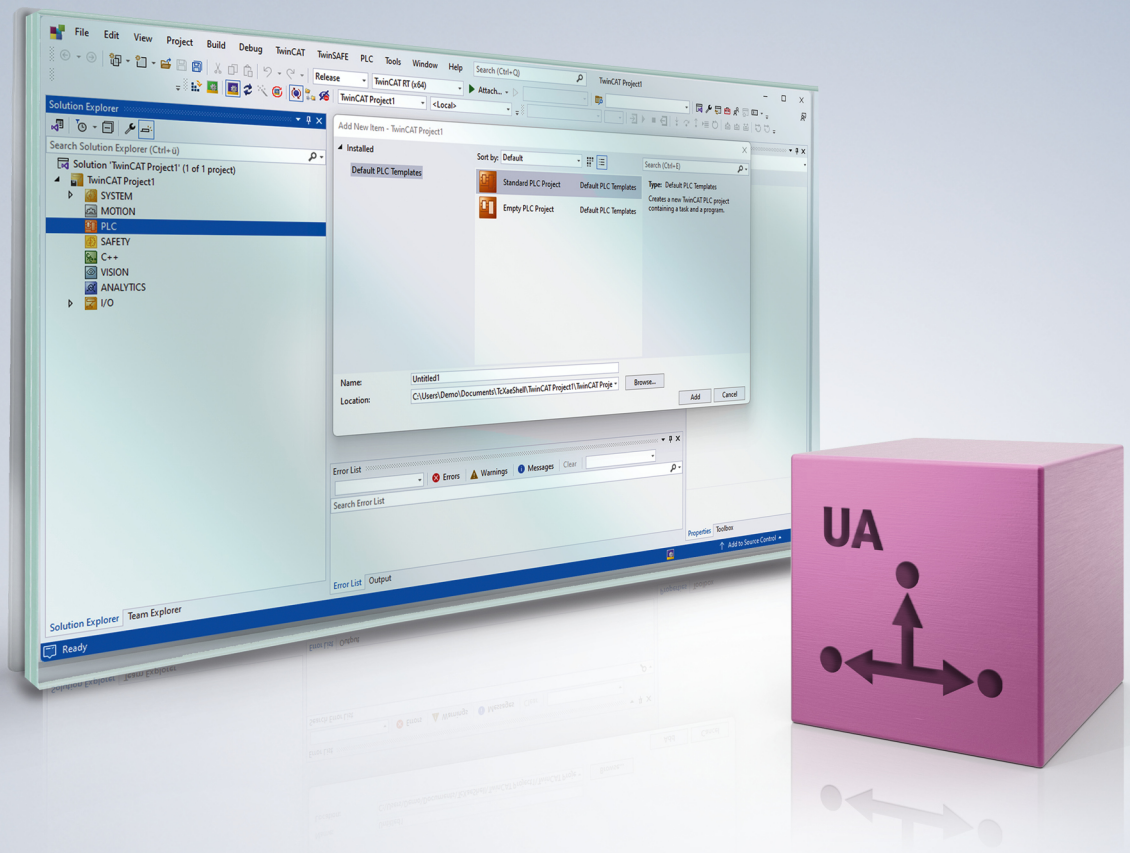
BECKHOFF New Automation Technology

Manual | EN

TF6100

TwinCAT 3 | OPC UA Server



2026-09-02 | Version: 1.6.0

Table of contents

1 Foreword 7
1.1 Notes on the documentation 7
1.2 For your safety 8
1.3 Notes on information security 9
1.4 Documentation issue status 9
2 Overview 10
2.1 TF6100 vs. TF6105 11
3 Installation 14
3.1 System requirements 14
3.2 Installation 15
3.3 Update installation 19
3.4 Installation variants 19
3.5 Container 21
3.6 Licensing 23
4 Technical introduction 27
4.1 Quick start 27
4.2 Quick Start (TwinCAT 2) 30
4.3 Initialization 34
4.4 Recommended steps 41
4.5 Supported features 43
4.6 Software architecture 46
4.7 Configurator 47
4.8 Optimizations 49
4.9 Application directories 54
4.10 Namespaces 55
4.11 Data Access 56
4.11.1 Overview 56
4.11.2 Connection with the runtime 56
4.11.3 Enabling symbols 59
4.11.4 Nodesets 76
4.11.5 Data types 77
4.11.6 Arrays 79
4.11.7 Enums 81
4.11.8 Structures 82
4.11.9 Properties 85
4.11.10 StatusCode 86
4.11.11 AnalogItemType 89
4.11.12 Description 90
4.11.13 ReadOnly 91
4.11.14 Pointers and references 92
4.11.15 Type system 92
4.11.16 DI Components 96
4.11.17 DeviceState 97

4.11.18 ServerState ... 97
4.12 Historical Access ... 98
4.12.1 Overview ... 98
4.12.2 Supported functions ... 99
4.12.3 Configuration ... 100
4.12.4 HistoryUpdate ... 101
4.12.5 TwinCAT Analytics ... 102
4.12.6 Access to historical data ... 102
4.13 Alarms and Conditions ... 105
4.13.1 Overview ... 105
4.13.2 Supported functions ... 105
4.13.3 Configuration ... 106
4.13.4 Additional application data ... 109
4.13.5 Access to alarms and events ... 111
4.14 Method calls ... 112
4.14.1 Overview ... 112
4.14.2 Job methods ... 112
4.14.3 RPC methods ... 115
4.15 TwinCAT EventLogger ... 117
4.15.1 Overview ... 117
4.15.2 Configuration ... 118
4.15.3 Access to alarms and events ... 118
4.16 Global Discovery Server ... 123
4.16.1 Overview ... 123
4.16.2 Push ... 123
4.16.3 Pull ... 124
4.17 Security ... 128
4.17.1 Overview ... 128
4.17.2 Endpoints ... 128
4.17.3 Certificate exchange ... 129
4.17.4 Authentication ... 131
4.17.5 Access rights ... 133
4.18 File Transfer ... 136
4.18.1 Overview ... 136
4.18.2 Configuration ... 137
4.19 Reverse Connect ... 138
4.20 Redundancy ... 141
4.20.1 ServiceLevel ... 144
4.21 Logging ... 145
4.22 System Tray ... 146
5 PLC API ... 148
5.1 Tc2_OpcUa ... 148
5.1.1 Data types ... 148
5.1.2 Function blocks ... 149
5.2 Tc3_OpcUa ... 151
5.2.1 Data types ... 151

5.2.2 Function blocks 158
6 Samples 160
7 Tutorials 161
8 Appendix 162
8.1 Attributes and comments 162
8.2 32-bit and 64-bit process 163
8.3 Error diagnosis 165
8.4 ADS Return Codes 168
8.5 Support and Service 171

1 Foreword

1.1 Notes on the documentation

This description is intended exclusively for trained specialists in control and automation technology who are familiar with the applicable national standards.
The documentation and the following notes and explanations must be complied with when installing and commissioning the components.
The trained specialists must always use the current valid documentation.

The trained specialists must ensure that the application and use of the products described is in line with all safety requirements, including all relevant laws, regulations, guidelines, and standards.

Disclaimer

The documentation has been compiled with care. The products described are, however, constantly under development.
We reserve the right to revise and change the documentation at any time and without notice.
Claims to modify products that have already been supplied may not be made on the basis of the data, diagrams, and descriptions in this documentation.

Trademarks

Beckhoff®, ATRO®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, MX-System®, Safety over EtherCAT®, TC/BSD®, TwinCAT®, TwinCAT/BSD®, TwinSAFE®, XFC®, XPlanar® and XTS® are registered and licensed trademarks of Beckhoff Automation GmbH.
If third parties make use of the designations or trademarks contained in this publication for their own purposes, this could infringe upon the rights of the owners of the said designations.





EtherCAT® is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany.

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.
The distribution and reproduction of this document, as well as the use and communication of its contents without express authorization, are prohibited.
Offenders will be held liable for the payment of damages. All rights reserved in the event that a patent, utility model, or design are registered.

Third-party trademarks

Trademarks of third parties may be used in this documentation. You can find the trademark notices here: https://www.beckhoff.com/trademarks.

1.2 For your safety

Safety regulations

Read the following explanations for your safety.
Always observe and follow product-specific safety instructions, which you may find at the appropriate places in this document.

Exclusion of liability

All the components are supplied in particular hardware and software configurations which are appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation, and drive technology who are familiar with the applicable national standards.

Signal words

The signal words used in the documentation are classified below. In order to prevent injury and damage to persons and property, read and follow the safety and warning notices.

Personal injury warnings

⚠ DANGER
Hazard with high risk of death or serious injury.

⚠ WARNING
Hazard with medium risk of death or serious injury.

⚠ CAUTION
There is a low-risk hazard that could result in medium or minor injury.

Warning of damage to property or environment

NOTICE
The environment, equipment, or data may be damaged.

Information on handling the product



This information includes, for example:
recommendations for action, assistance or further information on the product.

1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at https://www.beckhoff.com/secinfo.

1.4 Documentation issue status

Version	Change
1.6.0	**New:** TF6100 vs. TF6105 [▶ 11]
1.5.0	Manually adding the TOFU user [▶ 40] Error diagnosis [▶ 165]: New error recorded **New:** Installation [▶ 15]: information on: TwinCAT 3 Usermode Runtime, Beckhoff RT Linux® and TwinCAT/BSD Container [▶ 21] Namespaces [▶ 55] ServiceLevel [▶ 144] Tc3_OpcUa [▶ 151]
1.4.x	**New:** Strings [▶ 78]
1.3.x	Installation/System requirements [▶ 14]: Change system requirements server
1.2.x	**New:** Installation [▶ 15]: TwinCAT 2 Setup Licensing [▶ 23]: Licensing under TwinCAT 2 Redundancy [▶ 141]
1.1.0	**New:** Quick Start [▶ 30] (TwinCAT 2) Supported features [▶ 43] Tutorials [▶ 161]

2 Overview

OPC Unified **A**rchitecture (OPC UA) is the next generation of the familiar OPC standard. This is a globally standardized communication protocol via which machine data can be exchanged irrespective of the manufacturer and platform. OPC UA already integrates common security standards directly in the protocol. Another major advantage of OPC UA over the conventional OPC standard is its independence from the COM/DCOM system.

