Path ..... 72
- Browse ..... 72
- Read/Write ..... 72
  - Diagnostics** ..... 73
    - Channel ..... 73
    - Parameters ..... 73
    - Log file ..... 73
  - Entities** ..... 74
    - Device ..... 74
    - Channel ..... 74
- Folders & Files** ..... 74
- Folders ..... 74
- Files ..... 75
  - About Versions** ..... 75
- This Document ..... 75
- Plugin ..... 75
- Assembly ..... 75
- Prerequisites ..... 76

- Configuration** ..... 76
  - Channel ..... 76
- Settings ..... 76
- Adding a Channel ..... 77
  - Variables ..... 77
- Path ..... 77
- Examples ..... 78
  - Diagnostics** ..... 78
    - Channel ..... 78
    - Variables ..... 79
    - Log file ..... 79
  - Entities** ..... 80
    - Device ..... 80
    - Channel ..... 80
  - Folders & Files** ..... 80
- Folders ..... 80
- Files ..... 80
  - About Versions** ..... 80
- This Document ..... 81
- Plugin ..... 81
- Assembly ..... 81
  - General** ..... 82
  - What Does the Plugin Do?** ..... 82
  - Features** ..... 82
  - Supported Servers** ..... 82
  - Purpose & Use Cases** ..... 82
  - Installation** ..... 82
    - Requirements ..... 82
  - Configuration** ..... 83
- Overview ..... 83
- Usage ..... 83
- Settings ..... 84
- Change Settings ..... 84
- Overview ..... 84
- Variables ..... 85
- OPC NodeId ..... 85
  - Diagnostics** ..... 86
    - Channel ..... 86
    - Variables ..... 86
    - Log File ..... 87
  - Status Codes ..... 87
  - Entities** ..... 87
    - Device ..... 87
    - Channel ..... 87
  - Folders & Files** ..... 88
- Folders ..... 88
- Files ..... 88
  - About Versions** ..... 88
- This Document ..... 88
- Plugin ..... 88
- Assembly ..... 88
  - Features** ..... 89
  - Purpose & Use Cases** ..... 89

Purpose ..... 89

Use Cases ..... 89

**Installation** ..... 90

Requirements ..... 90

PLC Settings ..... 90

**Configuration** ..... 90

    Using UKI-4.0 ..... 90

Channel ..... 90

Variables ..... 92

Import/Export ..... 93

    Using the S7 XML Configuration File ..... 94

Structure ..... 94

Usage ..... 98

Synchronization ..... 98

Example Configuration File ..... 98

Example Configuration Scheme File ..... 99

    Using the Application (Windows only) ..... 101

Overview ..... 101

Usage ..... 102

**Diagnostics** ..... 103

    Channel ..... 103

    Variables ..... 103

    Log File ..... 104

Status Codes ..... 104

**Entities** ..... 104

Device ..... 104

Channel ..... 104

Variable ..... 105

**Folders & Files** ..... 105

Folders ..... 105

Files ..... 105

**About Versions** ..... 105

This Document ..... 105

Plugin ..... 105

Assembly ..... 106

**Exchange Model** ..... 107

**Configuration** ..... 107

    Using UKI-4.0 ® ..... 107

**General** ..... 108

**What Does the Plugin Do?** ..... 108

**Features** ..... 108

**Supported Contents** ..... 108

**Purpose & Use Cases** ..... 108

Purpose ..... 108

Use Cases ..... 108

**Installation** ..... 109

    Requirements ..... 109

**Configuration** ..... 109

    Using the Configuration File ..... 109

Structure ..... 109

Servers Element ..... 109

Server Element ..... 109

Trigger-Element ..... 110

- File Element ..... 111
- Bindings Element ..... 112
- Binding Element ..... 112
- Node Query Expression ..... 114
- Usage ..... 114
- Synchronization ..... 114
- Example Configuration File ..... 114
- Example Configuration Scheme File ..... 115
- Diagnostics** ..... 118
- Entities** ..... 118
- Folders & Files** ..... 119
- Folders ..... 119
- Files ..... 119
- About Versions** ..... 119
- This Document ..... 119
- Plugin ..... 119
- Assembly ..... 119
- General** ..... 120
- What Does the Plugin Do?** ..... 120
- Features** ..... 120
- Supported Database Engines** ..... 120
- Purpose & Use Cases** ..... 120
- Installation** ..... 121
- Requirements ..... 121
- Configuration** ..... 121
- Structure ..... 121
- Example Configuration File ..... 126
- Diagnostics** ..... 127
- Entities** ..... 127
- Folders & Files** ..... 128
- Folders ..... 128
- Files ..... 128
- About Versions** ..... 128
- This Document ..... 128
- Plugin ..... 128
- Assembly ..... 128
- General** ..... 129
- What Does the Plugin Do?** ..... 129
- Features** ..... 129
- Supported Database Engines** ..... 129
- Purpose & Use Cases** ..... 129
- Installation** ..... 130
- Requirements ..... 130
- Configuration** ..... 130
- Overview ..... 130
- Database-Specific Settings ..... 130
- Table-Specific Settings ..... 132
- Columns ..... 132
- Diagnostics** ..... 133
- Database ..... 133
- Columns ..... 134
- Log File ..... 134
- Entities** ..... 135

- Exchange ..... 135
- Channel ..... 135
- Table ..... 135
- Column ..... 135
- Folders & Files** ..... 135
- Folders ..... 135
- Files ..... 136
- About Versions** ..... 136
- This Document ..... 136
- Plugin ..... 136
- Assembly ..... 136
- Configuration** ..... 137
- Document ..... 137
- Settings ..... 137
- Adding a Document (Channel) ..... 137
- Variables ..... 138
- Example ..... 138
- Read/Write ..... 139
- Diagnostics** ..... 140
- Channel ..... 140
- Variables ..... 141
- Log file ..... 141
- Entities** ..... 142
- Exchange ..... 142
- Channel ..... 142
- Folders & Files** ..... 142
- Folders ..... 142
- Files ..... 142
- About Versions** ..... 143
- This Document ..... 143
- Plugin ..... 143
- Assembly ..... 143
- Interface Model** ..... 144
- Configuration** ..... 144
- Using UKI-4.0 ® ..... 144
- General** ..... 145
- What Does the Plugin Do?** ..... 145
- Features** ..... 145
- Supported OPC Protocols** ..... 145
- Purpose & Use Cases** ..... 145
- Installation** ..... 145
- Configuration** ..... 145
- User Configuration ..... 146
- Accessing the OPC UA Server ..... 147
- Diagnostics** ..... 148
- Entities** ..... 148
- Interface ..... 148
- Channel ..... 148
- Folders & Files** ..... 148
- Folders ..... 148
- Files ..... 148
- About Versions** ..... 149
- This Document ..... 149

- Plugin ..... 149
- Assembly ..... 149
  - General** ..... 150
  - What Does the Plugin Do?** ..... 150
  - Features** ..... 150
  - Supported Clients** ..... 150
  - Purpose & Use Cases** ..... 151
- Documentation of the RESTful API ..... 151
  - Installation** ..... 151
  - Configuration** ..... 151
  - Diagnostics** ..... 151
  - Entities** ..... 151
  - Folders & Files** ..... 151
  - About Versions** ..... 151
- This Document ..... 151
- Plugin ..... 151
- Assembly ..... 151
  - Access** ..... 153
  - Request & Response** ..... 153
- Specifications of the Request ..... 153
- Tools for Example Request ..... 153
- General Request ..... 154
- General Response ..... 155
  - Read & Browse** ..... 157
- Request ..... 157
- Response ..... 158
  - Write** ..... 160
- Request ..... 160
- Response ..... 161
  - Create** ..... 162
- Request ..... 162
- Response ..... 163
  - Update** ..... 164
  - Delete** ..... 164
  - General** ..... 165
  - What Does the Plugin Do?** ..... 165
  - Features** ..... 165
  - Supported Specifications** ..... 165
    - Purpose & Use Cases ..... 166
- Script Interface Plugin Development Guide ..... 166
  - Installation** ..... 166
  - Configuration** ..... 166
  - Diagnostics** ..... 166
  - Entities** ..... 166
  - Folders & Files** ..... 166
  - About Versions** ..... 167
- This Document ..... 167
- Plugin ..... 167
- Assembly ..... 167
- Example Code Script ..... 168
  - Managing Scripts** ..... 168
  - Script Handling** ..... 169
  - The Built-in Script Editor** ..... 170



Going Live .....	171
Useful Shortcuts .....	172
<b>Viewing the Script Log</b> .....	172
<b>Writing Script Code</b> .....	172
JavaScript Basics .....	172
Script API .....	174
Accessing UKI-4.0 .....	174
Find a Node and Log its Value .....	174
Node Events .....	175
Write a Value to a Node .....	176
Async Functions .....	176
Reading Node's Values Using a Synchronous Read .....	178
File Access .....	178
Basic File Operations .....	178
Reading and Writing Text Files .....	179
HTTP Handlers .....	181
Dynamically Generate a Text Page .....	181
Generate an HTML page with an input form .....	182
Display History Values as Diagram .....	184
Extended HTTP Programming .....	189
WebSocket Connections in an HTTP Handler .....	189
Simple Echo WebSocket .....	192
"Chat" with Background Send Operations .....	193
Sending HTTP Requests .....	197
Simple GET requests .....	197
Accessing a JSON-based REST API .....	198
<b>Recommended Best Practices</b> .....	199
Storing Passwords Securely .....	200



UKI-4.0UKI-4.0UKI-4.0  
UKI-4.0UKI-4.0UKI-4.0

# ® - Middleware for Industry

## 4.0



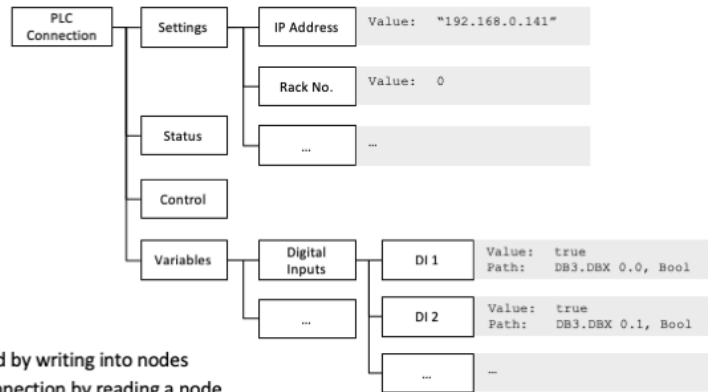
The industrial Middleware for any type of connection.



 Unified architecture over all interfaces

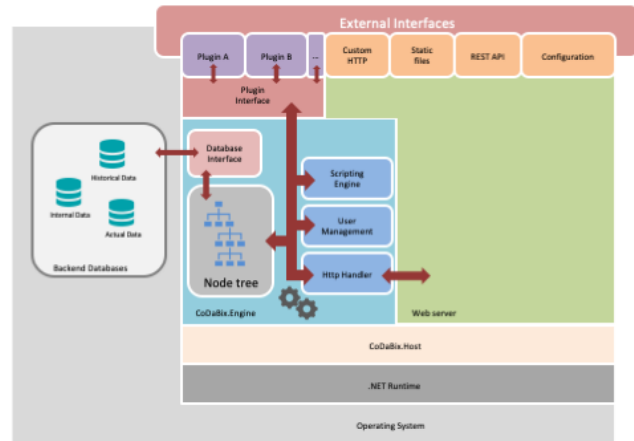
- **Everything is a node**
  - Only thing to handle is a node
  - Interface will not change
  - Completely mappable to **OPC UA**

- **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



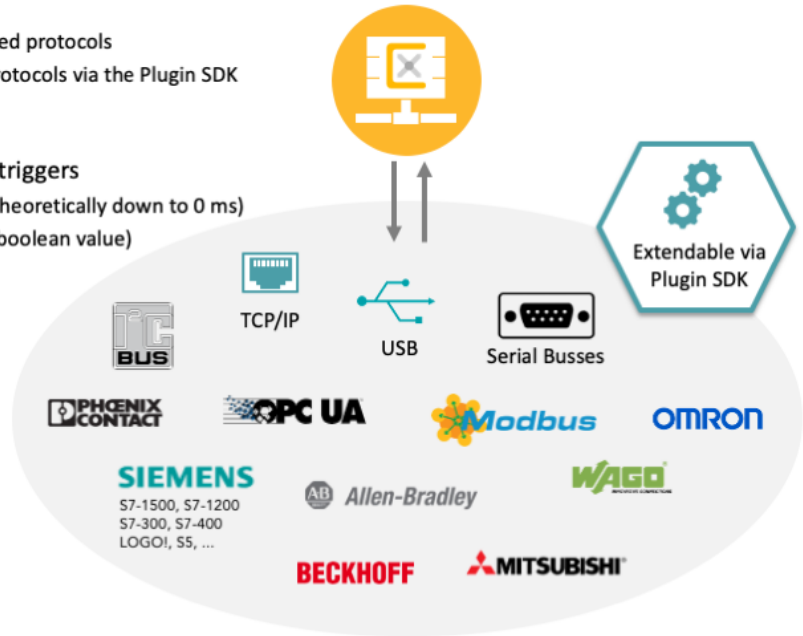
## System architecture

- **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



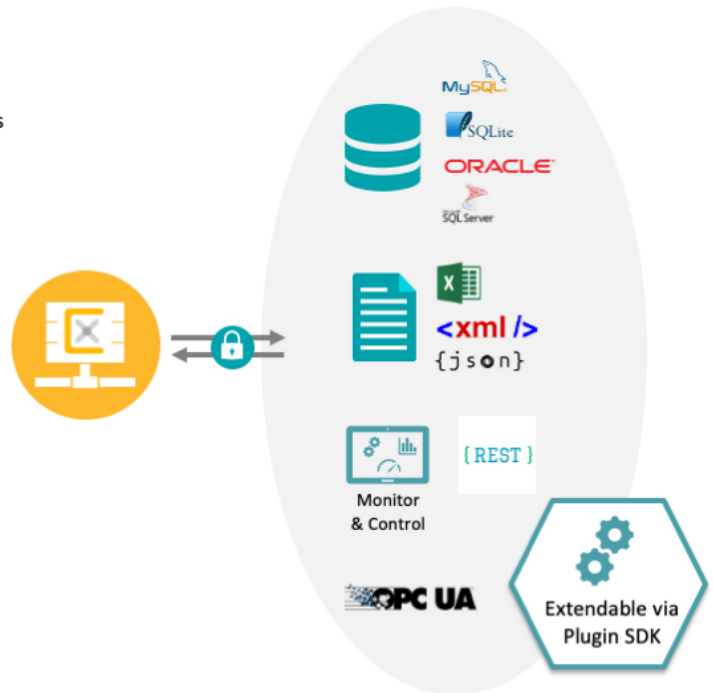
**Connect to production machines and sensors**

- **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

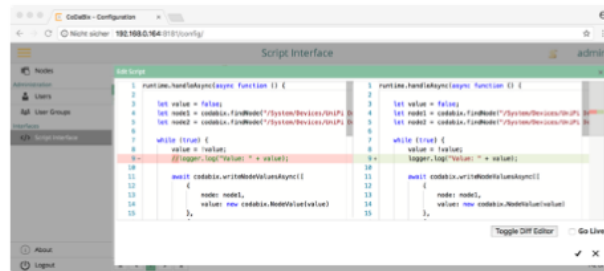
- **Publish your data**
  - Convert data into required format
  - Execute on triggers
- **Remote monitoring**
  - Interpret data in real time and publish results
- **Remote control**
  - Fetch and validate data from cloud
  - Execute user-defined function (Remote Procedure Call)



## 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

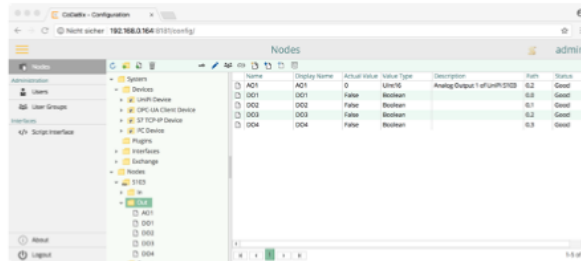
- Online editor
  - IntelliSense
    - Syntax highlighting
    - Tooltips
    - Autocompletion
  - Diff view



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

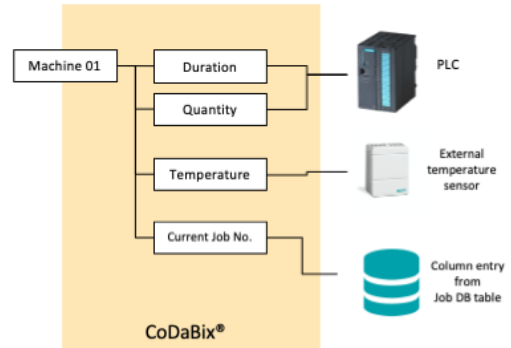
## ⚙️ Configuration and administration

- **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

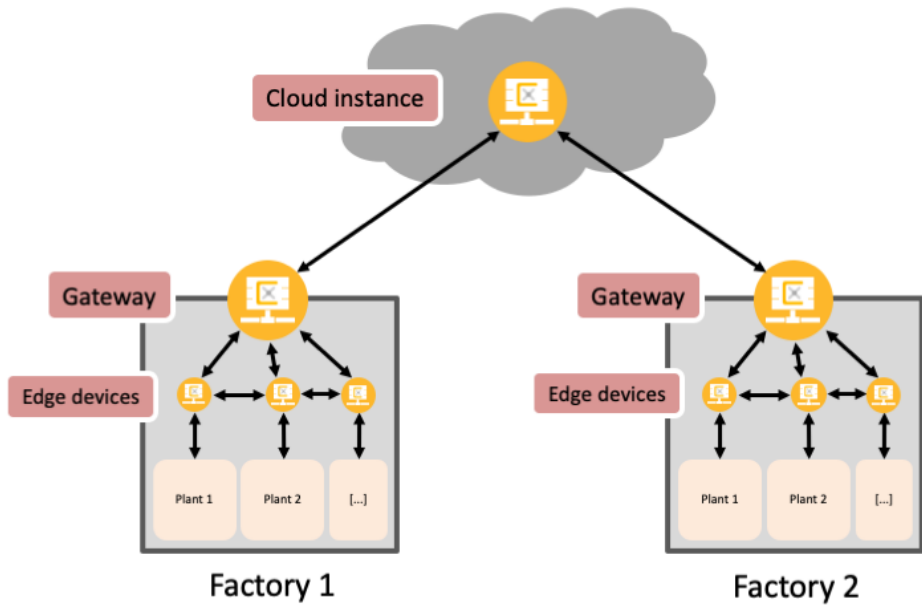


**Unify, harmonize and extend interfaces**

- **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



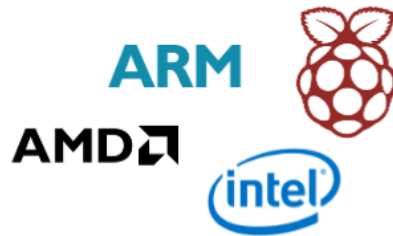
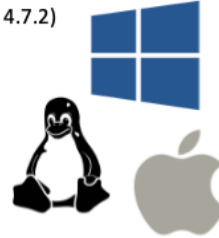


## Cascade multiple CoDaBix instances

- **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](#)

[See product presentation](#)

## UKI-4.0® is a bidirectional universal 'Communication Data Bridge'

UKI-4.0® is the key element in Industry 4.0 - used in projects for factory automation, building control and much more.

### The System is used as:

- **Middleware**
  - connection to MES/PPS
  - realizing Industry 4.0 networks (e.g. cloud of your organisation in intranet or internet)
- **Edge Device**
  - UKI-4.0® runs on tiny systems too (MiniPC, Raspberry Pi). The networking property provide excellent requirements to be used as a Industry 4.0 Edge Device.

UKI-4.0® is fully developed under C# (.NET Standard bzw. .NET Core).

So UKI-4.0® runs on any platform which is supported by .NET Core / .NET Standard.

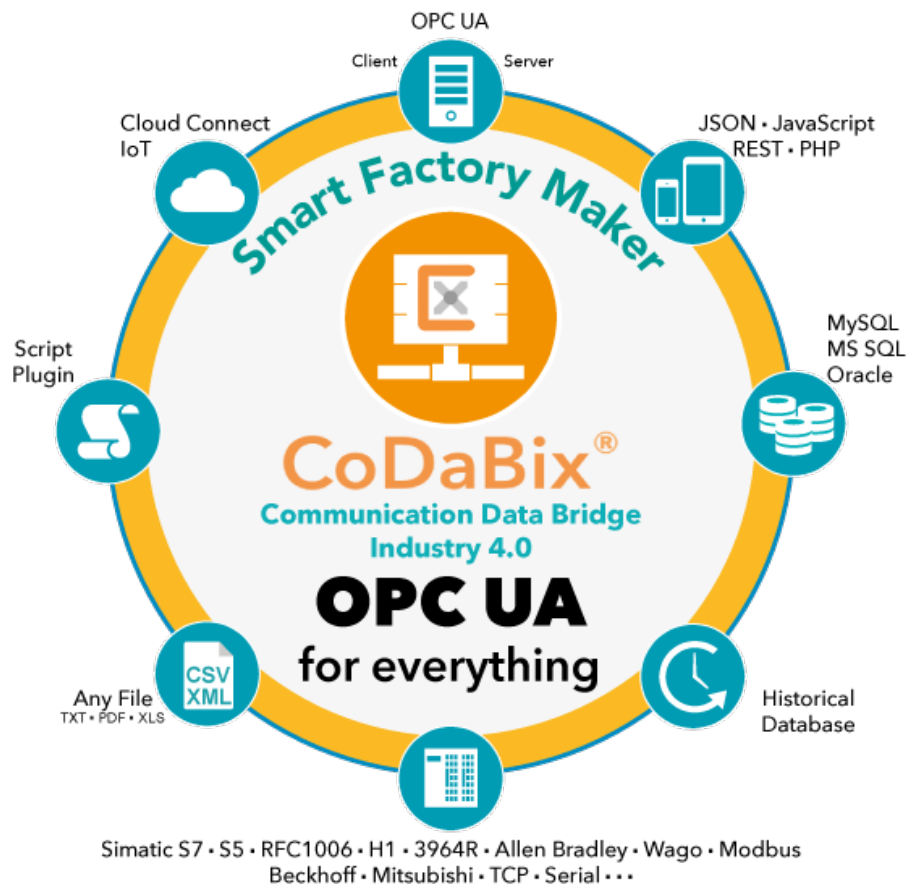
### OPC UA Companion Specification

Due to the flexible modeling of the data structure in UKI-4.0®, we have created the best conditions for mapping complete OPC UA Companion specifications.

The logical functionality can be easily implemented by the user with integrated Typescript. Everything in one system.

No matter OPC UA Methods or dynamic nodes need to be created / removed - anything is possible.





## Use / Application

With UKI-4.0® a heterogeneous machine environment with different intelligence characteristics, control systems, data formats and connection (bus) protocols can be lifted (and harmonized) to a customer-specific standard.

The core is the OPC UA conform structure as well as the central OPC UA connection to all “UKI-4.0® data”.

“Real” machine data and “virtual” variables (= data from / to database, text file, or web interface etc.) is processed in the same manner.

The Node tree ( “Variable tree”) by OPC UA standard thus plays a major part in UKI-4.0®. The data of the connected sources can be mapped arbitrarily in a logical and hierarchical tree structure.

Each element is treated as a “Node” in UKI-4.0®.

All variables and their properties such as name, current value, timestamp, min / max value and so on are provided in the internal high-speed cache. The access is possible in a bidirectional way as a read and write access.

The integrated database allows historical storing of any desired process variables.

For further processing by higher-level systems (for example MES, ERP) UKI-4.0® provides its own standardized, but also customized interfaces in form of “plugins”.

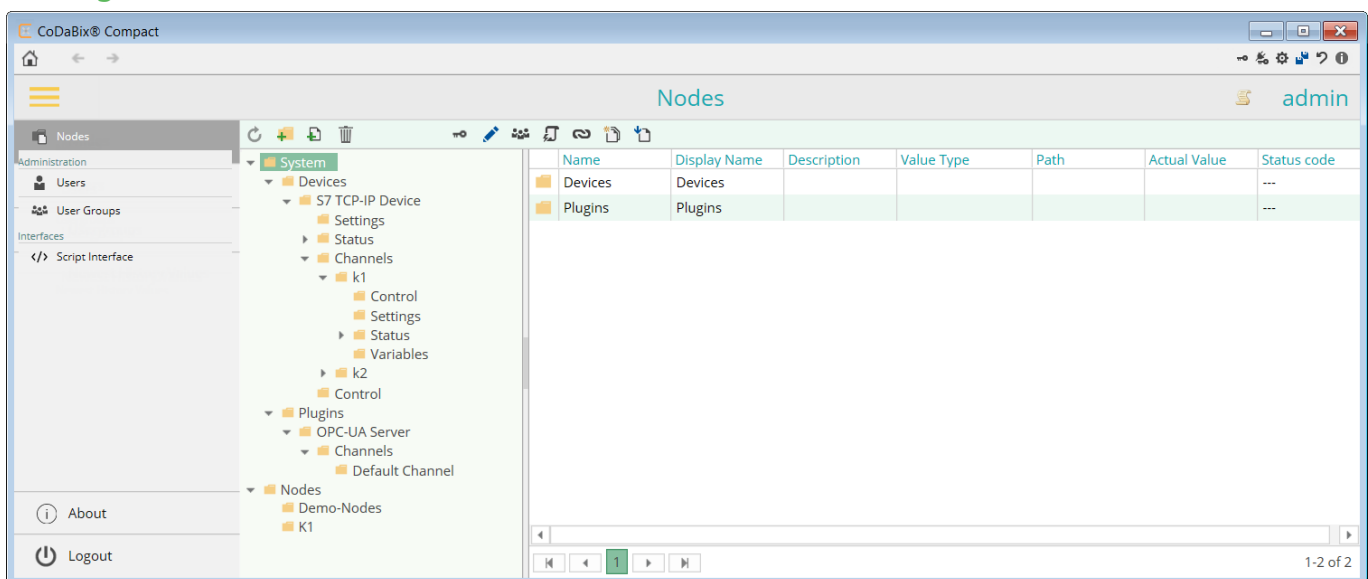
Access to current values and stored historical values is easily possible via OPC UA, REST / JSON, directly via the database or by script plugin (JavaScript).

This allows UKI-4.0® to be connected to any data source or data storage in horizontal and vertical direction. The data is automatically exchanged (after a change), occurring event- or trigger-controlled.

## Plugins for Devices / Interfaces and Data Exchanges

- OPC UA (Server and Client)
- OPC Classic (DA)
- SQL databases
  - MySQL
  - MSSQL
  - Oracle SQL
  - any other database system if required
- CSV / XML / text files
- web application via REST / JSON interface
- SAP
- Devices
  - SIMATIC S7, S5
  - RFC-1006 (ISO on TCP)
  - SINEC H1
  - Allen Bradley
  - Beckhoff
  - Schneider
  - Mitsubishi
  - Omron
  - your PLC is missing - no problem contact us
- OPC Classic

## Configuration



The UKI-4.0 ® configuration is done via the integrated web interface. For the plugins a configurator for each on its own is available.

In general, the parameterization via an XML config file is possible. The format of the XML config file is freely selectable and documented accordingly.

Therefore, UKI-4.0 ® can be combined and connected in a simple way with available project planning system / parameterization applications (e.g. COMOS).

## Requirements

UKI-4.0 ® is supported by following operation systems:

- Windows
  - Workstation Windows 7/8/10 (32/64 Bit)

- Server Windows Server 2008 R2/2012/2016/2019
- Linux
  - Debian 9 or higher
  - Fedora 32 or higher
  - Ubuntu 18.04 or higher
  - OpenSuse Leap 15.0 or higher
- Raspberry Pi (z.B. UniPI, KUNBUS)
  - Raspbian 9 (Stretch) or higher
- macOS

[Detailed requirements see here.](#)

The requirements for CPU power, memory and hard drive are dependent on the desired data throughput and data volume.

# Installing

UKI-4.0 UKI-4.0 UKI-4.0  
UKI-4.0 UKI-4.0 UKI-4.0

UKI-4.0 is available for the following operating systems and hardware configurations:

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

## UKI-4.0 for Windows

You can install UKI-4.0 on both workstation and server editions of Windows.

UKI-4.0 runs on both 32-bit and 64-bit versions of Windows, though we **recommend** to run the **64-bit** version of UKI-4.0 for stability reasons.

## System Requirements

### Supported Operating Systems

UKI-4.0 **for Windows** is supported on the following operating systems:

Windows Version	Workstation Operating System	Server Operating System
6.1.7601	Windows 7 SP1 <sup>1)</sup>	Windows Server 2008 R2 SP1 <sup>2)</sup> (Option "Full Installation")
6.2.9200	Windows 8 (Windows Embedded 8 Standard)	Windows Server 2012 (Option "Server with GUI")
6.3.9600	Windows 8.1	Windows Server 2012 R2 (Option "Server with GUI")
10.0.14393	Windows 10 [IoT] Enterprise 2016 LTSC	Windows Server 2016 (Option "Server with Desktop Experience")
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 with Desktop Experience", or "Server Core" with <a href="#">Server Core App Compatibility FOD</a> installed)
	Windows 10 Version 1809 (October 2018 Update)	Windows Server, version 1809 (with <a href="#">Server Core App Compatibility FOD</a> installed)
10.0.18363	Windows 10 Version 1909 (November 2019 Update)	Windows Server, version 1909 (with <a href="#">Server Core App Compatibility FOD</a> installed)
10.0.19041	Windows 10 Version 2004 (May 2020 Update)	Windows Server, version 2004 (with <a href="#">Server Core App Compatibility FOD</a> installed)
10.0.19042	Windows 10 Version 20H2 (October 2020 Update)	Windows Server, version 20H2 (with <a href="#">Server Core App Compatibility FOD</a> installed)
10.0.19043	Windows 10 Version 21H1	Windows Server, version 21H1 (with <a href="#">Server Core App Compatibility FOD</a> installed)

### Hardware Requirements

Recommended: 64-bit Quad-Core CPU, 8 GB RAM

### Back-end Database Requirements

By default, UKI-4.0 uses an **embedded database (SQLite)** which doesn't have any additional

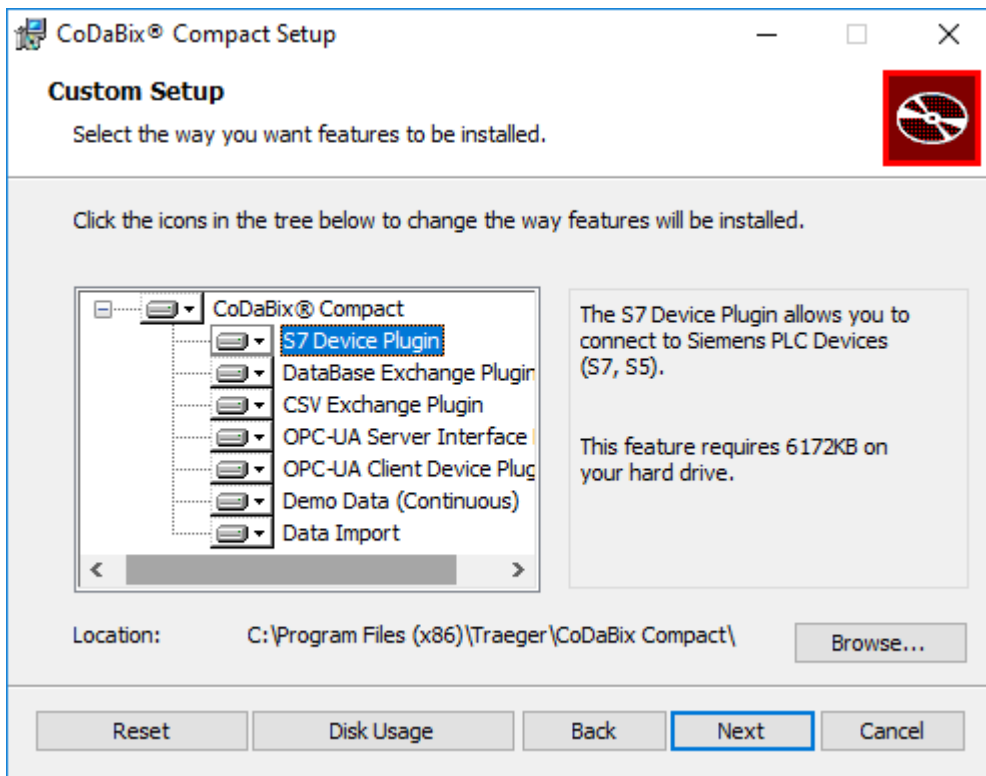
requirements.

However, if you plan on using MySQL, MariaDB, or Microsoft SQL Server as a back-end database, please make sure it is MySQL 8.0 or higher, MariaDB 10.3 or higher, or Microsoft SQL Server 2012 or higher. We recommend running MySQL/MariaDB/MSSQL on the **same machine** as UKI-4.0 .

## UKI-4.0UKI-4.0 UKI-4.0UKI-4.0 Setup and First Start

### UKI-4.0 Setup

In order to install UKI-4.0 , download and run the UKI-4.0 [installer](#) (MSI file). You can select which Plugins are to be installed with UKI-4.0 .



### Install and Uninstall Plugins

If you want to change your UKI-4.0 installation at a later time (e.g. to remove or add plugins), simply start the UKI-4.0 Installer again (alternatively, you can also open the Windows Control Panel, click on "Programs and Features", select "UKI-4.0 " and click "Change").

You can then click "Change" in the installer to change the UKI-4.0 plugins to be installed or uninstalled.


### Updating UKI-4.0

When you have already installed UKI-4.0 and want to update to a newer version, you don't need to uninstall the older version. Simply run the UKI-4.0 Installer of the newer version, and it will automatically update UKI-4.0 .

**Note:** If you had installed UKI-4.0 as a service, you will need to reinstall the service in the UKI-4.0 Settings dialog after the update.

### First Start



You can start UKI-4.0 by double-clicking the UKI-4.0 shortcut  on the Desktop, or by running the following command on the command line (e.g. on Windows Server Core):

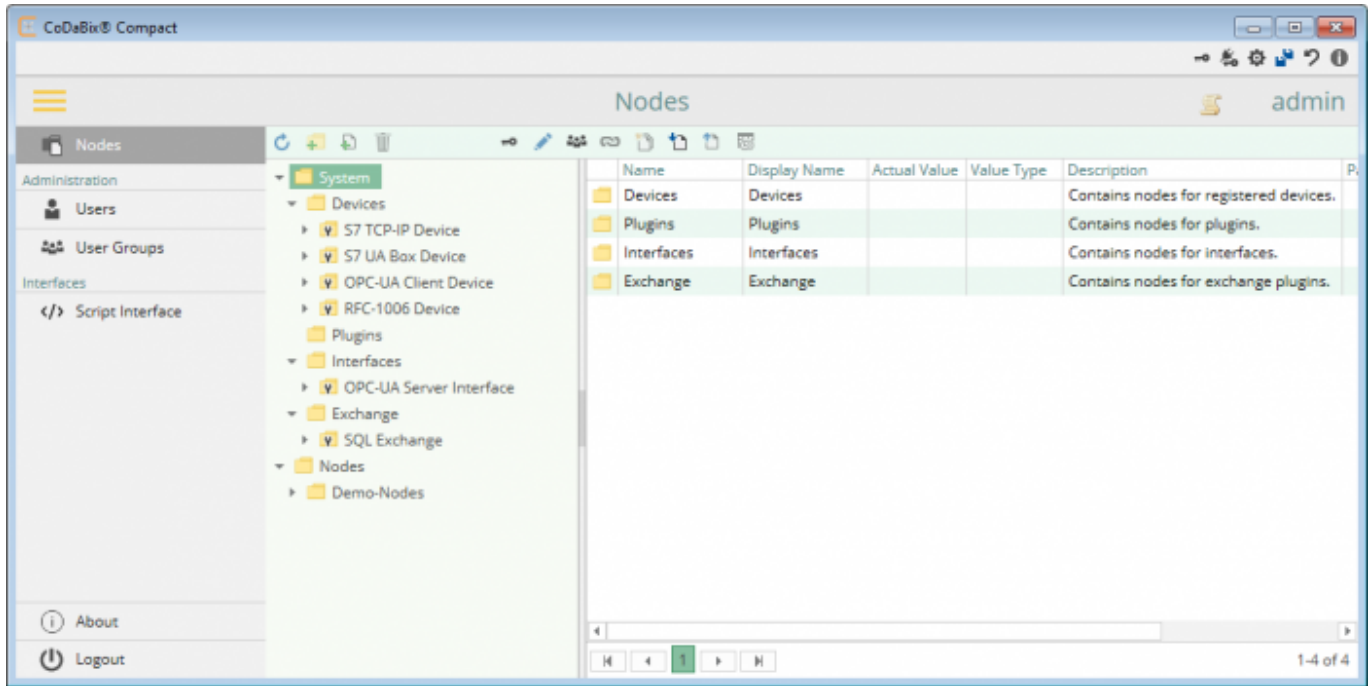
```
"%ProgramFiles%\TIS\UKI-4.0 \UKI-4.0 -ui.exe"
```

The first time you start UKI-4.0 , you will be asked to specify a **project directory** (best would be an empty folder) where UKI-4.0 will be allowed to store its data (settings, database, configuration, logfiles etc).

After you confirm the selection, the **Settings Dialog** (see next section) will appear. Click OK to apply the settings. Then, a dialog box will ask you to set an admin password. You will need this password later for the configuration.

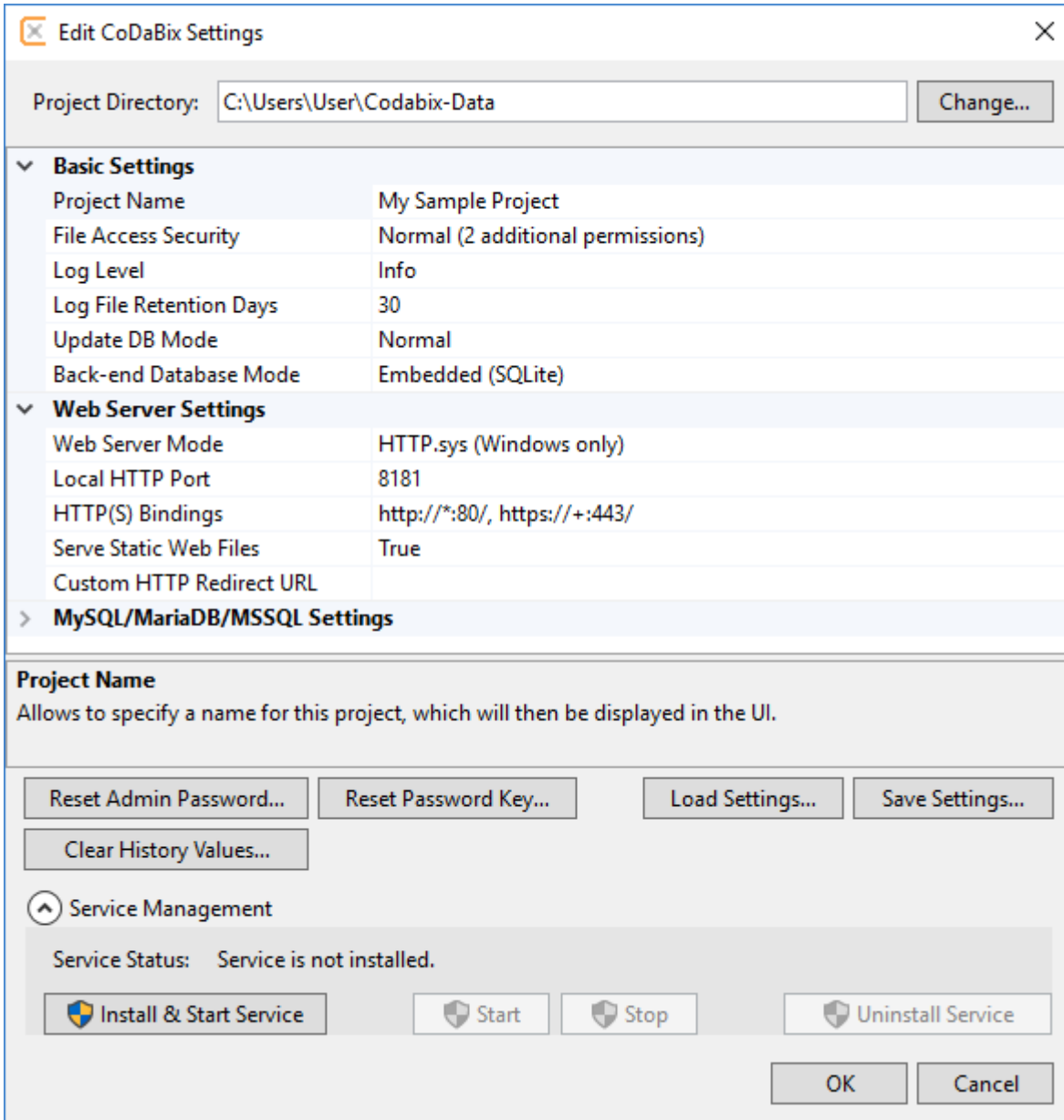
**Note:** A dialog of the Windows Firewall may appear asking to allow access to UKI-4.0 . This is caused by the OPC UA Server Plugin, which creates an OPC UA Server at port 4840 by default. If you click on “Allow access”, other machines in the network may access this OPC Server.

After you have set the password and UKI-4.0 has started, the **Web Configuration** login screen appears. You can now log in with the username **admin** and your previously set password.



## UKI-4.0UKI-4.0 Project Settings

UKI-4.0 provides a number of settings which you can configure for the selected project. To edit the settings, click on the gear toolbar item at the top right (⚙️) which opens the UKI-4.0 settings dialog.



## Basic Settings

Setting Name	Description
<b>Project Directory</b>	<p>The project directory specifies the folder where UKI-4.0 stores the <b>project settings</b> (all of the following settings), the <b>back-end database</b> (if <i>Back-end Database Mode</i> is set to "Embedded (SQLite)"), log files and configuration files for the plugins. It can be retrieved using the environment variable <code>%UKI-4.0 ProjectDir%</code>, e.g. in Scripts.</p> <p>It contains the following folders:</p> <ul style="list-style-type: none"> <li>• <b>log</b>: Contains the UKI-4.0 runtime log files and log files from the Entity Model.</li> <li>• <b>plugins</b>: Contains configuration files for plugins.</li> <li>• <b>webfiles</b>: In this folder, you can place static files that should be accessible through the embedded UKI-4.0 web server for URLs starting with <code>/webfiles/</code>, if the setting <i>Serve Static Web Files</i> is enabled.</li> <li>• <b>dashboard</b>: Similar to <b>webfiles</b>, but for URLs starting with <code>/dashboard/</code> in order to override the embedded Dashboard and use your own one.</li> <li>• <b>userdata</b>: You can place custom files in this folder (e.g. for use by Scripts). This folder is guaranteed to not be used differently in future UKI-4.0 versions.</li> </ul> <p>When creating a <b>backup</b>, the contents of these subfolders will be included in the backup.</p>
<b>Project Name</b>	<p>Allows you to specify a name for the current project, which will be displayed in the UKI-4.0 application's title bar and used for the default backup file name.</p>



Setting Name	Description
<b>Access Security</b>	<p>Within UKI-4.0 , the project logic can access files on your system by creating File Nodes (and then accessing them using the HTTP Access URL or using an OPC UA Client), or by creating a Script that uses the <code>io.file</code> and <code>io.directory</code> namespaces. To prevent a user that has the UKI-4.0 admin password (or is able to create Scripts or File Nodes in UKI-4.0 ) from accessing arbitrary files on the system (especially when running as a service), you can allow or deny access to specific paths.</p> <p>Furthermore, you can define alternative access credentials to be used when accessing the path (which is done by impersonation), or additionally add the path as a network resource.</p> <p>By default, level <code>Normal</code> is set wich permits the folders <code>plugins</code>, <code>log</code>, <code>userdata</code>, <code>webfiles</code>, <code>dashboard</code> within the <b>project directory</b> for read/write access.</p>
<b>Log Level</b>	<p>Specifies the detail level of logging which UKI-4.0 shall use when writing logfiles. Only those log entries are written that have at least a severity of this level. 'Off' means no logfiles are created.</p>
<b>Back-end Database Mode</b>	<p>Specifies which back-end database to use. By default, UKI-4.0 uses an embedded database (SQLite) which doesn't need any additional configuration. The embedded database will be stored in the files UKI-4.0 <code>db.db</code> and UKI-4.0 <code>historydb.db</code> in the specified project directory.</p> <p>However, you can also use MySQL 8.0 or higher, MariaDB 10.3 or higher, or Microsoft SQL Server 2012 or higher as a back-end database. If you select “MySQL/MariaDB” or “Microsoft SQL Server”, you will need to fill in the settings grouped under the “MySQL/MariaDB/MSSQL Settings” category.</p> <p>When creating a <b>backup</b>, the contents of the back-end database will be included in the backup. You can also migrate the current back-end database from SQLite to MySQL/MariaDB/MSSQL (or MySQL/MariaDB/MSSQL to SQLite) by creating a backup, switching the Database Mode and then restoring the backup.</p>

## Web Server Settings

Setting Name	Description
<b>Web Server Mode</b>	<p>Specifies the mode of the embedded web server that UKI-4.0 uses in order to provide the UKI-4.0 Web Configuration, the REST/JSON interface, registered Script HTTP Handlers and optionally static files.</p> <p><b>Kestrel</b> is built on socket APIs and is used by default as it generally provides the best performance. When using HTTPS, SSL certificates need to be provided as PFX (PKCS #12) files (<code>.pfx</code>, <code>.p12</code>) and will be stored in the UKI-4.0 Settings file.</p> <p>The <b>Windows HTTP Server API</b> is supported on Windows 10 Version 1607/Windows Server 2016 and higher, and allows to share ports with other IIS web sites. When using HTTPS, SSL certificates need to be placed in the Windows Certificate Store (see below). In order to use remote bindings, UKI-4.0 will need to be installed as a service.</p>
<b>Local HTTP Port</b>	<p>Specifies the TCP at which the embedded web server listens for local connections. By default, UKI-4.0 uses the port <b>8181</b>.</p> <p><b>Note:</b> When you use <b>Windows HTTP Server API</b> as <i>Web Server Mode</i> and you want to use ports ≤ 1024 (e.g. port 80), or you want to access the UKI-4.0 Web Configuration or the REST/JSON interface from other machines on the network, you need to install UKI-4.0 as a service (see below). In the latter case you will also need to open the “Service HTTP(S) Bindings” options and enable the option “Use the Local Port as remote HTTP Binding”. Otherwise, only ports &gt; 1024 are allowed and only the local machine can access UKI-4.0 via HTTP.</p>

Setting Name	Description
<b>Service HTTP(S) Bindings</b>	<p>You can enable HTTP and HTTPS bindings, containing the hostname, port and the SSL certificate (for HTTPS), for remote connections. This allows you to connect to UKI-4.0 from remote machines, optionally over an authenticated and encrypted <b>TLS connection</b>.</p> <p><b>Note:</b> To use a SSL certificate for <b>Windows HTTP Server API</b> Web Server Mode, it must be stored in the <b>Personal</b> or <b>Web Hosting</b> Certificate Store of the <i>Local Computer</i> account in Windows and you must have a <i>private key</i> for the certificate. Intermediate certificates (also called CA Cert) must be stored in the <b>Intermediate Certificate Authorities</b> Store.</p> <p>To manage certificates in Windows, you can use MMC by running <code>certlm.msc</code> and then opening the <i>Personal</i> folder (or <i>Web Hosting</i>, resp.) for your certificate and the <i>Intermediate Certificate Authorities</i> folder for your intermediate certificates or you can use PowerShell by switching to the path <code>Cert:\LocalMachine\My</code> (or <code>Cert:\LocalMachine\WebHosting</code>) for your certificate and <code>Cert:\LocalMachine\CA</code> for your intermediate certificates.</p>
<b>Serve Static Web Files</b>	<p>If enabled, the embedded web server will serve static files placed in the <code>webfiles</code> folder of the project directory for URLs starting with <code>/webfiles/</code>. This allows you in combination with the Script HTTP Handler feature to develop Web Apps served by UKI-4.0 (e.g for visualization).</p>
<b>Custom HTTP Redirect URL</b>	<p>If not specified, requests for the root path (<code>/</code>) will be redirected to the UKI-4.0 Web Configuration. Alternatively, you can specify a custom redirect URL. This is useful if you want users to directly get to your custom Web App served by the embedded web server, when entering the UKI-4.0 address in their browser.</p>

### MySQL/MariaDB/MSSQL Settings

(applies only if *Back-end Database Mode* is “MySQL/MariaDB” or “Microsoft SQL Server”)

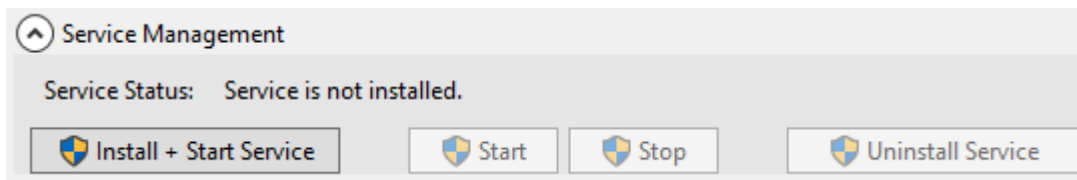
Setting Name	Description
<b>Hostname</b>	<p>Specifies the hostname of the MySQL/MariaDB/MSSQL server. When using MSSQL and the port is empty, this will be interpreted as <b>data source</b> (e.g. <code>&lt;ComputerName&gt;\&lt;InstanceName&gt;</code>). Otherwise, this is the hostname for TCP connections.</p>
<b>Port</b>	<p>Specifies the TCP port of the MySQL/MariaDB/MSSQL server. Can be empty to use the default port (for MySQL/MariaDB) or to use the hostname as data source (for MSSQL).</p>
<b>Database Name</b>	<p>Specifies the database name which UKI-4.0 shall use on the MySQL/MariaDB/MSSQL server. If the specified database does not exist, UKI-4.0 will automatically create the database and the necessary tables.</p>
<b>Login Name</b>	<p>Specifies the username which UKI-4.0 shall use to connect to the MySQL/MariaDB/MSSQL Server.</p>
<b>Password</b>	<p>Specifies the password for the username.</p>

### Installing UKI-4.0 as a Service

You can install UKI-4.0 as a service. This allows you to run UKI-4.0 automatically and permanently in the background (like on a Server) without having to explicitly start the UKI-4.0 application. Furthermore, this enables other machines in the network to access the UKI-4.0 instance per HTTP (e.g. opening the UKI-4.0 Web Configuration or using the REST/JSON interface), if you enable the option “Use the Local Port as remote HTTP Binding” in the “Service HTTP(S) Bindings” options.