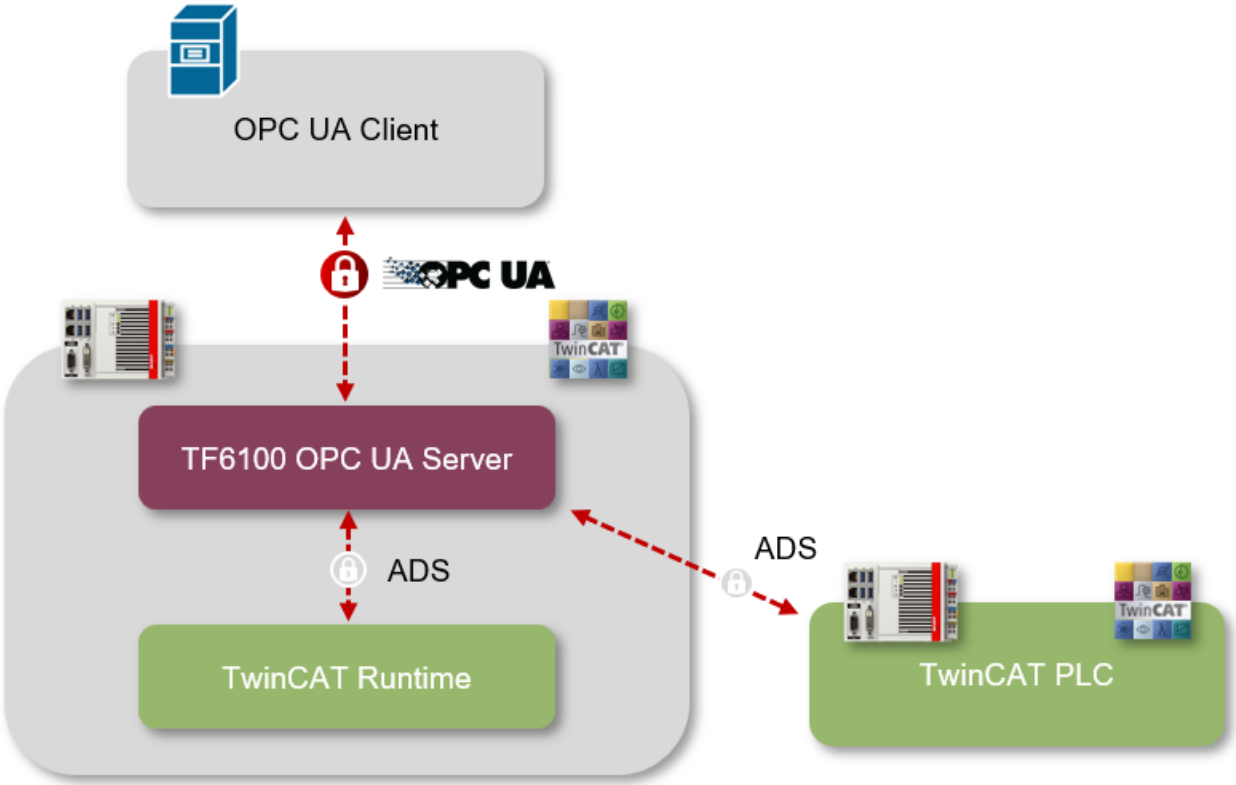


Detailed information on OPC UA can be found on the web pages of the OPC Foundation.

The TwinCAT 3 Function TF6100 OPC UA consists of several software components that enable data exchange with TwinCAT based on OPC UA.
The following table provides an overview of the individual product components.

Software component	Description
TwinCAT OPC UA Server	Provides an OPC UA Server interface so that UA clients can access the TwinCAT runtime.
TwinCAT OPC UA Client	Provides OPC UA Client functionality to enable communication with other OPC UA Servers based on PLCopen-standardized function blocks and an easy-to-configure I/O device.
TwinCAT OPC UA Configurator	Graphical user interface for configuring the TwinCAT OPC UA Server.
TwinCAT OPC UA Sample Client	Graphical sample implementation of an OPC UA Client in order to carry out a first connection test with the TwinCAT OPC UA Server.
TwinCAT OPC UA Gateway	Wrapper technology that provides both an OPC COM DA Server interface and OPC UA Server aggregation capabilities.

This documentation describes the TwinCAT OPC UA Server, which is an OPC UA server application that enables access to symbols from the TwinCAT real-time environment. The server can be operated either directly on the controller or on a gateway PC.
Both relationships are shown in simplified form in the following diagram and are explained in more detail in chapter Installation variants [▶ 19].



2.1 TF6100 vs. TF6105

With TF6100 TwinCAT 3 OPC UA and TF6105 TwinCAT 3 OPC UA Pub/Sub, Beckhoff offers two TwinCAT 3 functions for OPC UA communication. Both products are based on OPC UA, but they implement different communication models and address different requirements.

TF6100 is primarily used when applications require service-oriented access to an OPC UA information model. Clients can establish a connection to a server, browse its address space, read and write values, subscribe to value changes, call methods, and use other OPC UA functions.

TF6105 is designed for the cyclic or event-driven distribution of predefined datasets. Publishers send messages without having to process individual read requests for each recipient. Depending on the selected transport method, the messages are distributed directly via UDP or via an MQTT message broker.

The two products therefore complement each other. Which product is suitable for a particular application depends primarily on how information is to be identified, retrieved, and exchanged.

TF6100: Service-oriented access with OPC UA client/server

TF6100 implements the established OPC UA client/server communication model. An OPC UA client establishes a secure connection and session with an OPC UA server and then uses standardized services to interact with the server's address space.

The TF6100 product package includes the following components:

- TwinCAT OPC UA Server, which makes data and functions from TwinCAT runtime environments available via an OPC UA address space.
- TwinCAT OPC UA Client, which allows TwinCAT applications to access remote OPC UA servers.
- TwinCAT OPC UA Gateway, which can aggregate subordinate OPC UA servers and provide an OPC COM DA integration.

TwinCAT OPC UA Server can represent PLC variables and other TwinCAT runtime data as nodes in a structured address space. Clients can examine the available nodes and their metadata at runtime. It is therefore not necessary to specify in advance which elements are to be exchanged.

Typical client/server operations include:

- browsing the address space
- reading and writing variable values
- creating subscriptions for value changes
- calling methods
- receiving alarms and events
- accessing historical data
- examining data types, references, and semantic information
- importing companion specifications

This makes the TF6100 particularly well-suited for communication with HMIs, SCADA systems, MES and ERP applications, engineering tools, diagnostic applications, and other systems that require flexible access to machine information.

TF6105: Distribution of datasets using OPC UA Pub/Sub

TF6105 implements the publisher/subscriber communication model defined in Part 14 of the OPC UA specification. The product is integrated into the TwinCAT runtime via the so-called OPC UA RT device and supports both publisher and subscriber configurations.

A publisher aggregates selected process values into a PublishedDataSet and sends the corresponding NetworkMessages at a configured interval. Subscribers receive the messages, identify the relevant dataset, decode its fields, and link the received values to their local process image.

TF6105 supports two basic transport modes:

- UDP with UADP encoding for direct communication without a broker, including unicast and multicast scenarios.
- MQTT with UADP or JSON encoding for broker-based distribution across network boundaries as well as in IT or cloud environments.

Since the datasets to be exchanged and their mappings are configured in advance, the runtime does not need to browse a remote address space or perform individual read operations for each value. As a result, the publisher-subscriber model is particularly well-suited for efficient, cyclic data distribution and for communication with multiple recipients.

Typical applications include:

- controller-to-controller and machine-to-machine data exchange
- distribution of process values to multiple recipients
- cyclic communication with defined publishing intervals
- broker-based distribution of machine data via MQTT
- integration of automation data into edge or cloud applications
- decoupling data producers and data consumers

TF6105 does not provide a browsable server address space and does not offer client/server services such as Read, Write, Call, Historical Access, or Alarms & Conditions. The product transfers the fields from the configured datasets.

An overview of the key differences

Feature	TF6100	TF6105
OPC UA communication model	Client/server	Publisher/subscriber
Main purpose	Flexible, service-oriented access to an information model	Efficient distribution of predefined datasets
Main roles	Server and client	Publisher and subscriber
Communication relationship	A stateful connection and session between the client and the server.	The publisher and subscriber are decoupled at the application level.

Feature	TF6100	TF6105
Data selection	A client browses the address space or knows the NodeIds and makes the choice of which nodes to use.	Dataset fields are chosen during the engineering phase and mapped in the configuration.
Runtime detection	The address space, nodes, references, data types, and metadata can be browsed.	Subscribers typically require the appropriate configuration or exchanged publisher-subscriber model configuration information.
Value access	Read, write, and client/server subscriptions	Cyclic or event-driven dataset messages
Method calls	Supported via the client/server service model	Not part of the publisher-subscriber model dataset exchange
Alarms and historical data	Supported by the TwinCAT OPC UA Server.	Are not provided as corresponding client/server services.
Topology	Typically, a client connects to a server; multiple clients can set up independent connections.	Suitable for one-to-one, one-to-many, and many-to-many distribution patterns.
Transport examples	OPC UA client/server transport with secure sessions	UDP or MQTT
Message encoding	OPC UA client/server service messages	UADP; JSON is also available with MQTT
Broker required	No	No for UDP; yes for MQTT
Typical focus	Interoperability, information models, diagnostics, commands, and flexible data access	Cyclic process data exchange, scalable distribution, and decoupling of producers and consumers

3 Installation

3.1 System requirements

The following system requirements apply for the installation and operation of this product.

Server

Technical data	Description
Operating system	Windows 10 Windows 11 Windows CE 6/7 (only up to and including setup version 5.1.x) Windows Server 2022 TwinCAT/BSD Beckhoff RT Linux®
Target platforms	PC architecture (x86, x64, Arm®)
.NET Framework	---
TwinCAT version	TwinCAT 2, TwinCAT 3
TwinCAT installation level	TwinCAT 2 CP, PLC, NC PTP TwinCAT 3 XAE, XAR, ADS
Required TwinCAT license	TS6100 TwinCAT OPC UA (for TwinCAT 2) TF6100 TC3 OPC UA (for TwinCAT 3)

Sample Client

Technical data	Description
Operating system	Windows 10 Windows 11 Windows Server 2022
Target platforms	PC architecture (x86, x64, Arm®)
.NET Framework	4.7.2
TwinCAT version	TwinCAT 2, TwinCAT 3
TwinCAT installation level	TwinCAT 2 CP, PLC, NC PTP TwinCAT 3 XAE, XAR, ADS
Required TwinCAT license	---

TwinCAT 2/TwinCAT 3 compatibility

This documentation is valid for both the TwinCAT 2 Supplement (TS6100) and the TwinCAT 3 Function (TF6100).
If there are differences in individual sub-functions, an information box will indicate this and provide additional explanations, if necessary.

Firewall port

To enable communication with the TwinCAT OPC UA Server, the following network port must be opened in the firewall of the device:

```
4840/tcp (incoming)
```

If, for example, the TwinCAT OPC UA Server is installed on a Beckhoff Industrial PC, this port must be opened as incoming communication in the operating system firewall.

Hardware requirements

The requirements of the TwinCAT OPC UA Server on the underlying industrial PC hardware depend strongly on the respective application scenario. In general, there are two metrics that are critical to server utilization: Main memory and CPU load.

The main memory load of the server varies depending on the number of symbols that are imported into the address space of the server via the Data Access [▶ 56] devices and on the number of simultaneous connections, subscriptions, historical access calls, … that an OPC UA client creates. However, the basic load of the server is based on the number of symbols and can be set to about 1024 bytes per symbol. With a symbol count of 1,000,000 symbols, for example, this means a main memory use of approx.

```
1024 Bytes * 1.000.000 Symbole = 1.024 Mbyte
```

Note the main memory load of the industrial PC

Please ensure that the industrial PC you are using has enough main memory.

For an initial test, you can start up your project on your engineering PC and read the main memory load of the TwinCAT OPC UA Server in the Windows Task Manager. Be aware of operating system limitations, such as the 2 GB main memory limit for 32-bit processes. This affects, for example, the 32-bit variant of the TwinCAT OPC UA Server. If you need more than 2 GB of main memory for the process, you must use the 64-bit version of the server.

The CPU load of the server, on the other hand, depends solely on the conditions prevailing at runtime, in particular the number of subscriptions and MonitoredItems and their configuration parameters that an OPC UA client requests from the server. See chapter Optimizations [▶ 49] for more information.

3.2 Installation

Depending on the TwinCAT version and operating system used, this TwinCAT 3 Function can be installed in different ways, which are described in more detail below.

NOTICE

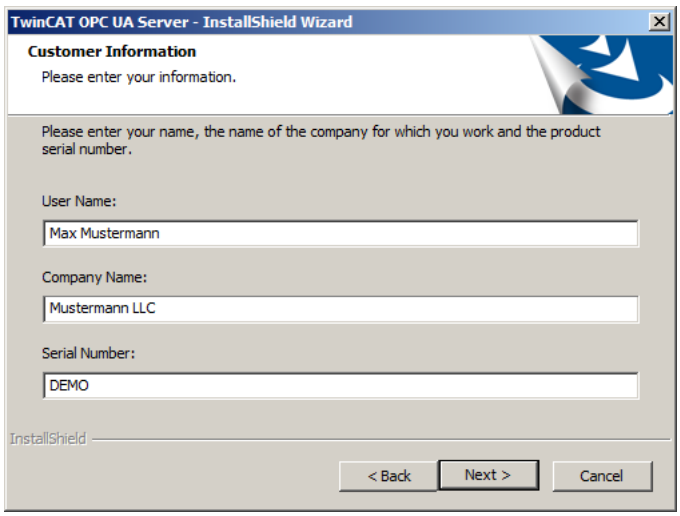
Update installation

An update installation always uninstalls the previous installation. Please make sure that you have backed up your configuration files beforehand.

TwinCAT 2 Setup

If you are using TwinCAT 2, you can install this supplement via a setup package, which you can download from the Beckhoff website at https://www.beckhoff.com/download.

The installation can be done on either the engineering or runtime side, depending on the system you need the supplement for. Please note that under TwinCAT 2, the supplement is licensed directly during installation. A separate dialog prompts you to enter the license key. If you would like to install a 30-day trial version of the supplement, please enter DEMO as the license key.



There is a separate setup download for TwinCAT 2 on Windows CE with the name TS6100-0030 OPC UA. This setup installs the CAB files, which you can then transfer to a Windows CE device and install them there.

TwinCAT 3 Package Manager

If you are using TwinCAT 3.1 Build 4026 (and higher) on the Microsoft Windows operating system, you can install this function via the TwinCAT Package Manager, see Installation documentation.

Normally you install the function via the corresponding workload; however, you can also install the packages contained in the workload individually. This documentation briefly describes the installation process via the workload.

NOTICE

Unprepared TwinCAT restart can cause data loss

The installation of this function may result in a TwinCAT restart.
Make sure that no critical TwinCAT applications are running on the system or shut them down in an orderly manner first.

Command line program TcPkg

You can use the TcPkg **C**ommand **L**ine **I**nterface (CLI) to display the available workloads on the system and to install a specific workload.

```
tcpkg list -t workload
tcpkg install TF6100.OpcUaServer.XAE
tcpkg install TF6100.OpcUaServer.XAR
```

TwinCAT Package Manager UI

You can use the **U**ser **I**nterface (UI) to display all available workloads and install them if required. To do this, follow the corresponding instructions in the interface.

TwinCAT 3 Usermode Runtime

This product supports installation on a TwinCAT 3 Usermode Runtime. When installed in such an environment, the corresponding UM package is automatically deployed.

The UM package installs a configuration file that instructs the TwinCAT Usermode System Service which applications to start during its startup phase. By default, this configuration file is placed in the TwinCAT Usermode Runtime directory:

C:\ProgramData\Beckhoff\TwinCAT\3.1\Runtimes\UmRT_Default\3.1\Target\StartManConfig

Each product can provide its own configuration file, so it is normal to find multiple configuration files in this directory. Every configuration file defines startup-related information for its associated TwinCAT 3 Function, such as:

- The installation path of the executable to be launched

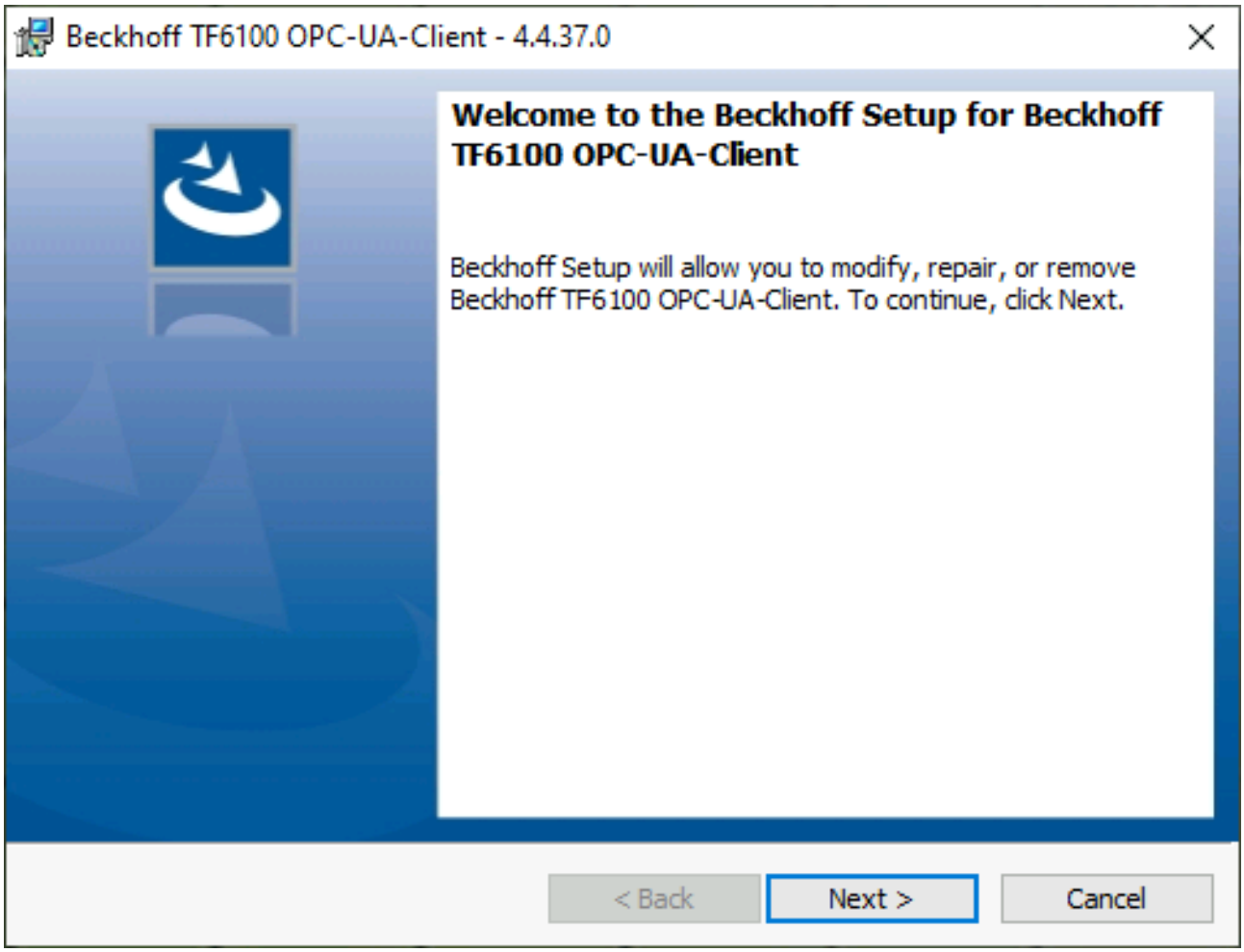
- Any parameters required to start the TwinCAT 3 Function

These entries are used by the TwinCAT Usermode System Service to determine which applications are started and when they are executed during runtime initialization.

TwinCAT 3 Setup

If you are using TwinCAT 3.1 Build 4024 on the Microsoft Windows operating system, you can install this function via a setup package, which you can download from the Beckhoff website at https://www.beckhoff.com/download.

Depending on the system on which you need the function, the installation can be done on either the engineering or runtime side. The following screenshot shows an example of the setup interface using the TF6100 TwinCAT OPC UA Client setup.



To complete the installation process, follow the instructions in the Setup dialog.

NOTICE

Unprepared TwinCAT restart can cause data loss

The installation of this function may result in a TwinCAT restart.
Make sure that no critical TwinCAT applications are running on the system or shut them down in an orderly manner first.

TwinCAT/BSD

If you use TwinCAT/BSD as your operating system, you can install this function via the TwinCAT/BSD Package Server. For further information on the TwinCAT/BSD Package Server and its configuration, please refer to the TwinCAT/BSD documentation.

To search for a package, you can use the following call in the TwinCAT/BSD console:

```
pkg search TF6100
```

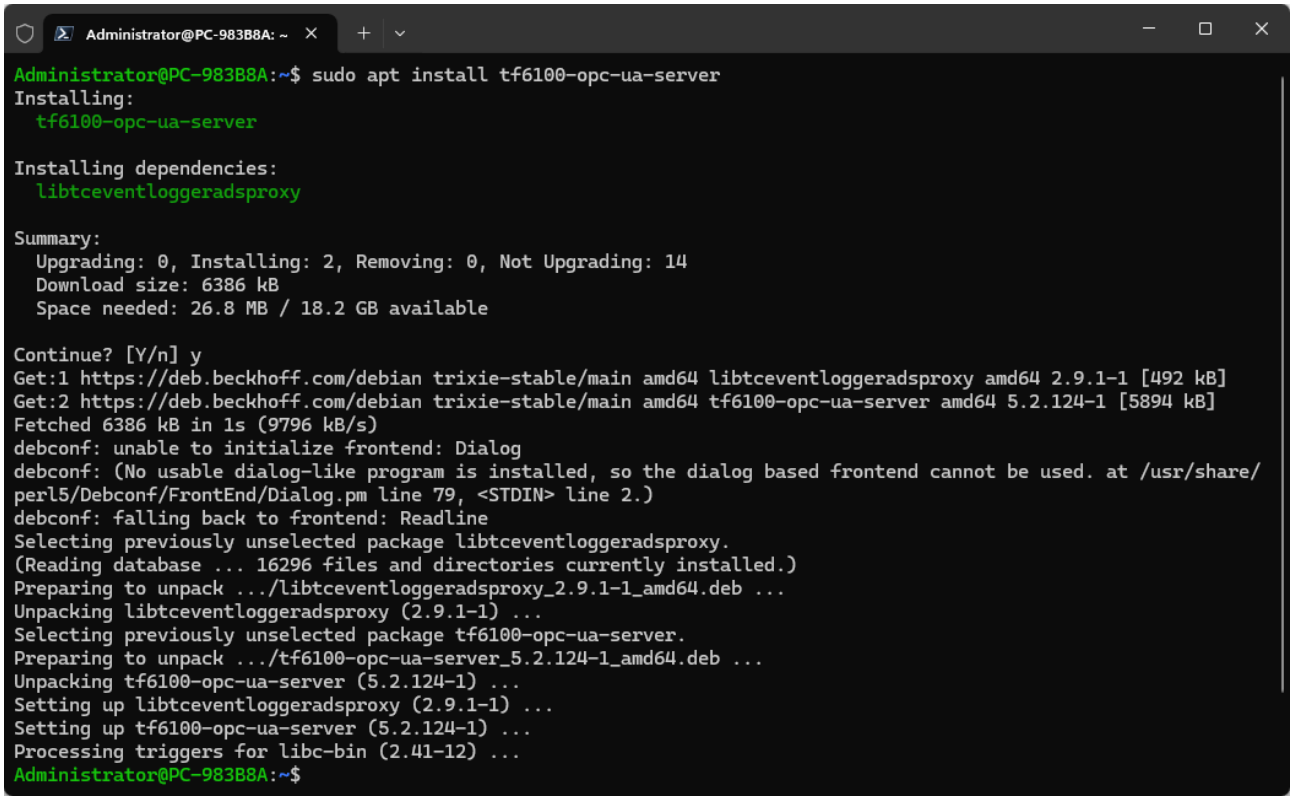
To install a function, you can use the following call in the TwinCAT/BSD console:

```
doas pkg install TF6100-OPC-UA-Server-<version>
```