Running UKI-4.0 as a service also allows to use HTTP Ports ≤ 1024, e.g. using Port 80.

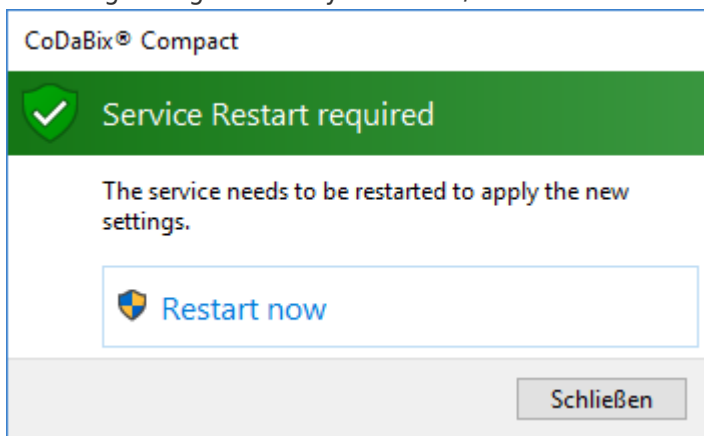
After opening the UKI-4.0 settings dialog by clicking on the gear toolbar item at the top right (⚙️), it will show the section “Service Management” that allows you to install, start, stop and uninstall UKI-4.0 as a service:



You can install and start UKI-4.0 as a service by clicking the **“Install & Start Service”** button. This will show an UAC dialog (because installing a service requires administrator rights).

To uninstall the service, click the “Uninstall Service” button.

**Note:** When you change a setting like the project directory or the HTTP port while UKI-4.0 is running as a service, the service needs to be restarted (or reinstalled) to apply the new settings. This is indicated by the following dialog that lets you restart/reinstall the service:



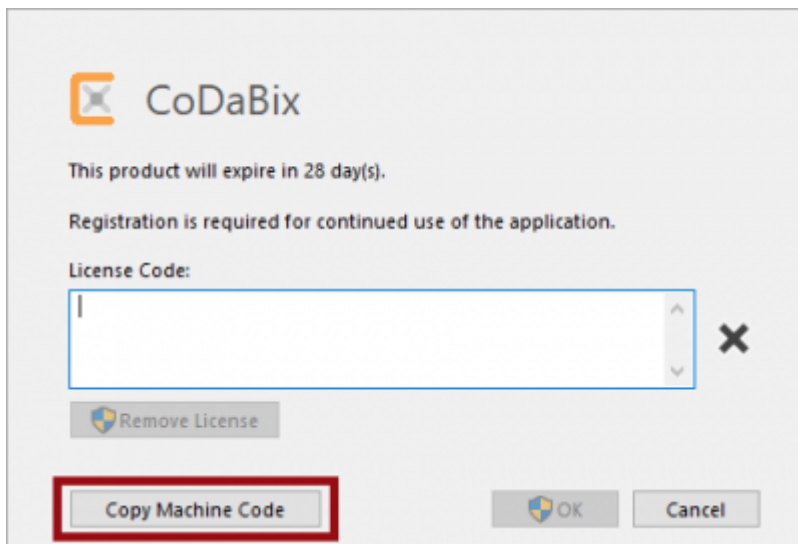
**Note:** The service is configured to automatically restart after a failure. This ensures UKI-4.0 can restart in a clean, defined state after a serious error has occurred.

## License Management

### Machine Code

When you order a UKI-4.0 license, you will need to provide the machine code for the machine where UKI-4.0 is running.

To get the machine code, in the UKI-4.0 application click on the key symbol (🔑) to open the license dialog. Then, click on the button “Copy Machine Code” to copy the local machine code into the clipboard.



UKI-4.0UKI-4.0  
UKI-4.0UKI-4.0

# for Linux

## System Requirements

### Supported Operating Systems

UKI-4.0 **for Linux** is supported on the following Linux distributions:

- Debian 9 (Stretch) or higher (x64, ARM64, ARM32)
  - including derivates such as Raspberry Pi OS (for Raspberry Pi)
- Fedora 32 or higher (x64)
- Ubuntu 18.04 or higher (x64, ARM64, ARM32)
- OpenSUSE Leap 15.0 or higher (x64)

### Hardware Requirements

- **Raspberry Pi:**
  - UKI-4.0 for Linux (ARM64) can run on a Raspberry Pi 4 or newer (with a 64-Bit OS).
  - UKI-4.0 for Linux (ARM32) can run on a Raspberry Pi 2 or newer.
  - For optimal performance, we recommend running UKI-4.0 (**ARM64**) on a **Raspberry Pi 4** (or newer) with 4 GB or 8 GB RAM, using an ARM64 OS.
- Other machines:
  - Recommended: 64-bit Quad-Core CPU (x64/ARM64), 8 GB RAM, 64-Bit OS and UKI-4.0

### Back-end Database Requirements

By default, UKI-4.0 uses an **embedded database (SQLite)** which doesn't have any additional requirements.

However, if you plan on using MySQL, MariaDB, or Microsoft SQL Server as a back-end database, please make sure it is MySQL 8.0 or higher, MariaDB 10.3 or higher, or Microsoft SQL Server 2012 or higher.

### Unsupported Features

Some features are currently not available in UKI-4.0 for Linux. This includes:

- Windows-only plugins like Melsec QJ Device Plugin, H1 Device Plugin, AKLAN Device Plugin
- Accessing OPC Classic (COM) Servers using the OPC UA Client Device Plugin

- Impersonation for files with different credentials (or establishing of SMB network connections), e.g. for File Nodes or when using the CSV Exchange Plugin
- GUI window (you can open the UKI-4.0 Web Configuration in a browser, but administrative tasks like setting the Project Directory, creating a backup etc. need to be done in the UKI-4.0 Shell console application)

## Installing UKI-4.0 and First Start

### Installing UKI-4.0 for Linux

To install UKI-4.0 for Linux, download the .setup file and execute the following command to make the file executable:

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

Start the setup by typing:

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

The setup will guide you through the installation.

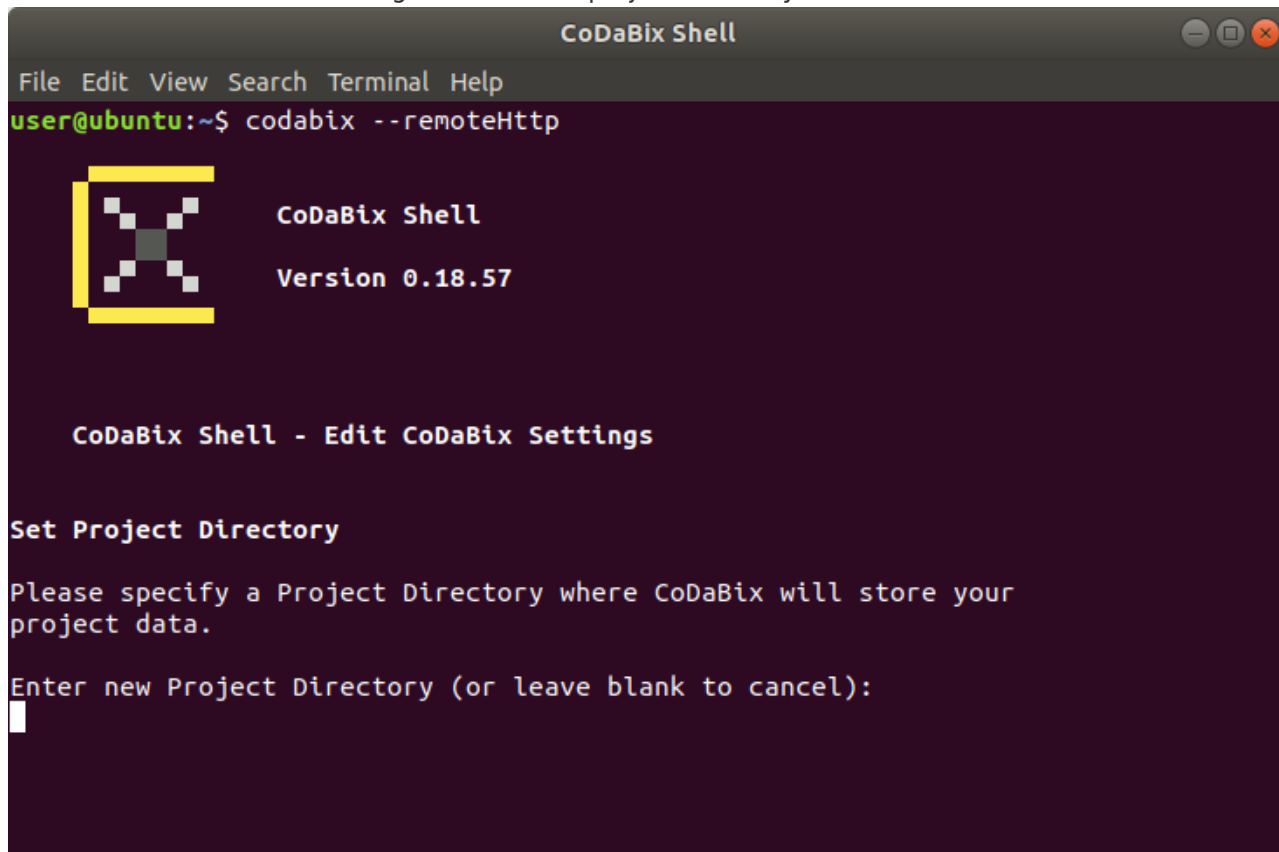
Older installations of UKI-4.0 will not be removed to enable a later roll-back. The database backup that is needed for such a roll-back can be created during the setup.

### First Start

To start the console-based UKI-4.0 Shell, run the following command:

```
UKI-4.0
```

- UKI-4.0 should now be starting and ask for a project directory:



- Enter the full path where you want to store the UKI-4.0 project directory (usually a folder in your

home directory) (see UKI-4.0 [Project Settings](#)). Then, press enter two times to apply the settings and restart UKI-4.0 .

- UKI-4.0 will now ask to reset the Admin password; please enter a new password here:

```

My Sample Project - CoDaBix Shell
File Edit View Search Terminal Help
Select an option (or leave blank to apply the current settings):
Settings applied!

CoDaBix Engine starting (Project Directory: '/home/user/My Sample Project', Local/Remote HTTP Port: '8181')...
Connecting to back-end database...
Initializing back-end database... Please do not turn off your computer.
Initializing back-end database... Please do not turn off your computer.
Loading nodes...
Loading plugins...
Starting plugin 'CoDaBix Socket Device Plugin'...
Starting plugin 'CoDaBix RFC-1006 Device Plugin'...
Starting plugin 'CoDaBix I²C Device Plugin'...
Starting plugin 'CoDaBix Modbus Device Plugin'...
Starting plugin 'CoDaBix S7 Device Plugin'...
Starting plugin 'CoDaBix SQL Exchange Plugin'...
Starting plugin 'CoDaBix OPC UA Client Device Plugin'...
Starting plugin 'CoDaBix Extension XML Plugin'...
Starting plugin 'CoDaBix Extension AutoSubscribe Plugin'...
Starting plugin 'CoDaBix OPC UA Server Interface Plugin'...
Starting plugin 'CoDaBix CSV File Server'...
Starting plugin 'CoDaBix File Import Plugin'...
Starting plugin 'CoDaBix Database Plugin'...
CoDaBix Engine started.

CoDaBix Shell (My Sample Project) - Reset Admin Password

Please enter the new Admin password for the CoDaBix Web Configuration (or leave blank to cancel).

Username: admin
Password:

```

- Now, you can open <http://localhost:8181/config/> in a browser on your machine (if your Linux distribution was installed with GUI) to open the UKI-4.0 Web Configuration, or you can use <http://<IP-address>:8181/config/> in a browser from another machine to access it over the network.

## Run <sup>UKI-4.0</sup>UKI-4.0 <sup>UKI-4.0</sup> as a Service

### Install <sup>UKI-4.0</sup> as a Service

To install UKI-4.0 as a service, start the UKI-4.0 Shell by executing the following command:

```
UKI-4.0
```

- Choose the option **6** from the command line menu to invoke the Service Management:

```

CoDaBix® Compact (Shell)
Starting plugin 'CoDaBix Database Plugin'...
Starting plugin 'CoDaBix File Import Plugin'...
Starting plugin 'CoDaBix OPC UA Server Interface Plugin'...
Starting plugin 'CoDaBix CSV File Server'...
CoDaBix Engine started.

CoDaBix Compact (Shell) - Main Menu
CoDaBix Engine Status: Running.

To access the Web Configuration GUI from the local machine, open the
URL 'http://localhost:8181/config/'.

Options:
1) Edit Settings
2) Reset Admin Password
3) Create Backup
4) Restore Backup
5) License Management
6) Service Management
8) Open OS Shell
9) Exit
> 6
    
```

- By choosing option **1** in the Service Management you can now install and start UKI-4.0 as a Service:

```

CoDaBix® Compact (Shell)

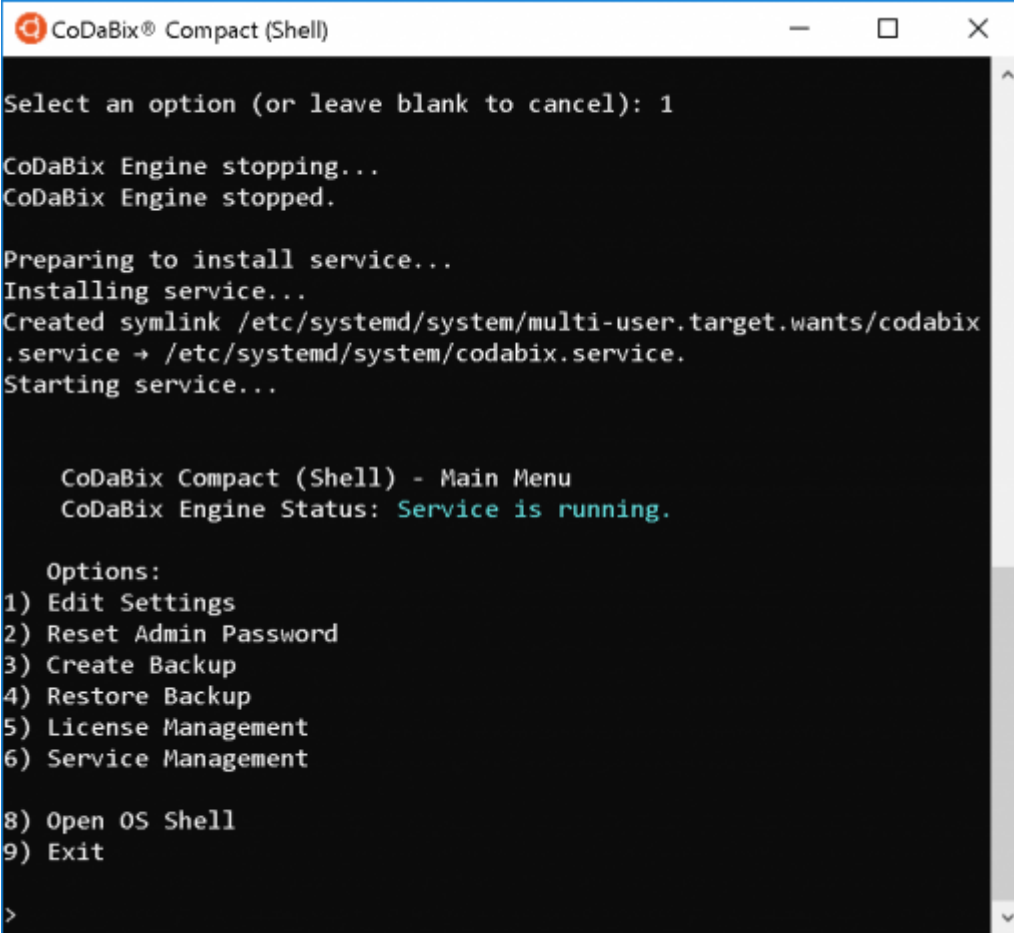
Options:
1) Edit Settings
2) Reset Admin Password
3) Create Backup
4) Restore Backup
5) License Management
6) Service Management
8) Open OS Shell
9) Exit
> 6

CoDaBix Compact (Shell) - Service Management
Current Service Status: Service is not installed.

1) Install + Start Service
2) (Re-)Start Service
3) Stop Service
4) Uninstall Service

Select an option (or leave blank to cancel): 1
    
```

- After successfully starting the service you will see the status **Service is running** on the command line:



```
CoDaBix® Compact (Shell)
Select an option (or leave blank to cancel): 1
CoDaBix Engine stopping...
CoDaBix Engine stopped.

Preparing to install service...
Installing service...
Created symlink /etc/systemd/system/multi-user.target.wants/codabix
.service → /etc/systemd/system/codabix.service.
Starting service...

CoDaBix Compact (Shell) - Main Menu
CoDaBix Engine Status: Service is running.

Options:
1) Edit Settings
2) Reset Admin Password
3) Create Backup
4) Restore Backup
5) License Management
6) Service Management

8) Open OS Shell
9) Exit
>
```

- Now you can close the UKI-4.0 Shell with **9** and the service will keep running in background.
- From now on the service will be automatically started in case of a reboot of the operating system.

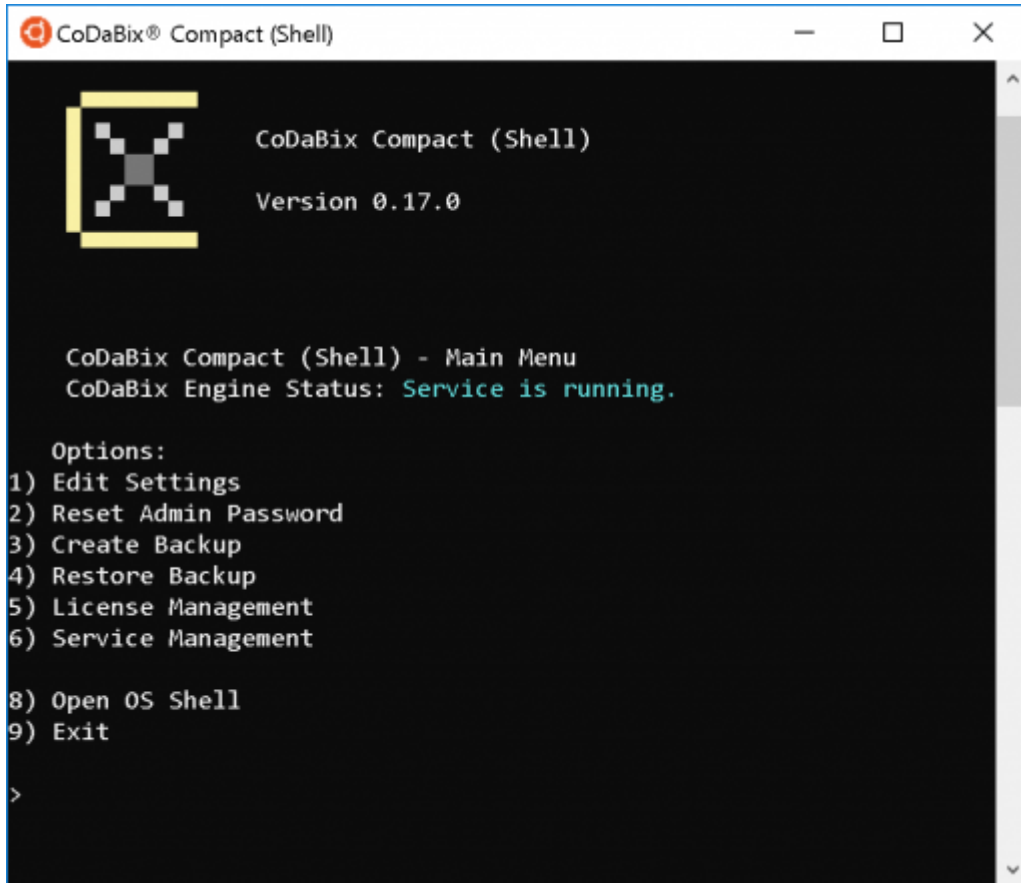
## Status of the UKI-4.0 Service

To retrieve the current status of the UKI-4.0 Service (*Running, Stopped*) start the UKI-4.0 Shell:

```
UKI-4.0
```

The current status of the service will now be displayed:





```
CoDaBix® Compact (Shell)
CoDaBix Compact (Shell)
Version 0.17.0

CoDaBix Compact (Shell) - Main Menu
CoDaBix Engine Status: Service is running.

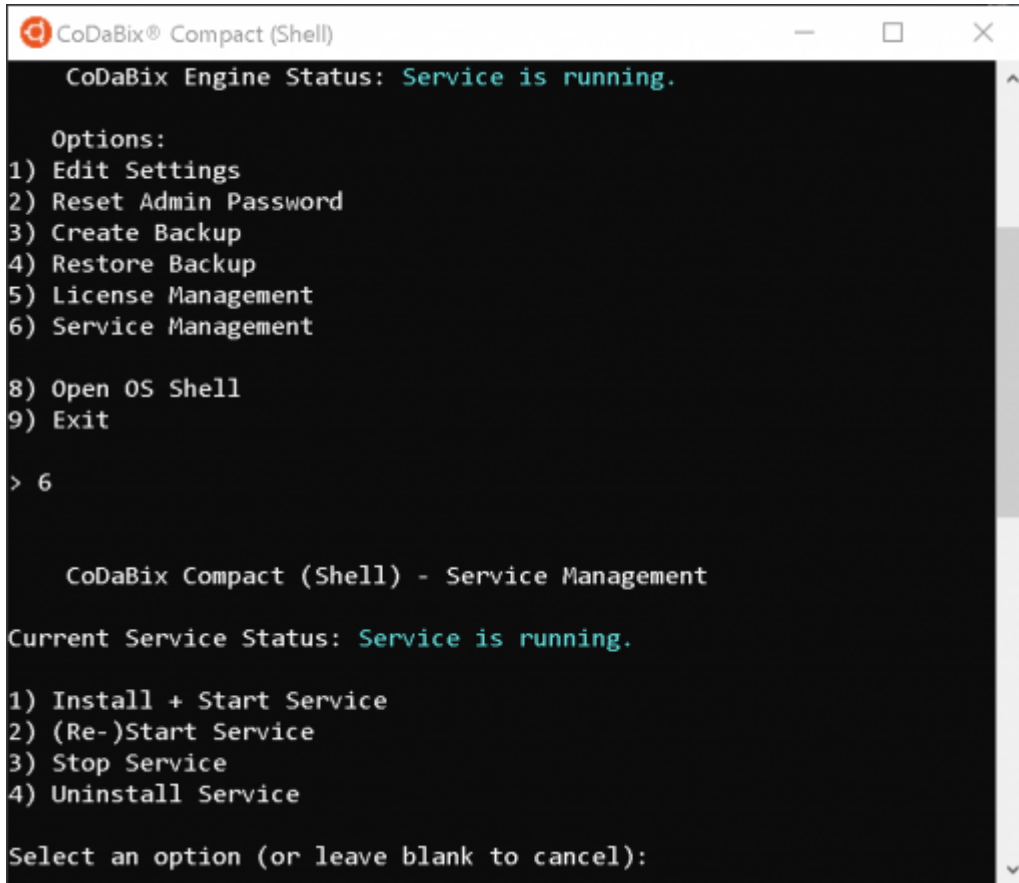
Options:
1) Edit Settings
2) Reset Admin Password
3) Create Backup
4) Restore Backup
5) License Management
6) Service Management
8) Open OS Shell
9) Exit
>
```

## (Re-)start, stop and uninstall the UKI-4.0 Service

In order to (re-)start, stop, or uninstall the UKI-4.0 Service open the UKI-4.0 Shell:

UKI-4.0

- Call the Service Management via option **6**
- Now choose the requested function:



```
CoDaBix® Compact (Shell)
CoDaBix Engine Status: Service is running.

Options:
1) Edit Settings
2) Reset Admin Password
3) Create Backup
4) Restore Backup
5) License Management
6) Service Management
8) Open OS Shell
9) Exit
> 6

CoDaBix Compact (Shell) - Service Management
Current Service Status: Service is running.

1) Install + Start Service
2) (Re-)Start Service
3) Stop Service
4) Uninstall Service

Select an option (or leave blank to cancel):
```

## License Management

### Machine Code

When you order a UKI-4.0 license, you will need to provide the machine code for the machine where UKI-4.0 is running.

To get the machine code, execute the following command on the command line (bash):

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

This will output the local machine code to the terminal, for example:

```
$ UKI-4.0 --machinecode
U7f3lqK5bG04fIVUNX5yhWgNlG6PnSKbvHuY6Ml610gAAgAA9+w50G0p0AFgKyfQPvjB09rId87pLs29nC+CHQ==
```

Alternatively, you can display the machine code in the UKI-4.0 Shell application, by entering `license` in the main menu to open the license management. The local machine code will be displayed at the top:

```

My Sample Project - CoDaBix Shell
File Edit View Search Terminal Help
> license

CoDaBix Shell (My Sample Project) - License Management
Machine Code: U7F3lqK5bG04fIVUNX5yhWgNlG6PnSKbvHuY6Ml610gAAgAA9+w50G0p0AFgKyfQPvJb09rId87pLs29nC+CHQ==

1) CoDaBix: This product will run for 180 more minute(s).

Plugin Licenses:
2) CoDaBix CSV File Server: This product will run for 180 more minute(s).
3) CoDaBix Database Plugin: This product will run for 180 more minute(s).
4) CoDaBix File Import Plugin: This product will run for 180 more minute(s).
5) CoDaBix I²C Device Plugin: This product will run for 180 more minute(s).
6) CoDaBix Modbus Device Plugin: This product will run for 180 more minute(s).
7) CoDaBix OPC UA Client Device Plugin: This product will run for 180 more minute(s).
8) CoDaBix OPC UA Server Interface Plugin: This product will run for 180 more minute(s).
9) CoDaBix RFC-1006 Device Plugin: This product will run for 180 more minute(s).
10) CoDaBix S7 Device Plugin: This product will run for 180 more minute(s).
11) CoDaBix Socket Device Plugin: This product will run for 180 more minute(s).
12) CoDaBix SQL Exchange Plugin: This product will run for 180 more minute(s).

Options:
1) Show/Change CoDaBix license
2 ... 12): Show/Change plugin license
A) Save Machine Code into file

Select an option (or leave blank to cancel):

```

## Siemens IOT2050 Image

This section describes the necessary steps, to put the UKI-4.0 [Siemens IOT2050 Image](#) into service.

### Content of the image

For this image the Siemens IOT2050 Example Image was used as base image.

Additionally the following components have been installed:

- Components for DNS Discovery
  - Avahi Daemon
    - This service allows the so-called DNS-SD (DNS Service Discovery) based on the Apple Bonjour Protocol. This way the device can be discovered in the local network by its hostname and the “local” domain without the need to know its IP address.
  - Automatic hostname
    - During startup an automatic hostname which is based on the serial number of the device is generated. So every device gets a unique hostname that can be determined with the serial number which is printed on the device housing.
- UKI-4.0 + UKI-4.0 service
- Set `root` password: UKI-4.0 `-iot`

### Writing the image

For writing the image to an SD card we recommend the following tool:

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

# Putting into service

After writing the image to the SD card, put the card into the SD card slot of the IOT2050 device and connect the device to the power supply. To enable access via network, the device needs to be part of the same network as your computer.

## Automatic Hostname

The hostname of the device is generated according to the following pattern:

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

For the next steps we assume that the device has the serial number **M4A98757**. So make sure to exchange it with the actual number of your device.

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

To check if the device is accessible you can execute the following command on the command line:

Attention: The Apple Bonjour service has to be installed on your system for the next steps to work.

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

If this command is not successful either the IOT2050 has not yet completely booted or it is not accessible from your network.

## Accessing the UKI-4.0 web configuration

If the ping command has been successful you can access the UKI-4.0 web configuration in a browser under the URL <http://UKI-4.0 -iot-M4A98757.local:8181>.

Login data:

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

## Access via SSH

To access the device via SSH, execute the following command on a command line:

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

Login data:

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

1) 2)

Update **KB3063858** (x64 or x86) is required.

Additionally, to use licenses with a machine code, the Windows Management Framework 3.0 (**KB2506143**) must be installed.


UKI-4.0UKI-4.0UKI-4.0  
UKI-4.0UKI-4.0UKI-4.0

# Web Configuration

## Starting the <sup>UKI-4.0UKI-4.0</sup><sub>UKI-4.0UKI-4.0</sub> Application

1. Install UKI-4.0 , see UKI-4.0 [Installation](#).
2. Make the necessary adjustments UKI-4.0 [Setup and First Start](#).



3. Double-click the icon  on the Desktop to start UKI-4.0 , or run the following command on the command line (e.g. on Windows Server Core):

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

4. The UKI-4.0 Login dialog will show.

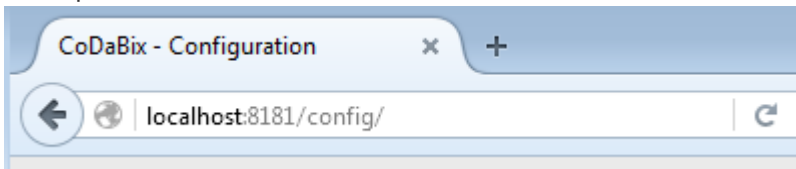
## Configuration

UKI-4.0 can be configured using a web-based GUI. On Windows, UKI-4.0 uses an embedded web browser to display the Web Configuration.

You can also open the Web Configuration with your own browser (e.g. Edge, Chrome, Firefox, Safari). By default, UKI-4.0 uses a local HTTP binding (default port 8181), so you can use the following URL:

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

Example:

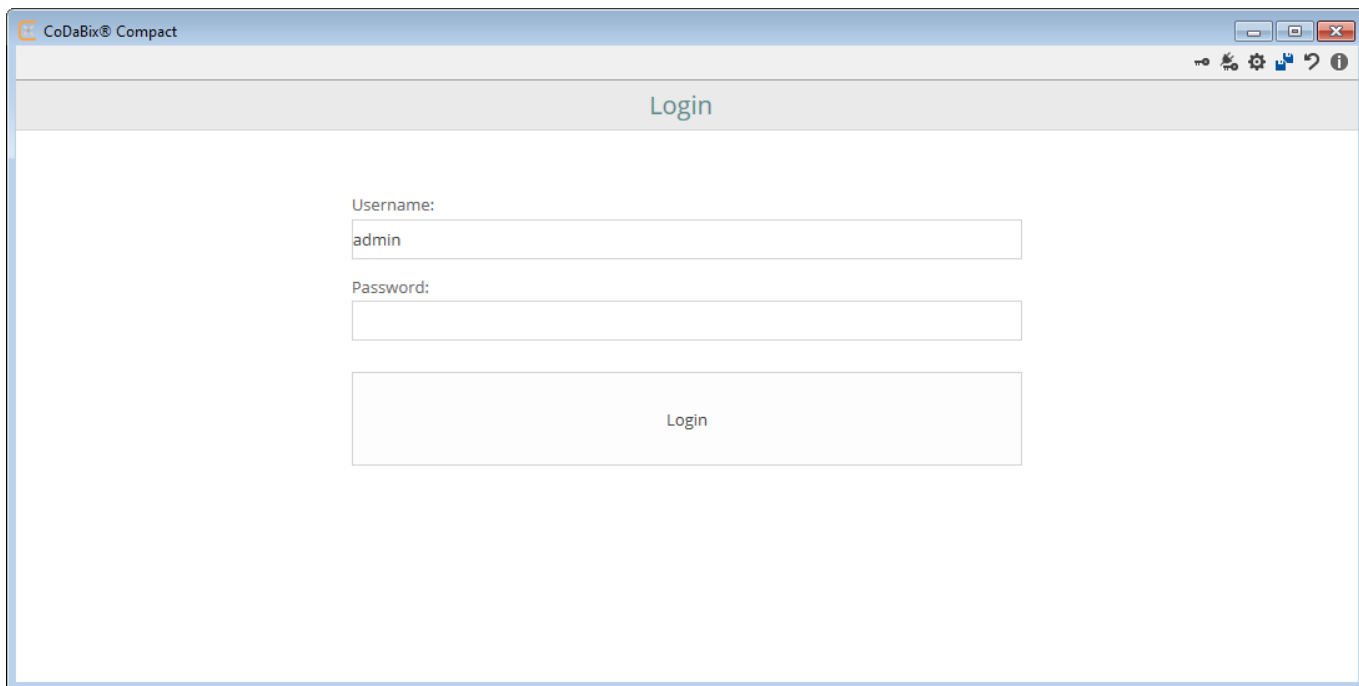


**Please note:** If you want to access the UKI-4.0 Web Configuration from a **remote computer**, you must add a remote HTTP binding in the UKI-4.0 Settings (for UKI-4.0 for Linux this already done by default). To access UKI-4.0 with a browser from a remote computer using HTTP, you can use the following URL:

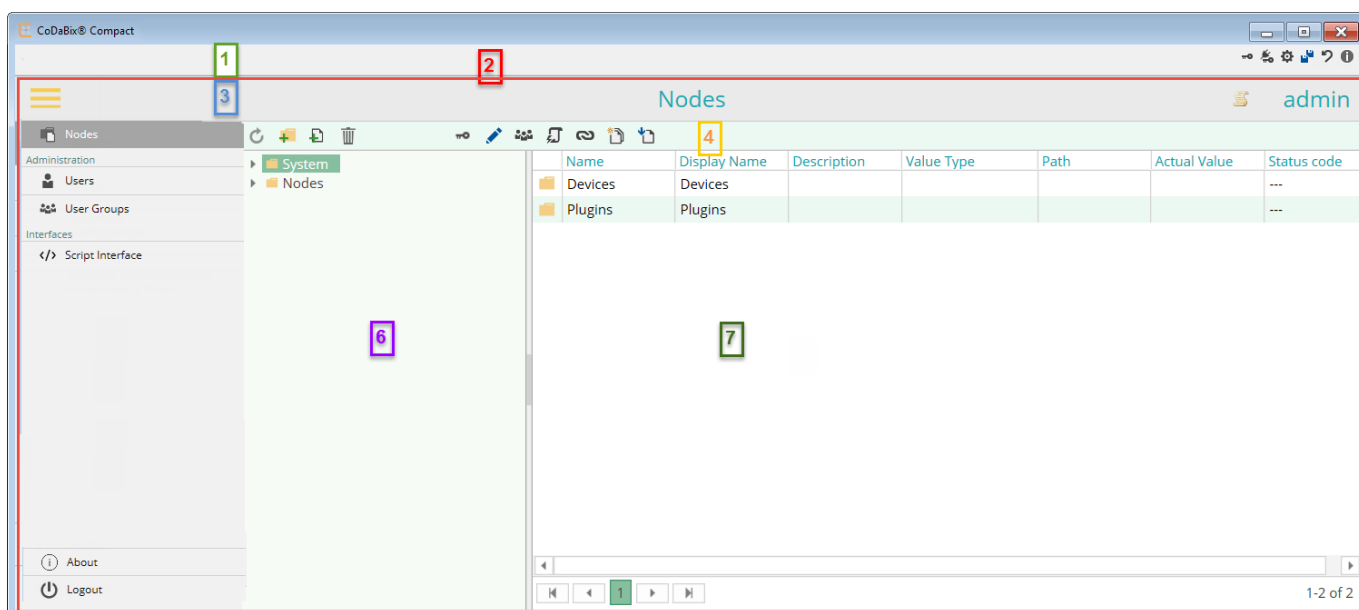
```
http://<hostname>:<port>/config/
```

## General

After starting or calling the URL the login dialog is displayed:



After the login you reach the UKI-4.0 main window.



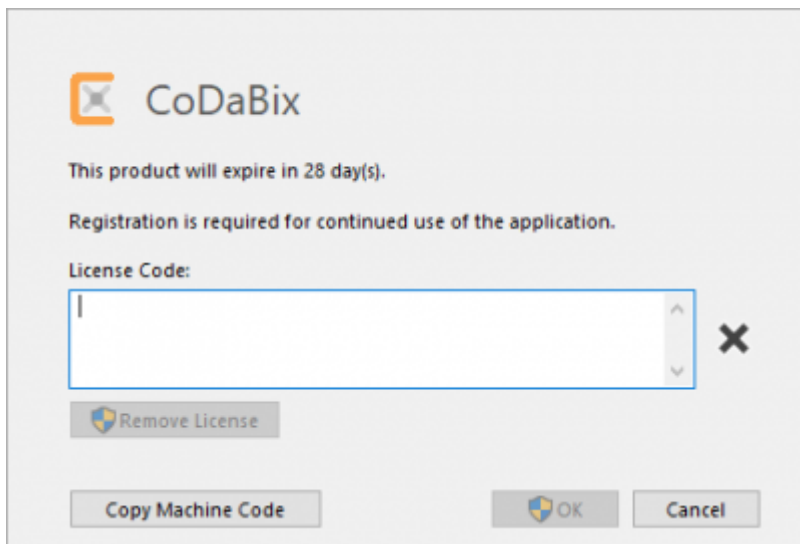
The UKI-4.0 Web Configuration is separated in different parts:

### (1) UKI-4.0 **Toolbar**

This toolbar is only visible if you use the UKI-4.0 application for the configuration. So this bar is only referring to the UKI-4.0 application.

Icon	Description
	Opens the license dialog, in which you can insert, change or delete the license
	See UKI-4.0 <a href="#">Settings</a>
	Shows all information on the license and application

### [Licence Dialog](#)



Field	Description
License Code:	Insert your license key here
Remove License	Remove the license from the computer
Copy Machine Code	Copies the machine code to the clipboard. This is required for the license creation.
OK	The entered license is stored on the computer. If "grayed out" no valid license key was entered.

## About

Left area: Information on the application

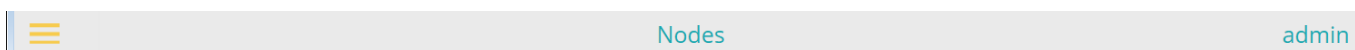
Right area: Information on the last installed license

## (2) Working Area

The working area consists of an embedded browser (red border).

This area is divided into 3 to 4 areas, depending on the choice of menu.

## (3) Title Bar



In the title bar the following actions are possible and the following data is displayed:

Field	Description
	Opens and closes the menu
Nodes	Display of the title of currently selected menu
admin	Display of the name of the user logged in

#### (4) Toolbar

Below the title bar is the toolbar. It is automatically adjusted depending on which menu you are in and which item you have selected.

Folder Node:

Name	Display Name	Description	Value Type
JobName	JobName		String
Machine runni...	Machine runni...		Boolean

Datapoint Node:

Name	Display Name	Description	Value Type
JobName	JobName		String
Machine runni...	Machine runni...		Boolean
Temperature	Temperature		Single

User:

First Name	Last Name	Login Email	Phone Number
Demo	User	demo@user.org	+1
Max	Mustermann	m.m@email.com	+49123456789

User Group:

Name	Type
Test Group	A
Rotating Cutter	A
Injection molding	A

Script Plugins:

Name	Description	Editor Strictness Level	Script State	Current Script State
test	test	Low	Enabled	NotRunning

General:

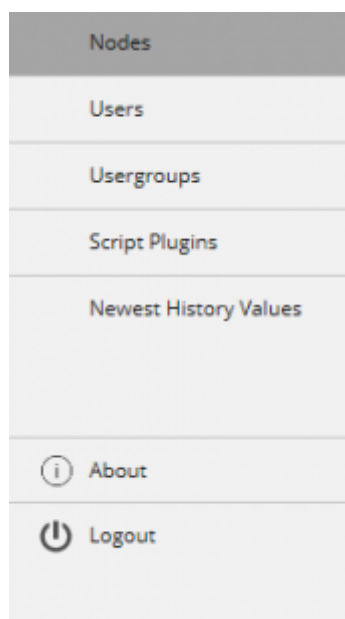
Grayed out icons are not applicable to the selected item.

Icon	Description	Visible in
	Refresh view	all menu
	UKI-4.0 Folder Node	Folder Node, Datapoint Node
	UKI-4.0 Datapoint Node	Folder Node, Datapoint Node
	UKI-4.0 -linked Folder Node	Folder Node, Datapoint Node



Icon	Description	Visible in
	UKI-4.0 -linked Datapoint Node	Folder Node, Datapoint Node
	All icons with this symbol add a new item on the basis of an input dialog	Folder Node, Datapoint Node, User, User Group, Script Plugins
	Delete selected item	Folder Node, Datapoint Node, User, User Group, Script Plugins
	Cancel action	Datapoint Node, User, User Group, Script Plugins
	Saves the edited record or records	Folder Node, Datapoint Node, User, User Group, Script Plugins
	Access to the selected Folder Node or Datapoint Node	Folder Node, Datapoint Node
	Edit selected item	Datapoint Node, User, User Group, Script Plugins
	Add or delete user	User Group
	Add or deselect Folder Node or Datapoint Node	User Group
	Select or deselect the user group for the selected item	Folder Node, Datapoint Node
	create or delete a link	Folder Node, Datapoint Node
	Duplicate the selected item	Folder Node, Datapoint Node
	If you click on a Folder Node, all values of the Datapoint Node are updated. If a Datapoint Node is selected, only this value is updated	Folder Node, Datapoint Node
	Set Datapoint Node value	Datapoint Node
	If available, the historical data of the Datapoint Node is displayed	Datapoint Node
	Edit script	Script Plugin
	Log view of the selected script	Script Plugin

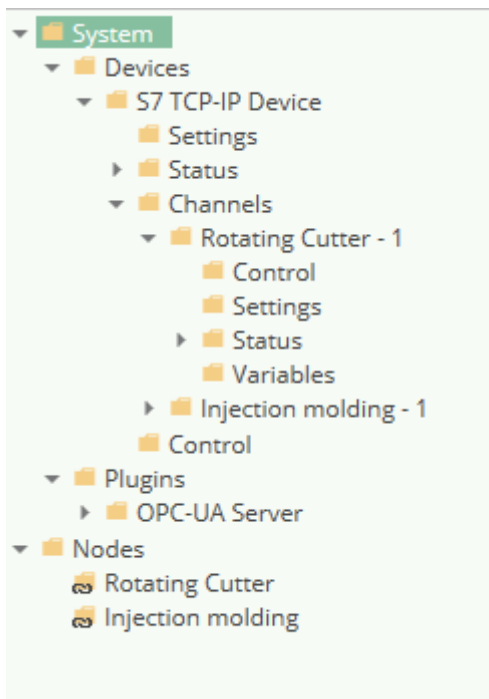
### (5) Menu



By default, the menu is displayed permanently.

Field	Description
Nodes	Current start page. Edit Folder Nodes and Datapoint Nodes
Users	Edit user
User Groups	Edit user groups
Script Plugins	Lightweight JavaScript (TypeScript) plugin that UKI-4.0 extends on custom functions. Executed in a secure environment
Newest History Values	View the of the last registered and stored historical values. Data will only be displayed if a Folder or Datapoint Node has enabled storing historical data
About	Information about the configuration page and, when appropriate, UKI-4.0
Logout	Logout user

## (6) Node Tree



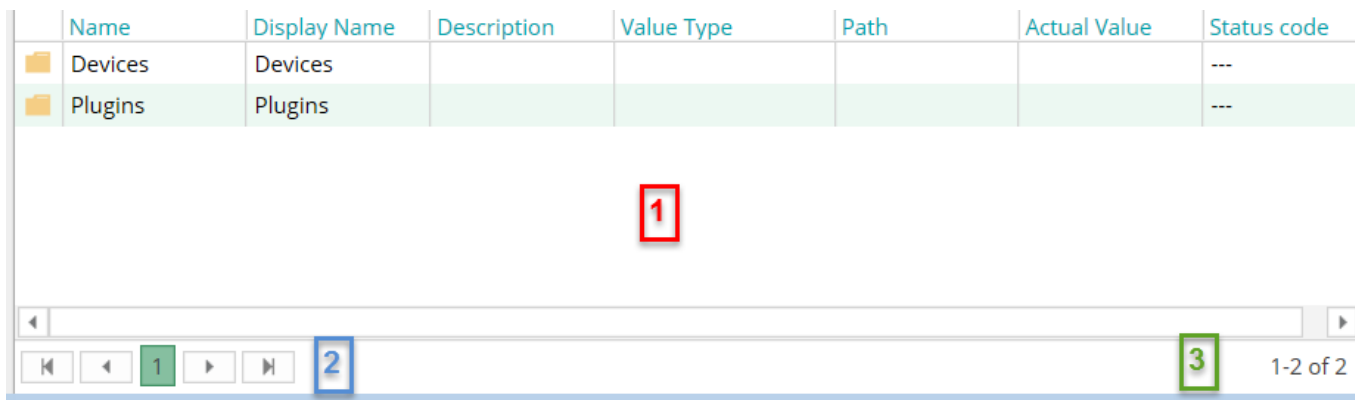
The available Folder Nodes are displayed in a tree structure.

Structure of the Node tree:

Node	Description	Purpose
System	System Nodes	Contains all Nodes that are necessary for the operation of UKI-4.0 . Most Nodes and variables are not editable here. Fixed Folder Nodes are: -System -Device -Plugins -Settings -Status -Channels -Variables -Control
Device	Device Nodes	Contains all Nodes of the devices registered in UKI-4.0
S7 TCP-IP Device	S7 Device Plugin Container	Contains all dataa of the plugins required for UKI-4.0 and the user, for example status, channels, etc.
Settings	Settings of the plugin	Possible Settings of the device
Status	Status of the plugin	General information on the plugin, for example error, plugin started

Node	Description	Purpose
Channels	Node for all channels	Contains all defined channels of the plugin
Channel e.g. Rotating Cutter - 1	Defined channel in the plugin	Contains - general settings of the channel - Status of the channel - all defined Datapoint Nodes
Plugin	Plugin Nodes	Root node for all plug-ins, similar to Devices
Nodes	Root node for user defined nodes	Contains all user defined nodes and variables

### (7) Data Area



Number	Description
1	Data of the selected menu or Folder Node is displayed here
2	Navigation bar, here you switch views of the respective data. Possible navigation: top of page previous page next page bottom of page
3	Number of all records

## Nodes

### What are Nodes?

“The OPC information model is a so-called Full Mesh Network based on nodes. These Nodes can include any kind of meta information, and are similar to the objects of object-oriented programming (OOP). A Node can have attributes for read access (DA, HDA) [...]. Nodes hold process data as well all other types of metadata. The OPC namespace contains the type model.”

Source: wikipedia.org

UKI-4.0 is similarly structured as specified in the OPC UA. Depending on the Node type, a Node has different functions and properties.

A distinction is made between:

- Folder Node
- Datapoint Node
- Link Node
- Directory Node

### Folder Nodes

A Folder Node can also be referred to as a root node, since it may provide additional Folder or Datapoint

Nodes but it itself may may not provide any value.

Set properties are automatically applied to the Child Nodes, except when they define a separate condition.

## Datapoint Nodes

A Datapoint Node in UKI-4.0 is like a variable with additional attributes (properties).

A Datapoint Node must not contain Child Nodes.

## Link Nodes

Link Nodes have only the properties they have in common.

A Link Node points to another Node that has already been created. This way it is possible to be assemble an individual record from several devices, Folder and Datapoint Nodes.

## Directory Nodes

The Directory Node is a Folder Node that directly points to a physical directory and automatically represents the substructure.

# Node Properties

## Common Properties

Field	Description	Data type
Lokal ID	Unique ID in the UKI-4.0 System e.g. used in access through the REST interface	Long
Global ID	Unique ID across UKI-4.0	GUID
Name	Unique name within the Parent Node, for example for the OPC UA addressing, JSON Interface e.g. job number	String
Display Name	Display name for the user e.g. job number	String
Path	e.g.: PLC address (according to S7 Syntax, DB1000.DBB 500, Word) OPC UA Node (3:AirConditioner_1.State) File path (R:\\MachineData) <b>NOTE:</b> The path is not dealt with at the <i>Folder</i> type	String
Max Value Age (ms)	Relevant for a synchronous read. When a synchronous read operation doesn't specify a <b>time to live</b> , this value (in milliseconds) is used instead. This means that if the currently set node value is <b>not</b> older than this age, it will be returned directly; otherwise, the value will be read from the device.	Integer

## Node Types

When creating a Folder Node you can decide between the types:

- Folder
- Directory

### Type: Folder

The Node is a regular Folder Node with no further special properties.

### Type: Directory

The Node is a Folder Node an represents a physical registry in the data system. The Path property of the Node states the registry path (it can contain environmental variables). When browsing Nodes (e.g. via the web configuration) subdirectories are automatically created and aligned in the path property as corresponding Folder Nodes(type **directory**) and Datapoint Nodes (type **file**) with the related file path.

**Note:** The specified path is subject to the **Access Security** restrictions that have been defined in the UKI-4.0 [Project Settings](#).

**Note:** On Windows 10 Version 1511 and older (and Windows Server 2012 R2 and older), e.g. on Windows 7, the maximum file path length is limited to 260 characters (**MAX\_PATH**). On Windows 10 Version 1607 and higher (as well as Windows Server 2016 and higher) longer path names can be used. However, for this you will need to enable the setting “Enable Win32 long paths” in the Windows Group Policy, see [Enabling Win32 Long Path Support](#).