Beckhoff RT Linux®

1. Use the CLI directly on your device. Alternatively, you can also connect via SSH.
2. Log in as an administrator.
3. Update the packages in the Package Manager.
   `sudo apt update`
4. Search for the existing TF6100 packages.
   `sudo apt search tf6100`
5. Install TF6100 OPC UA server via the Package Manager.
   `sudo apt install tf6100-opc-ua-server`

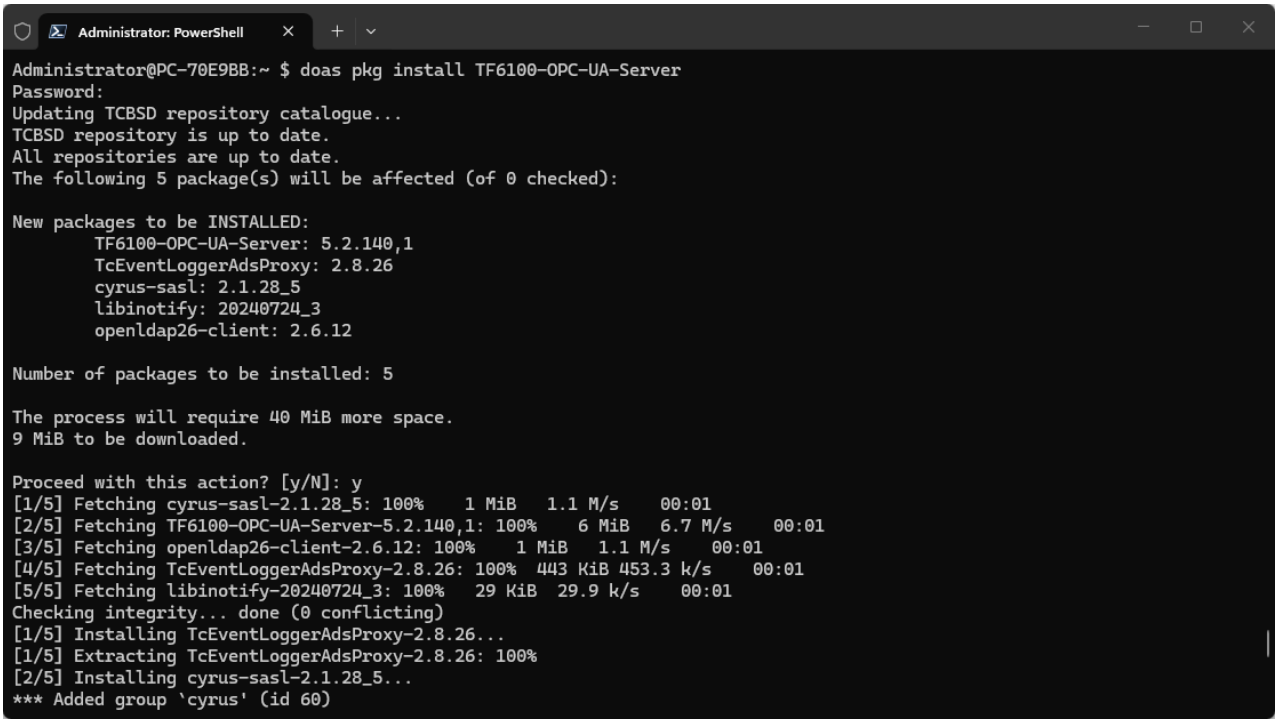
⇨ The TF6100 OPC UA server is installed.



TwinCAT/BSD

1. Use the CLI directly on your device. Alternatively, you can also connect via SSH.
2. Log in as an administrator.
3. Update the packages in the Package Manager.
   `doas pkg update`
4. Search for the existing TF6100 packages.
   `doas pkg search TF6100`
5. Install TF610x OPC UA client PubSub via the Package Manager.
   `doas pkg install TF6100-OPC-UA-Server`

⇨ The TF6100 OPC UA server is installed.



Windows CE

If you are using Microsoft Windows CE as your operating system, you can install this function via the respective CAB files, which you can download as separate files from the Beckhoff website at https://www.beckhoff.com/download.

After the download, the CAB files can be transferred to the Windows CE device via file transfer and executed there. The CAB files then install and register the function on the respective system.

Always use the appropriate CAB file for your system. Specifically, this means:

- CE-ARMV4I: Arm®-based devices, e.g. CX8190, CX9020
- CE-X86: x86-based devices, e.g. CX51xx, CX52xx, CX20xx

The CAB file can be transferred to the device either via the CF/SD card or the FTP server integrated in Windows CE.

Device restart

After installing this function, a device restart is required so that the function can be used.

3.3 Update installation

During an update installation, old application files are replaced by new ones. Backups of existing configuration files are created and saved in the form *.xml.bak in the application directory [▶ 54]. This also applies to the certificate directories.

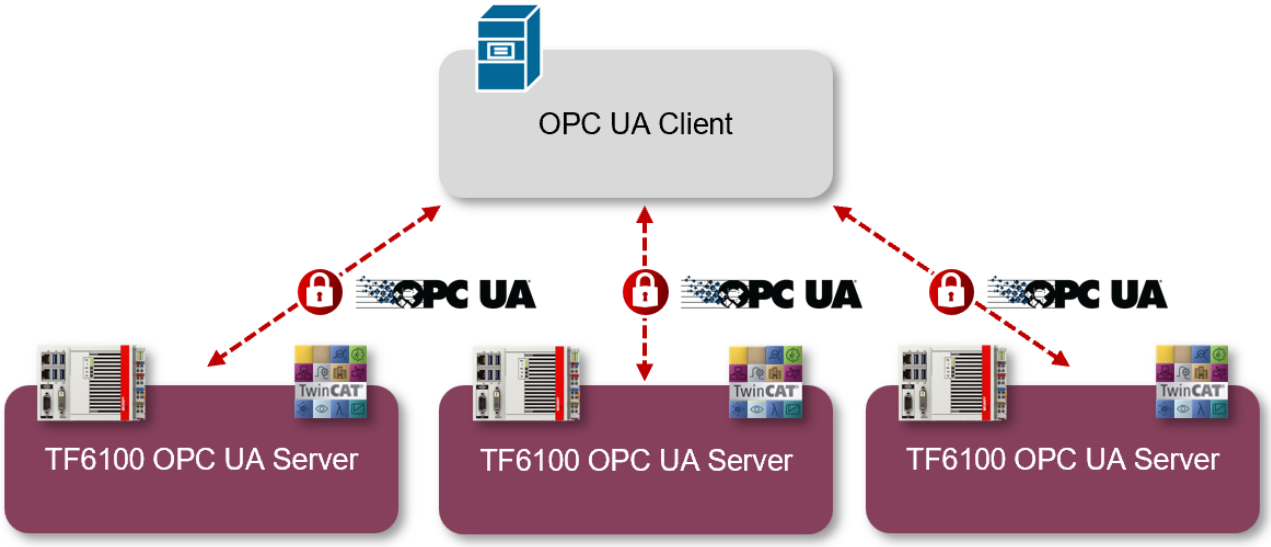
3.4 Installation variants

In the following, two installation variants are described, according to which the TwinCAT OPC UA Server can be installed depending on the application and infrastructure.

1. Server integrated directly into the control system
2. Operation of the server on a gateway PC

Server integrated directly into the control system

This scenario describes how the TwinCAT OPC UA Server should normally be operated. One of the biggest advantages of this software is that it can be integrated into even the smallest embedded platform, e.g. the CX8000 series. Thanks to this integration, general handling is very simple and convenient. OPC UA clients, e.g. HMI or MES/ERP systems, can establish a connection with the corresponding TwinCAT OPC UA Servers in the network and read and write symbol information originating from the TwinCAT runtime.

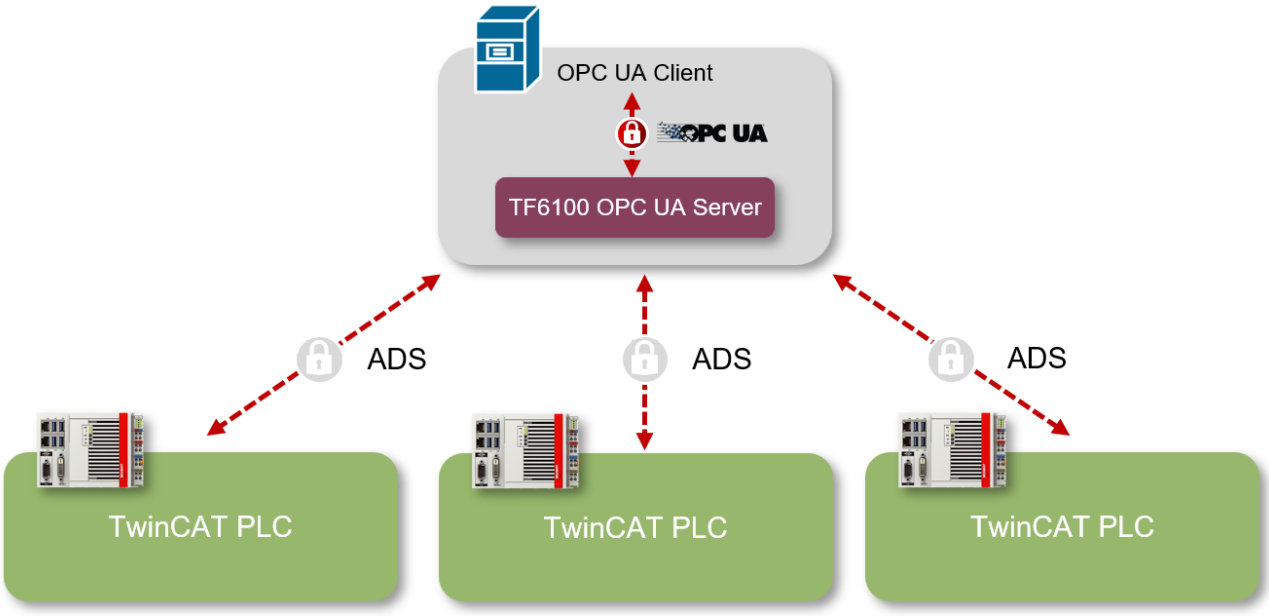


This scenario has the following advantages:

- Optimized network load, as OPC UA mechanisms such as subscriptions ("OnDataChange communication") can be used.
- Decentralized memory usage. Each TwinCAT OPC UA Server device is exclusively responsible for its own memory requirements, as only its "own" PLC symbols need to be provided in the server's address space.
- Secure communication right up to the control system. OPC UA features security mechanisms that are directly integrated in the protocol. Only the OPC UA server port is enabled in the controller firewall, which is then used for secure communication. In the case of the Reverse Connect [▶ 138] functionality, there is even no need to open the (incoming) firewall port.

Operation of the server on a gateway PC

This scenario describes the use of the TwinCAT OPC UA Server on a gateway PC. This use case is often used in Brownfield scenarios where existing control systems are to be given an OPC UA interface. In this case, the TwinCAT OPC UA Server is installed on a gateway PC (often referred to as an "edge PC") and connects one or more subordinate TwinCAT PLC controllers. Communication between the server and PLC then takes place via TwinCAT ADS.



From a financial perspective, this scenario is very attractive, as only one TwinCAT OPC UA Server software license needs to be purchased. However, this scenario also has some technical disadvantages compared to integrating the server into the controller:

- Network load can be very high depending on the number of devices and symbols present. The TwinCAT OPC UA Server uses cyclic ADS sampling to query symbol values quickly and efficiently from the TwinCAT runtime and must also be able to serve thousands of symbols simultaneously. The more symbols (and the more connected PLC controllers) there are, the more cyclic communication there is in the network.
- The memory requirement on the central server is very high because the TwinCAT OPC UA Server has to import the symbol information from each TwinCAT PLC and store it in its address space.
- Communication between the server and PLC is based on TwinCAT ADS, which only enables integrated encryption of the transport channel in newer TwinCAT versions. This may not yet be available for older systems as part of a Brownfield application.
- The symbol files must be exchanged between the TwinCAT PLC and the central server each time the PLC program is changed. This step is not necessary if the server is operated directly in the PLC controller.

3.5 Container

TwinCAT runtime for Linux® is a new, powerful operating system for Beckhoff controllers that combines the real-time capability of TwinCAT with the flexibility of advanced Linux® systems. One of the central innovations is the native support of container technologies such as Docker®, which enables TwinCAT functions to be installed and operated as container applications for the first time.

The use of Docker® containers opens up new possibilities for the provision, versioning and maintenance of TwinCAT functions. Functions can be encapsulated, reproducibly deployed and updated independently of the rest of the system. At the same time, users benefit from established Linux® and DevOps workflows without having to forego the deterministic real-time properties of the Beckhoff control platform.

This article describes how you can install and operate this TwinCAT function under Linux® in a Docker® container. The aim is to present the basic concepts, prerequisites and steps in an understandable way and thus create a basis for the practical use of container-based TwinCAT functions.

The following example setup shows a typical system architecture for the operation of a TwinCAT function as a Docker® container under Linux®. The TwinCAT function is executed as an independent container and integrated into the TwinCAT system via defined interfaces.

The controller is based on the TwinCAT runtime for Linux® and provides both the real-time environment for TwinCAT and the container runtime (Docker®). The TwinCAT function runs within a Docker® container and uses the resources provided by the system as well as the corresponding TwinCAT interfaces for communication with the controller and other applications. In this example, the function runs together with the TwinCAT PLC runtime in the same container, but it is also possible to separate these components.

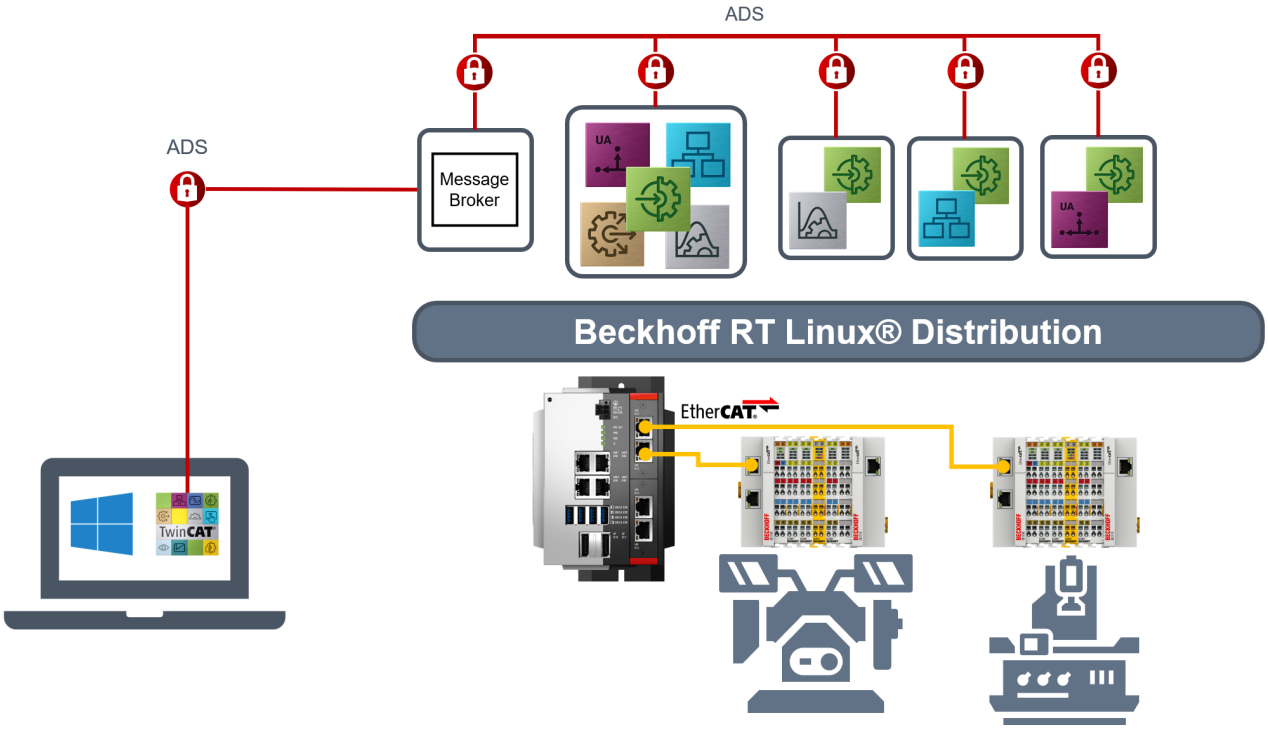
ADS-over-MQTT can be used for TwinCAT-side communication between several containers. By using MQTT as the mediating transport layer, the individual TwinCAT components are decoupled from each other, which significantly simplifies the setup and management of ADS routes.

Instead of static, point-to-point defined ADS routes, the containers involved communicate via a shared MQTT message broker. Changes to IP addresses, container names or deployment structures therefore generally do not require any adjustments to the ADS routes, which is particularly advantageous in containerized and dynamic environments.

The message broker required for ADS-over-MQTT can also be operated as a Docker® container on the Beckhoff controller or on a separate system and thus fits seamlessly into the container-based system structure.

Note

The following diagram provides a conceptual overview and does not represent a complete system configuration.



Overview of the setup

The following steps provide an overview of the basic procedure for installing and commissioning the TwinCAT function as a Docker® container. Detailed instructions and complete configuration examples can be found in the linked GitHub repository (see below).

Checking the requirements

Ensure that the controller is equipped with the TwinCAT runtime for Linux® and that Docker® is installed and activated as a container runtime.

Creating the container image

A new container image is created via a Docker® file, which contains this TwinCAT function and any dependent components.

Configuring the container

A Docker® Compose File is used to configure the parameters required to operate the container. Such parameters can be as follows, for example:

- Settings for network and communication interfaces
- Access rights and resources
- TwinCAT-specific settings (e.g. ADS routes)
- Volume mappings for storing user-specific configuration files

Starting the container

Start the Docker® container on the Beckhoff controller and check the proper start-up of the TwinCAT function.

Integrating into the TwinCAT system

Integration of the TwinCAT function into the existing TwinCAT project and testing of communication and functionality.

Monitoring and maintenance

Monitoring of the container status, evaluation of logs and execution of updates or restarts if required.

A complete, executable example with detailed documentation, configuration files and source code is available in the following GitHub repository:

https://www.github.com/Beckhoff/TF6100_Samples

3.6 Licensing

This TwinCAT product requires a license for full operation. In addition, a trial license can be activated in order to evaluate the product.