## Datapoint Nodes

Additionally to the previously mentioned [properties](#) the Datapoint Node has the following properties:

Name	Description	Data Type
Description	Description of the datapoint	String
Location	Used during evaluation of conditions. Used to display e.g. of the origin of the message. Arbitrarily assignable	String
Value Types	Data type of the node, further information see table <a href="#">Value Types</a>	enum
Min Value	Minimum value, is required for the evaluation of conditions	Double
Max Value	Maximum value, is required for the evaluation of Conditions	Double
Hysteresis	Threshold value used in the evaluation of conditions when e.g. minor temperature fluctuations are to be compensated.	Double
Scaling Factor	Factor that is added to the currently read value	Double
Scaling Offset	The value is added to the currently read value	Double
Unit	Unit of the value e.g. °C	String
Precision	Number of decimal digits	Number
History Options	Specifies how history values are to be captured and written into the database for this node. <b>No:</b> No history values are being captured for this node. <b>Yes; only on Value Change:</b> When a <b>History Interval</b> is set for the node, history values are being captured by the <i>History Timer</i> for this node from the actual value at the specified history interval; otherwise, history values are being captured when writing a value to the node. A captured value is inserted into the database only if it differs from the last written history value for that node. Additionally, in case of a device variable, a subscription is being created for this node. <b>Yes:</b> When a <b>History Interval</b> is set for the node, history values are being captured by the <i>History Timer</i> for this node from the actual value at the specified history interval; otherwise, history values are being captured when writing a value to the node. Captured history values are inserted into the database. Additionally, in case of a device variable, a subscription is being created for this node.	Enum
History Interval	The interval at which history values shall be captured.	Enum
History Resolution	Specifies the resolution to which numeric historic values are rounded when being captured.	Double

### Note:

- To determine if a captured history value differs from the last history value (for the setting **Yes; only on Value Change**) only properties **Value** and **Status** are considered, but not property **Timestamp** (CreationTimestamp).
- When the **History Options** of a node are changed from **No** to **Yes; only on Value Change** while UKI-4.0 is running, the first captured history value after this change will be written to the database in every case, even when it is not different from the last written history value for that node.
- When the **History Options** of a node are set to **Yes; only on Value Change** and UKI-4.0 is shut down and later restarted, the first captured history value of the node after the restart will be written

into the database in every case, even when it is not different from the last written history value for that node.

- When the **History Options** of a node are set to **Yes; only on Value Change** (or no **History Interval** is specified), the *Creation Timestamp* of the actual value will be kept when capturing the history value; otherwise, the *Creation Timestamp* will be set to the current time.
- In a **Script**, the setting **Yes; only on Value Change** corresponds to the value UKI-4.0 `.NodeHistoryOptions.Subscription | UKI-4.0 .NodeHistoryOptions.ValueChange` (contains UKI-4.0 `.NodeHistoryOptions.Active`), and the setting **Yes** corresponds to the value UKI-4.0 `.NodeHistoryOptions.Subscription` (contains UKI-4.0 `.NodeHistoryOptions.Active`)
- History Values are inserted into the database **asynchronously** in the background. This means that when you write a value to a node (where no *History Interval* is set), even though the write operation finished, the captured history value might not yet be in the database. This is especially true if you set the *DB Update Mode* to **Restricted** in the UKI-4.0 Settings, because captured history values are then inserted into the database only every 5 seconds.

## Value Types

UKI-4.0 provides the following value types:

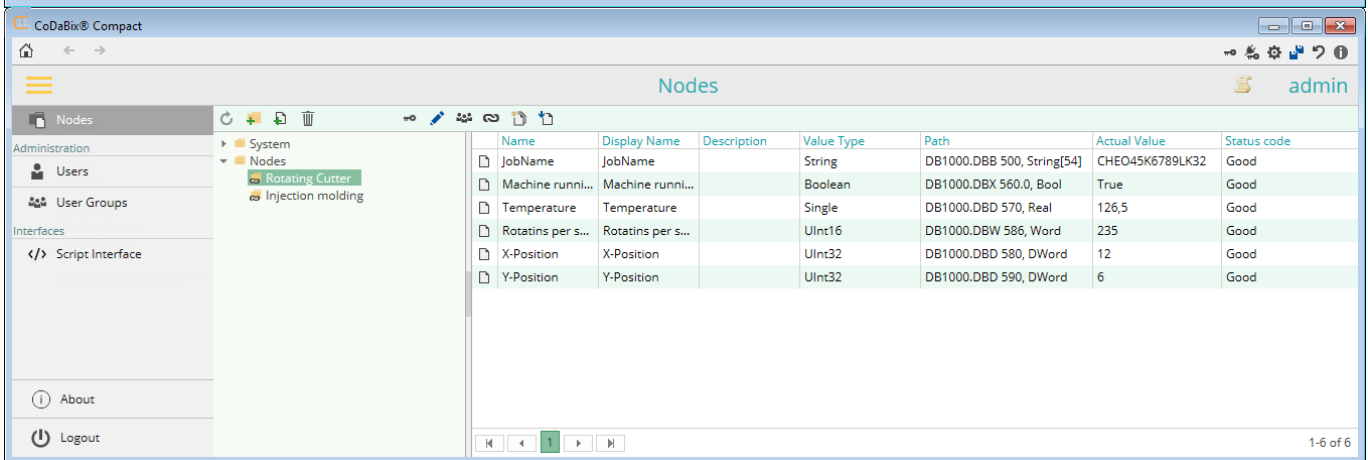
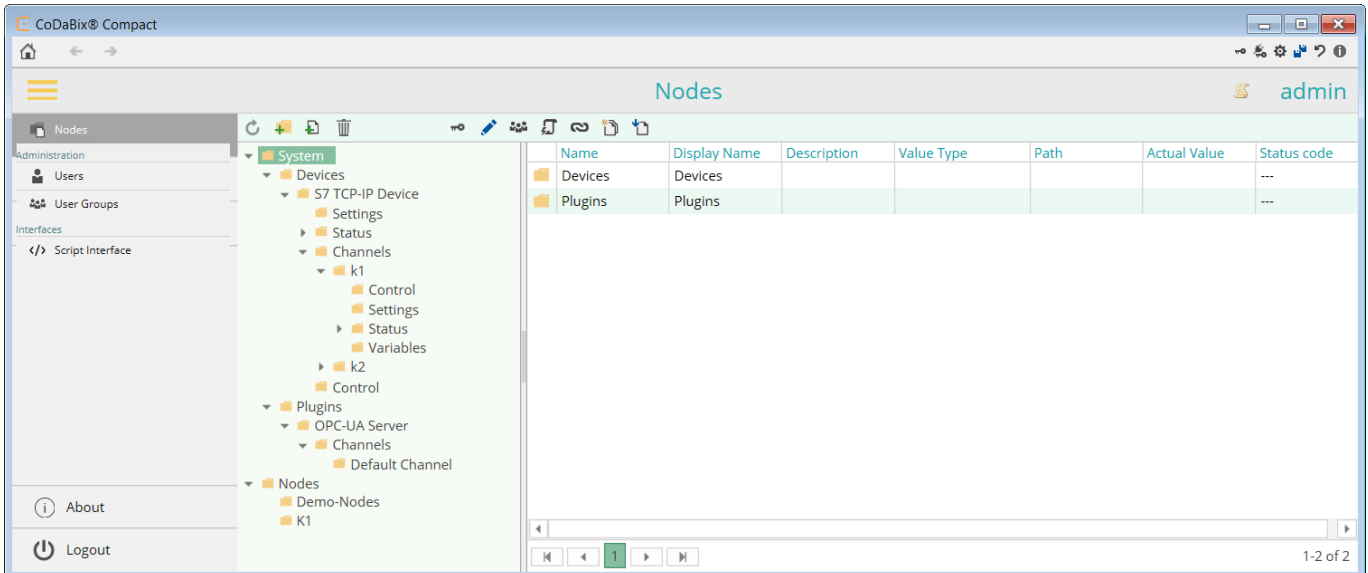
Name	Accordinging Data Type	Length in Bits
Blob	binary, optional statement of file name and MIME type possible	arbitrary
String	String	arbitrary
Null	without value	0
Boolean	Bool	1
SByte	signed Byte	8
Byte	unsigend Byte	8
Int16	signed Integer	16
UInt16	unsigned Integer	16
Int32	signed Integer	32
UInt32	unsinged Integer	32
Int64	signed Integer	64
UInt64	unsigned Integer	64
Single	single Floating Point	32
Double	double Floating Point	64
File	The Node represents a physical file that can be accessed via OPC UA Client or via HTTP Access URL. The "Path" property of the Node contains a file path. <b>Note:</b> The specified path is subject to the <b>Access Security</b> restrictions that have been defined in the UKI-4.0 <a href="#">Project Settings</a> .	

An array can also be created from each data type. Exceptions are the data types:

- Null
- File

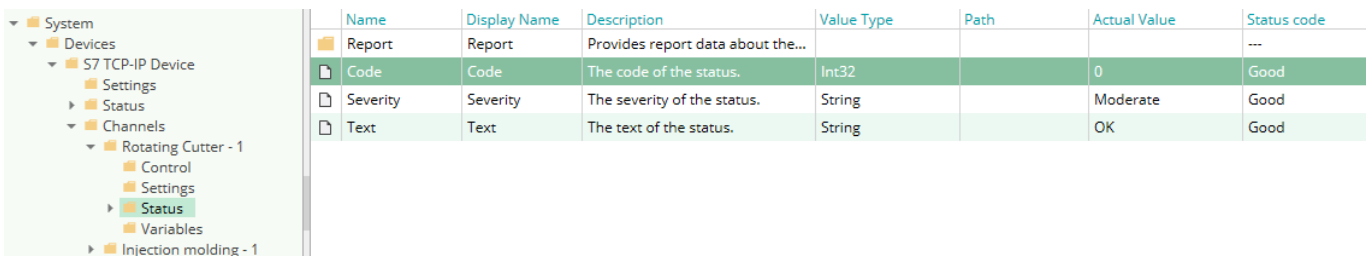
The length of the array is decided during the writing of the Datapoint Node.

## Node View



Field	Description
Name	internal unique name of the Folder Node
Display Name	Display name of the Node
Description	Description of the Node
Value Type	Data type or kind of data of the Node. See <a href="#">Properties</a>
Path	e.g. PLC Address, OPC Node, data path to the node. See <a href="#">Properties</a>
Actual Value	Value after click on . Live data is not displayed automatically. See <a href="#">(4) Toolbar</a>
Status	Current status of the Node. Possible status: Good Bad

Status at the device:



At System / Devices / Device Plugin / Channels / Channel / Status you will find the status for the selected channel with the associated message.

When the code is  $\geq 0$  the channel is running ok.

# Node Structure

There are System Nodes and User Nodes.

System Nodes are fixed, predetermined structures that are necessary for the operation of UKI-4.0 and its plugins.

User Nodes are created and managed by the user according to his needs.

Structure of System Nodes:

- System
  - Devices
    - Device Plugin
      - Settings (*settings of plugin*)
      - Status (*status of plugin*)
      - Channels (*defined channels*)
        - Channel 1 (*defined channel*)
          - Control (*Datapoint Nodes to control the channel*)
          - Settings (*settings of the channel*)
          - Status (*status informations of the channel*)
          - Variables (*defined variables*)
        - Channel 2
          - Control ...
      - Control (*Datapoint Nodes to control the plugin*)
  - Plugins
    - Plugin
      - Settings
      - Status
      - Channels
        - Channel 1
          - Control
          - Settings
          - Status
          - Variables
        - Channel 2
          - Control ...
      - Control

Structure of User Nodes:

- Nodes
  - User Folder Node
    - User Datapoint Node
    - User Folder Node
    - User Datapoint Node
    - User Linked Node
  - User Folder Node...



See also [\(6\) Node Tree](#).



# Create a Node

You can create your own Nodes in the *Nodes* view under [Nodes](#).

In order to be able to create a Node, you always have to select the superior Node first. You can find the following items for adding Nodes in the toolbar:

	Add Folder or Directory Nodes
	Add Datapoint Nodes

## Add a Folder Node

Add new Folder Node
✕

Name:

Display Name:

Node Type:

Path:

Refresh Interval:

Max Value Age (ms):

History Value Interval:

✓ ✕

- Enter name e.g. Press
- Enter display name e.g. Press
- Node type: select "Folder"
- when appropriate set intervals
- create a Node with ✓

## Add a Directory Node

Add new Folder Node
✕

Name:

Display Name:

Node Type:

Path:

Refresh Interval:

Max Value Age (ms):

History Value Interval:

✓ ✕

- enter name e.g. DirectoryR\_MData
- enter display name e.g. machine data
- path: enter directory path e.g. R:\MachineData
- when appropriate set intervals
- create a Node with ✓



In order for you to be shown folders and files you have to refresh the view, e.g. by clicking "User" and then clicking "Nodes". Depending on the directory size it is possible that you will not see all files and subdirectories at once.

## Add a Datapoint Node

Add new Datapoint Node
✕

<p>Name: <input type="text" value="Datapoint"/></p> <p>Display Name: <input type="text" value="Datapoint"/></p> <p>Description: <input type="text"/></p> <p>Location: <input type="text"/></p> <p>Value Type: <span style="border: 1px solid #ccc; padding: 2px;">String</span></p> <p>Path: <span style="border: 1px solid #ccc; padding: 2px;">String</span></p> <p>Min Value: <input type="text" value="Blob"/></p> <p>Max Value: <input type="text" value="Null"/></p> <p>Hysteresis: <input type="text" value="Boolean"/></p>	<p>Scaling Factor: <input type="text"/> <span style="float: right;">✕</span></p> <p>Scaling Offset: <input type="text"/> <span style="float: right;">✕</span></p> <p>Unit: <input type="text"/></p> <p>Precision: <input type="text"/> <span style="float: right;">✕</span></p> <p>Publishing Level: <span style="border: 1px solid #ccc; padding: 2px;">A</span></p> <p>Refresh Interval: <span style="border: 1px solid #ccc; padding: 2px;">&lt;Inherit&gt;</span></p> <p>Max Value Age (ms): <input type="text"/> <span style="float: right;">✕</span></p> <p>History Value Interval: <input type="text"/></p> <p>Has History Values: <span style="border: 1px solid #ccc; padding: 2px;">No</span></p>
--	---

✓
✕

- enter name e.g. Pressure (bar)
- enter display name e.g. Pressure (bar)
- select value type e.g. Single
- when appropriate enter boundaries, scales, historical data (as e.g. Min Value = 20,5, Scaling Factor = 2,398 ...)
- create a Node with ✓

The Node could look like this:

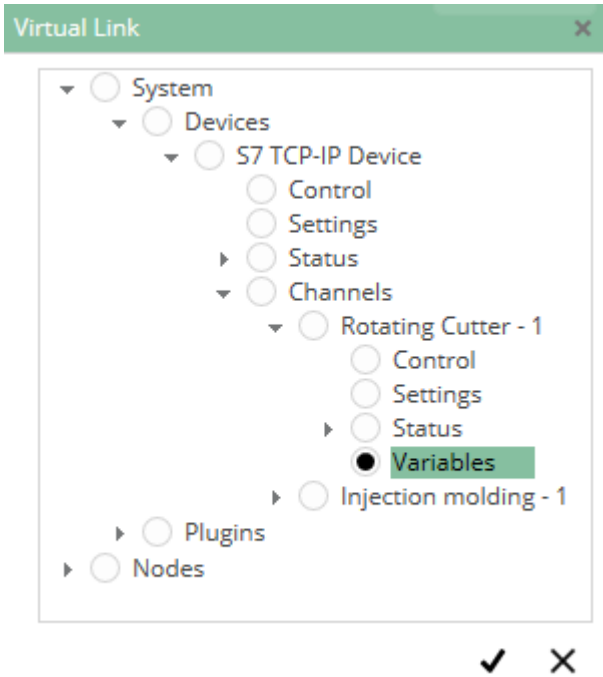
Edit Variable
✕

<p>Name: <input type="text" value="Pressure (bar)"/></p> <p>Display Name: <input type="text" value="Pressure (bar)"/></p> <p>Description: <input type="text"/></p> <p>Location: <input type="text"/></p> <p>Node Type: <span style="border: 1px solid #ccc; padding: 2px;">Single</span></p> <p>Path: <input type="text" value="DB1000.DBD 670, Real"/></p> <p>Min Value: <input type="text"/> <span style="float: right;">✕</span></p> <p>Max Value: <input type="text"/> <span style="float: right;">✕</span></p> <p>Hysteresis: <input type="text"/> <span style="float: right;">✕</span></p>	<p>Scaling Factor: <input type="text"/> <span style="float: right;">✕</span></p> <p>Scaling Offset: <input type="text"/> <span style="float: right;">✕</span></p> <p>Unit: <input type="text"/></p> <p>Precision: <input type="text"/> <span style="float: right;">✕</span></p> <p>Publishing Level: <span style="border: 1px solid #ccc; padding: 2px;">A</span></p> <p>Refresh Interval: <span style="border: 1px solid #ccc; padding: 2px;">&lt;Inherit&gt;</span></p> <p>Max Value Age (ms): <input type="text"/> <span style="float: right;">✕</span></p> <p>History Value Interval: <span style="border: 1px solid #ccc; padding: 2px;">100 ms</span></p> <p>History Options: <span style="border: 1px solid #ccc; padding: 2px;">on Interval</span></p>
--	--

✓
✕

# Link Nodes

For the selected item, the following dialog will be displayed:



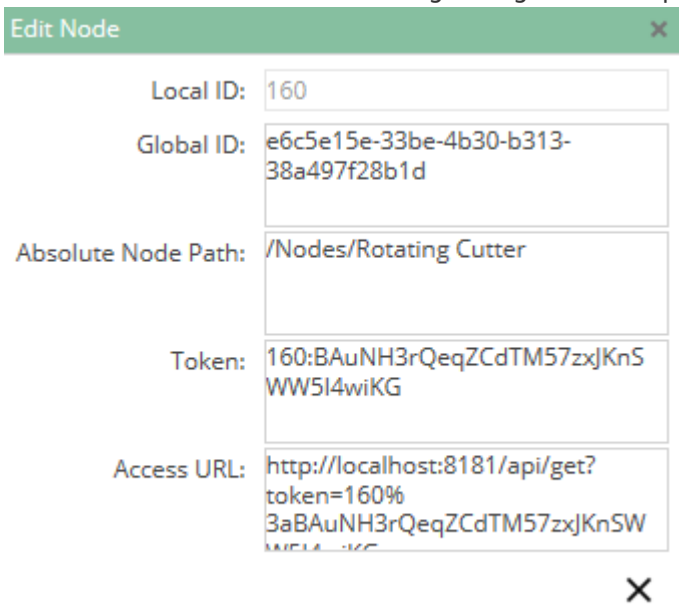
The selected item will be linked to the selected Node in the Node tree. The following icons are possible:

	linked Folder Node
	linked Datapoint Node

If a Folder Node is linked onto a Datapoint Node, it is automatically converted to a Folder node.

# Node Access

For the selected item the following dialog will be displayed:



Field	Description
Local ID	Unique UKI-4.0 internal ID of the selected item. See <a href="#">Abbreviations / Glossary</a>
Global ID	The GUID unambiguously identifies the Node beyond system, e.g. Use of UKI-4.0 at several locations that are to form a record together. <a href="#">Abbreviations / Glossary</a>

Field	Description
Absolute Node Path	Path for accessing the Node. In linked datapoints the path display to the origin of the datapoint is displayed.
Token	Act like a password for JSON to gain access to the Node.
Access URL	Read the actual value via HTTP directly and display it in a browser window.

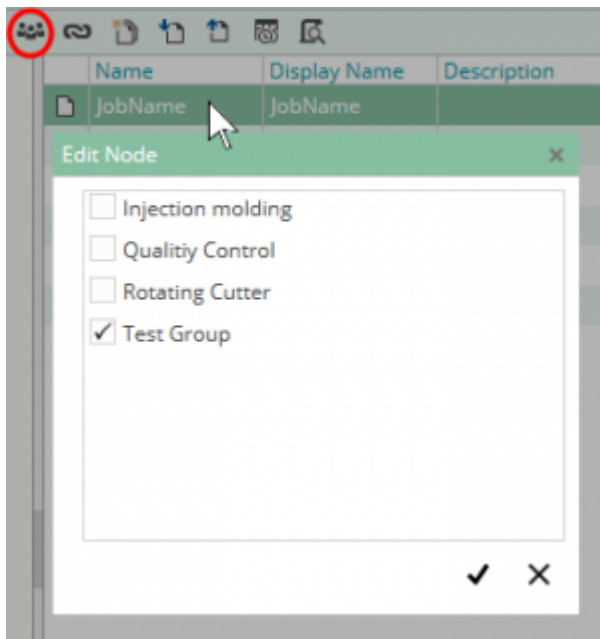
## Add / Delete Node Access to a User

**A node access can only be added or removed to / from a user group.**

There are two ways to add a Node to the user via the user group:

- via the Node view
- via the user group directly

### Add / Remove via the Node View

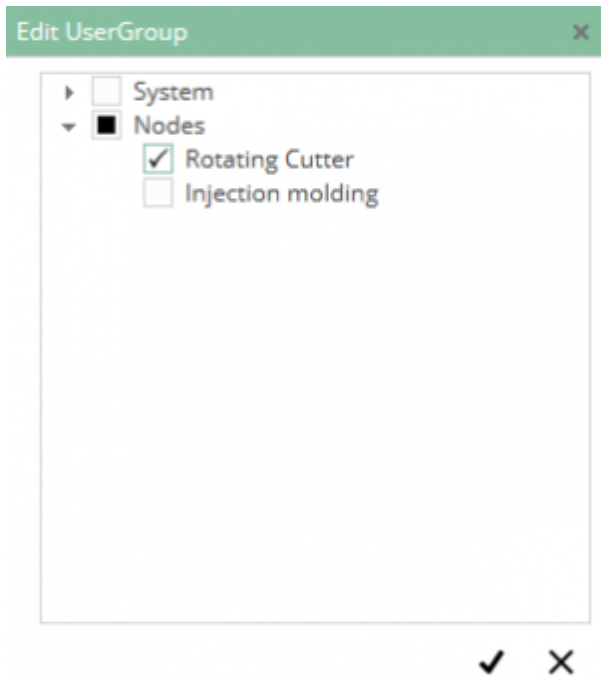


If a Folder Node is selected, automatically all the Nodes lying underneath inherit the selected user group.

The user group is not displayed in the underlying nodes, if you click .

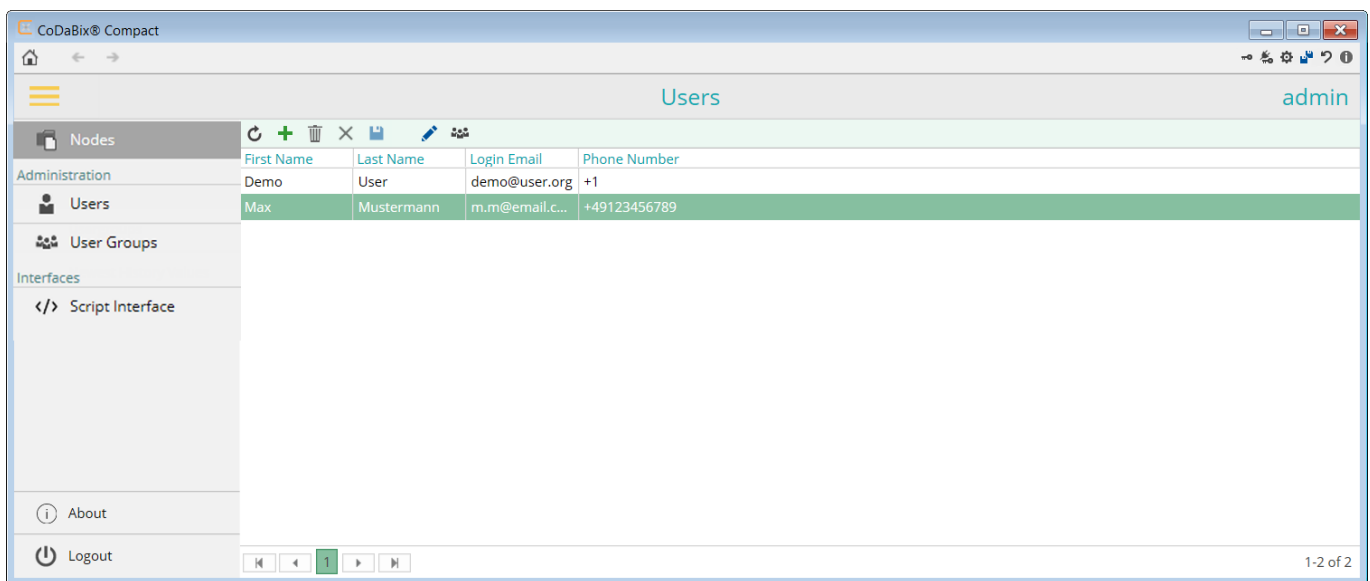
Thus, you have the option to explicitly set the user group to a specific Folder or Datapoint Node, if for example, the Parent Node loses access.

### Add / Remove via the User Group



All previously selected Folder and Datapoint Nodes are displayed by hooks. When you select a Folder Node, all nodes below will be automatically selected or deselected.

# User



Here you create the users who shall have access to the dataa provided by UKI-4.0 .

Only the administrator has the right to access the configuration page.

Functions:

- add / edit / delete users
- assign user group(s)

## Add Users

Add new User
✕

First Name:

Last Name:

Login Email:

Phone Number:

Login Password:

✓ ✕

Field	Description
First Name	First name
Last Name	Last name
Login Email	Email address: Can occur only once
Phone Number	Phone number: Can occur only once
Login Password	Password with repetition

### Edit Users

Edit User
✕

First Name:

Last Name:

Login Email:

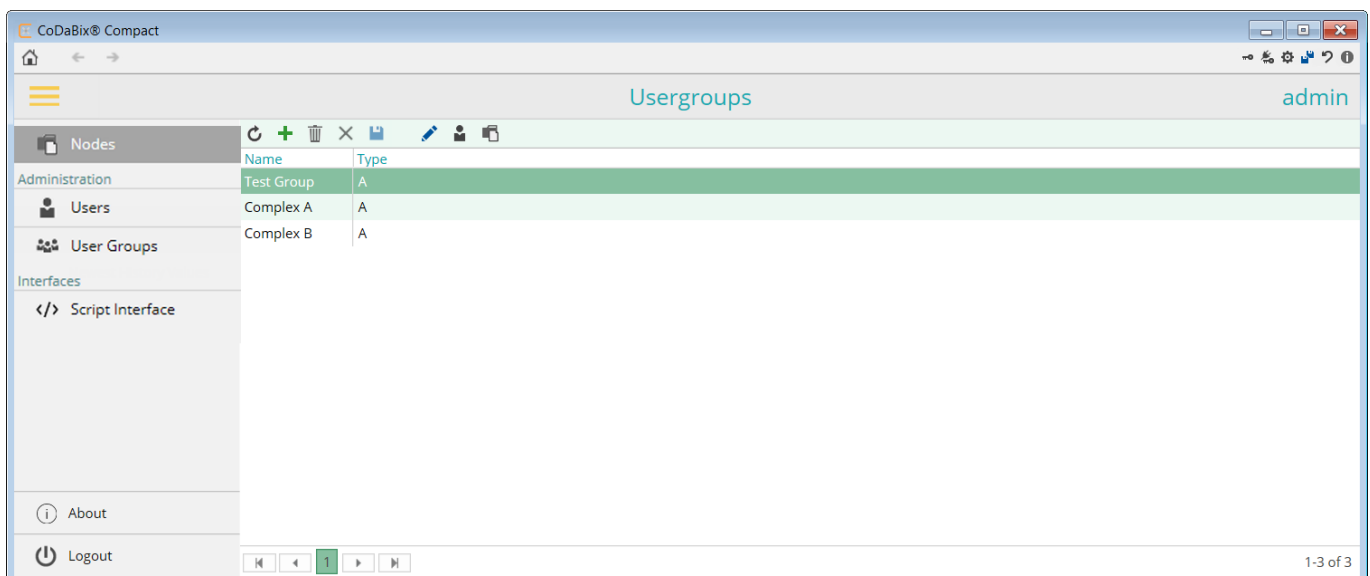
Phone Number:

Login Password:

✓ ✕

The same settings as in [Add Users](#) apply.

## User Group



A Node cannot be directly assigned to a user. You always need a user group. This simplifies the

administration of the Node access.

Functions:

- Add / edit/ delete user group
- Add user to user group
- Add / delete Node

## Add User Group

Field	Description
Name	Display name of the group
Type	A or B, currently not used

At the moment the usertype is not used.

## Edit User Group

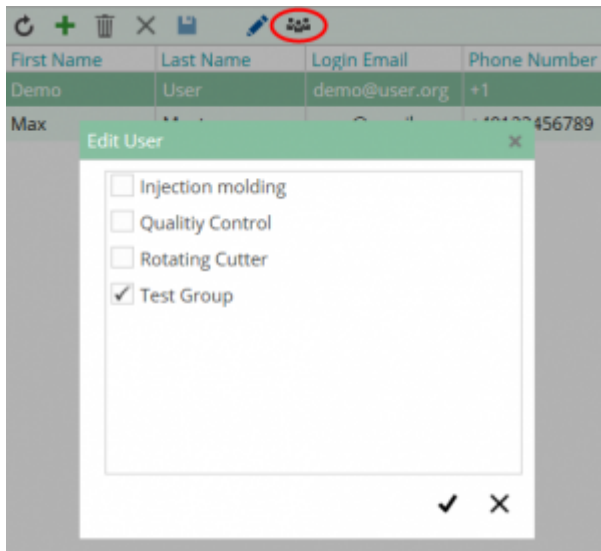
The same settings as in [Add User Group](#) apply.

## Add User to User Group

There are two ways to add a user to user group:

### Via the user menu:

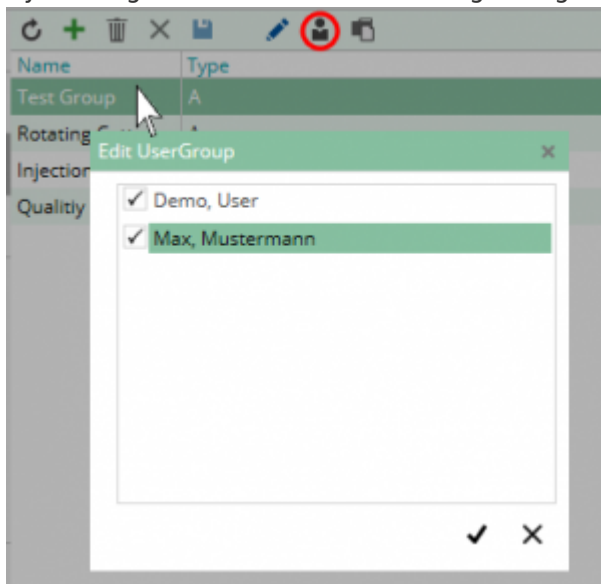
For the selected item the following dialog will be displayed:



Here you can add or deselect the user groups for the selected item.

**Via the user group:**

By clicking the user icon the following dialog is displayed for the selected user group:



All available users can be selected or deselected here.

# Plugins

UKI-4.0 can be extended to add new functions by plugins. These plugins are grouped by their classification. Depending on the category, the plugin offers a special range of services, organizations and Nodes. Some plugins can provide configuration files and additionally, as required by the circumstances, include a configuration application.

## Device Plugins

All device plugins are using the UKI-4.0 Device Model. Each device that is provided is registered and managed by the UKI-4.0 Device Manager. See [Device Plugins](#).

### S7Device

Connection to Siemens SIMATIC S7. See [S7 Device Plugin](#).

## Exchange Plugins

All Exchange Plugins are using the UKI-4.0 Storage Model. Each Exchange Plugin that is provided is



registered and managed by the UKI-4.0 Device Manager. See [Exchange Plugins](#).

### **Database**

UKI-4.0 supports connections to databases. More information can be found at [Database Plugin](#).

### **CSV**

For further information about the handling with CSV files see [CSV Exchange Plugin](#).

## **Interface Plugins**

UKI-4.0 contains an API interface. For more informations see [Interface Plugins](#).

### **REST**

The REST interface allows access to the Node UKI-4.0 via HTTP request formatted as a JSON object. For more informations see [REST Interface Plugin](#)

### **Script Plugins**

Script plugins extend UKI-4.0 with additional functionalities via JavaScript editor. They are executed in a secure environment.

See [Script Interface Plugin](#).

### **OPC UA Server Interface**

UKI-4.0 contains an OPC UA Server for the exchange of data. More information under [OPC UA Server Interface Plugin](#).

# Plugins

Different Plugins are available for UKI-4.0 ® which are grouped by their classification. Depending on its classification the plugin provides a specialized set of services, entities and Nodes. Some plugins can provide configuration files and additionally, as the circumstances require, a configuration application.

## Activation

During the startup progress of UKI-4.0 ® all plugins located in the plugins directory (<UKI-4.0 PluginsDir>) are recursively run through, loaded, instantiated and started in alphabetical order. Each plugin which is not in the plugin directory or in any sub directory is neither loaded nor started by the UKI-4.0 ® Plugin Manager.

While the UKI-4.0 ® Engine is started, it is possible to restart a plugin. This is done by sending a request to the UKI-4.0 ® Plugin Manager using the plugin's proxy instance (note that this function makes modifications without preannouncement).

## State Machine

During the startup progress and during the runtime of UKI-4.0 ® the plugin can run through different states. Each state does restrict the possible transition from one state to the other while some of them occur consecutively. The plugin instance itself stays in the state Created after the UKI-4.0 ® Plugin Manager has loaded and instantiated the plugin.

After the Plugin Manager has loaded and instantiated all plugins the start sequence calls the startup sequence of the plugins. During the startup the state is changed to Starting. After a successful start of the plugin the state is changed to Started. If the plugin cannot be started, the state is set back to the value it held before Starting. Only plugins with the state Started can be stopped.

A plugin is stopped during the restart sequence or during the shutdown process of the UKI-4.0 ® Plugin Manager. The plugin then processes its stop sequence (only if its state is Started). During this sequence the plugin state changes to Stopping. If the shutdown of the plugin fails, the state is set to Stopped. If the plugin cannot be stopped, it goes back to the state it had before being changed to Stopping.

## Diagnostics

The provided diagnostic information of a plugin depends on the classification of the plugin and whether the plugin provides the specific diagnostic information itself. The [S7 Device Plugin](#) e.g. is a [Device Plugin](#) and does provide status information and other diagnostic information through its UKI-4.0 ® Device, UKI-4.0 ® Device Channel and S7 Device Variables.

## Location

All plugin specific files are stored in a plugin directory in the `plugins` directory, which is located in the UKI-4.0 ® installation directory (selected during the installation). The following list illustrates the hierarchy:

- <UKI-4.0 InstallDir>
  - plugins\
    - <PluginName>

- `<PluginAssembly>`
- ...

This can look like as follows:

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

## Classification

### Device Plugins

All plugins classified as Device Plugin integrate physical or logical devices in UKI-4.0 ®. A device itself can be any kind of resource made accessible for UKI-4.0 ® by the channel model defined by the UKI-4.0 ® [Device Model](#).

In general a device plugin defines a specified type of UKI-4.0 ® Device using the UKI-4.0 ® Device Model (for more information see [Device Plugins](#)) and therefore provides the necessary accessibility layer for such a device.

### Exchange Plugins

All plugins classified as Exchange Plugin link storage engines with UKI-4.0 ®. A storage engine itself can be any kind of database management system (DBMS) of which the instances are accessible by the relational model using the UKI-4.0 [Storage Model](#). The storage engine itself has to provide at least one primitive relational model like e.g. a table (= entity in DBMS).

In general an Exchange Plugin defines a specified type of UKI-4.0 ® Storage using the UKI-4.0 [Storage Model](#) (for more information see [Exchange Plugins](#)) and therefore provides the necessary accessibility layer for such a storage engine.

### Interface Plugins

All plugins classified as Interface Plugin link UKI-4.0 ® with other platforms or technologies. The interface itself can be any kind of language, service, protocol, etc. The API model only has to depict the special environment onto the API defined by UKI-4.0 ®.

In general an Interface Plugin defines the a specialized type of platform or technology API (for more information see [Interface Plugins](#)) and therefore provides the necessary accessibility layer for such a platform or technology to the UKI-4.0 ® API.

## Configuration

### Using an Application

#### Location:

In case the plugin provides an application for configuration it can be located in the `<UKI-4.0 InstallDir>` directory). The following list illustrates the hierarchy:

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

- ...

For example:

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

## Using a Configuration File Location

In case the plugin uses a configuration file it can be located in the plugin directory within the UKI-4.0 project directory (configured in UKI-4.0 ). The following list illustrates the hierarchy:

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

For example:

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

## Structure

An XML based plugin configuration file defines at least the **PluginSettings** element, while the further defined elements depend on the classification of the plugin and their own custom elements and attributes.

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

## Entity Attributes

Some elements in the configuration file create or represent an entity in UKI-4.0 ®. For such elements some attributes are eserved in order to service those entities easier.

### The Entity **Identifier** Attribute

This attribute is of the type GUID. This ID is produced while producing the entity of UKI-4.0 ®. Through this identifier (= in UKI-4.0 ® the Global Node Identifier) it is possible to uniquely identify the entity and assign it uniquely to the configuration element.

In case a configuration element is missing or the element / attribute to be identified cannot be found, it is assumed that a new entity shall be created. If a new configuration element shall be created, it is not necessary to define the **Identifier** attribute. If an existing element shall be modified ,the **Identifier** attribute must be defined.

### The Entity **ChangeType** Attribute

The following values are valid for attributes of that type:

Value	Description
"None"	The entity configuration element hasn't been changed recursively.
"Created"	The entity configuration element is new and requires a new entity to be created.

Value	Description
"Updated"	The entity configuration element was changed and requires the entity depicted to be updated.
"Deleted"	The entity configuration element is obsolete and requires the entity represented to be deleted.

In case there the attribute has the value:

- **"None"**: Nothing special is to be done and the value itself is ignored. Then the plugin evaluates the attribute.
- **"Created"**: The **Identifier** attribute is used (if set) to find the entity. If the entity is not found, a new entity is created. The Global Node Identifier of the entity is added to the **Identifier** attribute. Then the plugin evaluates the attribute.
- **"Updated"**: The **Identifier** attribute is used to find the entity. If the entity is not found, a new entity is created. The transferred value is updated. Then the plugin evaluates the attribute.
- **"Deleted"**: The **Identifier** attribute is used to find the entity. If the entity exists it is deleted. Then the plugin checks if the attribute was removed.

## Synchronization

As soon as a plugin is loaded and started by the UKI-4.0 ® Plugin Manager its configuration file (if available) is read and synced by the plugin with the appropriate UKI-4.0 ® entities.

In case the configuration file changes the plugin manager notifies the according plugin. The plugin decides, if a restart or a synchronisation of the according configuration entities happens.

If at least one of configuration entity changes, the plugin has to handle the entity change event and synchronize the entity / entities with the configuration file.

# Device Plugins

All device plugins use the UKI-4.0® Device Model. Each device provided is registered and managed by the UKI-4.0® Device Manager.

Such devices can be:

- Physical devices like:
  - SIMATIC S7 PLCs (see [S7 Device Plugin](#))
  - Allen Bradley PLCs
  - Mitsubishi PLCs
  - Raspberry Pi's accessible via TCP/IP
  - Arduino / Netduino projects accessible e.g. via USB
- Logical devices like:
  - Files in the local or remote file system, e.g. a connection to a remote drive - which could be a file in a cloud drive or a file accessible via (S)FTP - needs to be established and maintained as long as the data is accessing the file and is administrated by the system.
  - Services like local services accessible via Shared Memory or Remote Services, accessible via RPC, REST, DCOM or any other kinds of protocols such as SOAP. Such a local / remote service can be also a OPC Server (see OPC UA Client Device Plugin).

## Device Model

The UKI-4.0® Device Model extends the basic UKI-4.0® Entity Model with entities typical for devices. Hereby a device entity defines the subordinated entities for control, settings, the status of the plugin and the various channels via which the plugin connects the supported devices to UKI-4.0®.

## Configuration

Each device plugin delivered with UKI-4.0® can be configured in the UKI-4.0® Host application. If a plugin has configuration parameters, those can be modified in the according device entity.

## Using UKI-4.0UKI-4.0®

The entire configuration of all device plugins can be found under the Node path `/System/Devices`. This root Node of the device plugins allows the complete configuration of the device plugins, provided that one of the actively used device plugins supplies its own device entities for the configuration.

## Using an Application

### Location

In case a plugin provides an application for configuration it can be located in the UKI-4.0® installation directory (selected during installation). The following list illustrates the hierarchy:

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

For example:

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

## Using a Configuration File

### Location

In case a plugin provides a configuration file it can be located in a plugin specific directory in the “plugins” directory in the UKI-4.0® data directory (configured in UKI-4.0®). The following list illustrates the hierarchy:

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

For example:

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

### Structure

Each UKI-4.0® plugin defines the root of its element tree by the `PluginSettings` element as described in [Plugin Configuration - Using a File](#). A device plugin expands its XML tree using the `Device` element, while the further defined child elements depend on the implementation of the plugin and its own custom elements and attributes.

#### Device Element

The `Device` element serves as the container for the `Channels` element. This element configures the whole type of device represented by the device plugin, while the further child elements depend on the implementation of the plugin and its own custom elements and attributes.

The `Device` element can look like the following:

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

#### Channels Element

The `Channels` element serves as the container for one or more `Channel` elements. This element maintains all channels associated with the type and provided by the device plugin, while the further child elements depend on the implementation of the plugin and its own custom elements and attributes.

The `Channels` element can look like the following:

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

#### Channel Element

The `Channel` element serves as the container for the `Settings` element, while the further child elements

depend on the implementation of the plugin and its own custom elements and attributes.

Each **Channel** element provides the following list of attributes:

	<b>Mandatory</b>	<b>Type</b>	<b>Purpose</b>
<b>Identifier</b>	no	GUID	The generic unique identifier of the channel.
<b>Name</b>	yes	String	The unique name (within the <b>Channels</b> element) of the channel.

The **Channel** element can look like the following:

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

### Settings Element

The **Settings** element defines the attributes to set up the channel. This attribute configures the channel connection parameters, while the further child elements depend on the implementation of the plugin and its own custom elements and attributes.

The **Settings** element can look like the following:

```
<Settings />
```

## Example Configuration File

### 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

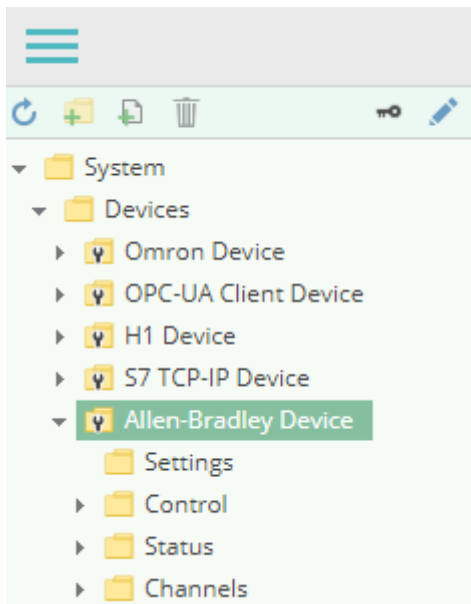
The Allen-Bradley Device Plugin allows reading and writing values from Allen-Bradley PLC devices over EtherNet/IP.

The following device types are supported:

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

## Configuration

The whole Allen-Bradley Device Plugin configuration is located in the node path `/System/Devices/Allen-Bradley Device`.



## Channel

An Allen-Bradley Device Channel represents the connection to a Allen-Bradley PLC.

### Settings

#### Address

IP address of the Allen-Bradley PLC.

#### Device Type

The device type of the Allen-Bradley PLC. The following device types are supported:

- ControlLogix/CompactLogix
- Micro800

- MicroLogix
- PLC-5
- SLC 500

## CIP Path

The CIP path to the PLC. This field must be specified for ControlLogix/CompactLogix as well as for a DH+ protocol bridge (i.e. a DHRIO module).

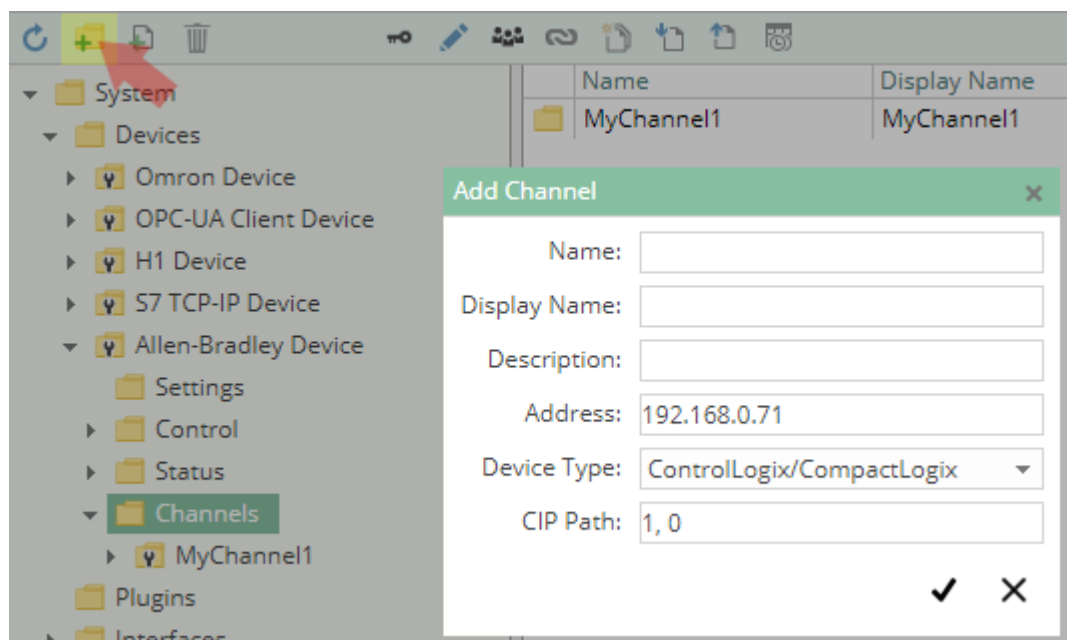
This field can have the format **A,B** where **A** specifies the port type (1=Backplane) and **B** specifies the slot in which the CPU is plugged.

Example: **1,0**

## Operation Timeout

The timeout which shall be applied for read and write operations.

## Adding a Channel



To add a new Allen-Bradley Channel, please follow these steps:

1. Add a Folder Node within the node **Allen-Bradley Device/Channels**, or right-click on the **Allen-Bradley Device/Channels** node and select **Add Channel**.
2. In the **Add Channel** dialog, specify the settings for the Allen-Bradley connection.
3. After clicking on “Save”, the channel node is created.
4. You can start the channel by selecting the channel node and clicking the start button.

## Variables

Within the node **Variables** you can create datapoint nodes which can be read and written from/to the Allen-Bradley PLC as **tags**.

The **Value Type** property must be set to the corresponding tag type. Currently the following tag types are supported:

Tag Type	UKI-4.0 Type
SINT	SByte or SByte-Array
USINT	Byte or Byte-Array
INT	Int16 or Int16-Array
UINT	UInt16 or UInt16-Array
DINT	Int32 or Int32-Array
UDINT	UInt32 or UInt32-Array
LINT	Int64 or Int64-Array
ULINT	UInt64 or UInt64-Array
REAL	Single or Single-Array
STRING (or user-defined String data type)	String or String-Array

For **String** data types, an ASCII/ISO-8859-1-like encoding is assumed (the high byte of the 16-bit char will be cut off). While this can mean that surrogate pairs (for characters outside of the Unicode BMP) are not correctly handled, it will mean that the number of bytes is the same as the string length.

Note: Each signed/unsigned pendant of a datatype can also be used in place of the datatype. For example, **SINT** can also be used as **Byte** instead of **SByte**, but the MSB (most significant bit) will have a different interpretation, as is interpreted as sign bit for signed data types.

## Path

The **Path** property of the node is used to specify the tag name, optionally a type specification, and (for arrays) optionally the offset and the length of the data:

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

Placeholder	Description
<TagName>	The full name of the tag. If the tag is a program tag, it needs to be prefixed with <b>Program:&lt;Programmname&gt;</b> . using the corresponding program name. If the tag is a structure, you can specify the name of the field to access after a dot (.). If the tag (or a structure field) is an array, you can specify an array offset using the syntax <b>[Offset]</b> .
<Length>	If the tag is an array, you can specify the array length here. It will only be used for reading, because when writing an array, its length will be determined from the value that is written.
<Type>	If present, contains a type specification, if it cannot be derived from the UKI-4.0 type. Currently, the following types can be specified: • <b>STRING:&lt;MaxLen&gt;</b> : Specifies the maximum string characters if the variable is of type <b>String</b> or <b>String-Array</b> . This can be used for user-defined String types; otherwise the predefined <b>STRING</b> data type with a maximum of 82 characters will be used.

## Examples

Value Type	Path	Meaning
Int16	Local:1:0.Data	Data field (INT) of the digital output (Int16 contains bitmask of the 16 digital outputs (Digital Output Points))
Int16	Local:1:I.Data	Data field (INT) of the digital input (Int16 contains bitmask of the 16 digital inputs (Digital Input Points))

Value Type	Path	Meaning
Int32	MyControllerTag	Controller Tag <b>MyControllerTag</b> (DINT)
Int32	Program:MainProgram.MyTag1.MyField1	From program <b>MainProgram</b> the tag <b>MyTag1</b> (Structure) using field <b>MyField1</b> (DINT)
Byte-Array	Program:MainProgram.MyTag2, 20	From program <b>MainProgram</b> the tag <b>MyTag2</b> (SINT[20]) with length 20 (range 0..20)
Byte-Array	Program:MainProgram.MyTag2[2], 16	From program <b>MainProgram</b> the tag <b>MyTag2</b> (SINT[20]) from index 2 with length 16 (range 2..18)
Byte-Array	Program:MainProgram.MyTag3[4].MyField1[1], 5	From program <b>MainProgram</b> the tag <b>MyTag3</b> (Structure[5]) at index 4 the field <b>MyField1</b> (SInt[10]) from index 1 with length 5 (range 1..6)
String	MyStringTag1	Controller Tag <b>MyStringTag1</b> (STRING with max. 82 characters)
String	MyStringTag2, STRING:20	Controller Tag <b>MyStringTag2</b> (user-defined String type with max. 20 characters)
String-Array	MyStruct.MyStringTag3[2], STRING:20[3]	Controller Tag <b>MyStruct</b> (Structure) using field <b>MyStringTag3</b> (user-defined String type with max. 20 characters) from index 2 with length 3 (range 2..5)

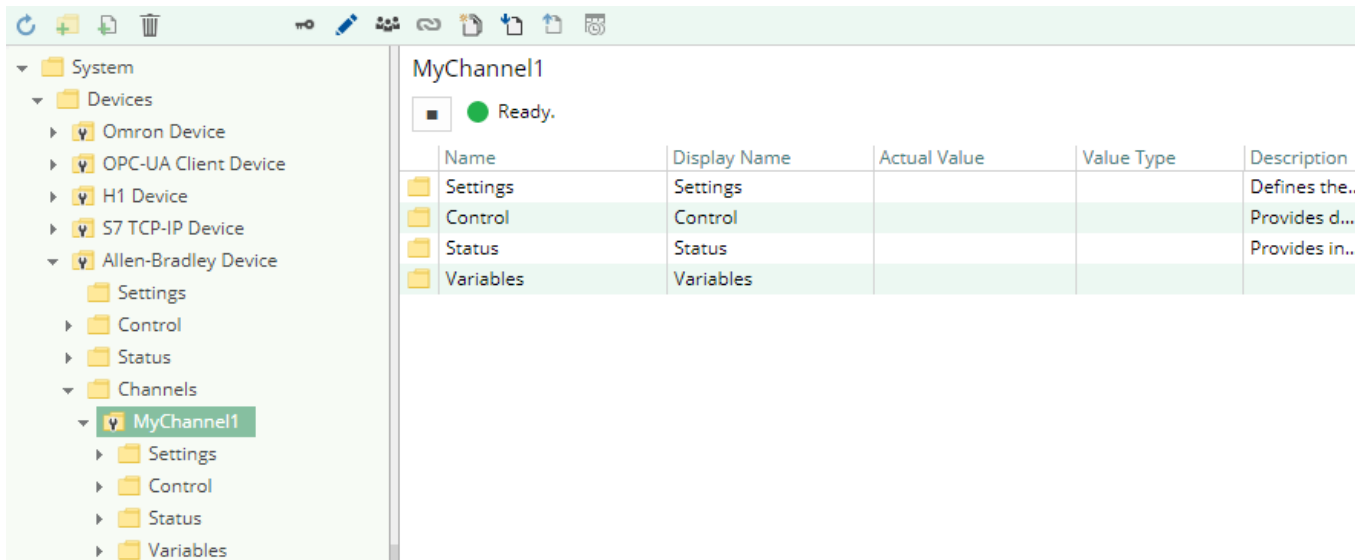
	Name	Display Name	Actual Value	Value Type	Description	Path	Status
📁	Output	Output	234	Int16		Local:1:O.Data	Good
📁	Input	Input	0	Int16		Local:1:I.Data	Good
📁	InputFault	InputFault	0	Int32		Local:1:I.Fault	Good
📁	MyRealControllerTag	MyRealControllerTag	0	Single		MyRealControllerTag	Good
📁	Tag1	Tag1	20	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

# Diagnostics

The Allen-Bradley Device Plugin provides different status information depending on the layer to inspect. In general the channel-based diagnostic information is produced by the connection status of the channel to the PLC. The variable-based diagnostic information is produced during the read/write access of the different variables.

## Channel

To monitor and diagnose the status of a Allen-Bradley Channel, take a look at the following image:



The image above depicts the Allen-Bradley Channel's Control Panel which displays all status relevant information. The control panel will automatically update its status information when a new status is available.

### Status Circle

Color	Meaning
	The channel is stopped. Click the  button to start it.
	The channel is currently in the progress of starting or stopping or is waiting to establish a connection.
	The channel is ready for doing read/write operations. You can stop it by clicking the  button.
	The channel is running, but the connection is currently in an error state. Please check the status text for more information.

## Variables

To monitor and diagnose the status of the different variables, take a look at the variable's **Status** property displayed in UKI-4.0 . Use the button “Read actual Value” to read the values from the PLC and store the result into the variables.

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

Common error messages:

- **PLCTAG\_ERR\_NOT\_FOUND:** The tag was not found in the PLC. Check that the name is spelled correctly.
- **PLCTAG\_ERR\_BAD\_PARAM:** The type or syntax of the tag is not valid. Check the tag name and if the correct value type is used.
- **PLCTAG\_ERR\_TOO\_LARGE:** The read value cannot be stored in a variable of this type, because more data was returned than what fits into the type. Chose a value type which corresponds to the PLC type. If the tag is a structure, please specify the name of the field you want to access.
- **PLCTAG\_ERR\_OUT\_OF\_BOUNDS:** It was tried to access array indexes that are out of the array bounds. Check the specified length and the offset of the array.
- **The operation has timed out.:** The PLC access has exceeded a specific timeout, or the PLC isn't available.

# Log file

All channel related status information is also logged into the channel-specific log file stored in the [\[LoggingFolder\]](#). Each log file is named in the naming scheme [Allen-Bradley Device.<ChannelName>.log](#).

The content of such a log file can look as follows:

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

# Entities

Like every device plugin the Allen-Bradley Device Plugin extends the basic UKI-4.0 [Device Model](#).

## Device

The plugin's device type [AllenBradleyDevice](#) also defines the [AllenBradleyDeviceChannel](#) and therefore extends the basic UKI-4.0 [Device](#) and UKI-4.0 [DeviceChannel](#) entities. While the [AllenBradleyDevice](#) just represents a concretization of the UKI-4.0 [Device](#), the [AllenBradleyDeviceChannel](#) extends the UKI-4.0 [DeviceChannel](#) with the Allen-Bradley Variable Entities.

## Channel

Each channel is handled by a channel worker which establishes a TCP socket connection to the PLC.

By default, the worker does not read any values. When a client or plugin requests a synchronous read of the channel's variables in UKI-4.0 (e.g. using the UKI-4.0 Web Configuration's function "Read actual value"), the channel worker reads them from the underlying PLC and then writes them into the corresponding UKI-4.0 Nodes.

Similarly, when a client or plugin writes values into the channel's variables, the channel worker will write those values to the underlying PLC.

To have an Allen-Bradley variable being read steadily, you can edit the Node in the UKI-4.0 Web Configuration and set "History Options" to [Yes](#) (which will create an internal subscription), or you can use e.g. a OPC UA Client connected to the OPC UA Server plugin and create a subscription for the Allen-Bradley variable Nodes. In these cases, the channel worker reads the variables from the PLC at a regular interval and, if the value of one of the variables has changed, writes the new value into the corresponding UKI-4.0 Node.

# Folders & Files

## Folders

Content	Path	Usage
AssemblyFolder	<UKI-4.0 InstallDir>/plugins/AllenBradleyDevicePlugin/	Contains the plugin's assembly file.
ConfigFolder	<UKI-4.0 ProjectDir>/plugins/AllenBradleyDevicePlugin/	Contains the plugin's configuration file.
LoggingFolder	<UKI-4.0 ProjectDir>/log/	Contains the plugin's log files.

## Files

Type	Path	Usage
Assembly	[AssemblyFolder]/UKI-4.0 .AllenBradleyDevicePlugin.dll	The plugin's assembly file.
Logging	[LoggingFolder]/Allen-Bradley Device.<ChannelName>.log	The log file.

# About Versions

## This Document

<b>Date</b>	2019-11-11
<b>Version</b>	1.0

## Plugin

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

## Assembly

<b>Name</b>	UKI-4.0 .AllenBradleyDevicePlugin.dll
<b>Date</b>	2019-11-11
<b>Version</b>	1.0.0.0

# EUROMAP Device Plugin

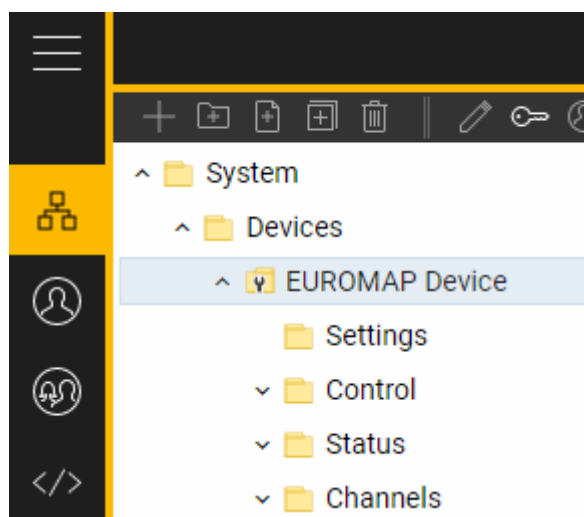
The EUROMAP Device Plugin allows reading and writing values from devices implementing the **EUROMAP 63** specification (file-based access):

- Reading values using **REPORT** commands
- Writing values using **SET** commands
- Browsing the machine using **GETID** commands

**Note:** The EUROMAP 63 specification uses file-based access, so the **session directory** of the machine must be accessible to UKI-4.0 for the plugin to work. You will need to add a permission to access this path in the **Access Security** section of the UKI-4.0 Project Settings.

## Configuration

The whole EUROMAP Device Plugin configuration is located in the node path `/System/Devices/EUROMAP Device`.



## Channel

An EUROMAP Device Channel represents the connection to an EUROMAP machine.

### Settings

#### Protocol

The protocol to use (currently, only EUROMAP 63 is supported).

#### Session Path

The path to the session directory in the local file system.

**Note:** To allow UKI-4.0 to access this path, you must add a read+write permission in the **Access Security** section of the UKI-4.0 Project Settings.

#### Min Session Number

The lowest session number to use.



## Max Session Number

The highest (exclusive) session number to use.

## Encoding

Specifies the encoding to use when reading response file from the machine, in order to support characters outside of the ASCII range.

## Line Ending

The line ending to be used. (This setting is currently ignored; CR+LF is always used.)

## List Delimiter

The list delimiter to be used. (This setting is currently ignored; “,” is always used.)

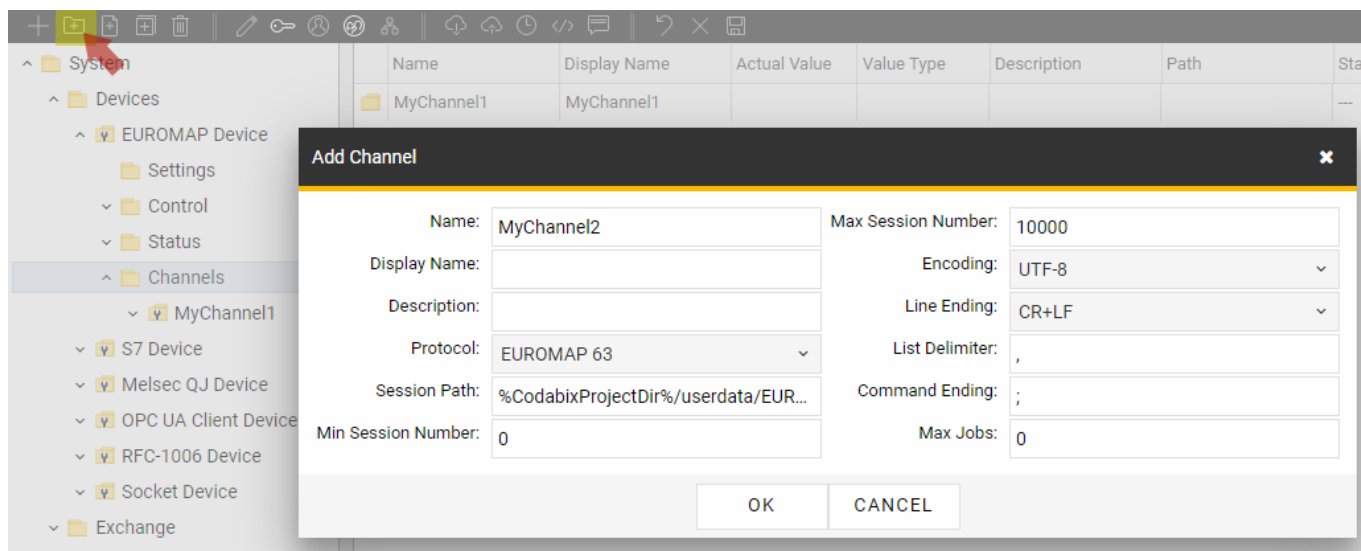
## Command Ending

The command ending to be used. (This setting is currently ignored; “;” is always used.)

## Max Jobs

The maximum number of jobs that can be active at the same time. (This setting is currently not used.)

## Adding a Channel



To add a new EUROMAP Channel, please follow these steps:

1. Add a Folder Node within the node **EUROMAP Device/Channels**, or right-click on the **EUROMAP Device/Channels** node and select **Add Channel**.
2. In the **Add Channel** dialog, specify the settings for the EUROMAP connection.
3. After clicking on “OK”, the channel node is created.
4. You can start the channel by selecting the channel node and clicking the start button.

## Parameters

Within the node **Parameters** you can create datapoint nodes which can be read and written from/to the EUROMAP machine as **parameters**.

Supported Node Value Types:

- String
- Numeric types like Int32, Double
- Boolean

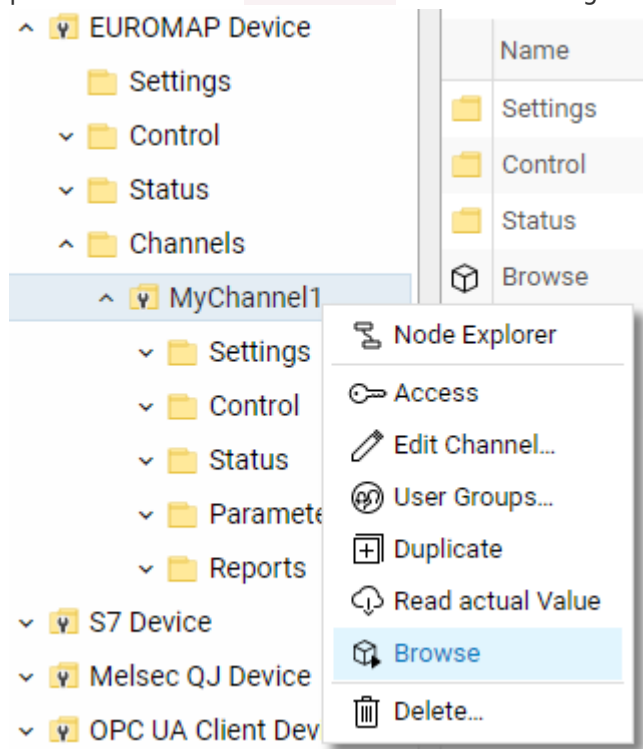
You can also create folder nodes to group datapoint nodes.

## Path

The Path property of the node is used to specify the parameter name in the machine.

## Browse

You can call the Browse method in order to browse the machine and automatically create the resulting parameters in the Parameters node. Browsing is done by executing a GETID command.



## Read/Write

When parameter nodes are read (through a synchronous read or through a subscription), the plugin executes a REPORT command that records a single sample of the parameters and then reads the resulting report file back. The report job file that is executed looks like this:

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

When parameter nodes are written, the plugin executes a SET command that sets the parameters to the specified values. The set command that is executed looks like this:

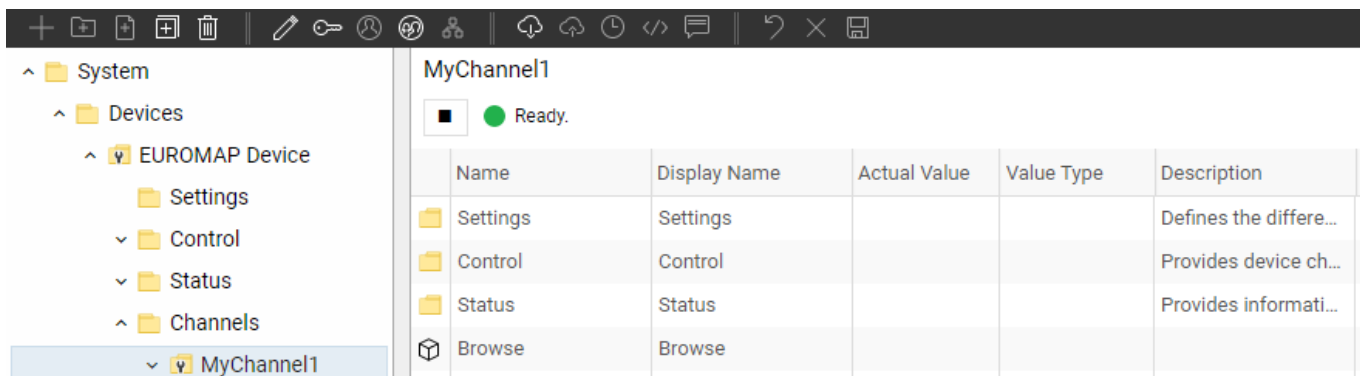
```
JOB CbxJ-LRC9LR6NF9A RESPONSE "CBXJ6RR7.RSP";
SET @P1 "Abcd";
SET @P2 123.45;
```

# Diagnostics

The EUROMAP Device Plugin provides different status information depending on the layer to inspect. In general the channel-based diagnostic information is produced by verifying whether access to the specified session directory is allowed. The parameter-based diagnostic information is produced during the read/write access of the different parameters.

## Channel

To monitor and diagnose the status of a EUROMAP Channel, take a look at the following image:



The image above depicts the EUROMAP Channel's Control Panel which displays all status relevant information. The control panel will automatically update its status information when a new status is available.

### Status Circle

Color	Meaning
Grey	The channel is stopped. Click the ► button to start it.
Yellow	The channel is currently in the progress of starting or stopping or is waiting to establish a connection.
Green	The channel is ready for doing read/write operations. You can stop it by clicking the ■ button.
Red	The channel is running, but the connection is currently in an error state. Please check the status text for more information.

## Parameters

To monitor and diagnose the status of the different parameters, take a look at the parameter's **Status** property displayed in UKI-4.0 . Use the button "Read actual Value" to read the values from the machine and store the result into the parameters.

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...

## Log file

All channel related status information is also logged into the channel-specific log file stored in the [LoggingFolder]. Each log file is named in the naming scheme **EUROMAP Device.<ChannelName>.log**.

The content of such a log file can look as follows:

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

# Entities

Like every device plugin the EUROMAP Device Plugin extends the basic UKI-4.0 [Device Model](#).

## Device

The plugin's device type [EuomapDevice](#) also defines the [EuomapDeviceChannel](#) and therefore extends the basic UKI-4.0 [Device](#) and UKI-4.0 [DeviceChannel](#) entities. While the [EuomapDevice](#) just represents a concretization of the UKI-4.0 [Device](#), the [EuomapDeviceChannel](#) extends the UKI-4.0 [DeviceChannel](#) with the EUROMAP Parameter Entities.

## Channel

Each channel is handled by a channel worker which exchanges files with the machine in the session directory, in order to start jobs and read the response files.

By default, the worker does not read any values. When a client or plugin requests a synchronous read of the channel's parameters in UKI-4.0 (e.g. using the UKI-4.0 [Web Configuration](#)'s function "Read actual value"), the channel worker reads them from the underlying machine and then writes them into the corresponding UKI-4.0 [Nodes](#).

Similarly, when a client or plugin writes values into the channel's parameters, the channel worker will write those values to the underlying machine.

To have an EUROMAP parameter being read steadily, you can edit the Node in the UKI-4.0 [Web Configuration](#) and set "History Options" to **Yes** (which will create an internal subscription), or you can use e.g. a OPC UA Client connected to the OPC UA Server plugin and create a subscription for the EUROMAP parameter nodes. In these cases, the channel worker reads the parameters from the machine at a regular interval and, if the value of one of the parameters has changed, writes the new value into the corresponding UKI-4.0 [Node](#).

# Folders & Files

## Folders

Content	Path	Usage
AssemblyFolder	<UKI-4.0 <a href="#">InstallDir</a> >/plugins/ <a href="#">EuomapDevicePlugin</a> /	Contains the plugin's assembly file.
ConfigFolder	<UKI-4.0 <a href="#">ProjectDir</a> >/plugins/ <a href="#">EuomapDevicePlugin</a> /	Contains the plugin's configuration file.
LoggingFolder	<UKI-4.0 <a href="#">ProjectDir</a> >/log/	Contains the plugin's log files.

## Files

Type	Path	Usage
Assembly	[AssemblyFolder]/UKI-4.0 .EuomapDevicePlugin.dll	The plugin's assembly file.
Logging	[LoggingFolder]/EUROMAP Device.<ChannelName>.log	The log file.

# About Versions

## This Document

<b>Date</b>	2020-12-04
<b>Version</b>	1.0

## Plugin

<b>Name</b>	EUROMAP Device Plugin
<b>Node</b>	/System/Devices/EUROMAP Device
<b>Version</b>	1.0.0

## Assembly

<b>Name</b>	UKI-4.0 .EuomapDevicePlugin.dll
<b>Date</b>	2020-12-04
<b>Version</b>	1.0.0.0

# H1 Device Plugin

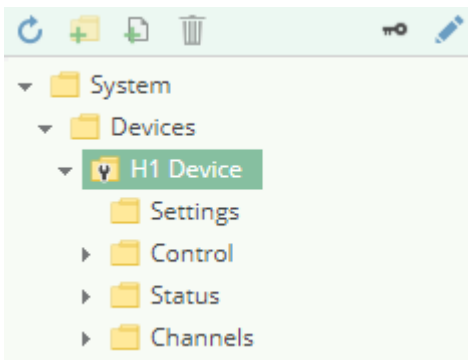
The H1 Device Plugin allows reading and writing values from SIMATIC S5 PLC devices using the SINEC H1 protocol based on ISO/MAC (Industrial Ethernet).

## Prerequisites

- This plugin only works on **Windows**.
- To use this plugin, [WinPcap 4.1.3](#) must be installed.

## Configuration

The whole H1 Device Plugin configuration is located in the node path [/System/Devices/H1 Device](#).



## Channel

An H1 Device Channel represents the connection to a H1 PLC.

## Settings

### Network Adapter

The NIC that should be used for the network communication.

### MAC Address

The MAC address of the remote device.

### Read SSAP/DSAP

The SSAP (Source SAP) and DSAP (Destination SAP) values that should be used when reading data. A maximum of 8 characters will be used.

### Write SSAP/DSAP

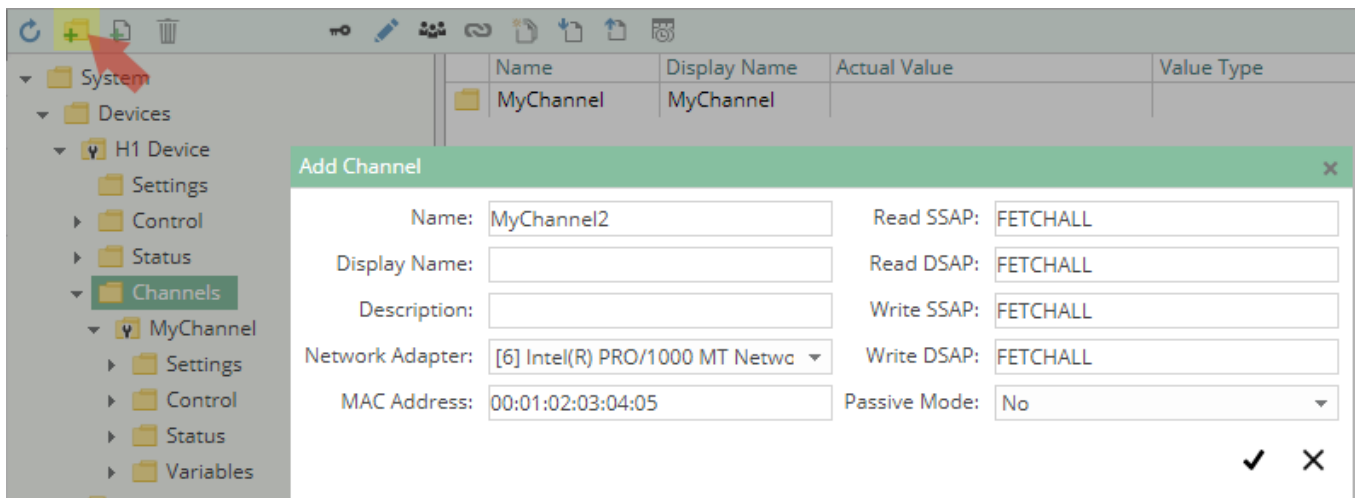
The SSAP (Source SAP) and DSAP (Destination SAP) values that should be used when writing data. A maximum of 8 characters will be used.

### Passive Mode

Specifies which party initiates the connection:

- **No:** The local machine initiates the connection.
- **Yes:** The remote device initiates the connection.

## Adding a Channel



To add a new H1 Channel, please follow these steps:

1. Add a Folder Node within the node **H1 Device/Channels**, or right-click on the **H1 Device/Channels** node and select **Add Channel**.
2. In the **Add Channel** dialog, specify the settings for the H1 connection.
3. After clicking on “Save”, the channel node is created.
4. You can start the channel by selecting the channel node and clicking the start button.

## Variables

Within the node **Variables** you can create datapoint nodes which can be read and written from/to the PLC. The **Value Type** property must be set to the corresponding operand type. Currently the following types are supported:

UKI-4.0 Type	Operand
Boolean or Boolean-Array	X
Byte/SByte or Byte-Array/SByte-Array	L (left=high byte) or R (right=low byte)
Int16/UInt16 or Int16-Array/UInt16-Array	W
Int32/UInt32 or Int32-Array/UInt32-Array	D
Float or Float-Array	D
TimeSpan or TimeSpan-Array	W

**Note:** 32-bit values (**Float**, **Int32**) will be read/written using two separate (16-bit) words. This can mean that if the value changes in the PLC while reading it, you might get an inconsistent value when one word has the new value but the other one has the old value.

**Note:** Bit values (**Boolean**) will be read/written as (16-bit) word. When writing such a boolean value, the word will be read from the PLC, then the bit(s) will be changed, and then the word will be written back to the PLC. This can mean that if at the same time some other party changes another bit in that word, that change might get lost.

## Path

The **Path** property of the node is used to specify the address and optionally the type and/or (for arrays) the

length of the data. Currently, only **Data Block (DB)** is supported as data area.

```
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>
```

Placeholder	Description
<DB number>	The number of the data block.
<Operand>	One of the operands as specified in the above table.
<Offset>	The <b>word (16-bit) offset</b> from where the data should be accessed.
<Bit>	When using a <b>Boolean/Boolean-Array</b> value you can specify the bit number in the word from 0 to 15.
<Length>	The length of the array.
<Type>	When using integer types, you can optionally specify a <b>BCD</b> type (Binary Coded Decimal) to store the values using BCD encoding. You can specify one of the following values: <ul style="list-style-type: none"> <li><b>BCD</b>: The value is BCD-encoded.</li> </ul>

## Examples

Value Type	Path	Meaning
Int16	DB1000.DW 20	In Data Block 1000, access the word at offset 20.
Int16-Array	DB1000.DW 30, 10	In Data Block 1000, start at offset 30 and read/write 10 words (exclusive end offset: 40).
Int32	DB1000.DD 40, BCD[4]	In Data Block 1000, start at offset 40 and read/write four 32-bit values (exclusive end offset: 48) and decode/encode the values as BCD.
Boolean-Array	DB1000.DX 50.12, 24	In Data Block 1000, access the bits from offset 50.12 to (exclusive) offset 52.4 (the bits are read/written as three 16-bit words).
Byte-Array	DB1000.DR 60, 2	In Data Block 1000, access the bytes from offset 60 (low byte) and 61 (high byte)

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

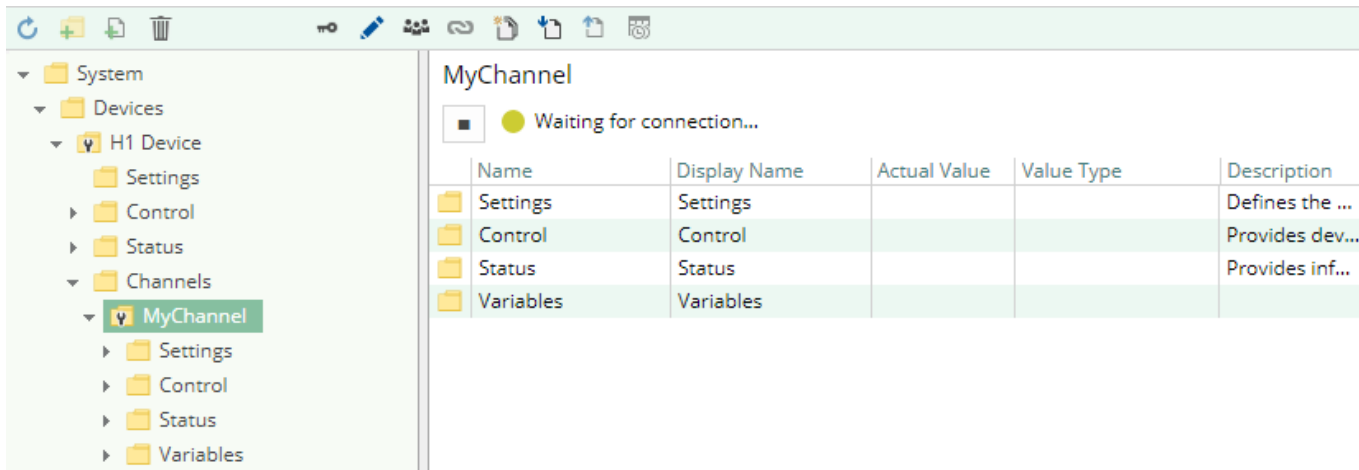
## Diagnostics

The H1 Device Plugin provides different status information depending on the layer to inspect. In general the channel-based diagnostic information is produced by the connection status of the channel to the PLC. The variable-based diagnostic information is produced during the read/write access of the different variables.

## Channel

To monitor and diagnose the status of a H1 Channel, take a look at the following image:





The image above depicts the H1 Channel's Control Panel which displays all status relevant information. The control panel will automatically update its status information when a new status is available.

### Status Circle

Color	Meaning
●	The channel is stopped. Click the ► button to start it.
●	The channel is currently in the progress of starting or stopping or is waiting to establish a connection.
●	The channel is ready for doing read/write operations. You can stop it by clicking the ■ button.
●	The channel is running, but the connection is currently in an error state. Please check the status text for more information.

## Variables

To monitor and diagnose the status of the different variables, take a look at the variable's **Status** property displayed in UKI-4.0 . Use the button “Read actual Value” to read the values from the PLC and store the result into the variables.

	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.

## Log file

All channel related status information is also logged into the channel-specific log file stored in the **[LoggingFolder]**. Each log file is named in the naming scheme **H1 Device.<ChannelName>.log**.

The content of such a log file can look as follows:

```

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

# Entities

Like every device plugin the H1 Device Plugin extends the basic UKI-4.0 [Device Model](#).

## Device

The plugin's device type `H1Device` also defines the `H1DeviceChannel` and therefore extends the basic UKI-4.0 `Device` and UKI-4.0 `DeviceChannel` entities. While the `H1Device` just represents a concretization of the UKI-4.0 `Device`, the `H1DeviceChannel` extends the UKI-4.0 `DeviceChannel` with the H1 Variable Entities.

## Channel

Each channel is handled by a channel worker which establishes a network connection to the PLC.

By default, the worker does not read any values. When a client or plugin requests a synchronous read of the channel's variables in UKI-4.0 (e.g. using the UKI-4.0 Web Configuration's function "Read actual value"), the channel worker reads them from the underlying PLC and then writes them into the corresponding UKI-4.0 Nodes.

Similarly, when a client or plugin writes values into the channel's variables, the channel worker will write those values to the underlying PLC.

To have an H1 variable being read steadily, you can edit the Node in the UKI-4.0 Web Configuration and set "History Options" to **Yes** (which will create an internal subscription), or you can use e.g. a OPC UA Client connected to the OPC UA Server plugin and create a subscription for the H1 variable Nodes. In these cases, the channel worker reads the variables from the PLC at a regular interval and, if the value of one of the variables has changed, writes the new value into the corresponding UKI-4.0 Node.

# Folders & Files

## Folders

Content	Path	Usage
AssemblyFolder	<UKI-4.0 <code>InstallDir</code> >/plugins/H1DevicePlugin/	Contains the plugin's assembly file.
ConfigFolder	<UKI-4.0 <code>ProjectDir</code> >/plugins/H1DevicePlugin/	Contains the plugin's configuration file.
LoggingFolder	<UKI-4.0 <code>ProjectDir</code> >/log/	Contains the plugin's log files.

## Files

Type	Path	Usage
Assembly	[ <code>AssemblyFolder</code> ]/UKI-4.0 <code>.H1DevicePlugin.dll</code>	The plugin's assembly file.
Logging	[ <code>LoggingFolder</code> ]/H1 Device.<ChannelName>.log	The log file.

# About Versions

## This Document

<b>Date</b>	2020-01-07
<b>Version</b>	1.0

## Plugin

<b>Name</b>	H1 Device Plugin
<b>Node</b>	/System/Devices/H1 Device
<b>Version</b>	1.0.0

## Assembly

<b>Name</b>	UKI-4.0 .H1DevicePlugin.dll
<b>Date</b>	2020-01-07
<b>Version</b>	1.0.0.0

# OPC UA Client Device Plugin

## General

The OPC UA Client Device Plugin provides linkage between UKI-4.0 and an OPC UA Server.

## What Does the Plugin Do?

The OPC UA Client Device Plugin establishes and maintains connections to one or more OPC Servers. Every OPC UA Client can be connected through a separate channel.

## Features

- Browse OPC UA Servers (or OPC Classic Servers) to automatically create available variables in UKI-4.0
- Read and write OPC UA/OPC Classic Data Variables
- Subscribe OPC UA/OPC Classic Data Variables when the corresponding Variable Nodes are subscribed in UKI-4.0

## Supported Servers

- OPC-UA Server using the `opc.tcp` or `http` protocol
- OPC Classic (COM) Servers (specified by the ProgID and ClassID) (only on Windows x86 and x64)

## Purpose & Use Cases

The linked OPC UA Servers can simply be controlled by using UKI-4.0. By linking the Nodes of the OPC UA Server to the Nodes defined in UKI-4.0 the OPC Node can directly interact with many other Nodes, devices, services, etc. maintained in UKI-4.0.

Also, other UKI-4.0 participants can interact with the OPC UA Server linked through the OPC UA Client Device Plugin.

## Installation

This plugin is part of the UKI-4.0 Setup. Please consult UKI-4.0 [Setup and First Start](#) for more information on how to install and uninstall this plugin.

## Requirements

- Basic requirements of UKI-4.0
- Enabled outgoing TCP/IP connection via the specified port

# Configuration

## Overview

The entire OPC UA Client Device Plugin configuration is located under the Node path `/System/Devices/OPC UA Client Device`. Using this root Node of the device plugin allows the full configuration of the OPC UA Client Device Plugin.

Name	Display Name	Description	Node Type	Path	Actual Value	Status code
ApplicationName	ApplicationName		String		CoDaBix OPC Client Device	Good
LoginCertificate	LoginCertificate		Blob			---
LoginName	LoginName		String			---
LoginPassword	LoginPassword		Blob			---
ServerAddress	ServerAddress		String		opc.tcp://192.168.0.116:12345	Good

## Usage

The Node tree in the image above depicts the OPC UA Client Device Plugin's default Node tree. To set up one or more OPC UA Client Device Channels, add a Folder Node beneath the Node `OPC UA Client Device/Channels` (left picture). Afterward the default Node tree for a channel will appear (right picture).

The dialog box 'Add new Folder Node' contains the following fields:

- Name: My Channel 1
- Display Name: My Channel 1
- Node Type: Folder
- Path: (empty)
- Refresh Interval: <Inherit>
- Max Value Age (ms): (empty)
- History Value Interval: (empty)

The resulting node tree on the right shows the 'Channels' folder expanded to reveal 'My Channel 1', which contains sub-folders for 'Control', 'Settings', 'Status', and 'Variables'.

Now the settings for the specified channel can be changed. Also new variables can be created below the `OPC UA Client Device/Channels/<Channel>/Variables` folder. The link to the OPC Node will be granted using the `Path` property of the new variable Node.

The `Path` property has to be a validly formatted OPC Nodeld, for example `2:Main-PLC/Office 1 - Lights/Front`.

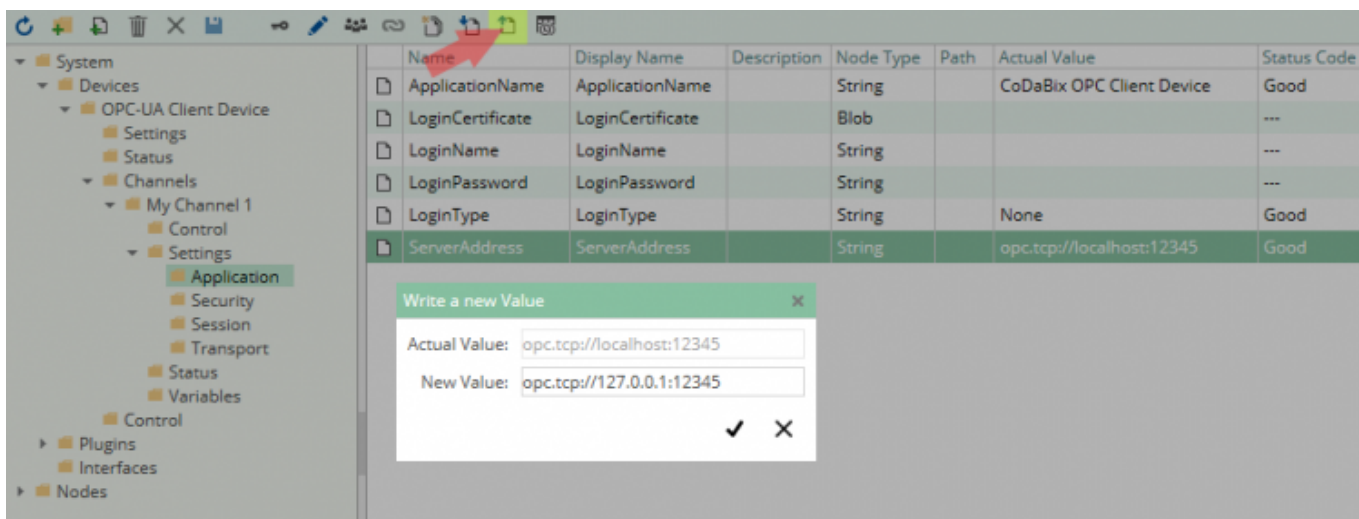
Every change in the `Settings` folder will apply as soon as the channel is restarted.

Every change in the `Variables` folder will automatically perform a reconfiguration of the OPC Client in the specified channel.

# Settings

## Change Settings

- Select the **Settings** property which shall be changed. (e.g. "Application/ServerAddress")
- Click "Write a new Value". (See example screenshot below)
- Enter the new value in the input field and save the changes.



## Overview

Name	Type	Default Value	Description
Application			
Application Name	String	UKI-	With this name the OPC UA Client will introduce itself to the OPC UA Server.
Server Address	String	opc.tcp://localhost:12345	<p>The URL to the OPC UA Server. Possible protocols are: <b>opc.tcp://</b>, <b>http://</b>, <b>https://</b>, <b>opc.com://</b>.</p> <p>To access OPC Classic (COM) servers on Windows (x86 or x64), use the format <b>opc.com://&lt;hostname&gt;(:&lt;port&gt;)/&lt;progId&gt;/&lt;classId&gt;</b>. The optional port number is used by the inline created wrapper server. In case the port number is missing it will be generated between 48000 and 48999 using the <b>&lt;classId&gt;</b>-value. Example for OPC Classic: <b>opc.com://localhost:4711/OPCManager.DA.XML-DA.Server.DA/{E4EBF7FA-CCAC-4125-A611-EAC4981C00EA}</b></p> <p><b>Please note:</b> Some OPC Classic servers support only <b>x86 (32-bit) clients</b>. Therefore, we recommend to install the <b>x86</b> version of <b>UKI-</b> instead of the <b>x64</b> version if you want to access such servers.</p>
Login Type	Enum	None	<p>Defines which authentication type will be used.</p> <p>Valid Values: <b>Anonymous, Certificate, UserPassword</b></p>
Login Certificate	Blob		The certificate of the OPC UA Server used for the authentication. Can be uploaded as certificate file. (Has to contain the key of the certificate)
Login Name	String		The user of the OPC UA Server used for the authentication.
Login Password	String		The password for the given user of the OPC UA Server.
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	Automatically detects if the connection to the OPC UA Server is interrupted. This is used for automatic reconnection if the connection to Server was lost. Valid Values: <b>True, False</b>
Security			
Policy Algorithm	String	Auto	Valid Values: <b>Auto, Basic128Rsa15, Basic256, Basic256Sha256, Custom, Https, None</b>
Policy Level	Int32	0	t.b.a.
Policy Mode	String	None	Valid Values: <b>None, Sign, SignAndEncrypt</b>
Use Domain Checks	Boolean	False	<p>Indicates if the OPC UA Client checks the OPC UA Server for a trusted certificate or not. This is a security feature to prevent e.g. man-in-the-middle attacks. Valid Values: <b>True, False</b></p> <p><b>Attention</b> If this option is set to <b>true</b> and the OPC UA Server doesn't have a trusted X.509 certificate (e.g. a self signed certificate) the connection will be rejected.</p>

Name	Type	Default Value	Description
Use Secure Endpoint	Boolean	True	Indicates if the OPC UA Client shall check if the endpoint is secure or not. Some OPC UA Servers might not support this option. Valid Values: <b>True, False</b>
Certificate	Blob	<Blob>	The Client certificate. With this certificate the OPC UA Client will introduce itself to the OPC UA Server. A new certificate is created by default.
<b>Transport</b>			
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.

### Variables

Every Node beneath **System/Devices/OPC UA Client Device/Channels/My Channel 1/Variables** can be linked to an OPC Node from the OPC Server of the specified Channel using the **Path** property.

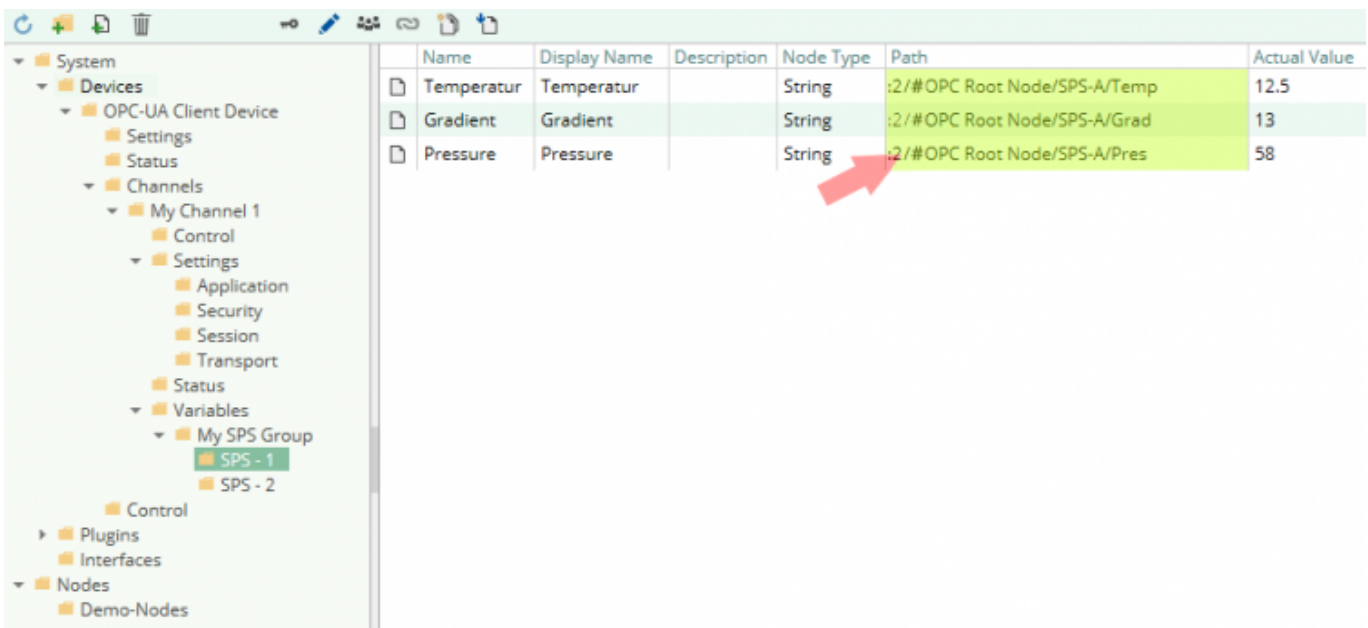
To connect a UKI-4.0 Node to an OPC UA Node the **Path** property has to be formatted as an OPC UA Nodeld.

### OPC Nodeld

- Example Nodeld: **ns=2;s=Machine\_1/IsActive**

The specified Nodeld is splitted in two parts: the Namespace Index and the Identifier of the OPC UA Node. The schema of a string identifier depends on the used OPC UA server.

- Namespace Index: **ns=<Number>** e.g. **ns=2**
- Separator: **;**
- Identifier (String): **s=<ParentNode1>/<ParentNode2>/.../<ParentNodeX>/<NodeName>**



# Diagnostics

The OPC UA Client Device Plugin provides different status information, depending on the layer to inspect. In general the channel-based diagnostic information is produced by the connection status of the channel to the OPC UA Server. The variable-based diagnostic information is produced during the read/write access of the different variables or directly from the OPC UA Server.

## Channel

To monitor and diagnose the status of the different device channels take a look at the following image:

Name	Display Name	Description	Node Type	Path	Actual Value	Status code
Category	Category	The category of the status.	String		Information	Good
Code	Code	The code of the status.	Int32		2100	Good
Report	Report	The log file used for status...	File	C:\Users\develop...		---
Severity	Severity	The severity of the status.	String		Moderate	Good
Text	Text	The text of the status.	String		Stopped!	Good

The image above depicts the channel's **Status** Node which displays all status relevant information. The following datapoint Nodes are used to persist the communication status between UKI-4.0 and the OPC UA Server.

Node	Description
<b>Category</b>	Classifies the status into <b>Information</b> , <b>Warning</b> and <b>Error</b> and therefore indicates the general type of status information.
<b>Code</b>	Defines the numeric expression/identifier of the status.
<b>Severity</b>	Rates the status information into <b>Low</b> , <b>Moderate</b> , <b>High</b> and <b>Critical</b> and therefore indicates the urgency of an intervention.
<b>Text</b>	Describes the status information identified by the <b>Code</b> property.

## Variables

To monitor and diagnose the status of the different OPC UA Client variables take a look at the variables' **Status** property displayed in UKI-4.0 .

In case that the **Status** column displays the value **Bad** the addressed data area is in most cases not accessible.

Name	Display Name	Description	Node Type	Path	Actual Value	Status Code
Temperatur	Temperatur		String	:2/#OPC Root Node/SPS-A/Temp	12.5	Good
Gradient	Gradient		String	:2/#OPC Root Node/SPS-A/Grad	13	Good
Pressure	Pressure		String	:2/#OPC Root Node/SPS-A/Pres	58	Good



## Log File

All device channel related status information is also logged into the channel-specific log file stored in the `[LoggingFolder]`. Each log file is named following the naming scheme `OPC UA Client Device.<ChannelName>.log`. The content of such a log file can look as follows:

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

Using the sample channel the name of the log file would be: `OPC UA Client Device.Channel 1.log`

## Status Codes

The following table displays the different status information possible.

Code	Category	Information Type
-22000 to -22999	Error	Error with exception
-21000 to -21999	Error	Internal error
-12000 to -12999	Warning	Warning with exception
-11000 to -11999	Warning	Internal warning
10000 to 10999	Information	Information
20000 to 20999	Information	Debug

# Entities

Like every device plugin the OPC UA Client Device Plugin extends the basic UKI-4.0 [Device Model](#).

## Device

The plugin's device type `OpcClientDevice` also defines the `OpcClientDeviceChannel` and therefore extends the basic UKI-4.0 `Device` and UKI-4.0 `DeviceChannel` entities. While the `OpcClientDevice` just represents a concretization of the UKI-4.0 `Device`, the `OpcClientDeviceChannel` extends the UKI-4.0 `DeviceChannel` with the OPC UA Client Variable Entities.

## Channel

Each channel is handled by a channel worker which establishes a physical connection to the OPC UA Server.

By default, the worker does not read any values. When a Client or plugin requests a synchronous read of the channel's variables in UKI-4.0 (e.g. using the UKI-4.0 Web Configuration's function "Read actual value"), the channel worker reads them from the underlying OPC UA Server and then writes them into the corresponding UKI-4.0 Nodes.

Similarly, when a Client or plugin writes values into the channel's variables, the channel worker will write those values to the underlying OPC UA Server.

To have an OPC UA Client variable being read steadily, you can edit the Node in the Configuration Web GUI and set "History Options" to **Yes** (which will create a subscription internally). In this case, the channel worker subscribes to the variables from the OPC Server (the OPC Client will get the new value from the OPC Server automatically if the value has been changed), and if the value of one of the variables has changed, writes the new value into the corresponding UKI-4.0 Node.

# Folders & Files

## Folders

Content	Path	Usage
AssemblyFolder	<UKI-4.0 InstallDir>/plugins/OpcUaClientDevicePlugin/	Contains the plugin's assembly file.
ConfigFolder	<UKI-4.0 DataDir>/plugins/OpcUaClientDevicePlugin/	Contains the plugin's configuration file.
LoggingFolder	<UKI-4.0 DataDir>/log/	Contains the plugin's log files.

## Files

Type	Path	Usage
Assembly	[AssemblyFolder]/UKI-4.0 .OpcUaClientDevicePlugin.dll	The plugin's assembly file.
Logging	[LoggingFolder]/OPC UA Client Device.<ChannelName>.log	The log file.

# About Versions

## This Document

<b>Date</b>	2021-01-25
<b>Version</b>	1.3

## Plugin

<b>Name</b>	OPC UA Client Device Plugin
<b>Node</b>	/System/Devices/OPC UA Client Device
<b>Version</b>	1.10.0

## Assembly

<b>Name</b>	UKI-4.0 .OpcUaClientDevicePlugin.dll
<b>Date</b>	2021-01-25
<b>Version</b>	1.10.0.0

# S7 Device Plugin

The S7 Device Plugin allows reading and writing values from physical SIMATIC S7 PLC devices via TCP/IP.

The following device types are supported:

- S7-1500 (see [PLC Settings](#))
- S7-1200 (see [PLC Settings](#))
- S7-300
- S7-400
- WinAC
- S7-SoftPLC
- LOGO! (see [PLC Settings](#))
- S7-200
- SIMATIC S5 over S5-LAN
- S7-LAN
- VIPA-S7 and any other S7-TCP/IP compatible PLC

## Features

- Optimized read and write requests by using the best utilization regarding the package size.
- Automated connection handling including auto-reconnect.
- Accessing PLC memory using user defined types.
- Use of existing PLC projects to set up channels and variables. The following project formats are supported:
  - STEP7 Project Files (\*.s7p)
  - IP-S7 Project Files (\*.ips7)
  - S7 Watch Project Files (\*.wproj)
  - CSV-to-S7 Project Files (\*.ini)

## Purpose & Use Cases

### Purpose

The linked devices can simply be controlled using UKI-4.0®. By linking the PLC memory to the Nodes defined in UKI-4.0® the PLC can directly interact with many other Nodes, devices, services, etc. maintained in UKI-4.0®. Also other UKI-4.0® participants can interact with the PLC devices linked through the S7 Device Plugin.

### Use Cases

- Dynamic generation of manufacturing jobs based on different conditions and data produced by machines, users, orders, condition, services, etc.
- Additional reliability by monitoring the whole plant including inter-plant interaction
- Centralized controller and flow surveillance for early diagnosis of possible failures
- Possibilities to improve the Shop Floor Management by monitoring and recording production data produced by the devices

# Installation

This plugin is part of the UKI-4.0® Setup. Please consult UKI-4.0® [Setup and First Start](#) for more information on how to install and uninstall this plugin.

## Requirements

- Basic requirements of UKI-4.0
- Enabled outgoing TCP/IP connection via port 102

## PLC Settings

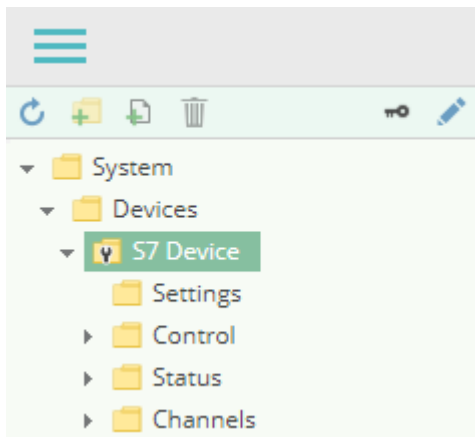
Please note that in order to access the S7-1200, S7-1500 and the LOGO!, the **optimized data block access** has to be **deactivated** at the data block attributes.

To do that, you can find instructions at: [Deactivate optimized data block access](#).

# Configuration

## Using UKI-4.0UKI-4.0 UKI-4.0UKI-4.0

The whole S7 Device Plugin configuration is located in the node path `/System/Devices/S7 Device`.



## Channel

An S7 Device Channel represents the connection to a S7 PLC.

### Settings

#### Address

IP address or hostname of the S7 PLC.

#### Rack

The rack number of the PLC device (ignored since S7-1200).

#### Slot

The slot number of the PLC device (ignored since S7-1200).

## Device Type

The device type of the SIMATIC S7 PLC. The following device types are supported:

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

## Channel Type

The type of device channel to use to communicate with the PLC.

## DateTime Interpretation

Specifies how the timezone of read and written DateTime values should be interpreted (are the date/time values in the PLC stored as UTC time or as local time).

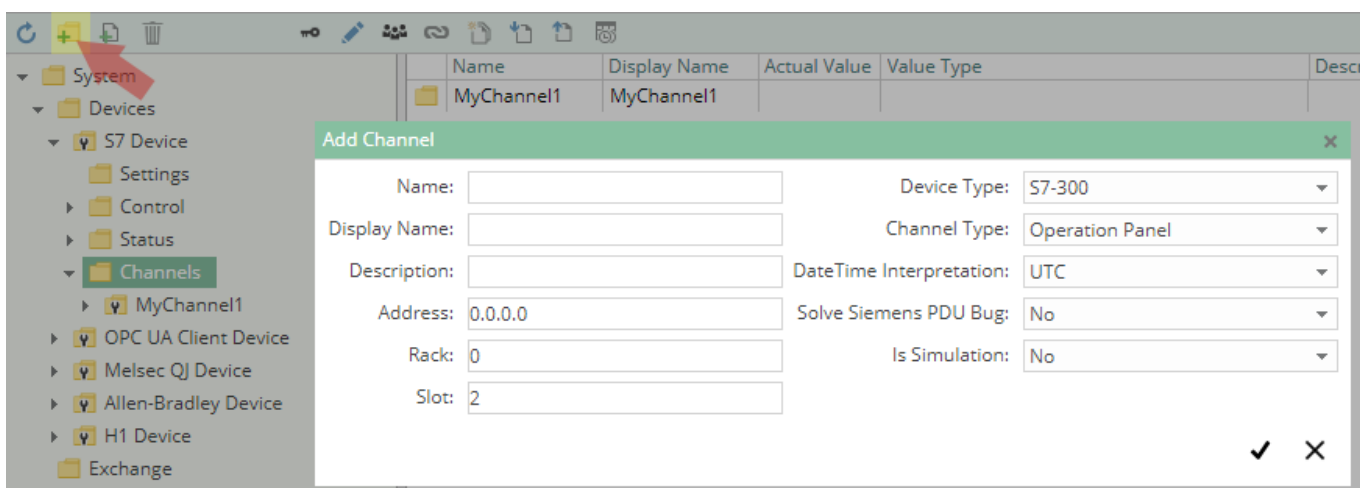
## Solve Siemens PDU Bug

Specifies if a reduced PDU size should be used to work-around an issue in some S7-400 devices which would otherwise randomly truncate a large data packet, resulting in an error like "specified data area doesn't exist".

## Is Simulation

Specifies whether this device runs in simulation mode, which means no connection to a physical device is established.

## Adding a Channel



To add a new S7 Channel, please follow these steps:

1. Add a Folder Node within the node **S7 Device/Channels**, or right-click on the **S7 Device/Channels** node and select **Add Channel**.
2. In the **Add Channel** dialog, specify the settings for the S7 connection.
3. After clicking on "Save", the channel node is created.

- You can start the channel by selecting the channel node and clicking the start button.

## Variables

Within the node **Variables** you can create datapoint nodes which can be read and written from/to the S7. Additionally, you can import variables e.g. from a **STEP7** project.

The **Value Type** property must be set to the corresponding S7 Device type. Currently the following types are supported in the S7 Device Plugin:

S7 Device Type	PLC Type	Preferred UKI-4.0 Value Type	Description
Bool	BOOL	Boolean or Boolean-Array	A variable of the PLC type BOOL.
Byte	BYTE	Byte or Byte-Array	A variable of the PLC type BYTE.
Char	CHAR	String or String-Array	A variable of the PLC type CHAR.
Int	INT	Int16 or Int16-Array	A variable of the PLC type INT. Representing a signed 16 bit integer.
Word	WORD	UInt16 or UInt16-Array	A variable of the PLC type WORD. Representing an unsigned 16 bit integer.
DInt	DINT	Int32 or Int32-Array	A variable of the PLC type DINT. Representing a signed 32 bit integer.
DWord	DWORD	UInt32 or UInt32-Array	A variable of the PLC type DWORD. Representing an unsigned 32 bit integer.
LInt	LINT	Int64 or Int64-Array	A variable of the PLC type LINT. Representing a signed 64 bit integer.
LWord	LWORD	UInt64 or UInt64-Array	A variable of the PLC type LWORD. Representing an unsigned 64 bit integer.
Real	REAL	Single or Single-Array	A variable of the PLC type REAL. Representing a single precision floating point number.
Double	REAL	Double or Double-Array	A variable of the PLC type REAL. Representing a double precision floating point number.
LReal	LREAL	Double or Double-Array	A variable of the PLC type LREAL. Representing a double precision floating point number.
Date	DATE	DateTime or DateTime-Array	A variable of the PLC type DATE.
Time	TIME	TimeSpan or TimeSpan-Array	A variable of the PLC type TIME.
TimeOfDay	TIME_OF_DAY	TimeSpan or TimeSpan-Array	A variable of the PLC type TIME_OF_DAY.
S5Time	S5TIME	TimeSpan or TimeSpan-Array	A variable of the PLC type S5TIME.
DateTime	DATE_AND_TIME	DateTime or DateTime-Array	A variable of the PLC type DATE_AND_TIME.
String	STRING	String	A variable of the PLC type STRING.
S5String	BYTE	String	A variable of the PLC type BYTE. A fixed number of bytes is interpreted as string.

### Path

The **Path** property of the node is used to specify the address, optionally a type, and (for arrays or strings) the length:

```

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

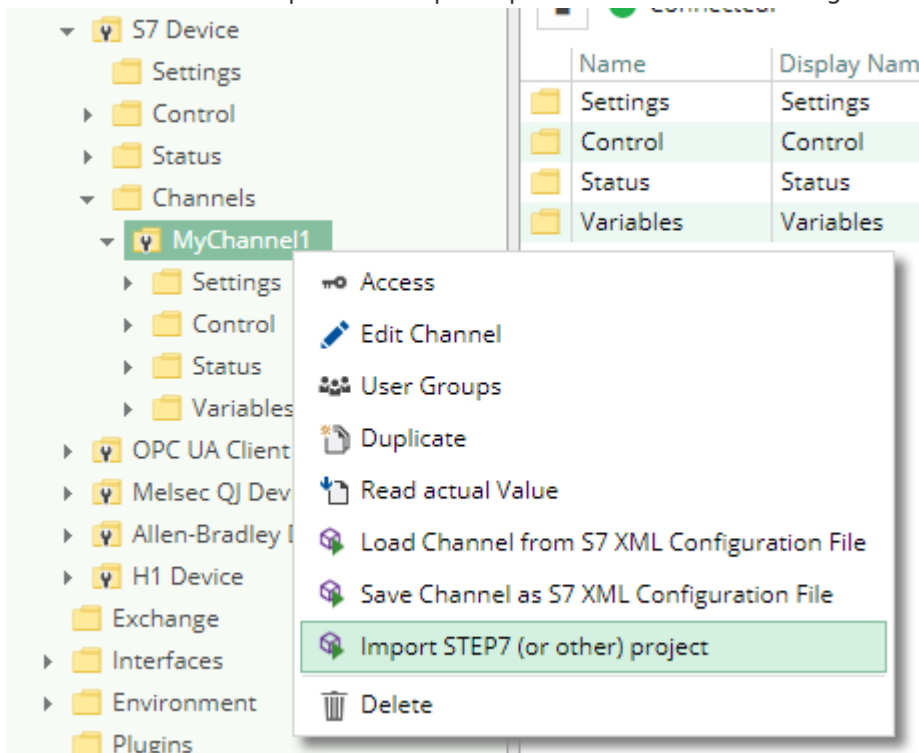
Placeholder	Description
<Address>	The address of the data. Example: <b>DB10.DBW 16</b>
<Length>	If the variable is an array or a string, specify the array/string length here. It will only be used for reading, because when writing an array/string, its length will be determined from the value that is written.
<Type>	If present, contains the S7 Device Type (see table above) which will override the default type inferred from the UKI-4.0 Value Type. Note: When using UKI-4.0 Value Type <b>Double</b> , The S7 Device Type is not automatically inferred; you will need to specify <b>Double</b> explicitly.

Examples

UKI-4.0 Value Type	Path	Meaning
Boolean	DB10.DBX 3.2	Single bit (BOOL) at address DB10.DBX 3.2
Boolean-Array	DB10.DBX 3.4, 18	Bit (BOOL) array from address DB10.DBX 3.4 to DB10.DBX 5.6 (exclusive)
Int32	DB10.DBD 12, DInt	Single DInt (DINT) at address DB10.DBW 12
Double-Array	DB10.DBD 20, Double[5]	Double-Array (REAL) from address DB10.DBD 20 to DB10.DBD 40 (exclusive)
String	DB10.DBB 40, 100	String (STRING) from address DB10.DBB 40 to DB10.DBB 142 (exclusive)
String	DB10.DBB 40, S5String[100]	S5String (BYTE) from address DB10.DBB 40 to DB10.DBB 140 (exclusive)

Import/Export

The S7 Device Plugins supports importing and exporting individual channels as S7 XML Configuration file (see next section). Additionally, you can import a **STEP7** project (.s7p) that has been packed into a ZIP file into a channel. The import and export options are shown when right-clicking on a specific channel:



To import a configuration file or STEP7 project as a new channel, create a new channel first (with default settings), then right-click on the channel to open the import dialog.

## Using the S7 XML Configuration File

### Structure

The S7 Device Plugin defines the root of its element tree by the `PluginSettings` element as described in [Plugin Configuration - Using a Configuration File](#) and continues its XML tree using the `Channel` element.

#### Channel Element

The `Channel` element serves as the container for the elements `Settings` and `Variables`. One channel identifies one PLC connection to which the channel configuration belongs. This information is therefore used by the S7 Device Plugin to link the PLC to UKI-4.0 ®.

The `Channel` element can look as follows:

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

#### Settings Element

The `Settings` element defines the attributes to setup the channel. These attributes configure the channels connection parameters.

The `Settings` element can look as follows:

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

The `Settings` element provides the following list of attributes:

	Mandatory	Type	Purpose
<b>Address</b>	yes	String	The IP Address of the PLC device to link.
<b>Rack</b>	no	Int32	The rack number of the PLC device (ignored since S7-1200).
<b>Slot</b>	no	Int32	The slot number of the PLC device (ignored since S7-1200).
<b>ChannelType</b>	no	S7 Device Channel Type	The type of device channel to use to communicate with the PLC.
<b>DeviceType</b>	no	S7 Device Type	The type of PLC device.

#### S7 Device Channel Type

The following values are valid for attributes of that type:

Value	Description
<code>"OperationPanel"</code>	To connect via OP channel to the device.
<code>"ProgrammerDevice"</code>	To connect via PG channel to the device.
<code>"Other"</code>	To connect to the device using alternative channels.

#### S7 Device Type

The following values are valid for attributes of that type:



Value	Description
"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

The **Variables** element serves as the container for one or more **Variable** elements. This element maintains all variables associated with the channel.

The **Variables** element can look as follows:

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

### Variable Element

The **Variable** element serves as the container for the element **Variables**. One variable identifies one addressable area in the memory of the PLC or a set of subsequent variables identify multiple addressable areas in the memory of the PLC. This information is therefore used by the S7 Device Plugin to link the PLC memory to UKI-4.0 ® Nodes.

Each **Variable** element provides the following list of attributes:

	Mandatory	Type	Purpose
<b>Identifier</b>	no	GUID	The generic unique identifier of the entity associated with the variable. <i>This is a generic entity attribute, for more information about its usage see <a href="#">Plugin Configuration - Using a Configuration File: The Entities Identifier Attribute</a>.</i>
<b>ChangeType</b>	no	ChangeType	The state of the entity configuration used to represent the variable. <i>This is a generic entity attribute, for more information about its usage see <a href="#">Plugin Configuration - Using a Configuration File: The Entities ChangeType Attribute</a>.</i>
<b>Name</b>	<b>yes</b>	String	The unique name (within the <b>Variables</b> element) of the variable.
<b>Description</b>	no	String	The description of the usage and purpose of the addressed memory area.
<b>Address</b>	<b>yes</b>	PLC Address	The operand to use to address the memory of the PLC (not supported in this case: Type="Object").
<b>Type</b>	<b>yes</b>	PLC Variable Type	The type of data stored at the addressed memory and how it is to be interpreted.
<b>Length</b>	no	Int32	The length of an array or string variable (only supported on Type="String" and numerical Types). If this attribute is defined the variable defines an array value (if supported), otherwise a scalar value.

#### PLC Variable Type

The following values are valid for attributes of that type:

Value	Description
"Bool"	A variable of the PLC type BOOL.
"Byte"	A variable of the PLC type BYTE.
"Char"	A variable of the PLC type CHAR.

Value	Description
"Int"	A variable of the PLC type INT. Representing a signed 16 bit integer.
"Word"	A variable of the PLC type WORD. Representing an unsigned 16 bit integer.
"DInt"	A variable of the PLC type DINT. Representing a signed 32 bit integer.
"DWord"	A variable of the PLC type DWORD. Representing an unsigned 32 bit integer.
"Real"	A variable of the PLC type REAL. Representing a single precision floating point number.
"Double"	A variable of the PLC type REAL. Representing a double precision floating point number.
"Date"	A variable of the PLC type DATE.
"Time"	A variable of the PLC type TIME.
"TimeOfDay"	A variable of the PLC type TIME_OF_DAY.
"S5Time"	A variable of the PLC type S5TIME.
"DateTime"	A variable of the PLC type DATE_AND_TIME.
"String"	A variable of the PLC type STRING.
"S5String"	A variable of the PLC type BYTE. A fixed number of bytes is interpreted as string.

### PLC Variable Address

The following Operand and Data Type combinations are valid to construct a valid PLC variable address:

#### Operands

Operand	Siemens, DE	IEC
Input	E	I
Output	A	Q
Flag	M	M
Peripherals	P	P
Counter	Z	C
Data Block	DB	DB
Timer	T	T

#### Data Types

Data Type	Operand	Bits	Range	Description
BOOL	X	1	0 to 1	A single bit representing true (1) or false (0).
BYTE	B	8	0 to 255	An unsigned 8-bit integer.
WORD	W	16	0 to 65.535	An unsigned 16-bit integer (Word).
DWORD	D	32	0 to $2^{32}-1$	An unsigned 32-bit integer (Double Word).
CHAR	B	8	A+00 to A+ff	An ASCII-Code as an unsigned 8-bit character.
INT	W	16	-32.768 to 32.767	A signed 16-bit integer.
DINT	D	32	$-2^{31}$ to $2^{31}-1$	A signed 32-bit integer (Double Word).
REAL	D	32	+ $-1.5e-45$ to + $-3.4e38$	An IEEE754 32-bit single precision floating point number.
S5TIME	W	16	00.00:00:00.100 to 00.02:46:30.000	A binary coded decimal (BCD) number representing a time span.
TIME	D	32	00.00:00:00.000 to 24.20:31:23.647	A signed 16-bit integer representing a time span in milliseconds.
TIME_OF_DAY	D	32	00.00:00:00.000 to 00.23:59:59.999	An unsigned 16-bit integer representing a time span in milliseconds.
DATE	W	16	01.01.1990 to 31.12.2168	An unsigned 16-bit integer representing a date in days.

Data Type	Operand	Bits	Range	Description
DATE_AND_TIME	D	64	00:00:00.000 01.01.1990 to 23:59:59.999 31.12.2089	A binary coded decimal (BCD) number representing a date and time.

*Examples*

Example	Data type	Siemens	IEC
Input Byte 1, Bit 0	BOOL	E 1.0	I 1.0
Output Byte 1, Bit 7	BOOL	A 1.7	Q 1.7
Flag Byte 10, Bit 1	BOOL	M 10.1	M 10.1
Data Block 1, Byte 1, Bit 0	BOOL	DB1.DBX 1.0	DB1.DBX 1.0
Input Byte 1	BYTE	EB 1	IB 1
Output Byte 10	BYTE	AB 10	QB 10
Flag Byte 100	BYTE	MB 100	MB 100
Peripherals Input Byte 0	BYTE	PEB 0	PIB 0
Peripherals Output Byte 1	BYTE	PAB 1	PQB 1
Data Block 1, Byte 1	BYTE	DB1.DBB 1	DB1.DBB 1

The **Variable** element can look as follows:

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

The **Variable** element with the **Type** attribute and the attribute value "Object" can look as follows:

```
<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>
```

## Usage

It is recommended to use a professional XML editor when editing the configuration file manually. To also benefit from the XML scheme definition mentioned at the end of this article you have to refer to the scheme definition by using the `xsi:noNamespaceSchemaLocation` attribute in the document root element `PluginSettings` as follows (the XSD file needs to be placed next to the XML file):

```
<?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>
```

Independent from editing / creating the configuration file manually or automated, the element tree above documented must be fulfilled to produce a valid, well-formed and usable configuration file.

## Synchronization

An S7 XML Configuration file is assigned to a S7 Device Channel in UKI-4.0 for **automatic synchronization**, if the following condition is true:

- A file with the name `UKI-4.0 .S7DevicePlugin.<ChannelName>.Settings.xml` exists in the ConfigFolder (`<UKI-4.0 ProjectDir>/plugins/S7DevicePlugin`) when starting the S7 Plugin (i.e. when starting UKI-4.0 ) or when creating a new channel in UKI-4.0 .

In that case, once the configuration file changes, it is automatically synchronized into UKI-4.0 . Also, when the variables in UKI-4.0 (or the channel settings) are changed, the changes are synchronized into the configuration file.

Regardless of the automatic synchronization, you can also use manual synchronization by right-clicking on a S7 Device Channel in UKI-4.0 and selecting the corresponding Import/Export menu item.

## Example Configuration File

UKI-4.0 [.S7DevicePlugin.Settings.xml](#)

```

<?xml version="1.0" encoding="utf-8"?>
<PluginSettings>
  <Channel>
    <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>
  </PluginSettings>

```

## Example Configuration Scheme File

### 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="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:enumeration value="S5String" />
    </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: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:restriction>
</xs:simpleType>

```

```

    <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: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:complexType>

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

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

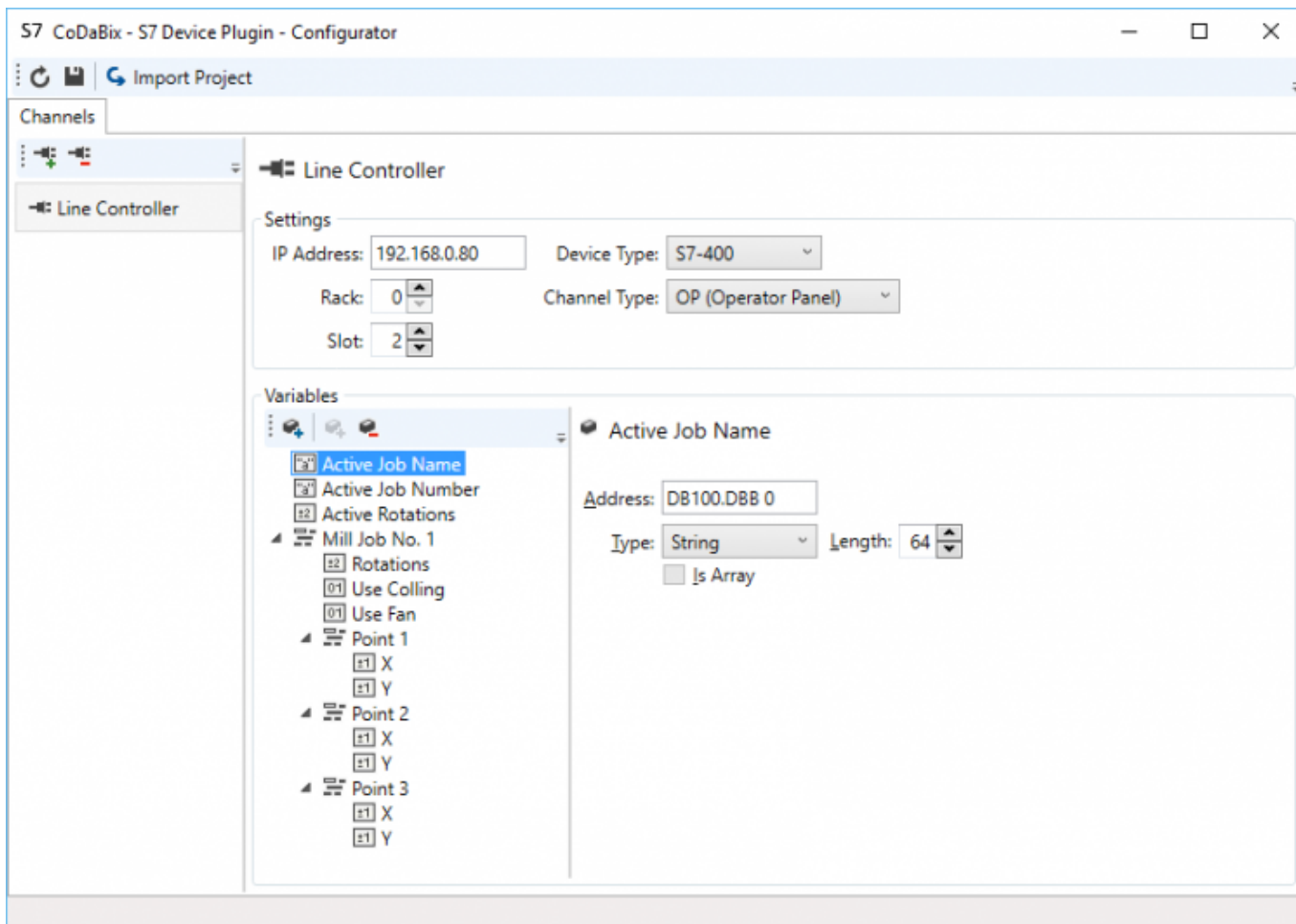
```

## Using the Application (Windows only)

**Note:** The configuration application allows you to load and save S7 XML Configuration Files (.xml) that can be imported and exported on a S7 Device Channel in UKI-4.0 (using the context menu), or that are assigned to an UKI-4.0 S7 Device Channel for automatic synchronization (see section [Using the S7 XML Configuration File](#)).

The configuration application resides in the folder `<UKI-4.0 InstallDir>` and is started by double-clicking the file with the name `UKI-4.0 .S7DevicePlugin.Configurator.exe`.

### Overview



## Usage

- Add a new **Channel** in the left list by clicking the upper plus button.
  - Set the name of the channel by clicking its name in the right pane.
  - Set up the **Channel Settings** like the **IP Address** (at least required).
  - Optionally set the appropriate **Device Type** and select the **Channel Type** to be used.
  - Optionally set the **Rack** and **Slot** (in case it differs from the defaults).
- Add a new **Variable** to the **Channel** in the second from the left list by clicking the upper first button with the plus sign.
  - Set up the name of the variable by clicking its name in the right pane.
  - Optionally set the description of the variable by clicking in the free area beneath the variable name.
  - Set up the PLC address operand to be used to access the PLC memory for that variable.
  - Set up the PLC variable type in the combo and enter, in case of a string or an array (check the option **Is Array**) variable, the appropriate length of the variable.
  - Optionally add additional variables in case the previously added variable is of the type "Object". Do that by selecting the object variable in the left tree and then click the second add button with the plus sign above.
- To remove a variable, select it in the tree and click on the button with the minus sign above.
- To remove a channel, select it in the left list and click on the button with the minus sign above.
- Using the **Import Project** button it is possible to import the following project formats:
  - STEP7 Project Files (\*.s7p)
  - IP-S7 Project Files (\*.ips7)
  - S7 Watch Project Files (\*.wproj)
  - CSV-to-S7 Project Files (\*.ini)

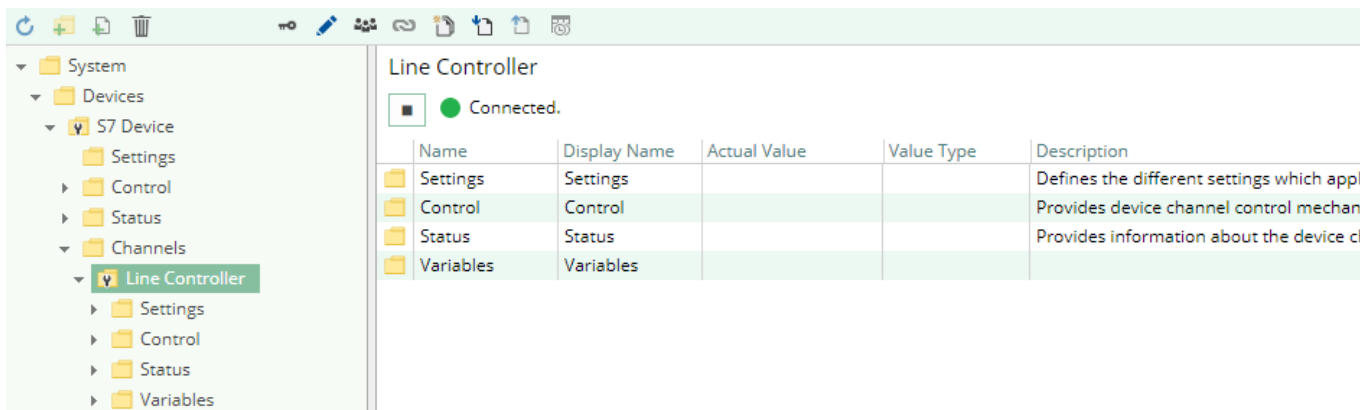


# Diagnostics

The S7 Device Plugin provides different status information depending on the layer to inspect. In general the channel based diagnostic information is produced by the connection status of the channel to the S7 PLC. The variable based diagnostic information is produced during the read/write access of the different variables.

## Channel

To monitor and diagnose the status of a S7 Channel, take a look at the following image:



The image above depicts the S Channel's Control Panel which displays all status relevant information. The control panel will automatically update its status information when a new status is available.

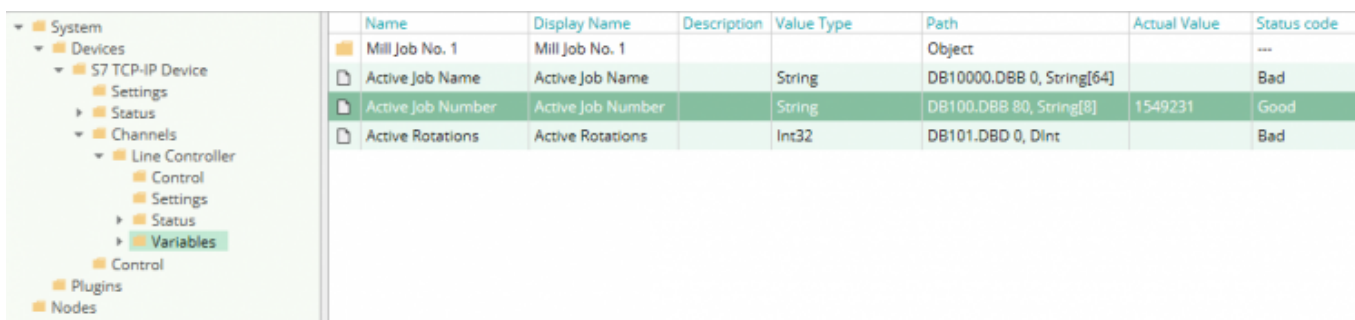
### Status Circle

Color	Meaning
Grey	The channel is stopped. Click the ▶ button to start it.
Yellow	The channel is currently in the progress of starting or stopping or is waiting to establish a connection.
Green	The channel is running and a connection has successfully been established. You can stop it by clicking the ■ button.
Red	The channel is running, but the connection is currently in an error state. Please check the status text for more information.

## Variables

To monitor and diagnose the status of the different S7 variables take a look at the variable's **Status Code** property displayed in UKI-4.0 ®.

In case the **Status Code** column displays the value **Bad** in most cases the addressed data area is not accessible.



# Log File

All device channel related status information is also logged into the channel-specific log file stored in the [\[LoggingFolder\]](#). Each log file is named in the naming scheme [S7 Device.<ChannelName>.log](#). The content of such a log file can look as follows:

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

Using the sample channel the name of the log file would be: [S7 Device.Line Controller.log](#)

## Status Codes

The following table displays the different status information possible:

Code	Category	Severity	Text
-88	Error	High	The operation has been canceled.
-34	Error	High	The requested PLC block could not be found.
-33	Error	High	The PLC does not support bit based operations.
-32	Error	High	The PLC supplied truncated data.
-31	Error	High	The requested PLC or PC type of data can not be transformed from one to the other.
-30	Error	High	The requested PLC or PC type of data is not available.
-21	Error	High	A connection to the device has already been established.
-20	Error	High	The size of the supplied buffer is lower than the amount of data available.
-11	Error	High	The type or format of data supplied is not supported.
-10	Error	High	The supplied access mode is not supported or unknown.
-9	Error	High	The operation failed upon a value range error.
-8	Error	High	Failed to allocate required memory.
-7	Error	High	The operation failed upon a socket error.
-6	Error	High	The specified CPU could not be found.
-5	Error	High	A general error occurred.
-2	Error	High	The necessary memory could not be allocated (out of memory).
-1	Error	High	The operation has timed out.
0	Information	Moderate	The operation completed without any kind of error.
1	Information	Moderate	The operation completed successfully.
2	Information	Moderate	The addressed data area does not exist.

# Entities

As each device plugin the S7 device plugin extends the basic UKI-4.0 [Device Model](#).

## Device

The plugin's device type [S7Device](#) also defines the [S7DeviceChannel](#) and therefore extends the basic UKI-4.0 [Device](#) and UKI-4.0 [DeviceChannel](#) entities. While the [S7Device](#) just represents a concretization of the UKI-4.0 [Device](#), the [S7DeviceChannel](#) extends the UKI-4.0 [DeviceChannel](#) with the S7 Variable Entities.

## Channel

Each channel is handled by a Channel Worker which establishes a physical connection to the S7 device.

For diagnostic purposes, the worker automatically reads the PLC address “MB 0” every 10 seconds to update the status code of the **Channel** and description to detect connection failures.

By default, the worker does not read any values. When a client or plugin requests a synchronous read of the **Channel** variables in UKI-4.0 (e.g. using the UKI-4.0 Web Configuration's function “Read actual value”), the channel worker reads them from the underlying S7 device and then writes them into the corresponding UKI-4.0 Nodes.

Similarly, when a Client or plugin writes values into the channel's variables, the channel worker will write those values to the underlying S7 device.

To have an S7 variable being read steadily, you can edit the Node in the UKI-4.0 Web Configuration and set “History Options” to **Yes** (which will create an internal subscription), or you can use e.g. a OPC UA Client connected to the OPC UA Server plugin and create a subscription for the S7 variable Nodes. In these cases, the channel worker reads the variables from the S7 at a regular interval and, if the value of one of the variables has changed, writes the new value into the corresponding UKI-4.0 Node.

## Variable

Each S7 variable can access the same PLC memory using the same PLC address operand. However, its interpretation depends on the PLC Data Type selection. Supported variable formats are scalar, array and object variables. Variables of the type **Object** can own further variables and are defined as **UserDefinedTypes** (UDT in STEP7).

# Folders & Files

## Folders

Content	Path	Usage
AssemblyFolder	<UKI-4.0 InstallDir>/plugins/S7DevicePlugin/	Contains the plugin's assembly file.
ConfigFolder	<UKI-4.0 DataDir>/plugins/S7DevicePlugin/	Contains the plugin's configuration file.
LoggingFolder	<UKI-4.0 DataDir>/log/	Contains the plugin's log files.

## Files

Type	Path	Usage
Assembly	[AssemblyFolder]/UKI-4.0 .S7DevicePlugin.dll	The plugin's assembly file.
Config App	<UKI-4.0 InstallDir>/UKI-4.0 .S7DevicePlugin.Configurator.exe	The config application file.
Config File	[ConfigFolder]/UKI-4.0 .S7DevicePlugin.<ChannelName>.Settings.xml	The optional channel config file.
Logging	[LoggingFolder]/S7 Device.<ChannelName>.log	The log file.

# About Versions

## This Document

<b>Date</b>	2020-12-01
<b>Version</b>	1.3

## Plugin

<b>Name</b>	S7 Device Plugin
-------------	------------------

<b>Node</b>	/System/Devices/S7 Device
<b>Version</b>	1.2.0

### Assembly

<b>Name</b>	UKI-4.0 .S7DevicePlugin.dll
<b>Date</b>	2020-12-01
<b>Version</b>	1.2.0.0

# Exchange Plugins

All exchange plugins make use of the UKI-4.0® Exchange Model. Each storage engine provided is registered and administrated by the UKI-4.0® Exchange Manager.

Such storage engines can be:

- Database management systems like
  - Microsoft SQL Server
  - MySQL
  - Oracle
- Database files like
  - Excel
  - CSV (see CSV Exchange Plugin)

## Exchange Model

The UKI-4.0® Exchange Model extends the basic UKI-4.0® Entity Model with entities typical for the exchange. Hereby an exchange entity defines the subordinated entities for control, settings, the status of the plugin and the various channels via which the plugin connects the supported storage engines to UKI-4.0®.

## Configuration

Each exchange plugin delivered with UKI-4.0® is directly and exclusively configurable in the UKI-4.0® Host application. If a plugin has configuration parameters, those can be modified in the according exchange entity.

## Using UKI-4.0UKI-4.0 UKI-4.0®

The entire configuration of all exchange plugins can be found under the Node path [/System/Interfaces](#). This root Node of the exchange plugins allows the complete configuration of the exchange plugins, provided that one of the actively used exchange plugins supplies its own exchange entities for the configuration.

# CSV Exchange Plugin

## General

The CSV Exchange Plugin provides a data exchange mechanism between UKI-4.0<sup>®</sup> and CSV (comma-separated values) based file formats.

## What Does the Plugin Do?

The CSV Exchange Plugin defines a storage structure for CSV files. Within this structure one or more data sets can be defined. Each data set represents a binding between the columns in the CSV storage and the Nodes in UKI-4.0<sup>®</sup>. Through that binding the plugin reads and writes the UKI-4.0<sup>®</sup> Node values as a data set to the CSV file and vice versa.

## Features

- File change monitoring
- File truncation on completion
- File deletion on completion
- Custom CSV separator configuration
- Relative Node binding in DataSet
- Access files in the local file system or use an SSH-based transfer protocol (SCP or SFTP)

## Supported Contents

- Microsoft Excel CSV Separator (default)
- Custom CSV separators

## Purpose & Use Cases

### Purpose

The configured CSV files can be used simply to read and write data from and to UKI-4.0<sup>®</sup>. This allows simple data import/export scenarios using comma-separated values. Also other UKI-4.0<sup>®</sup> participants can deliver and receive information stored in a CSV file through the CSV Exchange Plugin.

### Use Cases

- Automatic data synchronization between UKI-4.0<sup>®</sup> and third party (sub)systems to transfer data
- Data import to archive process protocols
- Data export to post-process UKI-4.0<sup>®</sup>-produced and -stored information

# Installation

This plugin is part of the UKI-4.0® Setup. Please consult UKI-4.0 [Setup and First Start](#) for more information on how to install and uninstall this plugin.

## Requirements

- Basic requirements of UKI-4.0®
- Read/write access to the CSV files configured

# Configuration

## Using the Configuration File

### Structure

The CSV Exchange Plugin defines the root of its element tree by the `PluginSettings` element as described in [Plugin Configuration - Using a Configuration File](#) and continues its XML tree using the `Servers` element.

### Servers Element

The `Servers` element serves as the container for one or more `Server` elements. This element maintains all file Servers associated with the exchange plugin.

The `Servers` element can look as follows:

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

### Server Element

The `Server` element serves as the container for the element `File`. One server identifies one CSV file to which the Server configuration belongs. The configuration also includes the actions to perform before, during and after the synchronization progress. This information is therefore used by the CSV Exchange Plugin to exchange CSV data with UKI-4.0®.

Each `Server` element provides the following list of attributes:

	Mandatory	Type	Purpose
<code>IsEnabled</code>	no	Boolean	Shows the value <code>true</code> if the data exchange is active for the file Server; otherwise shows the value <code>false</code> (default: <code>true</code> ).
<code>SyncMode</code>	no	CSV Sync Mode	The direction into which the synchronization is to be performed.
<b>Additional attributes for <code>SyncMode="FileToSystem"</code></b>			
<code>SyncTrigger</code>	no	CSV Sync Trigger	The trigger condition to initiate the synchronization.
<code>BeforeSyncAction</code>	no	CSV Sync Action	The action to perform, before the synchronization progress begins.
<code>AfterSyncAction</code>	no	CSV Sync Action	The action to perform, after the synchronization progress has ended.

	Mandatory	Type	Purpose
<b>BeforeSyncMoveFileTo</b> <b>(if BeforeSyncAction="MoveFile")</b>	yes	Node Query Expression	Specifies the new fully qualified file path or directory path to which the file is moved. If you specify a directory path (ending with / or \), the file name is kept when moving the file to the new directory. In that case, if a file with the same name already exists, the plugin will append a suffix like (1), (2) etc.
<b>AfterSyncMoveFileTo</b> <b>(if AfterSyncAction="MoveFile")</b>	yes	Node Query Expression	See above.

### CSV Sync Mode

The following values are valid for attributes of that type:

Value	Description
"FileToSystem"	The CSV file is synchronized to the bound UKI-4.0 ® Nodes.
"SystemToFile"	The bound UKI-4.0 ® Nodes are synchronized to the CSV file.

### CSV Sync Trigger

The following values are valid for attributes of that type:

Value	Description
"FileChanged"	The synchronization progress is to be triggered whenever the file gets changed. This means that the synchronization is invoked everytime as soon as the date changes at which the file was last written.

### CSV Sync Action

The following values are valid for attributes of that type:

Value	Description
"KeepFile"	The file is to be kept before/after synchronization.
"TruncateFile"	The files content is to be removed before/after synchronization.
"DeleteFile"	The file is to be deleted before/after synchronization.
"MoveFile"	The file is to be moved or renamed before/after synchronization. For this action, you will have to specify the BeforeSyncMoveFileTo or AfterSyncMoveFileTo attributes (see above).

The **Server** element can look as follows:

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

## Trigger-Element

The **Trigger** element allows to define a trigger (if SyncMode="SystemToFile"), which specifies when the data should be collected from UKI-4.0 ® by doing a synchronous read and then be written into the CSV file.

The **Trigger** element provides the following list of attributes:



	Mandatory	Type	Purpose
<b>Type</b>	yes	String	Specifies the type of the trigger. "ValueChange": The trigger fires when a new value is written to a specified node and the new value is different from the previous value of the node. "Edge": The trigger fires when a specific values is written to the node (specified by the <b>EdegValue</b> attribute) and the previous value was different. Optionally, you can specify a value that is to be written back to the node after synchronization. "Interval": The trigger fires in a regular interval (specified by the <b>interval</b> attribute).
<b>Node</b> (if Type="Edge" or Type="ValueChange")	yes	String	Specifies the fully qualified path to the node which is to be monitored.
<b>EdgeValue</b> (if Type="Edge")	yes	String	Specifies the value for which the trigger is monitoring. The trigger fires when this value is written to the node while the previous value of the node was a different one.
<b>ChangeBackValue</b> (if Type="Edge")	no	String	If specified, the CSV Exchange plugin will write this value into the node after the trigger fired and the dataset has been collected with an synchronous read.
<b>ChangeBackNode</b> (if Type="Edge")	no	String	If specified and if <b>ChangeBackValue</b> is specified, the CSV Exchange Plugin will write the value to the node specified by this attribute, instead of the node specified by the <b>Node</b> attribute.
<b>Interval</b> (if Type="Interval")	yes	Int32	Specifies the trigger interval in milliseconds. For example, a value of 2000 means that the trigger fires every two seconds.

The **Trigger** element can look as follows:


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

### File Element

The **File** element serves as the container for the element **Bindings** and defines the attributes to set up the file used for the exchange. These attributes configure the file and its format used to synchronize.

The **File** element provides the following list of attributes:

	Mandatory	Type	Purpose
<b>Path</b>	yes	Node Query Expression	The fully qualified accessible path to the CSV file to synchronize.  When using SyncMode <b>SystemToFile</b> , the nodes specified in the Node Query Expression will be included in the synchronous read of the column nodes, and then evaluated when writing the file. When using SyncMode <b>FileToSystem</b> , the specified nodes will be evaluated once when starting the channel, and additionally you can specify a filter like <b>XY*.csv</b> , which means the plugin will monitor and process all files in the directory matching that filter.  <b>Note:</b> For files accessed using the local file system, this path is subject to the <b>File Access Security</b> restrictions that have been defined in the UKI-4.0 ® <b>Settings</b> .
<b>Separator</b>	no	Char	The CSV separator in the file used to separate the values from each other (default: <b>;</b> ).

	Mandatory	Type	Purpose
<b>HasHeader</b>	no	Bool	Specifies if the first line of the CSV file contains a title for each column. Valid values: <b>True</b> , <b>False</b>  When specifying <b>True</b> , for SyncMode <b>FileToSystem</b> this means that the first line of the CSV file will be skipped when synchronizing it. For SyncMode <b>SystemToFile</b> , this means that if the CSV file is empty when synchronizing it, a header line will first be written with a title for each column that can be specified in the <b>&lt;Binding&gt;</b> elements using the <b>HeaderName</b> attribute.
<b>Type</b>	no	String	Specifies the type of the file transfer mechanism. <b>File</b> (default): The file is accessed using the local file system. <b>Scp</b> : The file is accessed via an SSH Server (SCP protocol). <b>Sftp</b> : The file is accessed via an SSH Server (SFTP protocol).
Additional attributes, when <b>Type="Scp"</b> or <b>Type="Sftp"</b> :			
<b>Hostname</b>	yes	String	The hostname of the SSH Server.
<b>Username</b>	yes	String	The username for the SSH Server.
<b>Password</b>	yes	String	The <b>encrypted password</b> for the SSH Server. To get the encrypted password, open UKI-4.0 , and in the Web Configuration click on  <b>Password Security</b> . Here, you can enter your original password and encrypt it, so that it can be specified for this attribute.

The **File** element can look as follows:

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

or, when using SCP:

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

## Bindings Element

The **Bindings** element serves as the container for one or more **Binding** elements. This element maintains all bindings associated with the file.

The **Bindings** element provides the following list of attributes:

	Mandatory	Type	Purpose
<b>BaseNode</b>	no	String	The Node path used as the base Node path for all subsequent Node bindings.

The **Bindings** element can look as follows:

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

## Binding Element

The **Binding** element binds a column in the CSV file to a UKI-4.0 ® Node. This information is therefore

used by the CSV Exchange Plugin to link the CSV comma-separated values to UKI-4.0 ® Nodes.

Each **Binding** element provides the following list of attributes:

	<b>Mandatory</b>	<b>Type</b>	<b>Purpose</b>
<b>ColumnIndex</b>	<b>yes</b>	Int32	The zero-based (index of the first column is zero) index of the CSV column to bind.
<b>Node</b>	<b>yes</b> (if <b>SystemValueType</b> is not specified)	String	The (relative, if <b>BaseNode</b> is used) Node path of the Node to bind.
<b>ValueType</b> (if <b>Node</b> is specified)	no	Value Type	If specified, forces the column value to be interpreted as the specified type. You can specify the formatting using the <b>ValueFormat</b> attribute.
<b>ValueFormat</b> (if <b>ValueType</b> is specified)	no	String	Specifies the formatting for the value.
<b>SystemValueType</b>	<b>yes</b> (if <b>Node</b> is not specified)	System Value Type	If specified, instead of using the value of a node, a system value is used for this column. See the table below for the possible system value types.
<b>SystemValueFormat</b> (if <b>SystemValueType</b> is specified)	no	String	Specifies the format of the system value (e.g. date format).
<b>HeaderName</b>	no	<b>Node Query Expression</b>	When <b>HasHeader</b> is set to <b>True</b> on the <b>&lt;File&gt;</b> element, specifies the title of the column. If not specified, a default name like "Column1" will be used.

### Value Type

The following values are valid for attributes of that type:

<b>Value</b>	<b>Description</b>
"DateTime"	The column value is to be interpreted as a DateTime value. You can specify the formatting using a .NET date/time format string ( <b>standard</b> or <b>custom</b> ), e.g. <b>yyyy-MM-dd HH:mm:ss</b> .
"TimeSpan"	The column value is to be interpreted as a TimeSpan value. You can specify the formatting using a .NET TimeSpan format string ( <b>standard</b> or <b>custom</b> ), e.g. <b>dd\ .hh\ :mm\ :ss</b> .

### System Value Type

The following values are valid for attributes of that type:

<b>Value</b>	<b>Description</b>
"TriggerTimestamp" (if <b>SyncMode</b> ="SystemToFile")	The column will contain the timestamp when the trigger has fired. You can specify the formatting using a .NET date/time format string ( <b>standard</b> or <b>custom</b> ), e.g. <b>yyyy-MM-dd HH:mm:ss</b> .
"CreationTimestamp" (if <b>SyncMode</b> ="FileToSystem")	The timestamp specified in the column will be used as CreationTimestamp for the node values that are to be written to UKI-4.0 . You can specify the formatting using a .NET date/time format string ( <b>standard</b> or <b>custom</b> ), e.g. <b>yyyy-MM-dd HH:mm:ss</b> .

The **Binding** element can look like as follows:

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

## Node Query Expression

Certain attributes (e.g. file path or column header) allow to specify a node path, which means that part will be replaced with the value of the node. A Query Expression has the form `${<Node-Path>|<ID>|<Guid>}` (the node path itself must not contain a `}` character). If you want to use a dollar sign (\$) directly, you have to escape it as two dollar signs (`$$`).

When evaluating an expression, a synchronous read is initiated. This allows you to provide the value with a script that registers a UKI-4.0 `.NodeReader` for the specified node.

Examples (assuming node `/Nodes/A` has value "First", node `/Nodes/B` has value "Second"):

Node Query Expression	Resulting output/file path
<code>C:\File-\${/Nodes/A}.csv</code>	<code>C:\File-First.csv</code>
<code>\$\$Header_\${/Nodes/A}\${/Nodes/B}\$\$</code>	<code>\$Header_FirstSecond\$</code>

## Usage

It is recommended to use a professional XML editor when editing the configuration file manually. To also benefit of the XML scheme definition mentioned at the end of this article you have to refer to the schema definition by using the `xsi:noNamespaceSchemaLocation` attribute in the document root `PluginSettings` as follows (the XSD file needs to be placed besides the XML file):

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

Independent from the technique of editing/creating the configuration file manually or automated the element tree documented above is to be fulfilled to produce a valid, well-formed and usable configuration file.

## Synchronization

As soon as the plugin is loaded and started by the UKI-4.0® Plugin Manager its configuration file is read and synchronized by the plugin with the appropriate UKI-4.0® entities.

In case the configuration file changes the plugin manager notifies the plugin. Through that notification the plugin restarts and uses the latest configuration changes on startup.

## Example Configuration File

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-HHmss"/>
        </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-HHmss" HeaderName="My Column 3"/>
        </Bindings>
      </File>
    </Server>
  </Servers>
</PluginSettings>

```

## Example Configuration Scheme File

### 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:restriction>
  </xs:simpleType>

```

```

    <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>

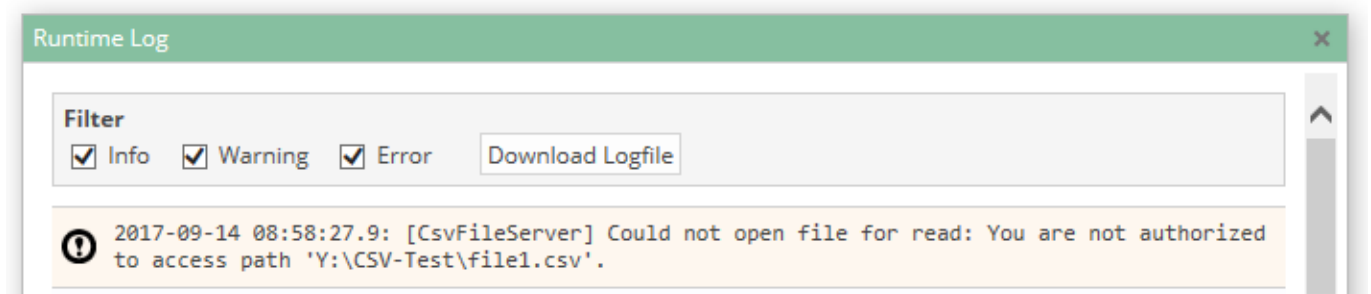
<xs:complexType name="ServersType">
  <xs:sequence>
    <xs:element name="Server" type="ServerType" minOccurs="0" maxOccurs="unbounded"
/>
  </xs:sequence>
</xs:complexType>

<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>
```

## Diagnostics

The CSV Exchange Plugin will log events that start with `[CsvFileServerPlugin]` or `[CsvFileServer]` into the UKI-4.0<sup>®</sup> runtime log. You can open the runtime log in the Web Configuration to view the last log entries:



## Entities

The CSV Exchange Plugin does not use the UKI-4.0<sup>®</sup> Entity Model because it is configured by an XML Configuration File (UKI-4.0 `.CsvExchangePlugin.Settings.xml`) and therefore does not provide entities.



# Folders & Files

## Folders

Content	Path	Usage
AssemblyFolder	<UKI-4.0 InstallDir>/plugins/CsvExchangePlugin/	Contains the plugin's assembly file.
ConfigFolder	<UKI-4.0 DataDir>/plugins/CsvExchangePlugin/	Contains the plugin's configuration file.
LoggingFolder	<UKI-4.0 DataDir>/log/	Contains the plugin's log files.

## Files

Type	Path	Usage
Assembly	[AssemblyFolder]/UKI-4.0 .CsvExchangePlugin.dll	The plugin's assembly file.
Config File	[ConfigFolder]/UKI-4.0 .CsvExchangePlugin.Settings.xml	The config file.

# About Versions

## This Document

<b>Date</b>	2019-04-17
<b>Version</b>	1.4

## Plugin

<b>Name</b>	CSV File Server
<b>Node</b>	/System/Plugins/Exchange/CSV
<b>Version</b>	1.1.0

## Assembly

<b>Name</b>	UKI-4.0 .CsvExchangePlugin.dll
<b>Date</b>	2019-04-16
<b>Version</b>	1.1.0.0

# Database Plugin

## General

The Database Plugin allows you to write UKI-4.0<sup>®</sup> Datapoint Node values to external databases, like MySQL or Microsoft SQL Server.

## What Does the Plugin Do?

The plugin allows you to:

- define database connections
- define triggers (e.g. interval trigger)
- define DataSets, consisting of
  - datapoint Nodes
  - system values

When a trigger fires, the plugin collects values for a “record set” by doing a [Synchronous Read](#) of specified UKI-4.0<sup>®</sup> datapoint Nodes (which means UKI-4.0<sup>®</sup> requests the underlying device to actually read its variables) and then writes the values of the record set to a database table.

To use the plugin, you need to create an XML configuration file.

## Features

- Collect node values and write them into a database when a trigger fires
- Supports multiple trigger types:
  - Interval trigger
  - Edge value trigger (fires when the value of a node changes to a specified value)
  - Value change trigger (fires when the value of a node changes)

## Supported Database Engines

- MySQL 5.5 or higher
- Microsoft SQL Server 2008 or higher

## Purpose & Use Cases

- mirror data from UKI-4.0<sup>®</sup> to an external database
- collect data from devices once a trigger fires

# Installation

This plugin is part of the UKI-4.0® Setup. Please consult UKI-4.0® [Setup and First Start](#) for more information on how to install and uninstall this plugin.

## Requirements

- The machine which runs UKI-4.0® must have access to one of the supported database engines.

# Configuration

This plugin can be only configured by using the **XML Configuration File** as described below. You will need to create the XML Configuration File ("UKI-4.0 .DatabasePlugin.Settings.xml") in the project directory (see [Folders & Files](#)). When the file is changed while UKI-4.0 is running, the Database Plugin will automatically restart and use the new configuration file.

## Structure

### PluginSettings Element

Each UKI-4.0® plugin defines the root of its element tree by the "PluginSettings" element as follows:

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

### Channels Element

The **Channels** element serves as a container for one or more **Channel** elements. The **Channel** element defines a group of database connections, triggers and datasets. For each channel there is a worker thread that handles a trigger by collecting Node values and then writing them to the specified database connections.

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

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

Each **Channel** element provides the following list of attributes:

Attribute	Mandatory	Type	Purpose
<b>id</b>	yes	String	Uniquely identifies this channel. This value can be referenced e.g. in one of the <b>SystemValue</b> elements.
<b>active</b>	no	Boolean	Determines if this channel is actually run. If "false", the channel is not started, otherwise it is.

### DbConnections Element

The **DbConnections** element serves as a container for one or more **DbConnection** elements.

The **DbConnection** element defines a physical connection to a database, including the table name. This determines to which database table is written.