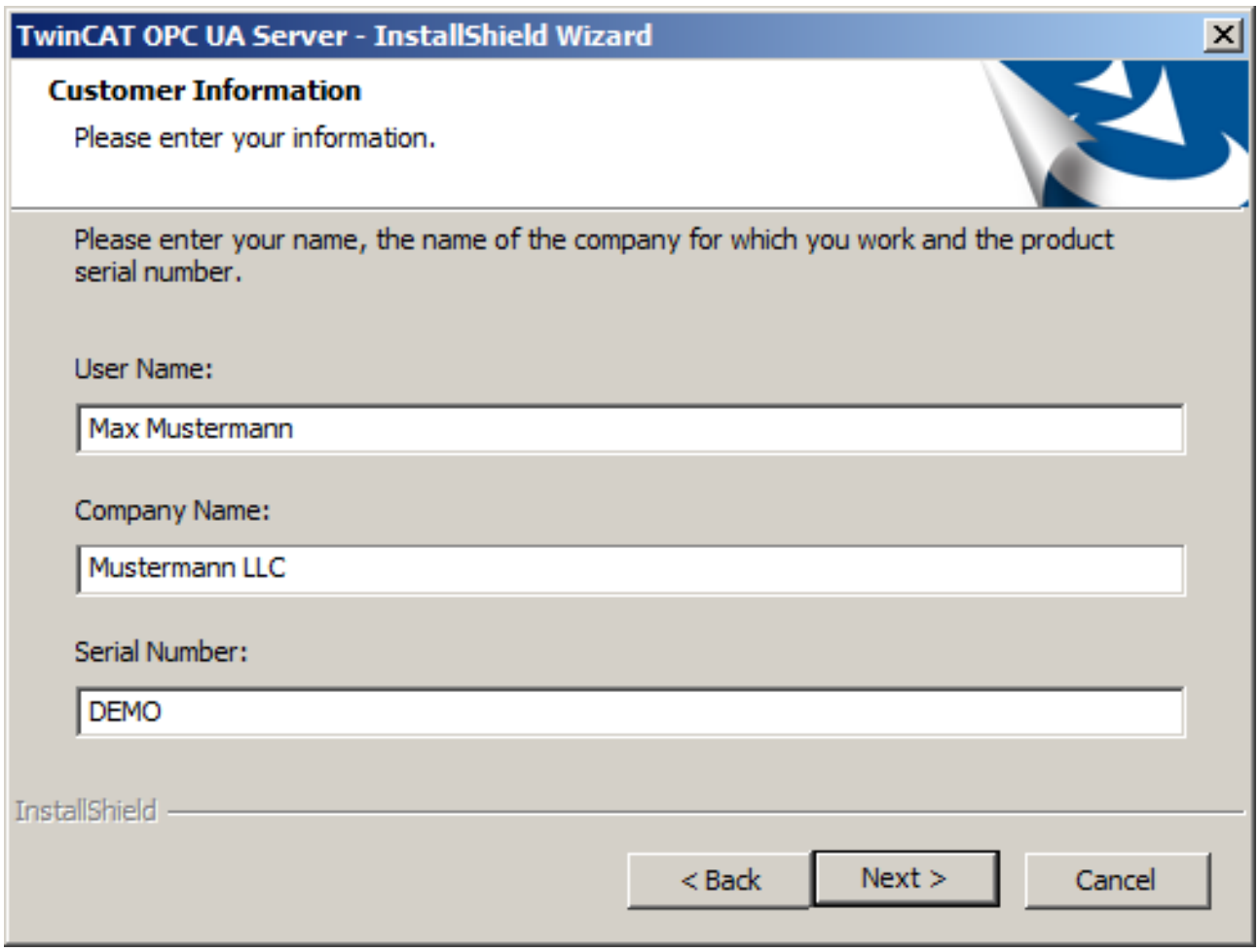
For more information about licensing the product, see the following sections:

- Licensing under TwinCAT 2
- Licensing of a 7-day trial version under TwinCAT 3
- Licensing of a full version under TwinCAT 3

Further information on TwinCAT 3 licensing can be found in the "Licensing" documentation in the Beckhoff Information System (TwinCAT 3 > Licensing).

Licensing under TwinCAT 2

The product is licensed under TwinCAT 2 during installation; a corresponding text field allows you to enter a license key. To evaluate this product, you can activate it for a period of 30 days. To do this, enter "Demo" as the license key.

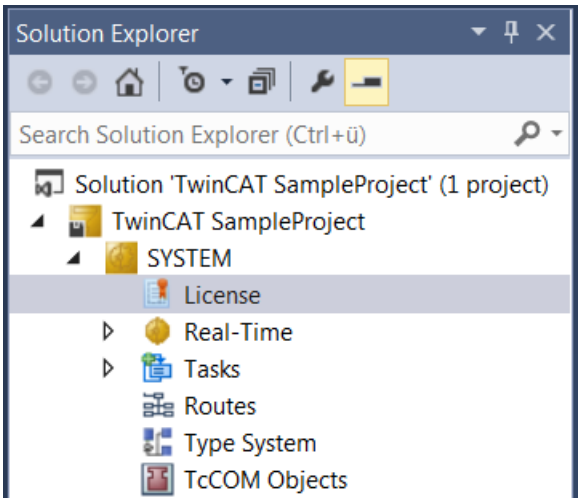


Licensing the 7-day test version of a TwinCAT 3 Function

A 7-day test version cannot be enabled for a TwinCAT 3 license dongle.

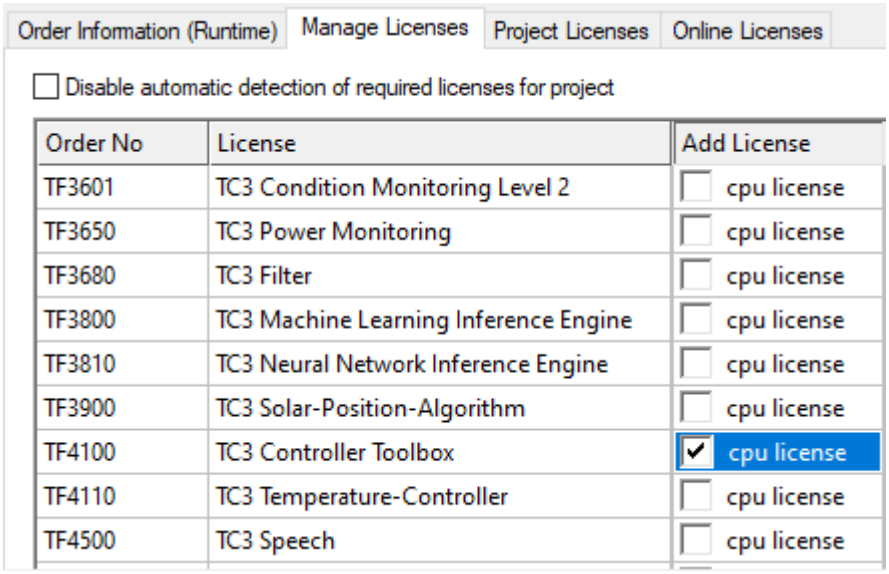
1. Start the TwinCAT 3 development environment (XAE).
2. Open an existing TwinCAT 3 project or create a new project.
3. If you want to activate the license for a remote device, set the desired target system. To do this, select the target system from the **Choose Target System** drop-down list in the toolbar.
   ⇨ The licensing settings always refer to the selected target system. When the project is activated on the target system, the corresponding TwinCAT 3 licenses are automatically copied to this system.

4. In the **Solution Explorer**, double-click **License** in the **SYSTEM** subtree.



⇨ The TwinCAT 3 license manager opens.

5. Open the **Manage Licenses** tab. In the **Add License** column, check the check box for the license you want to add to your project (e.g. "TF4100 TC3 Controller Toolbox").

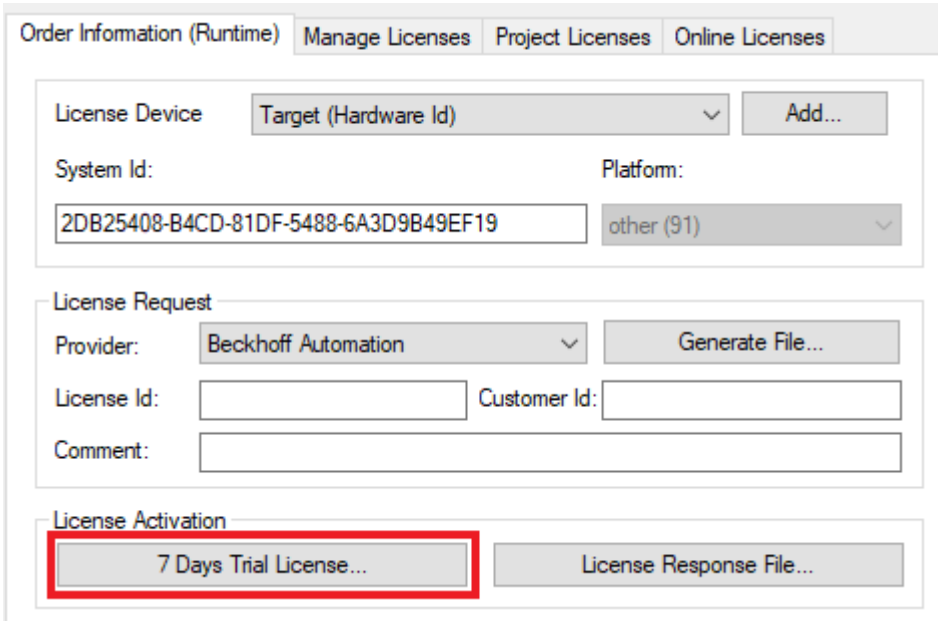


Order No	License	Add License
TF3601	TC3 Condition Monitoring Level 2	cpu license
TF3650	TC3 Power Monitoring	cpu license
TF3680	TC3 Filter	cpu license
TF3800	TC3 Machine Learning Inference Engine	cpu license
TF3810	TC3 Neural Network Inference Engine	cpu license
TF3900	TC3 Solar-Position-Algorithm	cpu license
TF4100	TC3 Controller Toolbox	cpu license
TF4110	TC3 Temperature-Controller	cpu license
TF4500	TC3 Speech	cpu license

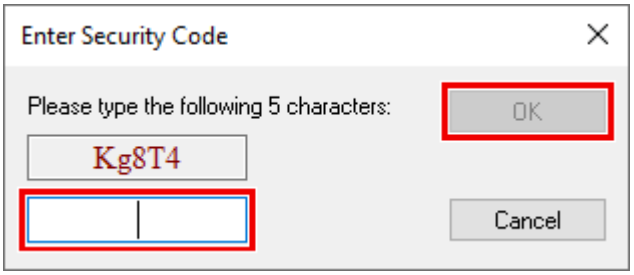
6. Open the **Order Information (Runtime)** tab.

⇨ In the tabular overview of licenses, the previously selected license is displayed with the status “missing”.

7. Click **7-Day Trial License...** to activate the 7-day trial license.



⇨ A dialog box opens, prompting you to enter the security code displayed in the dialog.



8. Enter the code exactly as it is displayed and confirm the entry.
9. Confirm the subsequent dialog, which indicates the successful activation.
   ⇨ In the tabular overview of licenses, the license status now indicates the expiry date of the license.
10. Restart the TwinCAT system.

⇨ The 7-day trial version is enabled.

Licensing the full version of a TwinCAT 3 Function

A description of the procedure to license a full version can be found in the Beckhoff Information System in the documentation "TwinCAT 3 Licensing".

4 Technical introduction

4.1 Quick start

The following chapter provides a quick start to the TwinCAT OPC UA Server. In these instructions, you initialize the TwinCAT OPC UA Server in the delivery state, then create a TwinCAT PLC project and finally enable a PLC variable by setting a pragma via OPC UA. The variable is then available in the address space of the server.
The action steps are described in more detail below in the order in which they are performed:

- Initialization of the server
- Creating a TwinCAT PLC project
- Activating the download of the symbol files
- Adding a product license
- Enabling a variable
- Connecting an OPC UA client

Initialization of the server

After installation, the server needs to be initialized [▶ 34] once according to the so-called TOFU principle (**T**rust-**O**n-**F**irst-**U**se). Here you configure a user account, which is then required to establish a connection with the server.

As this is a central and security-relevant topic, initialization [▶ 34] is explained in detail in a separate documentation chapter. The further steps in this section assume that this process has been carried out once and that you have initialized the server with a user account.