```
<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>
```

Each **DbConnection** element provides the following list of attributes:

Attribute	Mandatory	Type	Purpose
<b>id</b>	yes	String	Uniquely identifies this database connection. This is used in the <b>DataSet</b> elements to refer to this connection.
<b>type</b>	yes	Enumeration	Specifies the type of the database engine. <b>"MSSQL"</b> : Microsoft SQL Server <b>"MySQL"</b> : MySQL Server
<b>hostname</b>	yes	String	The hostname, DNS name or IP address of the database Server.
<b>port</b>	yes	Integer	The TCP port of the database server. When using <b>MSSQL</b> as Server Type, you can specify <b>0</b> as port, which means a connection to the default SQL Server instance shall be established.
<b>username</b>	yes	String	The username which is used to establish the database connection.
<b>password</b>	yes	String	The <b>encrypted password</b> for the username. To get the encrypted password, open UKI-4.0 , and in the Web Configuration click on <b>🔒 Password Security</b> . Here, you can enter your original password and encrypt it, so that it can be specified for this attribute.
<b>database</b>	yes	String	The database / scheme name.
<b>table</b>	yes	String	The name of the database table to which the values shall be written.
<b>connectTimeout</b>	no	Integer	The timeout in seconds to use when connecting to the database. If not specified, the value 30 is used.
<b>commandTimeout</b>	no	Integer	The timeout in seconds to use when running a database command. If not specified, the value 60 is used.
<b>maxPoolSize</b>	no	Integer	The maximum number of active database connections that are pooled. If not specified, the value 40 is used.

**Note:** Currently, the Database Plugin always writes values to a database table by using an **INSERT** statement (meaning new values are appended to the table at each write).

### Triggers Element

The **Triggers** element serves as a container for one or more **Trigger** elements.

The **Trigger** element defines an object which fires an event. A trigger can be referenced in a **DataSet** element to determine when the Database Plugin should collect and write Node values to the database.

```
<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>
```

Each **Trigger** element provides the following list of attributes:

Attribute	Mandatory	Type	Purpose
<b>id</b>	yes	String	Uniquely identifies this trigger. This is used in the <b>DataSet</b> elements to refer to this trigger.
<b>type</b>	yes	Enumeration	Specifies the type of the trigger. <b>"interval"</b> : Fires at a specific regular interval, specified by the <b>interval</b> attribute. <b>"edge"</b> : Fires when a specific value is written into the specified Node (and the previous Node value was a different value). <b>"valueChange"</b> : Fires when a value of the specified Node changes.
<b>interval</b> (when <b>type="interval"</b> )	yes	Integer	Specifies the interval of the trigger in milliseconds. For example, a value of <b>2000</b> means the trigger fires every two seconds.
<b>node</b> (when <b>type="edge"</b> or <b>type="valueChange"</b> )	yes	String	Specifies the Node path of the Node which the trigger should use to check for value changes.
<b>edgeValue</b> (when <b>type="edge"</b> )	yes	String	The value for which the trigger checks. The trigger fires when this value is written to a Node and the previous Node value was a different one.
<b>changeBackValue</b> (when <b>type="edge"</b> )	no	String	If specified, the Database Plugin will write this value into the Node after the trigger has fired and the record set has been collected using an synchronous read.
<b>instantChangeBackValue</b> (when <b>type="edge"</b> )	no	String	If specified, the Database Plugin will write this value into the Node immediately after the trigger has fired (before the record set has been collected).

### DataSets Element

The **DataSets** element serves as a container for one or more **DataSet** elements.

The **DataSet** element defines a group of datapoint Nodes and SystemValues which will be collected into a record set and then written to one or more database connections. In a **DataSet**, you can reference one or more **DbConnection** elements (meaning that a **DataSet** is written to all these database connections) and one or more triggers (meaning the Node values are collected and written when one of these triggers fires).

```

<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>
    
```

Each **DataSet** element provides the following list of attributes:

Attribute	Mandatory	Type	Purpose
<b>id</b>	<b>yes</b>	String	Uniquely identifies this DataSet.
<b>writeDelay</b>	no	Integer	Specifies the maximum delay (in ms) after the cached record sets are written to the database.
<b>writeBufSize</b>	no	Integer	Specifies the maximum number of cached record sets until they are written to the database.

For each DataSet, after a set of values has been collected (a record set), the record set is buffered to efficiently handle a number of record sets to write them in one go. If either the **writeDelay** or **writeBufSize** is specified (or both), the buffered record sets are not written to the database until the **writeDelay** time has passed since the last write, or the number of cached record sets specified by **writeBufSize** has been exceeded. If none of these attributes are specified, the record sets are written immediately.

Each **DbConnection** element and **Trigger** element provides the following list of attributes (in the **DataSet** element):

Attribute	Mandatory	Type	Purpose
<b>id</b>	<b>yes</b>	String	References a previously defined DbConnection or Trigger.

### Nodes Element

The **Nodes** element serves as a container for one or more **Node** elements.

The **Node** element references a UKI-4.0 ® datapoint Node using either an absolute or relative Node path. If the **Nodes** element specifies a path in its **root** attribute, the **path** of the **Node** elements is relative to the **Nodes**'s **root** path.

Example with root path:

```

<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>
    
```

Example without root path:

```
<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>
```

Each **Nodes** element provides the following list of attributes:

Attribute	Mandatory	Type	Purpose
<b>root</b>	no	String	Specifies the path to the parent Node. If specified, the path of the <b>Node</b> elements is relative to this root path. Otherwise, the path of the <b>Node</b> elements must be the complete path.

Each **Node** element provides the following list of attributes:

Attribute	Mandatory	Type	Purpose
<b>path</b>	<b>yes</b>	String	The relative path from <b>root</b> to the Node, or the absolute path if the <b>root</b> of the <b>Nodes</b> element is not specified.
<b>column</b>	<b>yes</b>	String	The name of the database table column in which the value shall be written.
<b>property</b>	no (default: <b>value</b> )	String	The property of the Node which shall be written. <b>"value"</b> (default): The Node's value. <b>"timestamp"</b> : The timestamp of the Node's value (UTC). <b>"timestampLocal"</b> : The timestamp of the Node's value (local time). <b>"nodeID"</b> : The identifier (local identifier) of the Node. <b>"nodeName"</b> : The name of the Node. <b>"nodeDisplayName"</b> : The display name of the Node. <b>"nodeUnit"</b> : The unit of the Node.

### SystemValues

The **SystemValues** element serves as a container for one or more **SystemValue** elements.

The **SystemValue** element either contains a literal value or references a predefined system value, which should be written to the database.

```
<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>
```

Each **SystemValue** element provides the following list of attributes:

Attribute	Mandatory	Type	Purpose
<b>column</b>	<b>yes</b>	String	The name of the database table column in which the value shall be written.
<b>type</b>	<b>yes</b> (if <b>value</b> is not specified)	Enumeration	Specifies which system value to use. <b>"triggerTimestamp"</b> : The timestamp when the trigger has fired (UTC). <b>"triggerTimestampLocal"</b> : The timestamp when the trigger has fired (local time). <b>"channelID"</b> : The identifier of the containing channel.

Attribute	Mandatory	Type	Purpose
<b>value</b>	<b>yes</b> (if <b>type</b> is not specified)	String	Specifies a direct (literal) value which shall be written.

**Note:** Either the **type** or the **value** attribute must be specified, but not both.

### AfterSyncActions Element

The **AfterSyncActions** element serves as a container for one or more **AfterSyncAction** elements.

The **AfterSyncAction** element allows to write a value to a Node after the record set has been collected.

```
<AfterSyncActions>
  <AfterSyncAction type="writeNodeValue" node="/Path/to/Node" value="MyValue" />
</AfterSyncActions>
```

Each **AfterSyncAction** element provides the following list of attributes:

Attribute	Mandatory	Type	Purpose
<b>type</b>	<b>yes</b>	Enumeration	Specifies the type of the AfterSyncAction. <b>"writeNodeValue"</b> : After collecting the record set, the specified value is written to the Node.
<b>node</b>	<b>yes</b>	String	Specifies the Node path of the Node to write the value.
<b>value</b>	<b>yes</b>	String	The value to write.

### Example Configuration File

The following is an example configuration file, using the elements as shown above:

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>

```

## Diagnostics

The Database Plugin writes the following events into the log file `[LoggingFolder]\logfile.txt` (see [Folders & Files](#) path of the log file):

- Channels are started or stopped (because UKI-4.0 ® is starting or stopping, or the configuration file has changed and the plugin restarts).
- A set of values has been successfully written to the database (after doing a synchronous read).
- An error occurred when trying to write a set of values to the database.

## Entities

The Database Plugin does not use the UKI-4.0 ® Entity Model because it is configured by an XML Configuration File (UKI-4.0 `.DatabasePlugin.Settings.xml`) and therefore does not provide entities.

# Folders & Files

## Folders

Content	Path	Usage
<b>AssemblyFolder</b>	<UKI-4.0 InstallDir>/plugins/DatabasePlugin/	Contains the plugin's assembly file.
<b>ConfigFolder</b>	<UKI-4.0 DataDir>/plugins/DatabasePlugin/	Contains the plugin's configuration file.
<b>LoggingFolder</b>	<UKI-4.0 DataDir>/plugins/DatabasePlugin/	Contains the plugin's log file.

## Files

Type	Path	Usage
<b>Assembly</b>	[AssemblyFolder]/UKI-4.0 .DatabasePlugin.dll	The plugin's assembly file.
<b>Config File</b>	[ConfigFolder]/UKI-4.0 .DatabasePlugin.Settings.xml	The config file.
<b>Logging</b>	[LoggingFolder]/logfile.txt	The log file.

# About Versions

## This Document

<b>Date</b>	2018-06-14
<b>Version</b>	1.5

## Plugin

<b>Name</b>	Database Plugin
<b>Version</b>	1.0.10

## Assembly

<b>Name</b>	UKI-4.0 .DatabasePlugin.dll
<b>Date</b>	2018-06-14
<b>Version</b>	1.0.10.0

# SQL Exchange Plugin

## General

The SQL Exchange Plugin allows you to bidirectionally exchange UKI-4.0® datapoint Node values with external databases, such as MySQL, Oracle or Microsoft SQL Server.

## What Does the Plugin Do?

The plugin allows you to:

- define databases (connections)
- browse or define tables
- browse or define table columns
- read and write columns, specifying a Read or Write SQL Expression

## Features

- Read columns of a database row (selected by the *Read SQL Expression*)
- Write columns of a database row by either inserting a new row or updating an existing row (selected by the *Write SQL Expression*)
- Subscribe to columns of a database row when the corresponding UKI-4.0® variable Nodes are subscribed to

## Supported Database Engines

- MySQL 5.5 or higher
- Microsoft SQL Server 2008 or higher
- Oracle
- SQLite

## Purpose & Use Cases

- Append a table row or update an existing row
- Read data from a table from the latest row or from a specific row
- Create subscriptions to get notified when data in the database is changed
- Call **stored procedures** using method nodes in UKI-4.0 (currently only supported for **Oracle** databases)

# Installation

This plugin is part of the UKI-4.0® Setup. Please consult UKI-4.0® [Setup and First Start](#) for more information on how to install and uninstall this plugin.

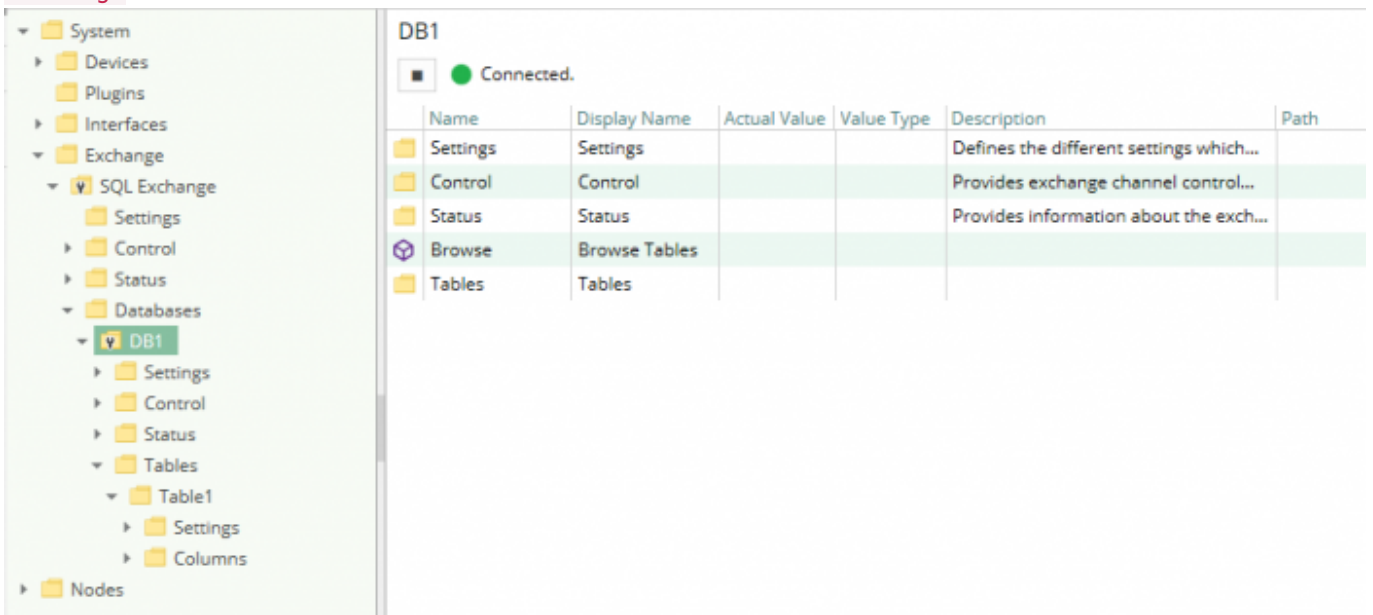
## Requirements

- The machine which runs UKI-4.0® must have access to one of the supported database engines.

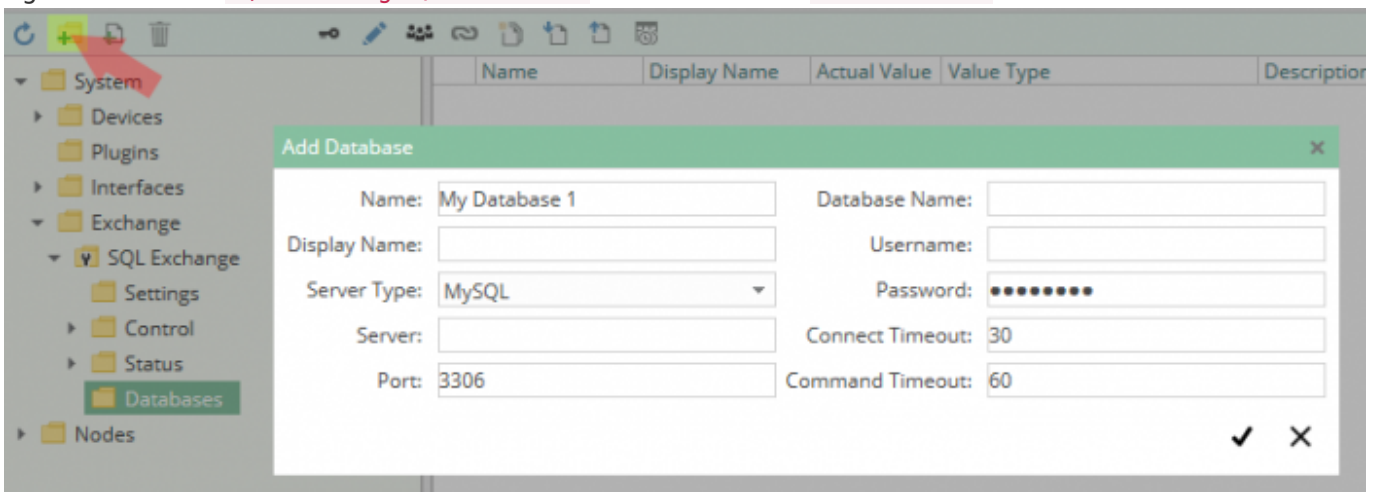
# Configuration

## Overview

The entire SQL Exchange Plugin configuration is located under the Node path [/System/Exchange/SQL Exchange](#).



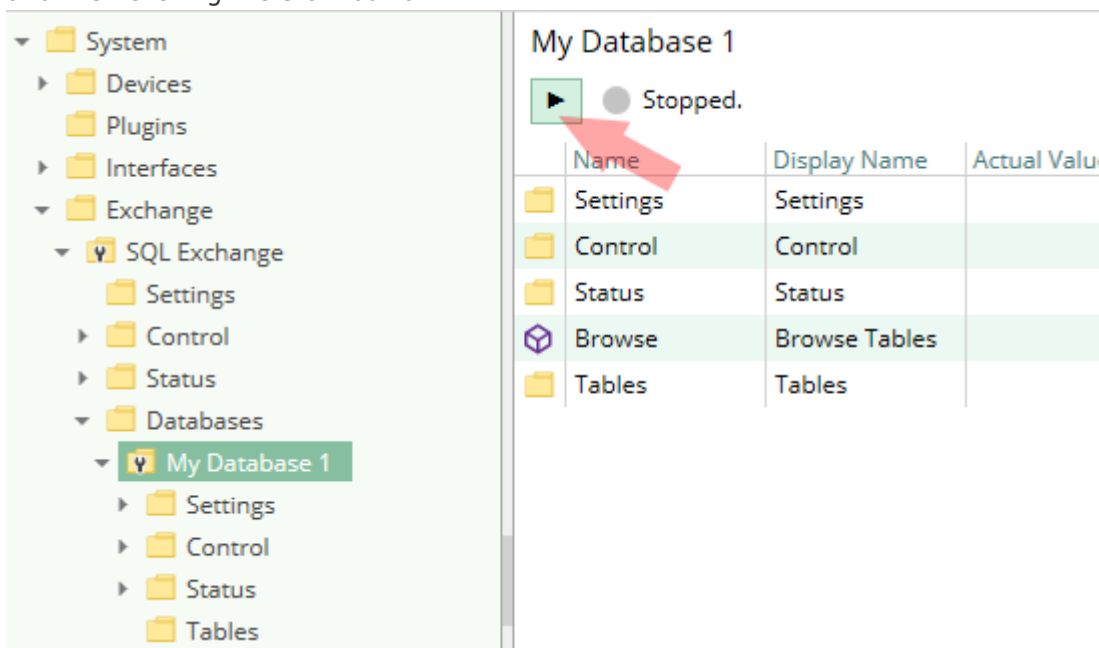
The Node tree in the image above depicts the SQL Exchange Plugin's default Node tree. To set up one or more SQL Exchange **Databases**, add a Folder Node beneath the Node [SQL Exchange / Databases](#), or right-click on the [SQL Exchange / Databases](#) Node and select [Add Database](#).



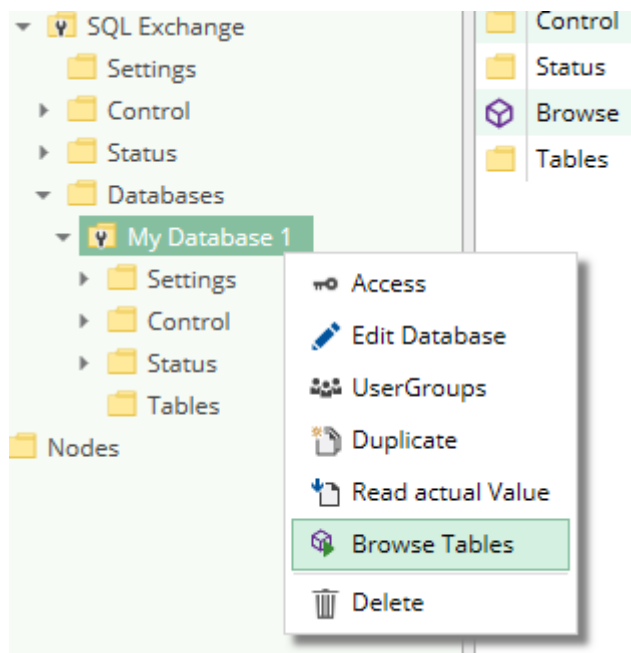
## Database-Specific Settings

Name	Type	Description
Server Type	Enum	Defines the server type to which the database connection is established. Valid values: <b>MySQL</b> , <b>Oracle</b> , <b>MSSQL</b> , <b>ODBC</b> , <b>SQLite</b>
Server	String	The hostname or IP address to which the connection shall be established. For <b>SQLite</b> , this is the file path to the database; other connection properties like <b>Port</b> will be ignored.
Port	Integer	The port to which the connection shall be established. When using <b>MSSQL</b> as Server Type, you can specify <b>0</b> as port, which means a connection to the default SQL Server instance shall be established.
Database Name	String	The name of the database/scheme. For <b>ODBC</b> , this is the data source name (DSN), and is the only field that needs to be specified. The DSN can be configured on Windows using the <b>ODBC Data Sources</b> app (User DSN or System DSN). Note that for UKI-4.0 (x86) you will need to use the 32-bit and for UKI-4.0 (x64) the 64-bit ODBC Data Sources app.
Username	String	The username for the connection.
Password	Password	The password.
Connect Timeout	Integer	The timeout in seconds after an attempt to connect is canceled.
Command Timeout	Integer	The timeout in seconds after a long-running command is canceled.

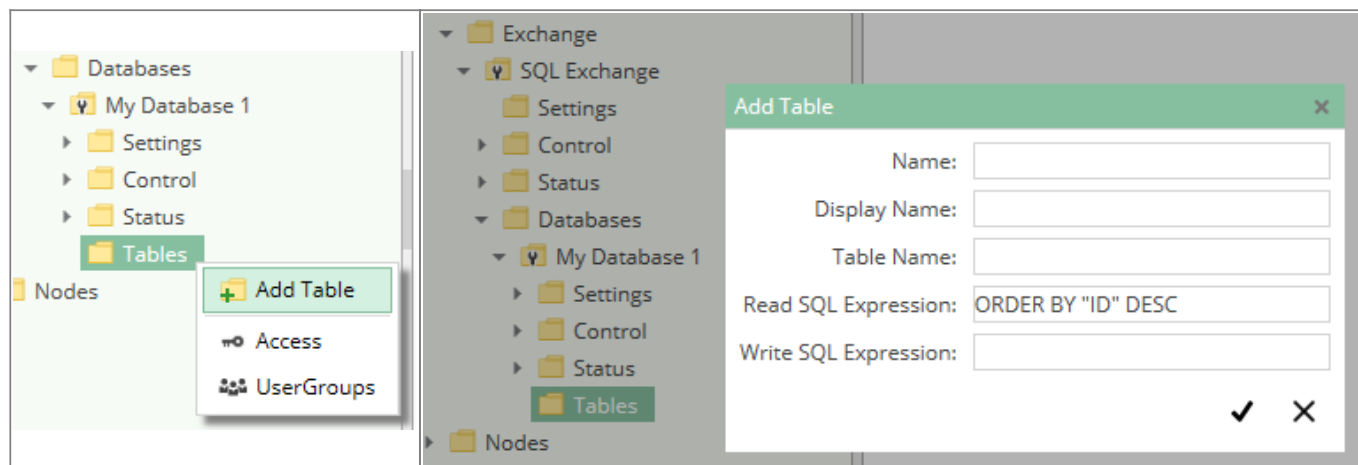
After clicking “Save”, the database Node will be created. You can start it by selecting the database Node and then clicking the Start button:



To setup one or more **tables** for the database, you can right-click on the database Node and then use the **Browse Tables** menu entry:



Alternatively, you can right-click on the **Tables** Node and select **Add Table** (to edit an existing table, right-click on the **Table** Node and select **Edit Table**):

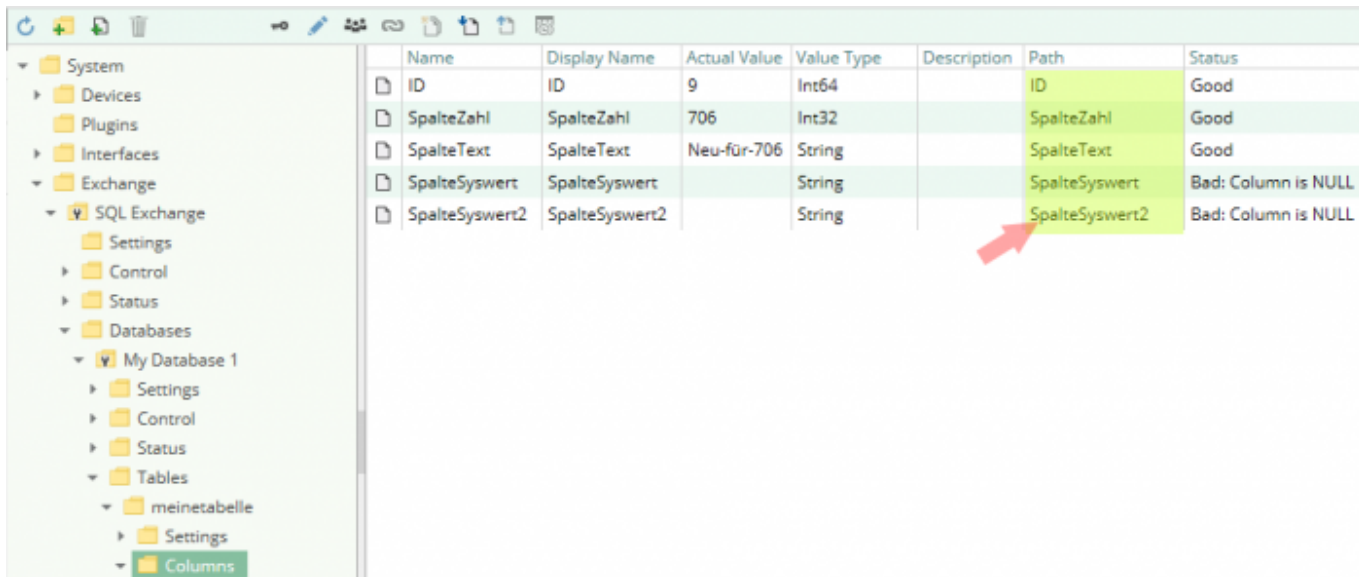


### Table-Specific Settings

Name	Type	Description
Table Name	String	The name of the table in the database.
Read SQL Expression	String	An SQL string (e.g. a <b>WHERE</b> clause or an <b>ORDER BY</b> clause) that is applied when reading the columns. The first row returned by the database is read. You can specify a <b>WHERE</b> clause (e.g. <b>WHERE "ID" = 123</b> ) to read only a specific column. The default is <b>ORDER BY "ID" DESC</b> to order the rows by ID descending, so that the row with the highest ID is used.
Write SQL Expression	String	An SQL String (e.g. a <b>WHERE</b> clause) that is applied when writing the columns. You can specify a <b>WHERE</b> clause (e.g. <b>WHERE "ID" = 123</b> ) to update an existing row. The default value is an empty string, which means a new row is inserted every time column Nodes are written.

### Columns

Every Node within the **Columns** Node of a table Node can be assigned to a column of the table, using the **Path** property.



**Syntax:**

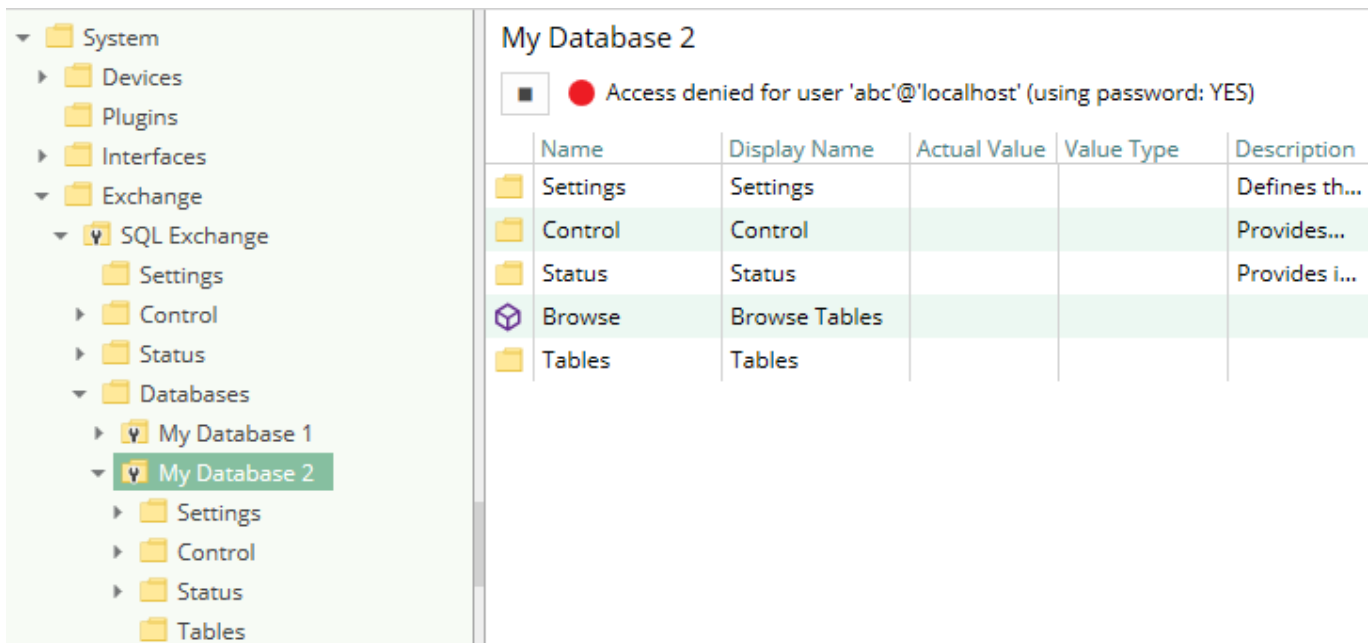
<code>&lt;ColumnName&gt;</code>	Only the column name is specified. When reading or writing from / to it, the table's Read SQL or Write SQL expressions are used. Example: <code>MyColumn1</code>
<code>&lt;ColumnName&gt;; &lt;ReadWriteExpression&gt;</code>	The column name is specified, along with an expression that is used when reading or writing the column. Example: <code>MyColumn1; WHERE ID = 123</code>
<code>&lt;ColumnName&gt;; &lt;ReadExpression&gt;; &lt;WriteExpression&gt;</code>	The column name is specified, along with an expression that is used when reading from the column, and a separate expression that is used when writing to the column. Example: <code>MyColumn1; WHERE ID = 123; WHERE ID = 456</code>

# Diagnostics

The SQL Exchange Plugin provides different status information depending on the layer to inspect. In general the channel-based diagnostic information is produced by the connection status of the channel to the database Server. The variable-based diagnostic information is produced during the read/write access of the different columns.

## Database

To monitor and diagnose the status of the different databases channels, take a look at the following image:



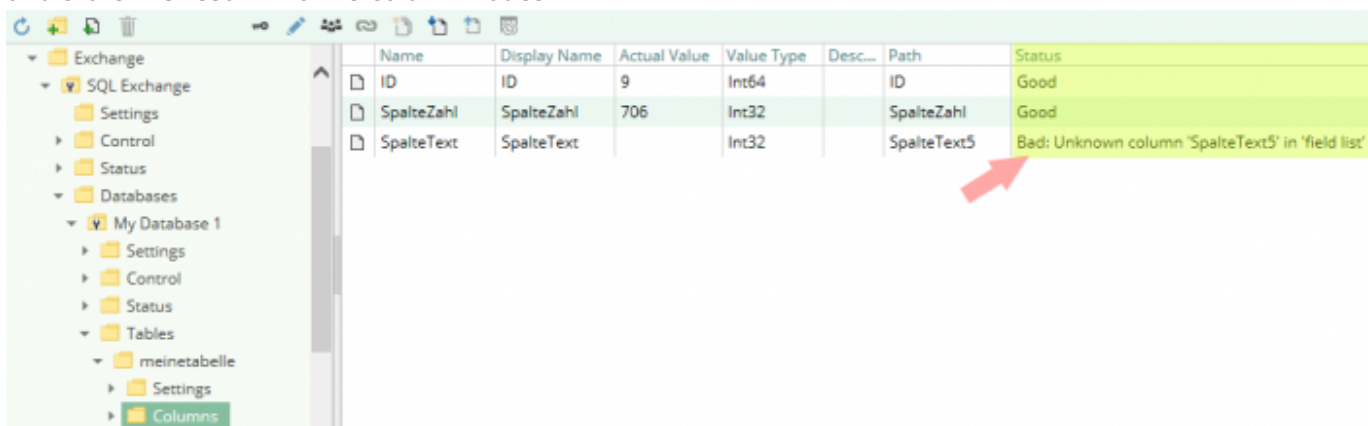
The image above depicts the database channel's Control Panel which displays all status relevant information. The control panel will automatically update its status information when a new status is available.

### Status Circle

Color	Meaning
● (Grey)	The database channel is stopped. Click the ▶ button to start it.
● (Yellow)	The database channel is currently in the progress of starting or stopping or is waiting to establish a connection.
● (Green)	The database channel is running and a connection has successfully been established. You can stop it by clicking the ■ button.
● (Red)	The database channel is running, but the connection is currently in an error state. Please check the status text for more information.

### Columns

To monitor and diagnose the status of the different table columns, take a look at the columns' **Status** property displayed in UKI-4.0 . Use the button "Read actual Value" to read the values from the database and store the result into the column nodes.



### Log File

All database related status information is also logged into the database-specific log file stored in the **[LoggingFolder]**. Each log file is named in the naming scheme **SQL Exchange.<DatabaseName>.log**. The content of such a log file can look as follows:



```

...
[14:55:34 25.07.2017] - Error (Severity=High): Code=[-1], Text=[Access denied for user
'abc'@'localhost' (using password: YES)], Details=[]
...
    
```

# Entities

As each exchange plugin the SQL exchange plugin extends the basic UKI-4.0 [Exchange Model](#).

## Exchange

The plugin's exchange type `SqlExchange` also defines the `SqlExchangeChannel` and therefore extends the basic UKI-4.0 `Exchange` and UKI-4.0 `ExchangeChannel` entities. While the `SqlExchange` just represents a concretization of the UKI-4.0 `Exchange`, the `SqlExchangeChannel` extends the UKI-4.0 `ExchangeChannel` with the SQL Table entities.

## Channel

Each channel is handled by a channel worker which establishes a physical connection to the database. For diagnostic purposes, the worker automatically checks the database connection every 10 seconds to update the status code of the `Channel` and description to detect connection failures.

By default, the worker does not read any values. When a Client or plugin requests a synchronous read of the `Channel`, the channel worker reads the variables in UKI-4.0 (e.g. using the UKI-4.0 `Web Configuration`'s function "Read actual value") from the database and then writes them into the corresponding UKI-4.0 `Nodes`.

Similarly, when a Client or plugin writes values into the channel's variables, the channel worker will write those values to the database.

To have an database variable being read steadily, you can edit the Node in the Configuration Web GUI and set "History Options" to `Yes` (which will create an internal subscription), or you can use e.g. an OPC UA Client connected to the OPC UA Server plugin and create a subscription for the S7 variable Nodes. In these cases, the channel worker reads the variables from the database at a regular interval and, if the value of one of the variables has changed, writes the new value into the corresponding UKI-4.0® Node.

## Table

Each table entity represents the information needed to access a database table. At table level, you can specify how to select the row which the values will be read from or written to.

## Column

Each column entity represents a single database table column.

# Folders & Files

## Folders

Content	Path	Usage
AssemblyFolder	<UKI-4.0 <code>InstallDir</code> >/plugins/SqlExchangePlugin/	Contains the plugin's assembly file.
ConfigFolder	<UKI-4.0 <code>DataDir</code> >/plugins/SqlExchangePlugin/	Contains the plugin's configuration file.
LoggingFolder	<UKI-4.0 <code>DataDir</code> >/log/	Contains the plugin's log files.

## Files

Type	Path	Usage
Assembly	[AssemblyFolder]/UKI-4.0 .SqlExchangePlugin.dll	The plugin's assembly file.
Logging	[LoggingFolder]/SQL Exchange.<DatabaseName>.log	The log file.

# About Versions

## This Document

<b>Date</b>	2018-06-15
<b>Version</b>	1.1

## Plugin

<b>Name</b>	SQL Exchange Plugin
<b>Node</b>	/System/Exchange/SQL Exchange
<b>Version</b>	1.0.5

## Assembly

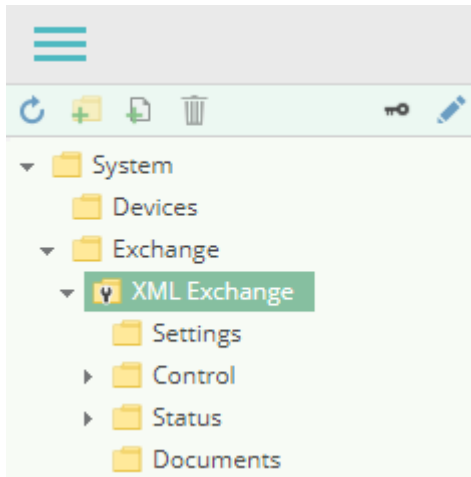
<b>Name</b>	UKI-4.0 .SqlExchangePlugin.dll
<b>Date</b>	2019-02-04
<b>Version</b>	1.0.5.0

# XML Exchange Plugin

The XML Exchange Plugin allows reading and writing values in XML files (with a fixed structure).

## Configuration

The whole XML Exchange Plugin configuration is located in the node path `/System/Exchange/XML Exchange`.



## Document

An XML Exchange Document (similar to a Channel for device plugins) represents an XML file.

## Settings

### File Path

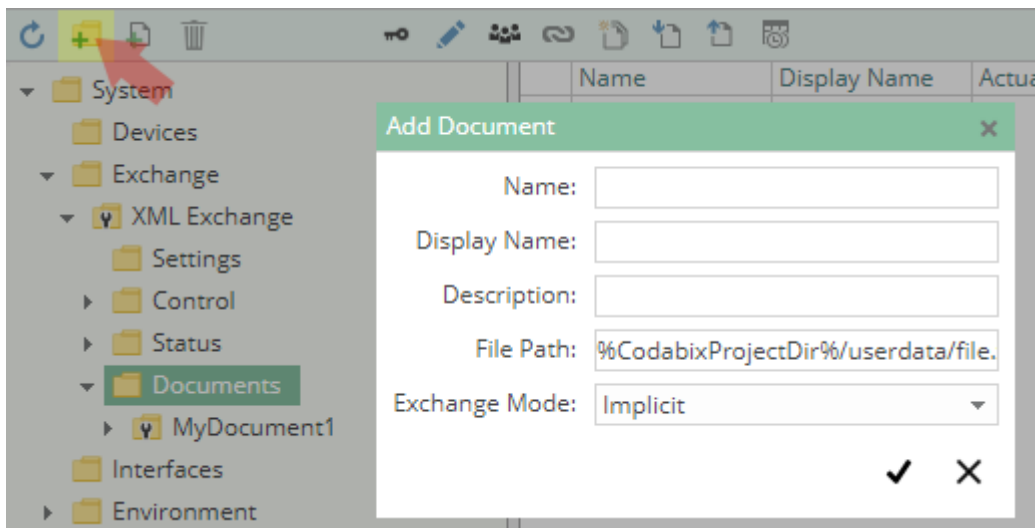
The path to the XML file in the file system.

### Exchange Mode

Specifies how the XML file is to be accessed:

- **Implicit:** The XML file will be read once one or more variables are read; and the XML file will be written once one or more variables are written.
- **Explicit:** No file access occurs when reading or writing variables. The file needs to be read or written explicitly, by calling the `Load` or `Save` method in order to transfer the file content to the variables and vice versa.

## Adding a Document (Channel)



To add a new **XML Document** (which corresponds to a **Channel** for device plugins), please follow these steps:

1. Add a Folder Node within the node **XML Exchange/Documents**, or right-click on the **XML Exchange/Documents** node and select **Add Document**.
2. In the **Add Document** dialog, specify the settings for the XML file.
3. After clicking on "Save", the document node is created.
4. You can start the channel by selecting the document node and clicking the start button.

## Variables

Within the **DocumentElement** node you can create variables (nodes) which correspond to the structure of the XML document (which means the tree structure of the UKI-4.0 variables corresponds to the structure of the XML nodes):

- A Folder Node variable maps to an **XML element**, where the **Path** property contains the name of the element; or, if it is empty, the name of the variable will be used as name for the XML element.
  - A Datapoint Node variable maps to an **XML attribute** or **XML text node** which can be read or written as **String**. The **Path** property can contain the following expressions:
    - **text()**: The variable maps to a **XML text node**.
    - Does **not** start with **/** (e.g. **abc**): The variable maps to an **XML attribute** of the parent XML element with the specified name (in the example, **"abc"**).
    - Starts with **/** (e.g. **/\*/A/B/text()[2]**): The variable represents an **XPath query**, which either returns a text as result (which means it cannot be written to), or an XML node set where then the first node is used (must be a text node, attribute node or comment node).
- Note:** When using an XPath query, the position of the variable within the node tree will not be considered.

When the XML file already exists, you can browse the Document (Channel) to automatically create the UKI-4.0 variables for the XML document structure.

## Example

[example.xml](#)

```

<Root>
  <A>Test</A>
  <B>
    <C x="Hello" y="World">
      12345
    <D />
    67890
  </C>
</B>
</Root>
    
```

After browsing the document, the following node structure is created in UKI-4.0 :

Name	Display Name	Actual Value	Value Type	Description	Path	Status
x	x	Hello	String		x	Good
y	y	World	String		y	Good
@Value	@Value	12345	String		text()	Good
D	D				D	---
@Value (1)	@Value (1)	67890	String		text()	Good

## Read/Write

For read operations as well as for write operations, the UKI-4.0 variables will be mapped to XML nodes by using their position within the node tree. When you write values and there are variables in UKI-4.0 which cannot be found in the XML file (which also happens e.g. if the XML file doesn't exist yet), the corresponding nodes will be created in the XML file.

**Read Behavior** (depending on the set **Exchange Mode** in the settings).

- Exchange Mode **“Implicit”**:
  - Synchronous read of one or more variables:
    - The XML file is read completely.
    - The XML nodes are mapped to the existing UKI-4.0 variables using the node tree structure.
    - Values are written only in those UKI-4.0 variables which were part of the synchronous read operation.
      - If reading the XML file fails, a value with status **Bad** is written to the variables.
  - Calling the **Load** method:
    - An error is raised because the method can only be called in **Explicit** mode.
- Exchange Mode **“Explicit”**:
  - Synchronous read of one or more variables:
    - No file access occurs; the current values of the read variables are returned as result of the synchronous read operation.
  - Calling the **Load** method:
    - The XML file is read completely.

- The XML nodes are mapped to the existing UKI-4.0 variables using the node tree structure.
- Values that were read from the corresponding XML nodes are written into all UKI-4.0 variables.
  - If reading the XML file fails, nothing is written into the variables, but the method raises an error.

### Write Behavior:

- Exchange Mode **“Implicit”**:
  - Writing values in one or more variables:
    - The XML file is read completely.
    - The XML nodes are mapped to the existing UKI-4.0 variables using the node tree structure.