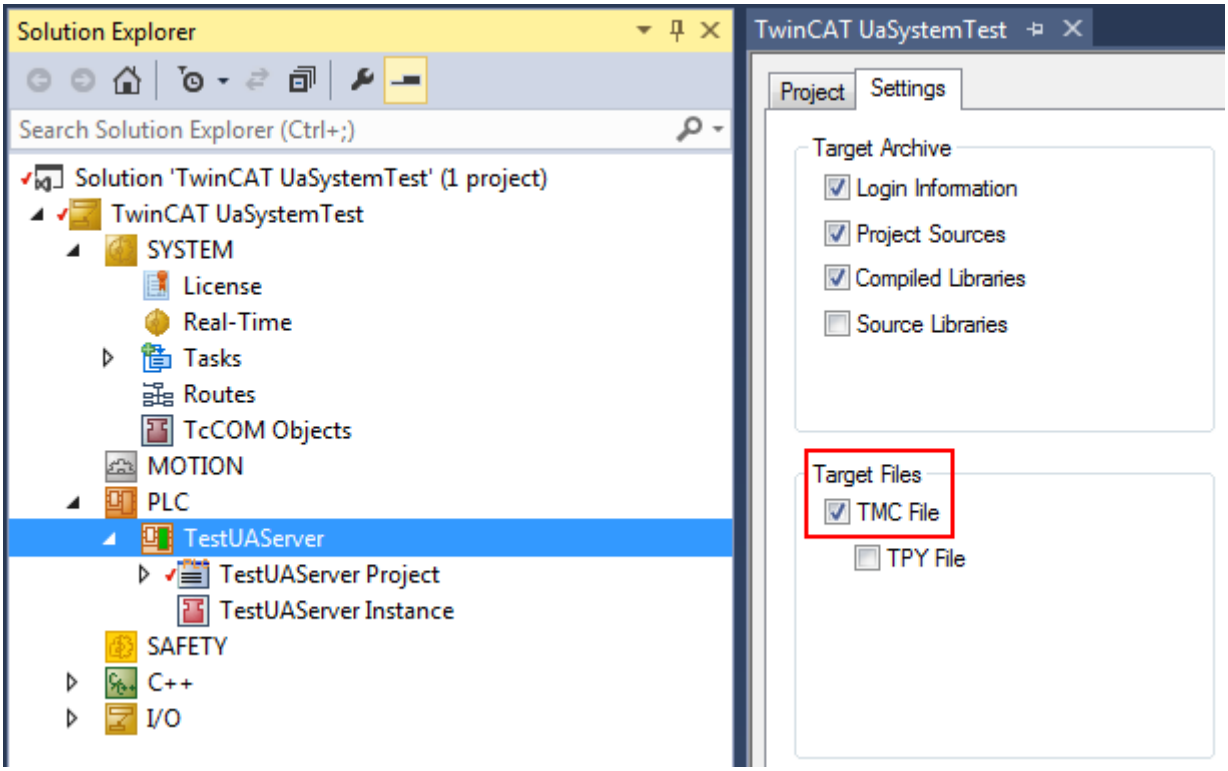
Tools for initializing the server

You can use the TwinCAT OPC UA Configurator to initialize the server. This may require the installation of an additional software package.

Creating a TwinCAT PLC project (download of symbol files)

1. Open the TwinCAT XAE Shell (or Visual Studio).
2. In the **File** menu, select the command **New > Project**.
3. Add an empty PLC project to the project.
4. Activate the automatic download of the TMC file in the properties of the PLC project, as shown in the following screenshot.

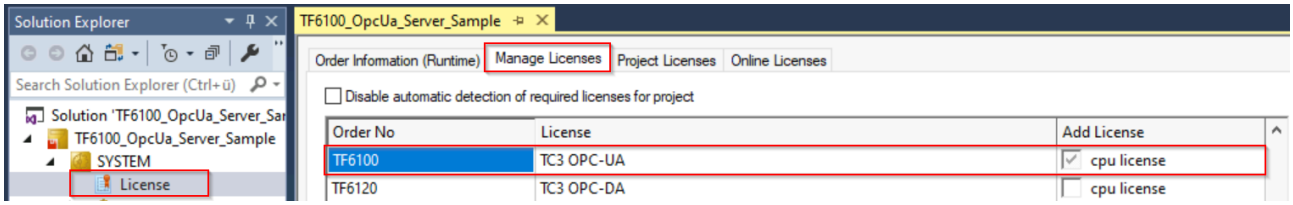
⇨ A new TwinCAT project has been created.



Adding a product license

✓ Check whether a TF6100 license is available in the TwinCAT XAE license dialog.

1. If not, you can use a 7-day trial license as part of this Quick Start Tutorial.



Enabling a variable

2. Add a new variable of data type INT to the MAIN program.
3. Set the pragma on this variable to enable the variables via OPC UA.

```
{attribute 'OPC.UA.DA' := '1'}
nMyCounter : INT;
```

4. In the implementation part of the MAIN program, increment this variable by 1 each cycle.

```
nMyCounter := nMyCounter + 1;
```

5. Activate the TwinCAT project on your system.

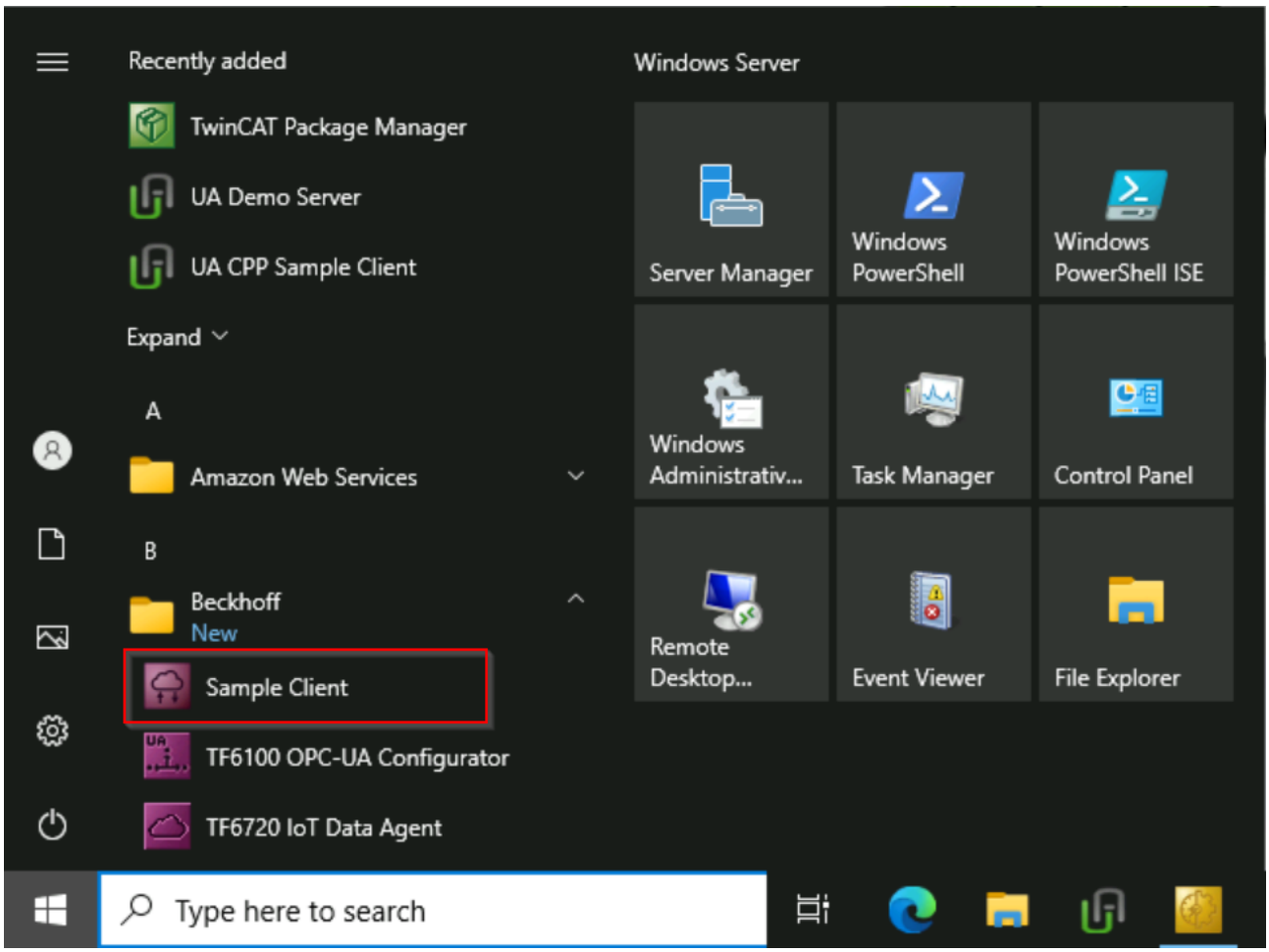
Connecting an OPC UA client

An OPC UA client uses the so-called ServerURL to connect to a server. The ServerURL contains the IP address or the host name of the device on which the server was installed. In this tutorial, we assume that the client and server are running on the same system. The client thus connects to the following ServerURL:

```
opc.tcp://localhost:4840
```

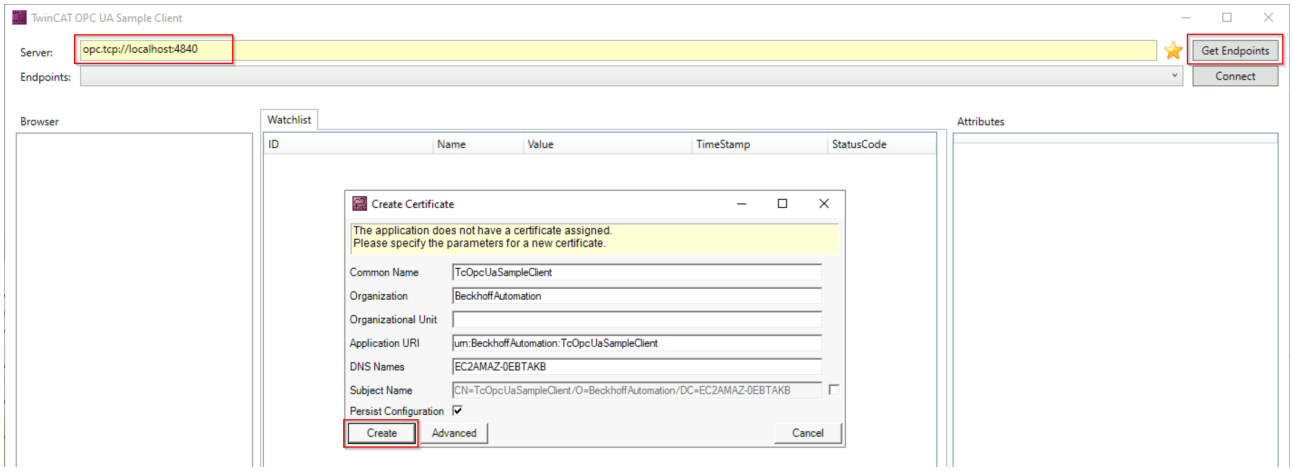
As a client, we use the TwinCAT OPC UA Sample Client, which is part of the TF6100 product package.

After installation, it can be called up via the Windows start menu.