If a XML node cannot be found for an existing UKI-4.0 variable, a new node is created in the XML file (even when that variable isn't part of the write operation), but initially without a value (text).
    - Values are inserted only in those XML nodes whose corresponding UKI-4.0 variables were part of the write operation.
    - The XML file is written.
  - Calling the **Save** method:
    - An error is raised because the method can only be called in **Explicit** mode.
- Exchange Mode **“Explicit”**:
  - Writing values in one or more variables:
    - No file access occurs; the status **Good** is used as result of the write operation.
  - Calling the **Save** method:
    - The XML file is read completely.
    - The XML nodes are mapped to the existing UKI-4.0 variables using the node tree structure.

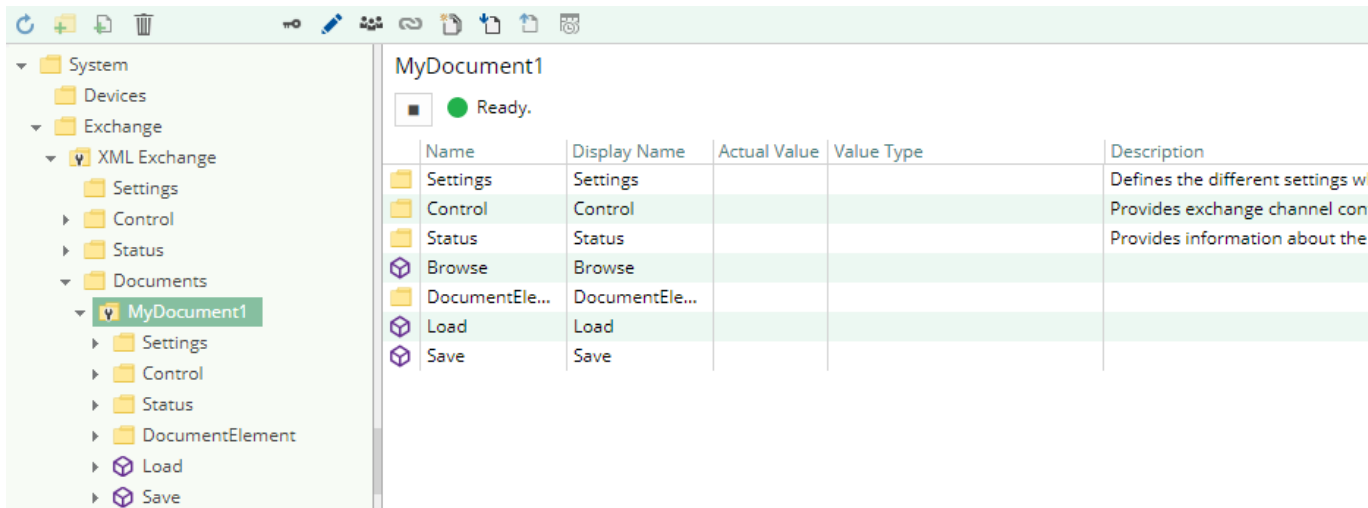
If a XML node cannot be found for an existing UKI-4.0 variable, a new node is created in the XML file, but initially without a value (text).
    - The current values of the UKI-4.0 variables are inserted into all XML nodes.
    - The XML file is written.

## Diagnostics

The XML Exchange Plugin provides different status information depending on the layer to inspect. In general the document-based diagnostic information is produced by the access status of the XML file. The variable-based diagnostic information is produced during the read/write access of the different variables.

## Channel

To monitor and diagnose the status of a XML Document (Channel), take a look at the following image:



The image above depicts the XML Document's Control Panel which displays all status relevant information. The control panel will automatically update its status information when a new status is available.

### Status Circle

Color	Meaning
	The channel is stopped. Click the  button to start it.
	The channel is currently in the progress of starting or stopping or is waiting to establish a connection.
	The channel is ready for doing read/write operations. You can stop it by clicking the  button.
	The channel is running, but the connection is currently in an error state. Please check the status text for more information.

## Variables

To monitor and diagnose the status of the different XML variables, take a look at the variable's **Status** property displayed in UKI-4.0 . If the Exchange Mode is set to **“Implicit”**, use the button “Read actual Value” to read the values from the PLC and store the result into the variables. Otherwise, you can call the **Load** method to verify if the file can be read correctly.

Name	Display Name	Actual Value	Value Type	Description	Path	Status
x	x		String		x	Bad: Name cannot begin with the '<' character, hexadecimal value 0x3C. Line 2, position 1.
y	y		String		y	Bad: Name cannot begin with the '<' character, hexadecimal value 0x3C. Line 2, position 1.
D	D				ns2:D	---

## Log file

All channel related status information is also logged into the channel-specific log file stored in the **[LoggingFolder]**. Each log file is named in the naming scheme **XML Exchange.<ChannelName>.Log**.

The content of such a log file can look as follows:

```

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

# Entities

Like every exchange plugin the XML Exchange Plugin extends the basic UKI-4.0 [Exchange Model](#).

## Exchange

The plugin's exchange type `XmlExchange` also defines the `XmlExchangeDocument` and therefore extends the basic UKI-4.0 `Exchange` and UKI-4.0 `ExchangeChannel` entities. While the `XmlExchange` just represents a concretization of the UKI-4.0 `Exchange`, the `XmlExchangeDocument` extends the UKI-4.0 `ExchangeChannel` with the XML Variable Entities.

## Channel

Each channel is handled by a channel worker which access an XML file using file system operations.

By default, the worker does not read any values. When the Exchange Mode is set to `Implicit` and a client or plugin requests a synchronous read of the channel's variables in UKI-4.0 (e.g. using the UKI-4.0 Web Configuration's function "Read actual value"), the channel worker reads them from the underlying PLC and then writes them into the corresponding UKI-4.0 Nodes.

If the Exchange Mode is set to `Explicit`, the worker reads values as soon as the `Load` method of the channel is called.

Similarly, the channel worker will write values to file when a client or plugin writes values into the channel's variables ("Implicit") or when the channel's `Save` method is called ("Explicit").

To have an XML Exchange Variable being read steadily (when using mode "Implicit"), you can edit the Node in the UKI-4.0 Web Configuration and set "History Options" to `Yes` (which will create an internal subscription), or you can use e.g. a OPC UA Client connected to the OPC UA Server plugin and create a subscription for the XML variable nodes. In these cases, the channel worker reads the variables from the XML file at a regular interval and, if the value of one of the variables has changed, writes the new value into the corresponding UKI-4.0 nodes.

# Folders & Files

## Folders

Content	Path	Usage
AssemblyFolder	<UKI-4.0 <code>InstallDir</code> >/plugins/XmlExchangePlugin/	Contains the plugin's assembly file.
ConfigFolder	<UKI-4.0 <code>ProjectDir</code> >/plugins/XmlExchangePlugin/	Contains the plugin's configuration file.
LoggingFolder	<UKI-4.0 <code>ProjectDir</code> >/log/	Contains the plugin's log files.

## Files

Type	Path	Usage
Assembly	[ <code>AssemblyFolder</code> ]/UKI-4.0 <code>.XmlExchangePlugin.dll</code>	The plugin's assembly file.
Logging	[ <code>LoggingFolder</code> ]/XML Exchange.<ChannelName>.log	The log file.



# About Versions

## This Document

<b>Date</b>	2020-02-06
<b>Version</b>	1.0

## Plugin

<b>Name</b>	XML Exchange Plugin
<b>Node</b>	/System/Exchange/XML Exchange
<b>Version</b>	1.0.0

## Assembly

<b>Name</b>	UKI-4.0 .XmlExchangePlugin.dll
<b>Date</b>	2020-02-06
<b>Version</b>	1.0.0.0

# Interface Plugins

All interface plugins make use of the UKI-4.0<sup>®</sup> API. Each interface provided is registered and administrated by the UKI-4.0<sup>®</sup> Interface Manager.

Such interfaces can be:

- RESTful applications (see REST Interface Plugin)
- Scripts (see Script Interface Plugin)
- other subsystems (see OPC UA Server Plugin)

## Interface Model

The UKI-4.0<sup>®</sup> API is extended by interfaces. Each interface does therefore extend the accessibility of UKI-4.0<sup>®</sup> for other platforms and technologies. Additionally, there is the option that an interface makes use of the UKI-4.0<sup>®</sup> Interface Model. Hereby the UKI-4.0<sup>®</sup> Interface Model extends the basic UKI-4.0<sup>®</sup> Entity Model with typical interface entities.

In this case the interface entity defines the subordinated entities for control, settings, the status of the plugin and the various channels, via which the plugin interacts with UKI-4.0<sup>®</sup> or other platforms.

## Configuration

Each interface plugin delivered with UKI-4.0<sup>®</sup> is directly and exclusively configurable in the UKI-4.0<sup>®</sup> Host application. If a plugin has configuration parameters, those can be modified in the according interface entity.

## Using <sup>UKI-4.0</sup><sub>UKI-4.0</sub> <sup>UKI-4.0</sup><sub>UKI-4.0</sub> <sup>®</sup>

The entire configuration of all interface plugins can be found under the Node path [/System/Interfaces](#). This root Node of the interface plugins allows the complete configuration of the interface plugins, provided that one of the actively used interface plugins supplies its own interface entities for the configuration.

# OPC UA Server Interface Plugin

## General

The OPC UA Server Interface Plugin allows you to connect to UKI-4.0 via the standardized OPC UA interface.

## What Does the Plugin Do?

The interface allows to you to create OPC UA Servers using the **opc.tcp** protocol and expose UKI-4.0 Nodes to OPC UA Clients.

## Features

- Browse UKI-4.0 ® Nodes
- (Synchronously) read and write values to Nodes
- Create subscriptions to Nodes
- Access historic values of UKI-4.0 ® Nodes using HDA (Historical Data Access)
- Remotely access files on the UKI-4.0 ® host machine with the OPC UA filetype
- Supports the **SignAndEncrypt** policy for encrypted transport

## Supported OPC Protocols

- opc.tcp (binary message encoding)
- http (binary and XML message encoding - in a future version)

## Purpose & Use Cases

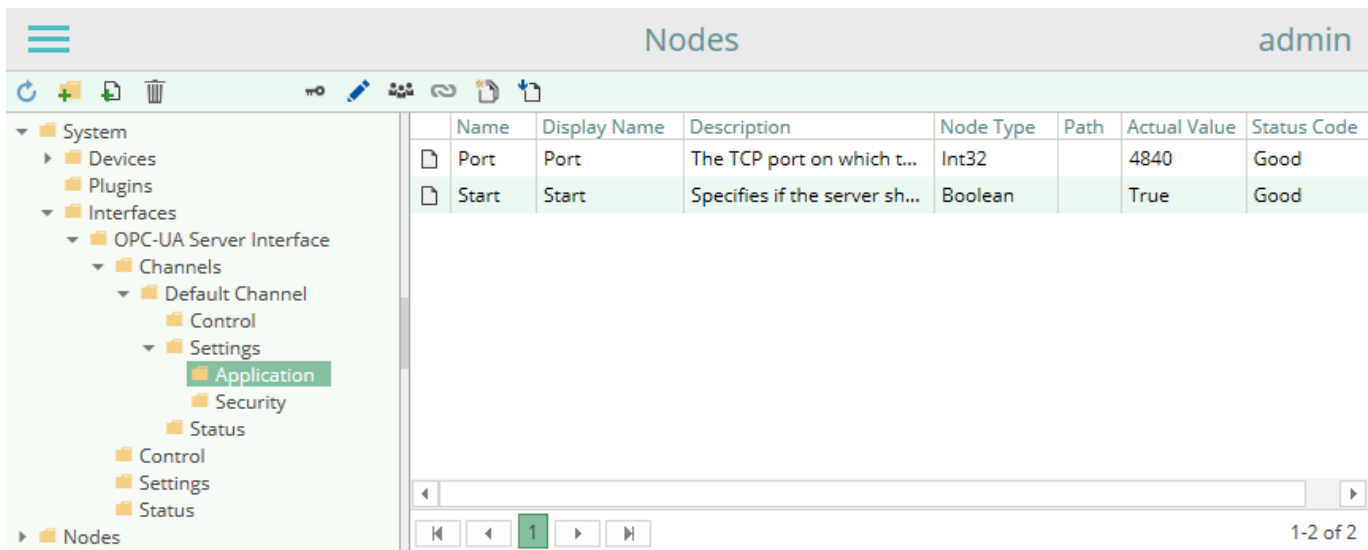
- Connect S7 Variables or other Nodes using OPC UA

## Installation

This plugin is part of the UKI-4.0 ® Setup. Please consult UKI-4.0 ® [Setup and First Start](#) for more information on how to install and uninstall this plugin.

## Configuration

The OPC UA Server Interface Plugin can be configured only in the Interface Node in UKI-4.0 ® ([/System/Interfaces/OPC UA Server Interface/Channels/Default Channel/Settings](#)):



**Nodes:**

Name	Type	Purpose
<b>Application</b>		
ApplicationCertificate	Blob	Contains the application instance certificate of the OPC UA Server in PKCS12 (.pfx) format.
Port	Int32	Specifies the TCP Port for the <code>opc.tcp</code> protocol.
<b>Security</b>		
AutoAcceptUntrustedCertificates	Boolean	Specifies if the OPC UA Server should automatically accept unknown Client application certificates when using the Sign or SignAndEncrypt mode.
PolicyAlgorithm	String	Valid values: <b>0</b> : None (default) <b>1</b> : Basic128Rsa15
PolicyLevel	Int32	
PolicyMode	String	Valid values: <b>0</b> : None (default) <b>1</b> : Sign <b>2</b> : SignAndEncrypt

To change the settings, you can write a new Node value in the UKI-4.0 ® Web Configuration by using the "Write a new value" button (📄).

To connect to the OPC UA Server, use the following URL (replace <Hostname> with your machine's hostname or IP Address (or "localhost" if connecting from the same machine), and replace <Port> with your port number):

```
opc.tcp://<Hostname>:<Port>/
```

### User Configuration


The OPC UA Server Interface Plugin requires that OPC UA Clients use the username authentication, which means you need to specify a username and password. The OPC UA Server uses the users configured in UKI-4.0 ® (see UKI-4.0 ® Web Configuration). Each user which is configured in UKI-4.0 ® can connect to the OPC UA Server using the corresponding username and password.

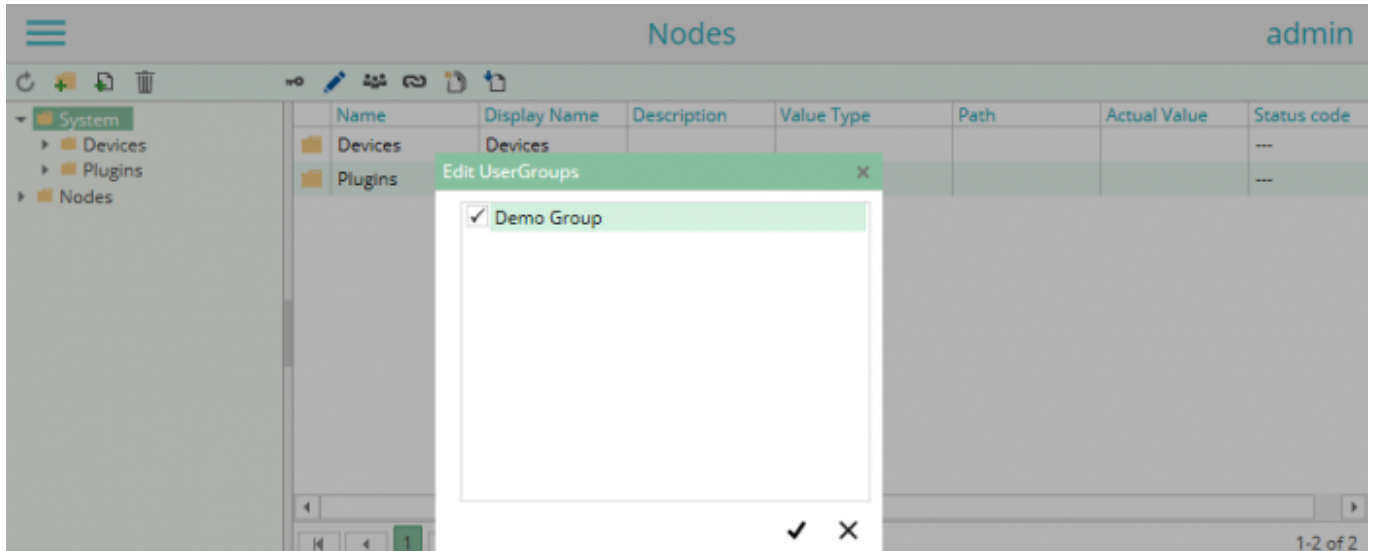
By default, UKI-4.0 ® comes with a demo user (username "demo@user.org", password "demo") which is added to the "Demo Group" usergroup.

By default, a user cannot access any UKI-4.0 ® Node. To change this, you need to add the user to a UserGroup, then allow access from this UserGroup to a Node. For further information, see UKI-4.0 ® Web

## Configuration.

For example, to allow the demo user (demo@user.org) to access the “System” Node:

1. Open the UKI-4.0<sup>®</sup> Web Configuration
2. Go to Nodes
3. Select the “System” Node
4. Click the “UserGroups” icon 
5. Select the “Demo Group” in the “Edit UserGroups” dialog



## Accessing the OPC UA Server

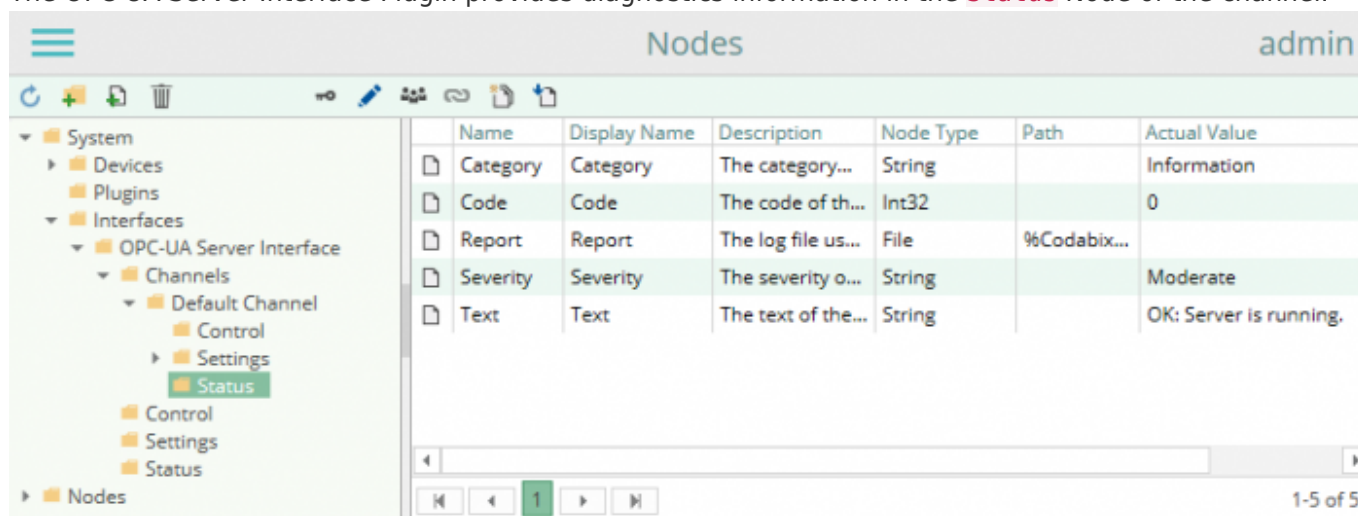
After configuring the users, you can then access the OPA UA Server e.g. with our tool **OPC Watch** using the following configuration:

- ServerAddress: `opc.tcp://localhost:4840/`
- UserIdentity / Username: `demo@user.org`
- UserIdentity / Password: `demo`

**OPC Namespace:** The UKI-4.0<sup>®</sup> Nodes use the OPC UA Namespace URI "`UKI-4.0 ://opc.server/`". By default, this namespace will have the namespace index “2”.

# Diagnostics

The OPC UA Server Interface Plugin provides diagnostics information in the **Status** Node of the channel.



Node	Description
Code	Defines the numeric expression / identifier of the status. A negative value means that the Server could not be started, whereas 0 or a positive value indicates that the Server started successfully.
Category	Classifies the status into <b>Information</b> , <b>Warning</b> and <b>Error</b> and therefore indicates the general type of status information.
Severity	Rates the status information into <b>Low</b> , <b>Moderate</b> , <b>High</b> and <b>Critical</b> and therefore indicates the urgency of an intervention.
Text	Describes the status information identified by the <b>Code</b> property.

# Entities

As an interface plugin, the OPC UA Server Interface Plugin extends the basic UKI-4.0 ® [Interface Model](#). **Interface**

The plugin's interface type **OpcServerInterface** also defines the **OpcServerInterfaceChannel** and therefore extends the basic UKI-4.0 **Interface** and UKI-4.0 **InterfaceChannel** entities. While the **OpcServerInterface** just represents a concretization of the UKI-4.0 **Interface**, the **OpcServerInterfaceChannel** extends the UKI-4.0 **InterfaceChannel**.

## Channel

Each channel represents an OPC UA Server instance that can be configured, started and stopped.

# Folders & Files

## Folders

Content	Path	Usage
Assembly	<UKI-4.0 <b>InstallDir</b> >/plugins/OpcUaServerInterfacePlugin/	Contains the plugin's assembly file.
Config	<UKI-4.0 <b>DataDir</b> >/plugins/OpcUaServerInterfacePlugin/	Contains the plugin's configuration file.

## Files

Type	Path	Usage
Assembly	[AssemblyFolder]/UKI-4.0 .OpcUaServerInterfacePlugin.dll	The plugin's assembly file.

# About Versions

## This Document

<b>Date</b>	2016-12-21
<b>Version</b>	1.3

## Plugin

<b>Name</b>	OPC UA Server
<b>Node</b>	/System/Interfaces/OPC-UA Server Interface
<b>Version</b>	1.0.9

## Assembly

<b>Name</b>	UKI-4.0 .OpcUaServerInterfacePlugin.dll
<b>Date</b>	2018-02-01
<b>Version</b>	1.0.9.0

# REST Interface Plugin

## General

The REST Interface allows access to the [Nodes](#) of UKI-4.0 ® via [HTTP request](#) formatted as an [JSON object](#).

## What Does the Plugin Do?

- Read [Nodes](#) (values, optionally historical values, children and [properties](#)) from UKI-4.0 ®.
- Write a new actual value to a specified [Node](#).
- Create a new [Node](#).

e.g. the following HTTP request is reading the [System Nodes](#) Tree of 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);
```

## Features

Supported actions for [Nodes](#):

Action	Description
<a href="#">Reading</a>	Reads the specified Node.
<a href="#">Browsing</a>	Reads the specified Node and the child Nodes.
<a href="#">Writing</a>	Writes a new actual value to the specified Node.

## Supported Clients

- HTTP Clients supporting HTTP/1.1 and supporting the JSON format



# Purpose & Use Cases

- Easy connection to other systems
- Easy read and write access to the [Nodes](#) of UKI-4.0 ® using any programming language without additional libraries
- Linking of UKI-4.0 ® to existing RESTable Software (e.g. SCADA systems)

## Documentation of the RESTful API

You can find the documentation of the REST API of the plugin here:

- [Documentation of the RESTful API](#)

# Installation

The REST Interface Plugin is an inherent part of UKI-4.0 ® and does not need to be installed.

# Configuration

The REST Interface Plugin currently doesn't provide a configuration.

# Diagnostics

The REST Interface Plugin provides a status code for functions for diagnostic purposes - see [Request & Response](#) of the REST API Documentation.

# Entities

The REST Interface Plugin does not use the UKI-4.0 ® Entity Model and therefore doesn't provide entities.

# Folders & Files

The REST Interface Plugin is an inherent part of UKI-4.0 ® and therefore doesn't consist of additional files or folders.

# About Versions

## This Document

<b>Date</b>	2017-07-12
<b>Version</b>	1.2

## Plugin

<b>Name</b>	REST Interface Plugin
<b>Version</b>	Corresponds to the UKI-4.0 ® Version

## Assembly

The REST Interface Plugin is an inherent part of UKI-4.0 ® and therefore doesn't consist of a separate assembly.



# REST API Documentation

- The interface has to be a [HTTP request](#) with a [JSON object](#) (Encoding: UTF-8) in the request body.

Example:

```
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": "" }
}
```

## Access

Access to the REST API can be granted using following link:

```
http(s)://<ip/domain>:<port>/api/json
```

e.g.

```
http://localhost:8181/api/json
```

See [Configuration of the Port](#)

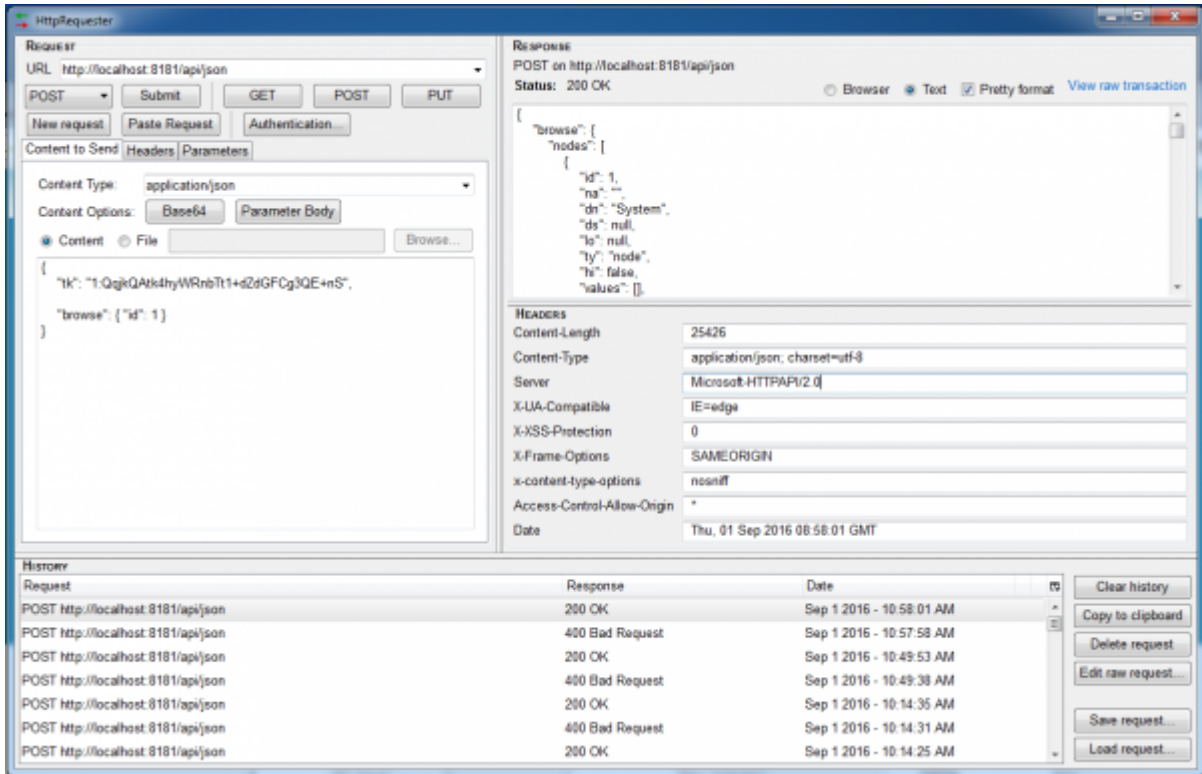
## Request & Response

### Specifications of the Request

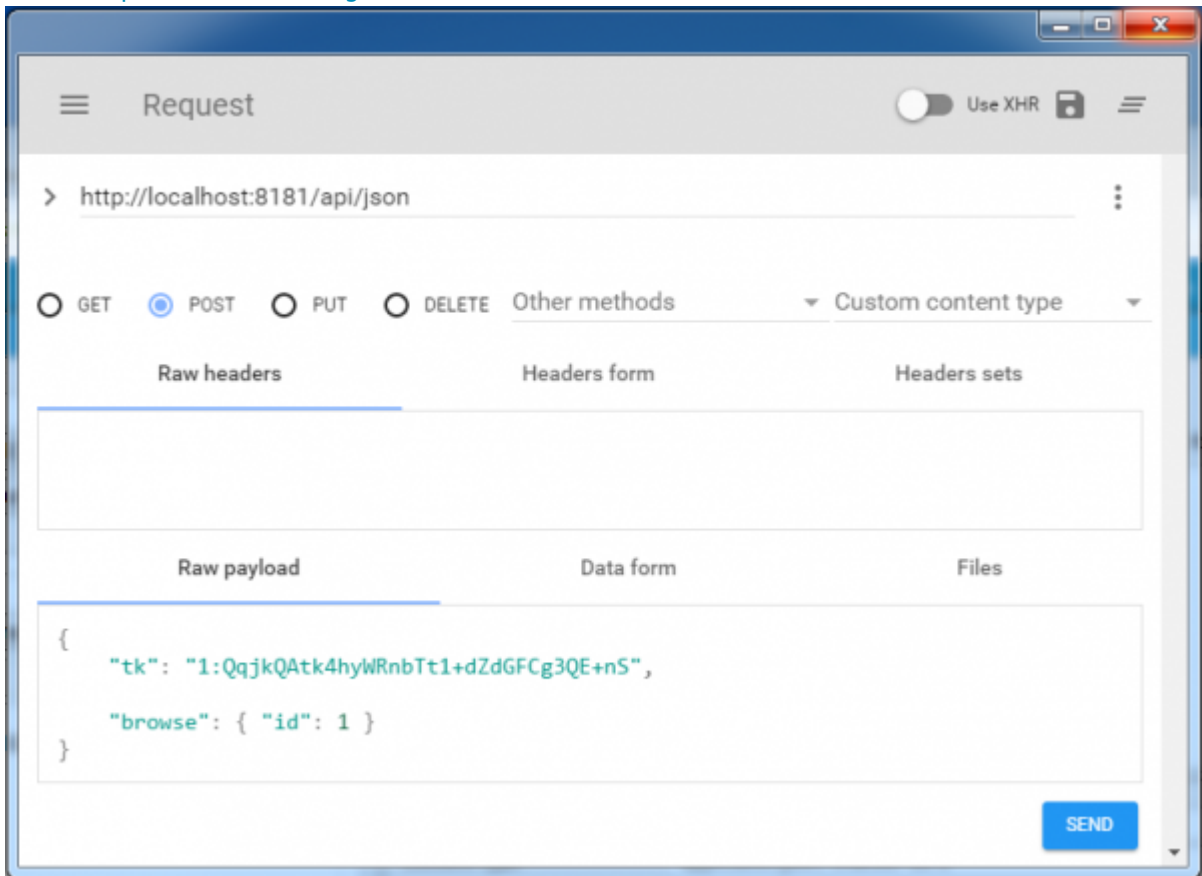
<b>Method</b>	POST
<b>Content type</b>	<a href="#">application/json</a>
<b>Encoding</b>	UTF-8
<b>Content</b>	<a href="#">JSON object</a>

### Tools for Example Request

- [HTTP Request Script](#)
- [HTTP Request Tool for Firefox](#)



- HTTP Request Tool for Google Chrome



## General Request

	Mandatory	Type	Purpose
tk	yes (if username and password are not specified)	string	The <b>Token</b> of the Node which is to be authenticated. From this Node, all children will be relatively addressed. For how to find the Token see <a href="#">Node Access</a> .  e.g. the token for Node <b>System/Plugins</b> is set, then "na": "OPC UA Server" is the relative path for Node <b>/System/Plugins/OPC UA Server</b> .
username	yes (if tk is not specified)	string	The email address or telephone number of the user to authenticate.
password	yes (if tk is not specified)	string	The password of the user.
get	no	object	<a href="#">Read Nodes</a>
browse	no	object	<a href="#">Read Nodes with all children (recursive)</a>
set	no	object	<a href="#">Write Nodes</a>
create	no	object	<a href="#">Create Nodes</a>
update	no	object	<a href="#">Update Nodes</a>
delete	no	object	<a href="#">Delete Nodes</a>

**Example Request (using a Token):**

```
{
  "tk": "961:oseIqCm8W00nPEE5g0tt1TZz2wbL/qUq",
  "get": {
    "na": "Temperature"
  }
}
```

**Example Request (using username-password):**

```
{
  "username": "demo@user.org",
  "password": "demo",
  "get": {
    "na": "/Demo-Nodes/Temperature"
  }
}
```

**Request frame:**

```
{
  "tk": /* Token */,
  "username": /* Username (if 'tk' not specified) */,
  "password": /* Password (if 'tk' not specified) */,

  "get": { /* "Get" elements */ },
  "browse": { /* "Browse" elements */ },
  "set": { /* "Set" elements */ },
  "create": { /* "Create" elements */ },
  "update": { /* "Update" elements */ },
  "delete": { /* "Delete" elements */ }
}
```

**General Response**

	Mandatory	Type	Purpose
get	no	object	<a href="#">Read &amp; Browse response</a>

	<b>Mandatory</b>	<b>Type</b>	<b>Purpose</b>
browse	no	object	Read & Browse response
set	no	object	Write response
create	no	object	Create response
update	no	object	Update response
delete	no	object	Delete response
res	<b>yes</b>	object	Result of the query

Possible **res** results:

	<b>Mandatory</b>	<b>Type</b>	<b>Purpose</b>
value	<b>yes</b>	number	Result codes: >= 0: OK - Everything is good! < 0: An error has occurred!
reason	no	string	Reason for the error

**Example Response:**

```

{
  "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
  }
}
    
```

**Response Frame:**

```

{
  "get": { /* "Get" elements */ },
  "browse": { /* "Browse" elements */ },
  "set": { /* "Set" elements */ },
  "create": { /* "Create" elements */ },
  "update": { /* "Update" elements */ },
  "delete": { /* "Delete" elements */ },
  "res": { /* "Result" elements */ },
}
    
```

# Read & Browse

## Request

- The “Get” function makes it possible to read one or more (historical) values. “Browse” is like “Get” but allows only to get the actual value of the Node, therefore it will recursively return all child Nodes including their actual value.

	Mandatory	Type	Purpose
<code>id</code>	<b>yes</b> (if <code>na</code> is not specified)	number	Local Identifier (“Id”) of the Node.
<code>na</code>	<b>yes</b> (if <code>id</code> is not specified)	string	Name of the Node. When using a token for authentication, this is the relative path from the authenticated Node to the destination Node (if empty, the authenticated Node itself is the destination Node). Otherwise (for username-password authentication), it is the absolute path to the destination Node.
<code>count</code> (if <code>tll</code> is not present)	no	number	The amount of history values (max. 1000). Default is <code>1</code> , which means the actual value (instead of history values) is returned.
<code>from</code> (if <code>tll</code> is not present)	no	number	If specified, the start timestamp from which to retrieve history values.
<code>to</code> (if <code>tll</code> is not present)	no	number	If specified, the end timestamp up to which to retrieve history values.
<code>tll</code> (if <code>count</code> is not present)	no	number or <code>null</code>	Time To Live. If this parameter is not present, the function directly returns the current (“cached”) value of the Node. If the parameter is present, a <b>synchronous read</b> is done on the Node in order to read from the underlying device. A value of <code>null</code> means to use the default maximum value age of the Node. Otherwise, the value is the age in milliseconds that the current Node value can have to be returned directly. If the value of the Node is older, it is read synchronously from the device. To force a synchronous read regardless of the node's Max Value Age, specify a value of <code>0</code> .

### Example Read Request (using a Token):

```
{
  "tk": "961:oseIqCm8W00nPEE5g0tt1TZz2wbL/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 */
  }
  ]
}
```

### Example Read Request (using username-password):

```

{
  "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 */
  }
]
}
    
```

## Response

- For each requested “Get” or “Browse” object the Server will respond with a Node (in array) which has the following properties:

	Mandatory	Type	Purpose			
res	yes	object	Result of the query			
id	yes	number	Identifier			
na	yes	string	Name			
dn	yes	string	Display name			
ds	yes	string	Description			
lo	yes	string	Location			
ty	yes	string	Type			
hi	yes	boolean	History Options			
min	yes	number	Minimum value			
max	yes	number	Maximum value			
values	yes	object array				
nodes	no (only browse)	object array	Contains all child Nodes (recursive). For each child the properties are the same as in this table.			

Possible res results:

	Mandatory	Type	Purpose
value	yes	number	Result Codes: >= 0: OK - Everything is good! < 0: An error has occurred!
reason	no	string	Reason for the error

Possible value results:

	Mandatory	Type	Purpose
va	yes	any	Value
ts	yes	number	Timestamp
ct	yes	number	Category
st	yes	number	Status
sttext	no	string	Status text (if set)

### Example Read Response:

```

{
  "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
}
```

```

}
}

```

# Write

## Request

- With the “Set” function it is possible to write values to a specified Node.
- If the timestamp is not set, it is important to send the requests for the same Node sequentially. So before sending another request it is important to wait for the successful response. Otherwise it could happen, that the values are sorted in another sequence as intended. Different Nodes of course can be written parallelly.

	Mandatory	Type	Purpose
id	yes (if na is not specified)	number	Identifier
na	yes (if id is not specified)	string	Name
va	yes	any	new value
ts	No	number	Timestamp
st	No	number	Status

### Example Write Request (using a Token):

```

{
  "tk": "961:oseIqCm8W00nPEE5g0tt1TZz2wbL/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
    }
  ]
}

```

### Example Write Request (using username-password):

```

{
  "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
    }
  ]
}
    
```

## Response

- For the “Set” function the Node array only has response objects for failed requests objects.

	Mandatory	Type	Purpose
res	yes	object	Result of the query
nodes	yes	object array	Contains all child Nodes (recursive). For each child the properties are the same as in this table.

Possible res results:

	Mandatory	Type	Purpose
value	yes	number	Result codes: >= 0: OK - Everything is good! < 0: An error has occurred!
reason	no	string	Reason for the error

### Example Write Response:

```

{
  "set": {
    "nodes": [],
    "res": {
      "value": 0
    }
  }
}
    
```

### Example Write Response (Error Case):

```

{
  "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"
    }
  }
}

```

# Create

## Request

	Mandatory	Type	Purpose
pid	yes (if pna is not specified)	number	Parent Identifier New Node will be created as a Child of this Node.
pna	yes	string	Parent Name New Node will be created as a Child of this Node. e.g if pna="Demo-Nodes", the new Node will be created in this folder. The new Node will be "/Nodes/Demo-Nodes/Name"
na	yes (if pid is not specified)	string	Name
dn	no	string	Display name
ds	no	string	Description
lo	no	string	Location
path	no	string	Path (e.g. for device variables)
ty	yes	string	Type
hi	no	number	Hysteresis
min	no	number	Minimum value
max	no	number	Maximum value

### Example Create Request:

```

{
  "tk": "938:843uqQeSaLAg+7TALd0hGDkL0HFZevZw", /* 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"
  }
  ]
}
    
```

## Response

	Mandatory	Type	Purpose
res	yes	object	Result of the query
nodes	yes	object array	Contains all child Nodes (recursive). For each child the properties are the same as in this table.

Possible **res** results:

	Mandatory	Type	Purpose
value	yes	number	Result Codes: >= 0: OK - Everything is good! < 0: An error has occurred!
reason	no	string	Reason for the error

### Example Create Response:

```

{
  "tk": "938:843uqQeSaLAg+7TALd0hGDkL0HFZevZw",
  "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"
  }
  ]
}
    
```

### Example Create Response (Error Case):

```
{
  "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."
    }
  }
}
```

## Update

- Modify all properties of a Node.
- This feature is not implemented yet but will be coming soon.

## Delete

- Remove a Node.
- This feature is not implemented yet but will be coming soon.

# Script Interface Plugin

## General

The Script Interface Plugin enables you to write **Scripts** in the *JavaScript* language to program UKI-4.0 . Scripts are **lightweight extensions** of UKI-4.0 .

- A fast and simple (yet powerful) way to extend UKI-4.0 's functionality
- Implemented in **JavaScript** which is a powerful scripting language, and **TypeScript** to avoid errors by providing static typing and to offer a rich developer experience
- Can be changed and restarted while UKI-4.0 is running
- Stored in the UKI-4.0 back-end database, so it is bound to the data, not to the installation
- Executed in an isolated environment, has only access to certain defined UKI-4.0 APIs, not OS APIs
- No need to install an development environment, create a project, compile it, copy the DLL, etc. ... Instead, simply write the script in the built-in and web-based Script Editor with **IntelliSense** support

## What Does the Plugin Do?

- Find, create and manipulate Nodes and read and write Node values
- Register for events, e.g. when a Node value has been written
- Do calculations, call mathematic functions, generate a random number
- Create an interval timer that continuously calls back a script function
- Read and write files
- Handle incoming HTTP(S) requests of the UKI-4.0 web server (including WebSocket connections)
- Execute HTTP(S) requests to external servers

If you have basic programming experience, you should be able to easily find your way when writing code of a script. JavaScript is one of the most popular scripting languages and in combination with TypeScript (that provides static type safety), a script can **scale from a one-liner** (e.g. adjust a Node value before it is written) **to complex code** with namespaces, classes, dependencies on other scripts and further more.

**Note:** You don't need to be familiar with TypeScript to be able to write scripts.

## Features

- High performance by compiling script code to CIL bytecode
- Type safety of variables, properties and so on through TypeScript
- Protects against unintended infinite loops by applying a timeout to script execution

## Supported Specifications

- Support for **TypeScript 4.1** in the Script Editor
- Full support for **ECMAScript 5.1** (syntax and library)

- Almost full support for **ECMAScript 2015+** syntax (`let` / `const`, `class`, `for-of`, `async` / `await` etc.) by downlevel compilation to ECMAScript 5.1 using TypeScript
- Partial support for **ECMAScript 2015** library (Collections, Typed Arrays, `Promise`)
- Supports **Async Functions** for long-running operations using the `async` / `await` keywords (enhancing JavaScript's event-driven, non-blocking model)

## Purpose & Use Cases

You can use scripts for

- simple circuitries (“When a button is pressed, the light shall be turned on and after 3 minutes be turned off”)
- adjusting values that are read from or written to a device, e.g. by doing some calculations
- generating complex Node hierarchies / structures
- generating demo data (e.g. write a random value to a Node every 2 seconds)
- big data: collect external data around specific points, e.g. store Berlin's current weather in a datapoint Node
- providing a complex, customized condition for triggers

### Script Interface Plugin Development Guide

You can find the Development Guide to the Script Interface Plugin here:

- [Script Interface Plugin Development Guide](#)

## Installation

The Script Interface Plugin is an inherent part of UKI-4.0 and does not need to be installed.

## Configuration

The Script Interface Plugin allows to configure Scripts in the UKI-4.0 Web Configuration using the “Script Interface” menu entry - see [Managing Scripts](#) in the Script Interface Plugin Development Guide.

## Diagnostics

The Script Interface Plugin allows to diagnose Scripts in the UKI-4.0 Web Configuration using the “Script Interface” menu entry - see [Managing Scripts](#) in the Script Interface Plugin Development Guide.

## Entities

The Script Interface Plugin does not use the UKI-4.0 Entity Model because it allows Scripts to be managed using the UKI-4.0 Web Configuration and therefore doesn't provide entities.

## Folders & Files

The Script Interface Plugin is an inherent part of UKI-4.0 and therefore doesn't consist of additional files or folders.



# About Versions

## This Document

<b>Date</b>	2017-11-14
<b>Version</b>	1.8

## Plugin

<b>Name</b>	Script Interface Plugin
<b>Version</b>	Corresponds to the UKI-4.0 Version

## Assembly

The Script Interface Plugin is an inherent part of UKI-4.0 and therefore doesn't consist of a separate assembly.

# Script Interface Plugin Development Guide

## Example Code Script

```
// Find the node by specifying the node path.
const pressureNode = UKI-4.0 .findNode("/Nodes/Injection molding/Pressure", true);

// Set an interval timer that will call our function every 3 seconds.
timer.setInterval(function () {

    // Generate a random number between 20 and 150.
    let randomNumber = Math.random() * 130 + 20;

    // Write the value to the Node.
    UKI-4.0 .writeNodeValueAsync(pressureNode, randomNumber);

}, 3000);
```

## Managing Scripts

The UKI-4.0 Web Configuration provides the item “Scripts” to create, edit and delete scripts. You can also stop scripts so that they are not executed anymore, without deleting them.

Property	Description
Name	Name of the script, used to identify it. This name will also be used in stacktraces when an exception occurred.
Description	You can enter a more detailed description of the script.
Editor Strictness Level	<p>Determines how strict the editor will handle certain code.</p> <p><b>Low</b> (default): The editor will not criticize the cases described for the following options.</p> <p><b>Medium</b>: The editor will criticize implicit <b>any</b> types, implicit returns, and fallthrough cases in <b>switch</b> statements.</p> <p><b>High</b>: Additionally to the cases described in the “Medium” level, the editor will criticize unused local variables and unused function parameters.</p> <p>Note: For Library Scripts (available in a later version) the level is always “High”.</p>
Script State	<p><b>Enabled</b>: The Script shall be run.</p> <p><b>Disabled</b>: The Script shall be stopped.</p>

Property	Description
CurrentScriptState	<p>Shows the actual state of the Script.</p> <p><b>NotRunning:</b> The Script is not running, either because it is disabled or because it has just been created and there is not yet a live code for it.</p> <p><b>Running:</b> The Script has been started and there was no error since then (this can also be the case if a Script has recently been restarted after an uncaught exception).</p> <p><b>Stopped:</b> The Script has been started and has been running, but no more event listeners or callbacks are active.</p> <p><b>StoppedAndScheduledForRestart:</b> The Script has been stopped because an uncaught exception occurred while executing it. It will be automatically restarted after a short period of time.</p>

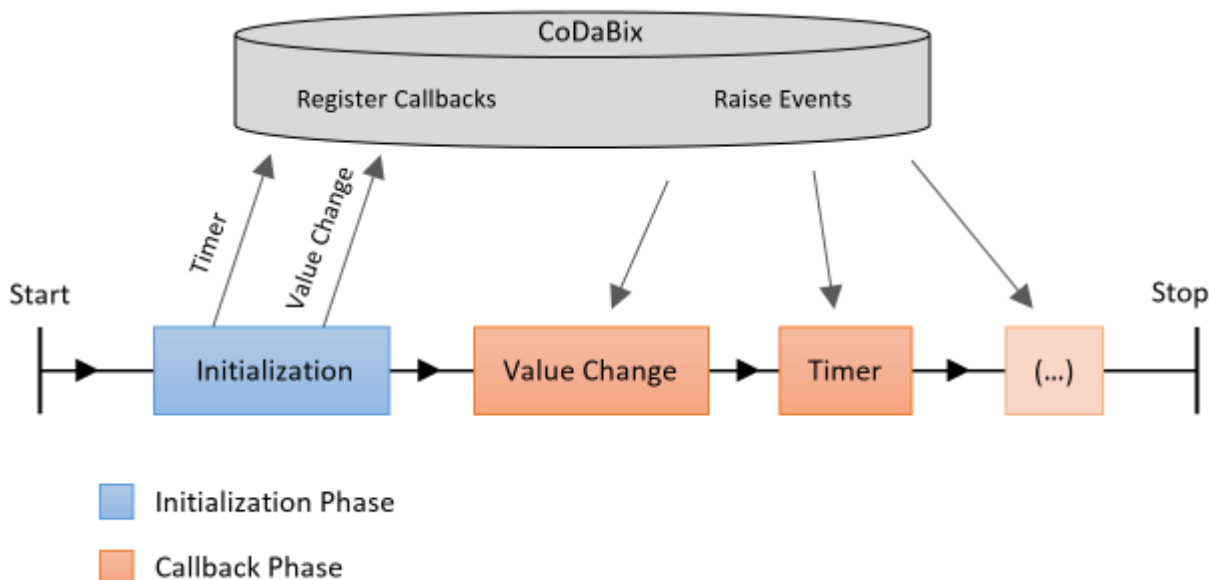
The default value of “Editor Strictness Level” is “Low”. We recommend this setting if you are just beginning with Scripts and JavaScript. If you are an experienced TypeScript developer, we recommend setting the value to “Medium” or “High” so that the editor can help you manage a clean code base.

**Note:** It can take up to 3 seconds until a change (e.g. enabling / disabling the script) will become effective and another 3 seconds until the CurrentScriptState and the Script Log are updated.

## Script Handling

Once you created a live version of a script (see section [Going Live](#)), it will automatically be started, as long as its state is set to “Enabled”. On startup, the script is in the “Initialization Phase”. During this phase, the script can register callbacks (e.g. for events or for a timer). When the callback is called, the script is in the “Callback Phase” (but the script can still register further callbacks in this phase).

The following diagram illustrates the phases of a script:



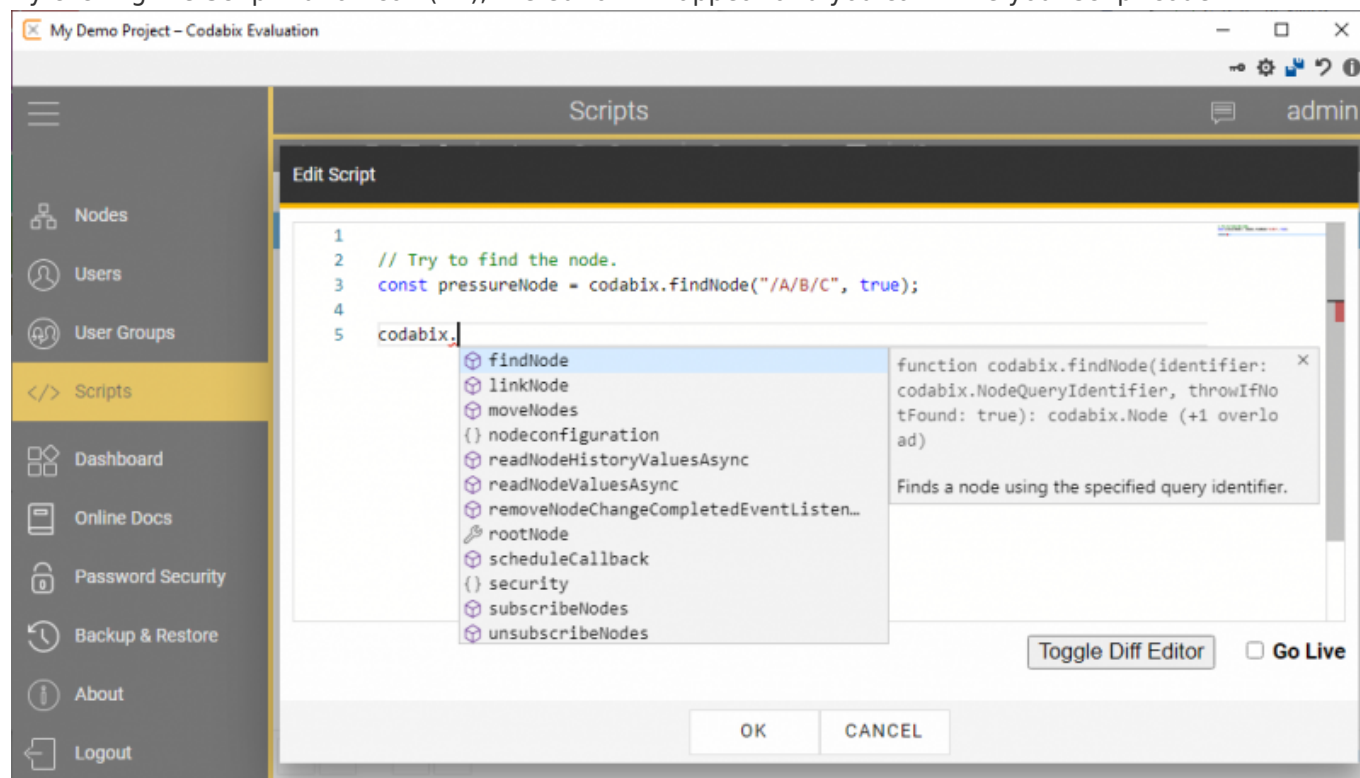
Note: Although not shown in the diagram above, the script can still register new callbacks for other events when it is already in the Callback Phase.

A timeout of about **15000 ms** is applied to the script to protect it against unintended infinite loops, for example `while (true) {}`. If a script is not finished after the timeout, it is stopped and treated as if an uncaught exception has occurred.

In both the Initialization Phase and the Callback Phase the script is automatically restarted after a short period of time (about 3 seconds) when an uncaught exception occurs.

# The Built-in Script Editor

By clicking the Script Editor icon (</>), the editor will appear and you can write your script code.



If you have already worked with Visual Studio or VS Code, the Script Editor will look familiar to you (in fact, the editor is based on the Monaco Editor from VS Code). The editor provides IntelliSense for UKI-4.0 API methods / interfaces and for built-in JavaScript classes as you type, as shown in the screenshot above.

If you hover on variable or method calls, a tooltip appears that shows the type:

```
// Write the value to the node. let randomNumber: number
codabix.writeNodeValueAsync(pressureNode, randomNumber);
```

If you have an error in your script, the editor will show red squiggly lines on it and show the error if you hover on it:

```
Property 'writeNNodeValueAsync' does not exist on type 'typeof codabix'.
// Write any
codabix.writeNNodeValueAsync(pressureNode, randomNumber);
```

When you right-click at a position in the code, a context menu appears with useful commands. For example, you can find all references to a specific variable in your code (and e.g. rename it):

```
11 // Write the value to the node.
12 codabix.writeNodeValueAsync(pressureNode, randomNumber);
```

References – 2 references

```
2 const pressureNode = codabix.findNode("/Nodes/Injection molding/Pressure");
3 if (pressureNode != null) {
4
5 // Set an interval timer that will call our function every 3 seconds.
6 timer.setInterval(function () {
7
8 // Generate a random number between 20 and 150.
9 let randomNumber = Math.random() * 130 + 20;
10
11 // Write the value to the node.
12 codabix.writeNodeValueAsync(pressureNode, randomNumber);
13
14 }, 3000);
15 }
16
17
```

let randomNumber = ...  
pressureNode, rando...

## Going Live

The Script Editor allows you to write and save code for the script (“draft”) without actually running it. Only when you select “Go Live”, the current draft script code will be saved as the “live version” and will actually be run. This allows you to progressively work on the script code without affecting the currently executed live version. With the “Toggle Diff Editor”, you can switch to a diff editor to compare your changes between the live version and your current draft.

Edit Script

```
1 // Try to find the node by specifying the node path.
2 const pressureNode = codabix.findNode("/Nodes/Injection mo
3 if (pressureNode != null) {
4
5 // Set an interval timer that will call our function e
6 timer.setInterval(function () {
7
8 // Generate a random number between 20 and 150.
9 let randomNumber = Math.random() * 130 + 20;
10
11 // Write the value to the node.
12 codabix.writeNodeValueAsync(pressureNode, randomNu
13
14 }, 3000);
15 }
16
17
```

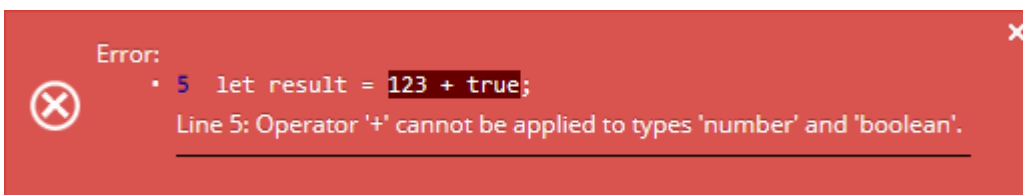
```
1 // Try to find the node by specifying the node path.
2 const pressureNode = codabix.findNode("/Nodes/Injection mo:
3 if (pressureNode != null) {
4
5 // Set an interval timer that will call our function e
6 timer.setInterval(function () {
7
8 // Generate a random number
9 // between 20 and 150.
10 let randomNumber = Math.random() * 130 + 20;
11
12 // Write the value to the node now.
13 codabix.writeNodeValueAsync(pressureNode, randomNu
14
15 }, 3000);
16 }
17
18
```

Toggle Diff Editor  Go Live

✓ ✕

Once you are finished with editing your script, make sure to check the “Go Live” checkbox and then click save. This will make your current script draft become the live version so that it is actually executed.

If you have a compile time error in your script when trying to go live, an error box will appear describing the error:



Otherwise, the Script Editor will close and the new script code will run after some seconds.

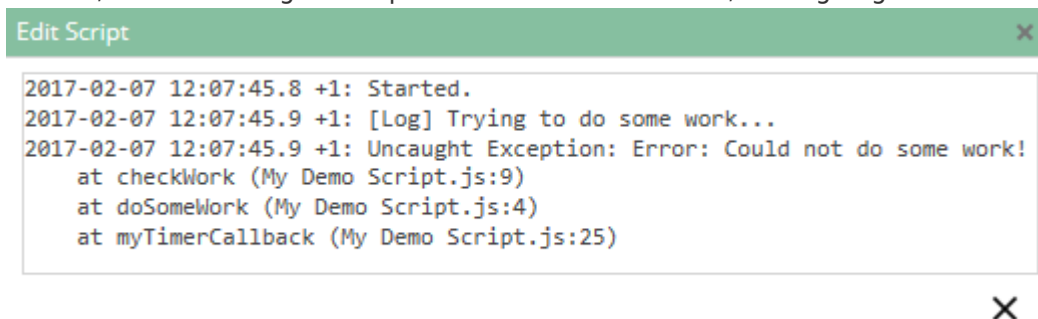
## Useful Shortcuts

- **Ctrl+F:** Find/Replace
- **Ctrl+F2:** Rename (change all occurrences)
- **Shift+F12:** Find all references
- **Ctrl+F12:** Go to definition
- **Ctrl+K, Ctrl+C:** Comment out the selected lines
- **Ctrl+K, Ctrl+U:** Uncomment the selected lines

## Viewing the Script Log

Each script has a log associated with it. When a script has been started (or an uncaught exception occurred), an entry will be made in the log. Additionally, you can create a log entry directly from the script code by calling `logger.log()`.

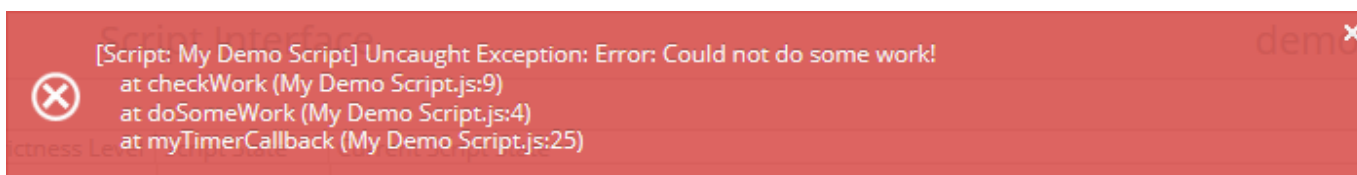
If you click on the log button (📄), a dialog with the log content will appear. For example, if the Script started, but an uncaught exception occurred in a callback, the log might look like this:



```
2017-02-07 12:07:45.8 +1: Started.
2017-02-07 12:07:45.9 +1: [Log] Trying to do some work...
2017-02-07 12:07:45.9 +1: Uncaught Exception: Error: Could not do some work!
  at checkWork (My Demo Script.js:9)
  at doSomeWork (My Demo Script.js:4)
  at myTimerCallback (My Demo Script.js:25)
```

In case an exception occurs, the log entry contains a stack trace showing the line numbers of the script (after the colon) that mark the position in the code at which the corresponding functions were executing when the exception occurred.

**Note:** When an uncaught exception occurs, an error message will also be shown in the Runtime Log:



```
[Script: My Demo Script] Uncaught Exception: Error: Could not do some work!
  at checkWork (My Demo Script.js:9)
  at doSomeWork (My Demo Script.js:4)
  at myTimerCallback (My Demo Script.js:25)
```

## Writing Script Code

### JavaScript Basics

Given below is a brief summarization of JavaScript basics. For a more detailed tutorial, please visit the [JavaScript Guide](#) on MDN.

In a script, you can declare variables that store values with `let` and `const` (`const` means the variable cannot be modified). You can assign values through the `=` operator (whereas `===` is used to check for equality):

```
let count = 123;  
const myText = "Hello World!";
```

JavaScript supports a number of basic value types:

- **number**: A number (which is a double precision floating point value) can represent both integers and decimals. You can use numbers to do calculations, e.g.:

```
let result = (2 + 0.5) * 3; // 7.5
```

- **boolean**: A boolean is either `true` or `false`. A boolean can be the result of comparison operators and used for control flows like `if`, `while` etc.
- **string**: A string can consist of an arbitrary number of characters and be used to represent text. Strings can be linked using the `+` operator:

```
let displayText = "The result of 5+6 is " + (5+6); // "The result of 5+6 is 11"
```

- **object**: An object stores properties that consist of a **key** (string) and a **value** (any value type). For example, the `UKI-4.0` object contains properties that are methods, e.g. `findNode`. Object properties are mostly accessed using dot (`.`) notation (`UKI-4.0.findNode(...)`, `UKI-4.0.rootNode`, ...).

You can use control flow statements to do comparisons:

```
let result = "The value is"  
if (count > 200) {  
    result += "greater than";  
}  
else if (count < 200) {  
    result += "lower than";  
}  
else {  
    result += "equal to";  
}  
result += " 200.";
```

You can create a function that will contain code which needs to be run more than once. For example, you could create a function that calculates the average of two values:

```
function average(value1, value2) {  
    return (value1 + value2) / 2;  
}  
  
// Calculate the average of 100 and 250 and write it to the Script Log.  
logger.log("The average of 100 and 250 is " + average(100, 250) + ".");
```

When you run this code, it will print something like this to the script's log:

```
2016-09-28 14:57:41.7 Z: [Log] The average of 100 and 250 is 175.
```

# Script API

The Script Interface Plugin provides the following API namespaces that can be used in a script:

- **UKI-4.0** : Contains all UKI-4.0 related functionality, e.g. to access and modify Nodes.
- **timer**: Contains methods to create a timer, so that you can let a function of your script be called back at a regular interval.
- **logger**: Contains a **log** method that allows you to write to the script log.
- **storage**: Allows you to persist information across restarts of the script.
- **io**: Provides I/O operations, e.g. [File Access](#).
- **net**: Provides network-related operations, e.g. to register an HTTP handler.
- **runtime**: Provides functions to interact with the script runtime environment.

Note: The Script Editor supports **IntelliSense**, so you can see which methods are available in the UKI-4.0 namespace just by typing UKI-4.0 . (notice the dot after “UKI-4.0 ”). Similarly, when a method returns an object (for example a Node), you can again type a dot after it to see which methods it has.

## Accessing UKI-4.0UKI-4.0 UKI-4.0UKI-4.0

### Find a Node and Log its Value

Let's assume you installed UKI-4.0 with the “Demo-Data (Continuous)” plugin and want to access the Node **Nodes → Demo-Data → Temperature**. To do this, you first need to get the Node path or the Identifier of the Node. To do this, open the Node view in the UKI-4.0 Web Configuration, select the relevant Node and click the **Access** symbol (🔑). Then copy the “Absolute Node Path”. We then specify this path in the UKI-4.0 `.findNode()` method as well as a `true` parameter so that the method throws an exception if the node could not be found:

```
// Find the "Temperature" node and check if the node
// has a value.
const temperatureNode = UKI-4.0 .findNode("/Nodes/Demo-Nodes/Temperature", true);
if (temperatureNode.value !== null) {
  // OK, node has a value. Now log that value.
  logger.log("Current Temperature: " + temperatureNode.value.value);
}
```

Your script log will then look like this:

```
2016-09-28 15:08:45.2 Z: Started.
2016-09-28 15:08:45.3 Z: [Log] Current Temperature: 71
2016-09-28 15:08:45.3 Z: Stopped.
```

However, in this example only one value is logged. This is because when the script is started, the code which finds the Node and logs the value is run, but after that the script is finished.

Now, if we want to log the value not only once but every 5 seconds, we can do this by creating a timer and supplying a function that the timer will call at a regular interval (note: Instead of `function () {...}`, for a callback you should use a **fat arrow function**: `() => {...}`).



```
// Find the "Temperature" node.
const temperatureNode = UKI-4.0 .findNode("/Nodes/Demo-Nodes/Temperature", true);

// Now create a timer that will log the node's value
// every 5 seconds.
const interval = 5000;
timer.setInterval(() => {

    // If the node has a value, log it.
    if (temperatureNode.value !== null) {
        logger.log("Current Temperature: " + temperatureNode.value.value);
    }

}, interval);
```

When you run this script your script Log will look like this after some seconds:

```
2016-09-28 15:15:42.6 Z: Started.
2016-09-28 15:15:47.6 Z: [Log] Current Temperature: 70
2016-09-28 15:15:52.6 Z: [Log] Current Temperature: 75
2016-09-28 15:15:57.6 Z: [Log] Current Temperature: 63
2016-09-28 15:16:02.6 Z: [Log] Current Temperature: 71
```

## Node Events

The example above uses a timer that calls a function in a regular interval. However, it is also possible to register for specific events of a Node:

- **ValueChanged:** Raised when a value has been written to the node (property `value`).  
**Note:** This event is also raised if the new value is equal to the previous value. To determine if the value has actually changed, you can check the `isValueChanged` property of the listener argument.
- **PropertyChanged:** Raised when a property (other than `value`) of the Node has been changed, e.g. `name`, `displayName` etc.
- **ChildrenChanged:** Raised when one or more children Nodes of the current Node have been added or removed.

You can handle the event by adding an **Event Listener** (callback) to the Node whose event you are interested in.

Example:

```
// Find the "Temperature" node and add a handler for the "ValueChanged" event.
const temperatureNode = UKI-4.0 .findNode("/Nodes/Demo-Nodes/Temperature", true);

temperatureNode.addValueChangedEventListener(e => {

    // Log the old and the new value of the node.
    logger.log("Old Value: " + (e.oldValue && e.oldValue.value)
        + ", New Value: " + e.newValue.value);

});
```

**Note:** You cannot (synchronously) read or write Node values (or do other changes to Nodes) from within an Node event listener. If you want to do this, use `UKI-4.0 .scheduleCallback()` to schedule a callback that is executed as soon as possible after the event listener is left.

## Write a Value to a Node

You can also write values to a Node from a script. For example, in the Node configuration, select the **“Nodes”** Node and create a datapoint Node with the name **“Counter”** and select **on Value Change** for **“History Options”**. Then create a script with the following code:

```
const counterNode = UKI-4.0 .findNode("/Nodes/Counter", true);

// Declare a counter variable.
let myCounter = 0;

// Set a callback that increments the counter and writes the
// current value to the node until the value is 5.
let timerID = timer.setInterval(() => {
  myCounter = myCounter + 1;
  UKI-4.0 .writeNodeValueAsync(counterNode, myCounter);

  if (myCounter == 5)
    timer.clearInterval(timerID);
}, 500);
```

Then, change back to the Nodes view, select the **“Counter”** Node and open the history values (📄):

History Values
✕

↻

Node	Category	Status	Receive Timestamp	Creation Timestamp	Value
[37] Counter	0	0	10.10.2016 11:25:38	10.10.2016 11:25:38	5
[37] Counter	0	0	10.10.2016 11:25:38	10.10.2016 11:25:38	4
[37] Counter	0	0	10.10.2016 11:25:37	10.10.2016 11:25:37	3
[37] Counter	0	0	10.10.2016 11:25:37	10.10.2016 11:25:37	2
[37] Counter	0	0	10.10.2016 11:25:36	10.10.2016 11:25:36	1

⏪
⏴
1
⏵
⏩
1-5 of 5

You can see now that the values 1, 2, 3, 4 and 5 have been written to the Node with a delay of 0.5 seconds.

Note: There is also an easier way of writing this code (withouth callbacks) using an **Async Function** as shown in the next chapter.

## Async Functions

The example shown in the previous chapter creates a timer by setting a callback. However, such code can quickly become confusing when you have more complex conditions. **Async Functions** allow to write the code in a simpler way as the code looks like synchronous code (without callbacks). UKI-4.0 provides a few async functions that can be recognized from their name ending in **Async**.

For example, the above code can be rewritten like this, using the `timer.delayAsync()` function:

```
const counterNode = UKI-4.0 .findNode("/Nodes/Counter", true);

// Write the values 1 to 5 to the node, and wait 0.5 seconds between each write.
for (let i = 1; i <= 5; i++) {
  await UKI-4.0 .writeNodeValueAsync(counterNode, i);
  await timer.delayAsync(500);
}
```

Notice using the keyword `await` here. **Await** means something like “interrupt the execution here until the asynchronous operation of the function has completed”. In this case the `delayAsync` function returns a **Promise** object that is fulfilled after 0.5 seconds have passed. As soon as the **Promise** is fulfilled the execution continues. Other async functions also return **Promise** objects, which may be fulfilled with a value, if applicable.

If you did not write `await`, your code would not wait 0.5 seconds but instead immediately continue with the next iteration. It is important to note that when `awaiting` an operation, in the meantime other code can still be executed, e.g. you may receive an event from the Node while your code pauses at the `await` position.

In the example above we also use `await` to wait for `writeNodeValueAsync`. Writing Node values may take some time if the Node is connected to a device, e.g. a S7 PLC device. In that case, execution would pause until the value has actually been written to the device. If you do not want to wait for that, you can also remove the `await` here (in which case you will need to write the `void` operator before the function call to signal to the compiler that the returned **Promise** has intentionally been discarded).

However, if you try this code directly it will not yet work because in order to use `await` you need to mark the surrounding function using the `async` keyword. If the surrounding async function is the **main** function, you should wrap it in a call to `runtime.handleAsync()` so that uncaught exceptions are not silently swallowed:

### Async-Template.js

```
runtime.handleAsync(async function () {

  // Your code here...

} ());
```

Here is a complete example of a Script using Async Functions:

```
runtime.handleAsync(async function () {

  for (let i = 0; i < 10; i++) {
    logger.log("Iteration " + i);
    await timer.delayAsync(1000);
  }

} ());
```

**Note:** If you want to use an async function as callback for an event listener, you should also wrap it in `runtime.handleAsync`, as shown in the following case where we handle the event when a Node gets a new value:

```
const myNode = UKI-4.0 .findNode("/Nodes/A", true);

myNode.addValueChangeListener(() => runtime.handleAsync(async () => {
  // Do async stuff...
}) ());
```

## Reading Node's Values Using a Synchronous Read

Another example of an async function is `UKI-4.0 .readNodeValuesAsync()`. This method invokes a [Synchronous Read](#) on a device and reading values from the device may take some time. Therefore you should use `await` to pause the execution until the resulting values arrive:

```
runtime.handleAsync(async function () {

  const node1 = UKI-4.0 .findNode("/Nodes/A", true);
  const node2 = UKI-4.0 .findNode("/Nodes/B", true);

  // Do a synchronous read, and pause until the values arrive from the device.
  let [nodeValue1, nodeValue2] = await UKI-4.0 .readNodeValuesAsync(node1, node2);

  logger.log("Node1 Value: " + (nodeValue1 && nodeValue1.value)
    + ", Node2 Value: " + (nodeValue2 && nodeValue2.value));

} ());
```

## File Access

The namespaces `io.file`, `io.directory` and `io.path` contain methods and classes to work with files and directories. For example, you can read and write text files, copy, move or delete files, and enumerate all files in a specific directory.

Note that file access is subject to the **Access Security** restrictions that have been defined in the [UKI-4.0 Project Settings](#).

Most of the I/O operations are implemented as **Async Functions** that return a **Promise** object. This is because I/O operations may take some time to complete (depending on the file system). In order to not block `UKI-4.0`, the I/O operations are run in the background. You can call them in the async functions using the `await` keyword.

**Note:** On Windows 10 Version 1511 and older (and Windows Server 2012 R2 and older), e.g. on Windows 7, the **maximum file path length** is limited to **260 characters** (`MAX_PATH`).

On Windows 10 Version 1607 and higher (as well as Windows Server 2016 and higher) longer path names can be used. However, for this you will need to enable the setting "Enable Win32 long paths" in the Windows Group Policy, see [Enabling Win32 Long Path Support](#).

## Basic File Operations

**Enumerate files of the UKI-4.0 project "log" directory:**

```
runtime.handleAsync(async function () {

    // Get the path to the UKI-4.0 "log" directory. We use the UKI-4.0 -defined environment
    // variable "%UKI-4.0 ProjectDir%" for this case.
    // combinePath is an OS independent way to combine path elements.
    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);

} ());
```

The result might look like this:

#### 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()`: Combines two or more path elements into one path in an OS independent way. For example, on Windows the path separator is `\`, while on Linux it is `/`. This method automatically uses the correct separator to combine the paths.
- `io.directory.GetFilesAsync()`: Returns a `string[]` array containing the file names of the specified directory (similarly, `io.directory.getDirectoriesAsync()` returns the subdirectory names).
- `io.file.copyFileAsync()`: Copies a file.
- `io.file.moveFileAsync()`: Renames or moves a file.
- `io.file.deleteFileAsync()`: Deletes a file.
- `runtime.expandEnvironmentVariables()`: Replaces the name of each environment variable (enclosed in two `%` characters) in the specified string with their value. UKI-4.0 defines the following environment variables in addition to the OS's variables:
  - UKI-4.0 `ProjectDir`: Contains the path to the currently used UKI-4.0 project directory.
  - UKI-4.0 `InstallDir`: Contains the path where UKI-4.0 has been installed.

## Reading and Writing Text Files

### Writing a text file in one step:

```
runtime.handleAsync(async function () {

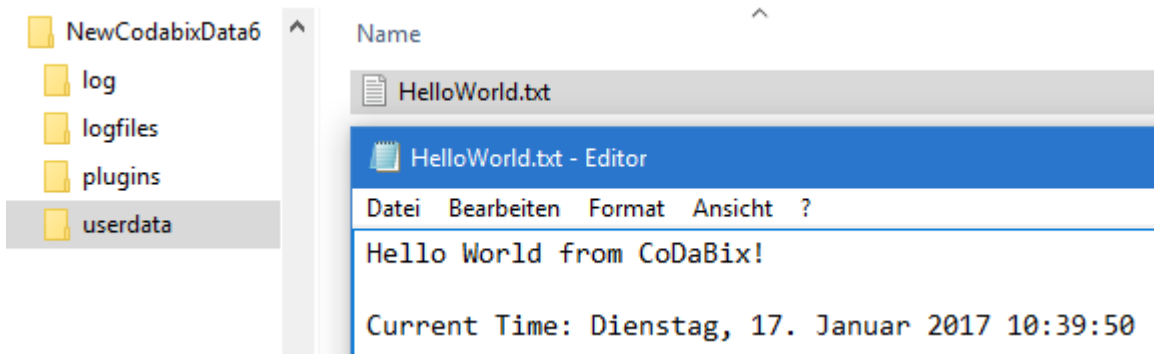
    // Create a string and write it into a textfile (HelloWorld.txt).
    let filePath =
    io.path.combinePath(runtime.expandEnvironmentVariables("%UKI-4.0 ProjectDir%"),
        "userdata", "HelloWorld.txt");
    let content = "Hello World from UKI-4.0 !\r\n\r\n" +
        "Current Time: " + new Date().toLocaleString();

    await io.file.writeAllTextAsync(filePath, content);

} ());
```

This will create a textfile `HelloWorld.txt` in the `userdata` directory of your UKI-4.0 project directory that

might look like this:



By calling `io.file.writeAllTextAsync()`, the file is written in one step (and with `io.file.readAllTextAsync()`, it is read in one step). However, you can also read or write text files on a line-by-line basis as in the following example:

### Reading a text file line-by-line:

```
runtime.handleAsync(async function () {

    // We want to read from the current UKI-4.0 Runtime Log file.
    const runtimeLogDir = io.path.combinePath(
        runtime.expandEnvironmentVariables("%UKI-4.0 ProjectDir%"), "log");
    const runtimeLogFiles = await io.directory.GetFilesAsync(runtimeLogDir);

    // The returned files are ordered ascending by their name; so we use the last
    // entry to get today's log file.
    const logFile = io.path.combinePath(runtimeLogDir,
        runtimeLogFiles[runtimeLogFiles.length - 1]);

    // Open the file using a Reader and read the first 5 lines.
    let result = "";
    let reader = await io.file.openReaderAsync(logFile);
    try {
        let lines = await reader.readLinesAsync(5);
        for (let i = 0; i < lines.length; i++) {
            // Append the line to the result string
            result += "\n[" + (i + 1) + "]: " + lines[i];
        }
    }
    finally {
        // Ensure to close the reader when we are finished.
        await reader.closeAsync();
    }

    // Log the result string.
    logger.log(result);

} ());
```

This code reads the first 5 lines of the current UKI-4.0 runtime logfile and prints it to the 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 Handlers

The namespace `net` provides methods to register **HTTP Handlers**, which you can use to specify a script function that shall be called every time a client sends an HTTP request to the UKI-4.0 web server. For example, you can dynamically generate HTML pages, similar to PHP or ASP.NET. Additionally, you can handle incoming **WebSocket connections** (see next chapter).

To register a HTTP handler you must call the `net.registerHttpRequestRoute()` method and pass a path as well as a callback to it. The callback will then be called each time an HTTP request for the URL path `/scripthandlers/<path>` arrives, where `<path>` represents the registered path.

Note that a particular path can only be registered once across all scripts at the same time. If another script has already registered a handler for the same path, the method throws an exception.

Also note that the path must be specified as “raw”, meaning it should be URL-encoded. For example, if the user should be able to enter `/scripthandlers/März` as address in the browser, you would need to specify `M%C3%A4rz` as path.

The callback needs to be a function that returns a `Promise` object (which happens automatically when using an **async function**). When a HTTP request arrives, the function is called, and the HTTP request stays active until the Promise object is “fulfilled” (the async function returns). The callback gets a `net.HttpContext` as parameter which contains the properties `request` and `response` in order to access objects representing the request of the client and the response to the client.

When the callback throws an exception (or rejects the returned `Promise`), the request is aborted and a warning is logged to the runtime log, but the script continues to run.

### Dynamically Generate a Text Page

Imagine we want to output the current time and the current number of UKI-4.0 Nodes as text page when the user enters `http://localhost:8181/scripthandlers/hello-world` in the browser on the current machine (provided that the **Local HTTP Port** is set to the default value of 8181). To do this, the following script code registers an HTTP handler for this path which outputs a text containing the information when requested:

```
runtime.handleAsync(async function () {

  net.registerHttpRequest("hello-world", async ctx => {
    let response = ctx.response;

    // Create a text with the current date/time and the number of UKI-4.0 nodes.
    let text = "Hello World! Time: " + new Date().toLocaleTimeString() + "\n\n" +
      "Node Count: " + countNodes(UKI-4.0.rootNode.children);

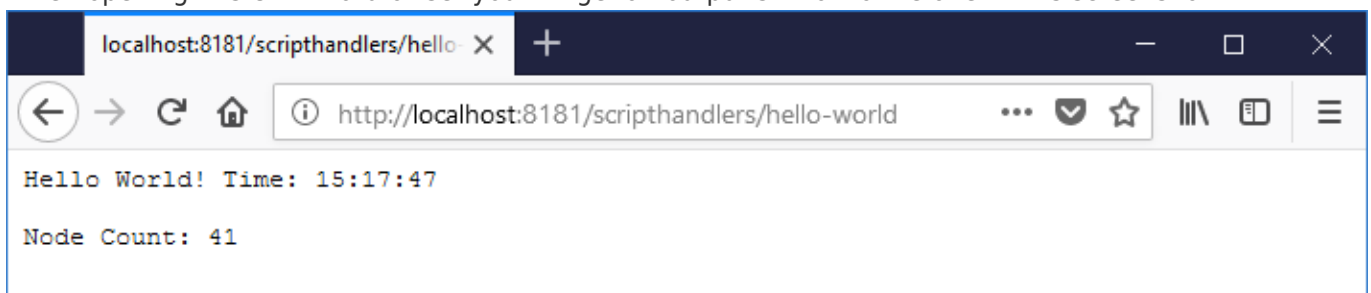
    // Set the Content-Type to the plain text format so that the browser knows
    // how to display the document.
    response.contentType = "text/plain";

    // Finally, write the generated text into the 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;
  }

}());
```

When opening this URL in a browser you will get an output similar to the one in this screenshot:



When refreshing the page with F5, you should see the current date and time being displayed.

The script uses the method `writeBodyCompleteAsync()` of the `response` object to write the generated text to the browser (using UTF-8 as text encoding). It is recommended to use this method instead of `writeBodyAsync()` when you can generate the whole response text in the script. In contrast, you can use `writeBodyAsync()` when you only want to generate small pieces of the response text and send them immediately to the client.

## Generate an HTML page with an input form

Imagine we want to create an HTML page where the user can enter a node path. When sending the form, the value of the specified node shall be read using a synchronous read and then be output on the page. This is done by the following script code:

```
runtime.handleAsync(async function () {

  const readNodeHandlerPath = "read-node-value";
  net.registerHttpRequest(readNodeHandlerPath, async ctx => {
    let request = ctx.request;
    let response = ctx.response;
```



```

    // Create an HTML page with a form in order to enter a node path.
    // When entered, we find the node and start a synchronous read.

    // Check whether the form has already been sent. If not, we write
    // an example path in the text box.
    const inputNodePathName = "nodePath";
    const inputNodePathValue = request.queryString[inputNodePathName];
    let resultMessage = "";

    if (inputNodePathValue !== undefined) {
        // The form has been sent, so find the node now.
        let node = UKI-4.0 .findNode(inputNodePathValue);
        if (!node) {
            // We couldn't find the node.
            resultMessage = `Error! Node with path '${inputNodePathValue}' was not
found!`;
        }
        else {
            // Start a synchronous read... (This may take a while,
            // depending on the device.)
            // The HTTP request stays active during that time.
            let resultValue = (await UKI-4.0 .readNodeValuesAsync(node))[0];
            if (resultValue == null) // The node doesn't yet have a value
                resultMessage = `Result: Node does not have a value!`;
            else
                resultMessage = `Value of Node '${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>`;

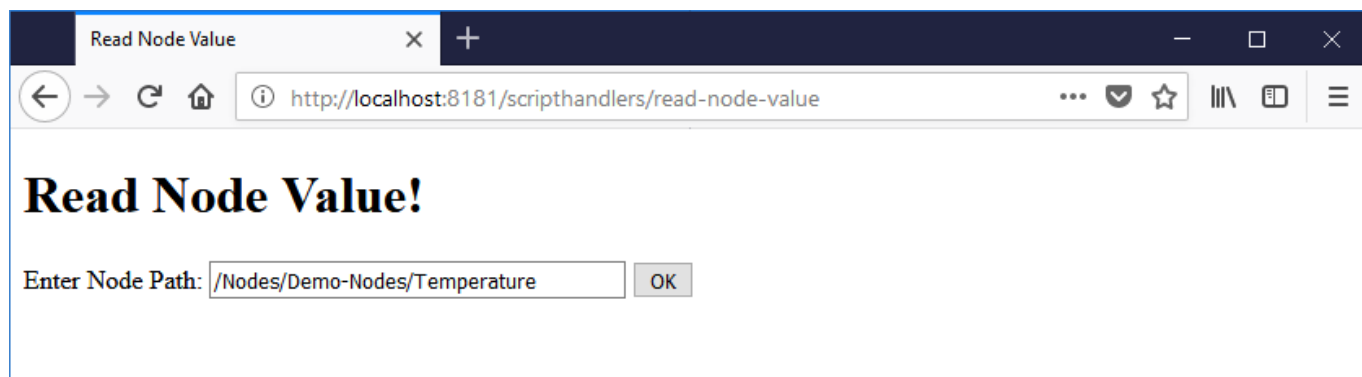
    // Set the Content-Type to HTML, and write the generated HTML into the response.
    response.contentType = "text/html";
    await response.writeBodyCompleteAsync(html);
  });
}());

```

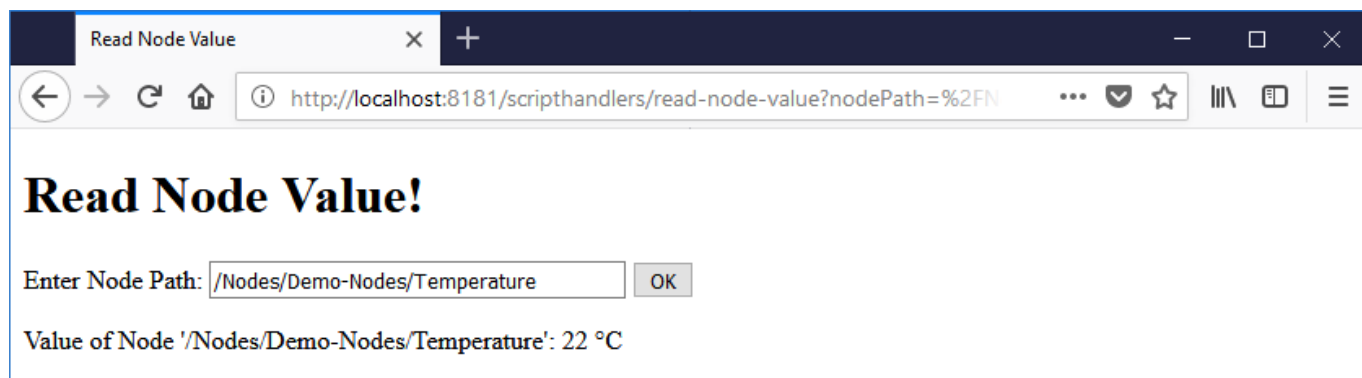
**Note:** The script uses the method `io.util.encodeXml()` in order to encode strings so that they can safely

be placed within HTML or XML text (or attribute values), without providing an attack surface for HTML injection or Cross-Site-Scripting (XSS). This is especially important when outputting strings in an HTML page that might originate from the user or external sources.

When you now open the URL <http://localhost:8181/scripthandlers/read-node-value> in the browser, the following form is shown:



When clicking on the “OK” button (and you have the Demo Data plugin installed), the current value of the temperature demo node will be shown:

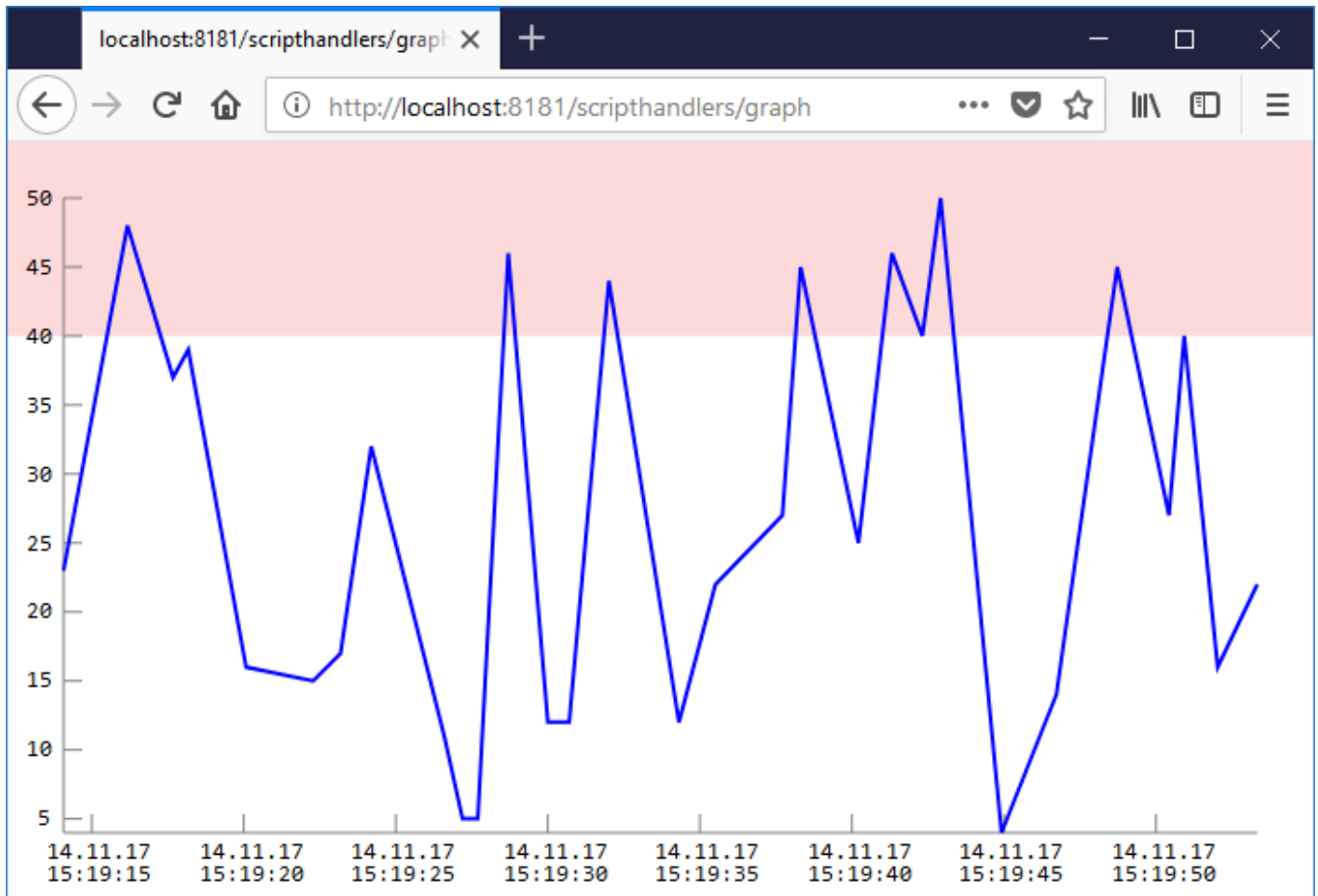


This means the script has reacted on the sent form and has output the current value of the node. Because the form uses the **GET** method, the parameter is visible in the URL as appended query string: `?nodePath=%2FNodes%2FDemo-Nodes%2FTemperature`

Similarly, you could enter a path to a S7 variable, an OPC UA Client variable, and so on. In these cases, not only the currently stored value of the node is output, but a synchronous read is started by the call to method UKI-4.0 `.readNodeValuesAsync()` which reads the value from the device and then outputs it in the HTML page.

### Display History Values as Diagram

With an HTTP handler you can also generate an SVG image that displays history values in a diagram (in the following example, using the Gradient demo node):



This can be done with the following script code:

```
// 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.registerHttpRequest("graph", async ctx => {
        const response = ctx.response;

        // Read the latest 50 history values of the gradient demo node, and
        // then display them in an SVG diagram.
        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()
            });
        }

        // Generate the SVG diagram.
        let resultString: string;
        try {
            resultString = SvgLibrary.generateValueChartSvg(svgValues, 700, 400, undefined,
40);

            // This worked, so set the Content-Type to SVG.
            response.contentType = "image/svg+xml";
        }
        catch (e) {
            // Output the error.
            resultString = String(e);
            response.statusCode = 500;
            response.contentType = "text/plain";
        }

        await response.writeBodyCompleteAsync(resultString);
    });
});

```

```
}());
```

## Extended HTTP Programming

The previously shown examples generate an HTML page or an SVG image that displayed in the browser, but doesn't change afterwards. To create dynamic HTML5 apps, you could also create a static HTML page with a JavaScript/TypeScript, which communicates with a script at the UKI-4.0 side, e.g. using JSON (to serialize and deserialize JSON you can use `JSON.stringify()` and `JSON.parse()`). This way, the script in the browser can regularly request new information.

The next chapter describes WebSocket connections which your HTML page can use to create bidirectional, socket-like connections in order to communicate with a UKI-4.0 Script.

## WebSocket Connections in an HTTP Handler

In a UKI-4.0 Script you can also run **server-side WebSocket connections**, that are created e.g. in a browser script using the **WebSocket** class that is available there. WebSocket allows to send data in both directions (server to client and client to server) at any time as long as the connection is established, in contrast to regular HTTP requests where the client would need to constantly send new HTTP requests in order to check if the server has any new data ("polling"). For example, in a UKI-4.0 Script you could add an event listener for the `ValueChanged` event to a node, and every time a new value is written to the node, send that new value using WebSocket to all currently connected browsers for displaying the value

In order to accept a WebSocket connection, first check with `ctx.isWebSocketRequest` if the request is actually a WebSocket request. If yes, call `ctx.acceptWebSocketAsync()` which returns a `net.RawWebSocket` object. On this object you can call `receiveAsync()` to receive message, and call `sendAsync()` to send message. The `Promise` object returned by these methods is fulfilled once the send/receive operation has completed. In case an error occurs when sending or receiving (e.g. when the connection has already been closed), the methods throw an exception (or reject the returned `Promise`).

**Note:** On a specific `RawWebSocket` object you can send and receive messages at the same time (meaning both a `receiveAsync()` and a `sendAsync()` operation can be outstanding).

For the following WebSocket examples, we use a static HTML page that establishes a WebSocket connection to UKI-4.0. Therefore, please first copy the following file UKI-4.0 `websocket.html` into the `webfiles` folder in your project directory:

### 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>
```

You should now be able to open this page when opening [http://localhost:8181/webfiles/UKI-4.0 WebSocket.html](http://localhost:8181/webfiles/UKI-4.0%20WebSocket.html) in your browser (provided that the Local HTTP Port for the UKI-4.0 web server in the UKI-4.0 Project Settings is set to the default value of 8181 and the option "Server Static Web Files" has not been disabled):



## Simple Echo WebSocket

In the simplest case you can receive messages from a WebSocket, and send them back directly to the client, as shown in the following UKI-4.0 Script:

```

runtime.handleAsync(async function () {

  const maxLength = 10000;

  net.registerHttpRequest("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(maxLength);
        if (message == null || message.length == maxLength)
          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);
      }
    }
  });
}());

```

The HTTP handler accepts a WebSocket connection, and then reads messages from the client and sends them back, until the client closes the connection.

In the HTML page, you can establish a connection with the “Connect” button and send messages to UKI-4.0 , which should be sent back immediately:

## "Chat" with Background Send Operations

Imagine we want to use WebSocket to send the current value of the Temperature demo node to all connected users as soon as a new value is written in UKI-4.0 . Additionally, users should be able to chat with one another.

Because the `sendAsync()` method (asynchronously) blocks until the data has been fully sent, we cannot just call it every time we want to send a new temperature value or a chat message, because the previous

message might not yet have been sent completely to the client. Instead, we use a **queue** where the messages to be sent are put. Another script function then removes a message from the queue and sends them using the WebSocket. Once the send operation is completed, it continues with removing the next message from the queue, or waits until at least one new message is in the queue.

In the following script code, this is done by the class `WebSocketHandler` which you can also reuse for other scripts.

(In case you have run the previously shown Echo WebSocket script, please disable it, as both scripts would register the same HTTP path.)

```
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
      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 occurred, 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.registerHttpRequest("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);
    });
  }
}

}());

```

Once a user connects, it is assigned a user name (e.g. User-6) and a message is sent to all connected users. The new user also receives the current value of the temperature node. As soon as a new value is written to the temperatur node or one of the users sends a chat message, it is send to all connected users (practically a broadcast):

## Sending HTTP Requests

The namespace `net.httpClient` provides methods that allow you to send a HTTP request to another server. This allows you e.g. to call external REST APIs using JSON objects.

**Important:** When sending requests over the internet (or other potentially insecure networks), please make sure that you always use `https:` URLs if possible, as `http:` URLs use an insecure connection and therefore do not provide server authenticity and data confidentiality/integrity. This is especially important when sending confidential login credentials.

**Note:** Currently, the request and response bodies can only use text data, not binary data.

### Simple GET requests

The following code issues a simple GET request and logs the response body (if present):

```
let response = await net.httpClient.sendAsync({
  url: "https://www.UKI-4.0.com/en/start"
});

if (response.content) {
  logger.log("Result: " + response.content.body);
}
```

## Accessing a JSON-based REST API

When you want to access an URL that can return a JSON result, you can use `JSON.parse()` to convert the JSON response body string into a JavaScript object, and access it.

The following example accesses an external REST API to get weather data (temperature) in a regular interval, and stores it in the `/Nodes/Temperature` node:

```
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);
}
```

The following code demonstrates how to send a POST request to the UKI-4.0 -internal REST API (for `<encrypted-password>` you will need to specify the encrypted user password):

```
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...
```



# Recommended Best Practices

The following list gives some recommendations for performant and clean scripts:

- If you are not writing a Library Script, place your whole code inside an IIFE (immediately-invoked function expression) as follows:

## Blank Template.js

```
runtime.handleAsync(async function () {  
  
    // Your code here...  
  
} ());
```

This avoids polluting the global scope and the TypeScript compiler can detect unused local variables and does not complain about using anonymous types in exported variables.

We recommend to mark the function with the `async` attribute so that you are able to call asynchronous functions using the `await` keyword. In that case you should pass the returned `Promise` object to `runtime.handleAsync()` to ensure uncaught exceptions are not silently swallowed, as shown in the code example above.

- Always use `let` or `const` to declare variables, not `var` as the latter is not block-scoped but function-scoped.
- Instead of using `arguments` which is a “magic”, array-like object, use rest parameters (`...args`) which is a real `Array` instance and doesn't contain previous function parameters.
- When enumerating an array, don't use `for-in` - use `for-of` instead, as the former doesn't guarantee any enumeration order.
- When throwing an exception, don't throw a primitive value like `throw "My Error Message";`. Instead, create and throw `Error` object instances, like `throw new Error("My Error Message");`. This ensures that you can retrieve the stack trace from where the exception occurred.
- If possible, avoid creating closures in code that is called very often (e.g. in a loop), because creating a closure is expensive in JavaScript. Instead, check if you can reuse a function by passing arguments.

For example, when you want to handle a node's `ValueChanged` event and write a new value in it (for which UKI-4.0 `.scheduleCallback()` must be used), avoid the following code that creates a closure every time the event is invoked:

```
myNode.addValueChangedEventListener(e => UKI-4.0 .scheduleCallback(() => {  
    void UKI-4.0 .writeNodeValueAsync(otherNode, e.newValue);  
}));
```

Instead, you can create a closure once and then using the following variant of UKI-4.0 `.scheduleCallback()` in the event listener that allows to pass arguments:

```
const scheduledHandler = (newVal: UKI-4.0 .NodeValue) => {  
    void UKI-4.0 .writeNodeValueAsync(otherNode, newVal);  
};  
myNode.addValueChangedEventListener(e => UKI-4.0 .scheduleCallback(scheduledHandler,  
e.newValue));
```

# Storing Passwords Securely

In some cases, you need to store passwords in script code, for example to access external password-protected HTTP services. To avoid having to store a password in plaintext, you can first encrypt it once using the UKI-4.0 Web Configuration by opening the menu item “Password Security”, and then at the start of your script code you can decrypt it by calling UKI-4.0 `.security.decryptPassword()`. This means that the password is **encrypted** with the **password key** of the back-end database which was generated using a cryptographically secure random number generator when creating the database.

The same mechanism is also used when storing passwords in datapoint nodes of type “Password”, which e.g. is used by device plugins.

This provides additional protection:

- Passwords are not displayed in clear text, so you can show Script code to other people, without allowing them to directly read the password.
- If you disable the option *Include Password Key* when creating a backup, such passwords cannot be extracted (decrypted) from the backup.

**Note:** You can generate a new password key in the UKI-4.0 Settings, which means passwords that were encrypted with the previous key are no longer readable.

For example, if you wanted to use the password “test” in your Script code, open the menu item “Password Security” in the UKI-4.0 Web Configuration, and then enter the password. After clicking on “Encrypt”, the encrypted password will be printed.

Example:

Password:    
Encrypted Password: `FNjeFt5k85PbWSly1q2h/Ctf7RqTW2JToUTNo/cwWFrL7oOUXJBewshFutKLosDJh6C8pGGgEa6m4MZqBF1RiQ==#`

You can now copy the encrypted password and store it in the Script code where you need it:

```
// Decrypt the password that we need to run a HTTP request.
const myPassword = UKI-4.0 .security.decryptPassword(
    "K8D/VWnVQG45HSF1D1G94RwXzrSxH3ARLzzhHekoAdf+prcwe5y6S4FhPUug1Ycw#" );

// ...

function doRequest() {
    let request = {
        user: "myUser",
        password: myPassword
    }
    // TODO: Send the request...
}
```