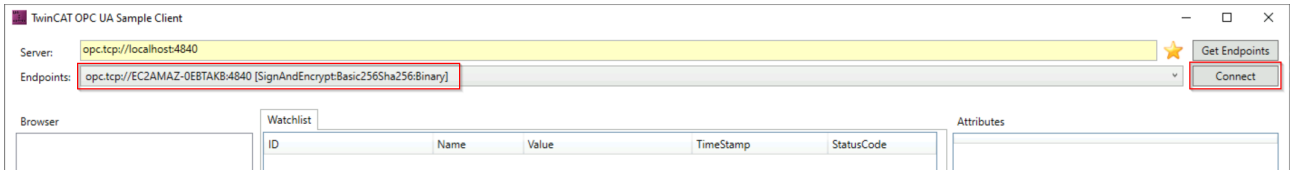


The ServerURL to the local host is already entered by default in the TwinCAT OPC UA Sample Client.

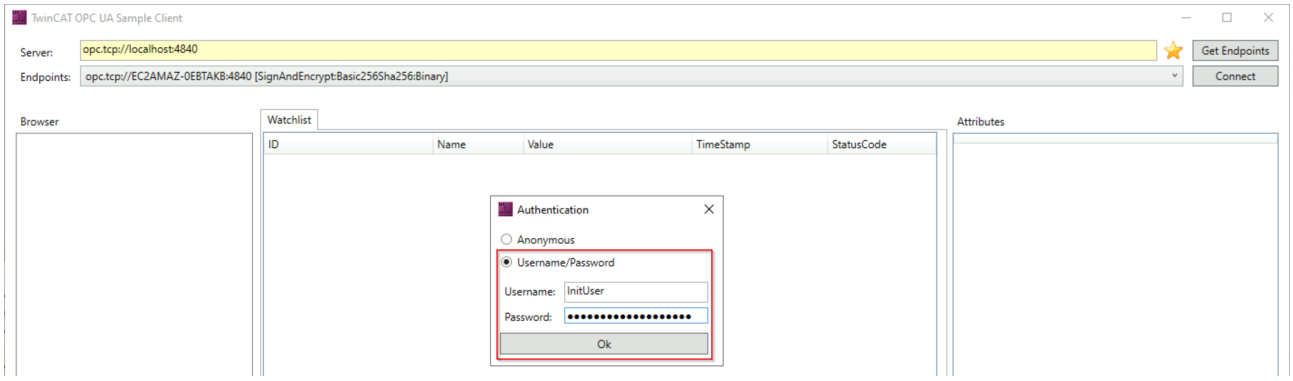
1. Click on the **Get Endpoints** button.
   ⇨ If you are using the Sample Client for the first time, a dialog box appears to accept settings for generating application certificates.



2. Confirm the dialog box with the **Create** button.
   ⇨ All connection endpoints are now read from the server and displayed.
3. Select the endpoint "SignAndEncrypt:Basic256Sha256:Binary" and click on the **Connect** button.



4. Enter the data for the user account that you configured in the first step of this documentation article for initializing the server.

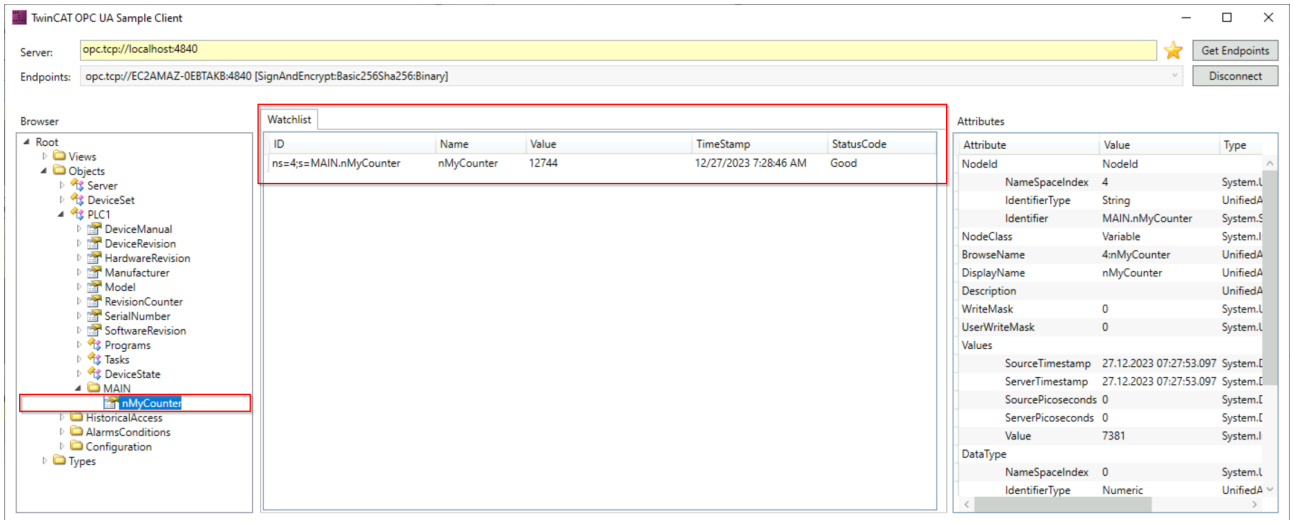


5. Click **OK**.

⇨ You are now connected to the server.
The address space of the server is displayed in a tree structure on the left-hand side of the application and you can navigate through the symbolism of the PLC program. In this example, we have enabled a PLC variable for OPC UA. These can be found at the following path:

```
Root \ Objects \ PLC1 \ MAIN \ nMyCounter
```

You can add the variable to the "Watchlist" by double-clicking on it. This means that a subscription is created for the variable and the variable value is transferred from the server to the client in the event of a value change.



4.2 Quick Start (TwinCAT 2)

The following chapter provides a quick start to the TwinCAT OPC UA Server if it is operated under TwinCAT 2. In these instructions, you initialize the TwinCAT OPC UA Server in the delivery state, then create a TwinCAT PLC project and finally enable a PLC variable by setting a comment via OPC UA. The variable is then available in the address space of the server.

The action steps are described in more detail below in the order in which they are performed:

- Initialization of the server
- Creating a TwinCAT PLC project
- Activating the download of the symbol files
- Enabling a variable
- Connecting an OPC UA client

Initialization of the server

After installation, the server needs to be initialized [▶ 34] once according to the so-called TOFU principle (**T**rust-**O**n-**F**irst-**U**se). Here you configure a user account, which is then required to establish a connection with the server.

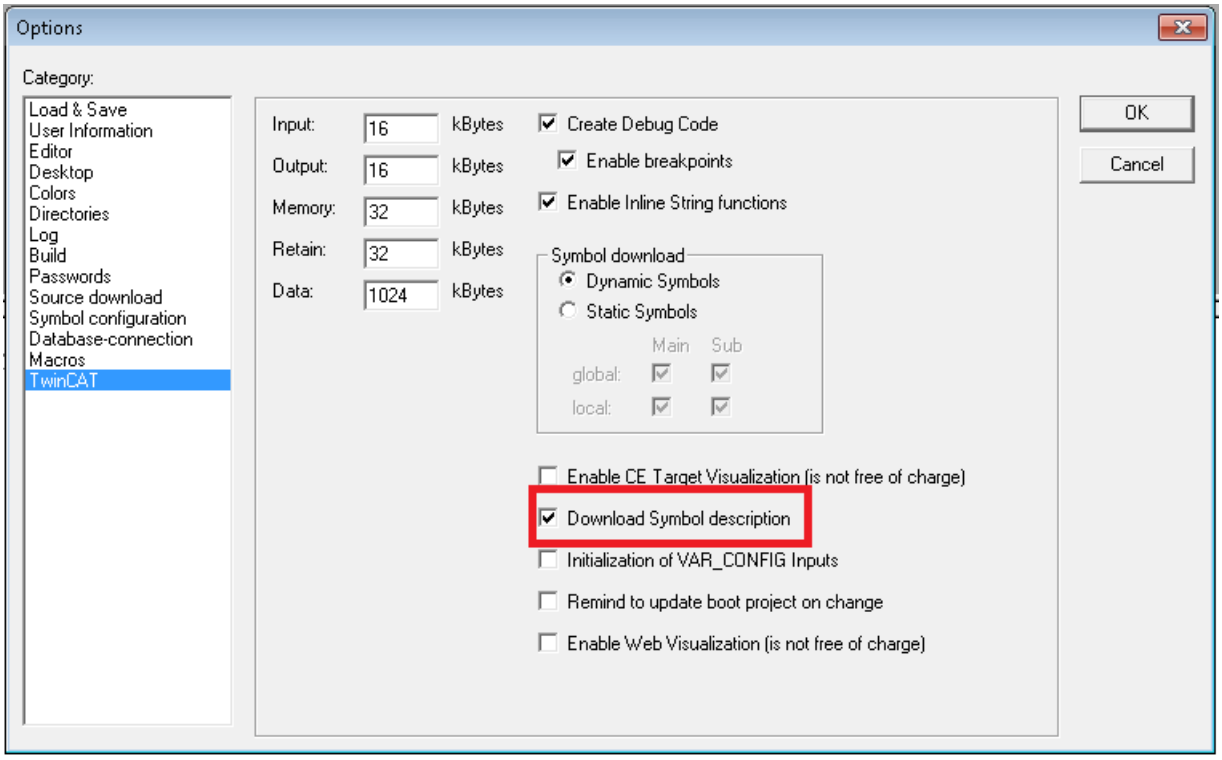
As this is a central and security-relevant topic, initialization [▶ 34] is explained in detail in a separate documentation chapter. The further steps in this section assume that this process has been carried out once and that you have initialized the server with a user account.

Tools for initializing the server

You can use the TwinCAT OPC UA Configurator to initialize the server. This may require the installation of an additional software package.

Creating a TwinCAT PLC project and activating the download of the symbol files

Create a new project in TwinCAT PLC Control or open your existing PLC project. Open the project properties and activate the “Download Symbol description” option so that the symbol file (*.tpy) of the PLC program is transferred to the boot directory of the target device when it is activated.



Enabling a variable

1. Add a new variable of data type INT to the MAIN program.
2. Set the comment on this variable to enable the variables via OPC UA. A text can be defined via the "Description", which is also displayed as a Description attribute at the respective node in the OPC UA address space.

```
nMyCounter : INT; (*~ (OPC:1:some description) *)
```

3. You can mark the variable as read-only with an additional comment. Write operations via OPC UA are then rejected by the server with the status code BadNotWriteable.

```
nMyCounter : INT; (*~ (OPC:1:some description)
                      (OPC_PROP[0005]:1:read-only flag) *)
```

4. You can use the following comment to define an alias for the variable, i.e. specify a different name for the variable in the OPC UA namespace. The value you specify for x then corresponds to the new variable name.

```
nMyCounter : INT; (*~ (OPC:1:some description)
                      (OPC_PROP[0005]:1:read-only flag)
                      (OPC_UA_PROP[5100]:x:alias name) *)
```

5. In the implementation part of the MAIN program, increment this variable by 1 each cycle.

```
nMyCounter := nMyCounter + 1;
```

6. Activate the TwinCAT project on your system.

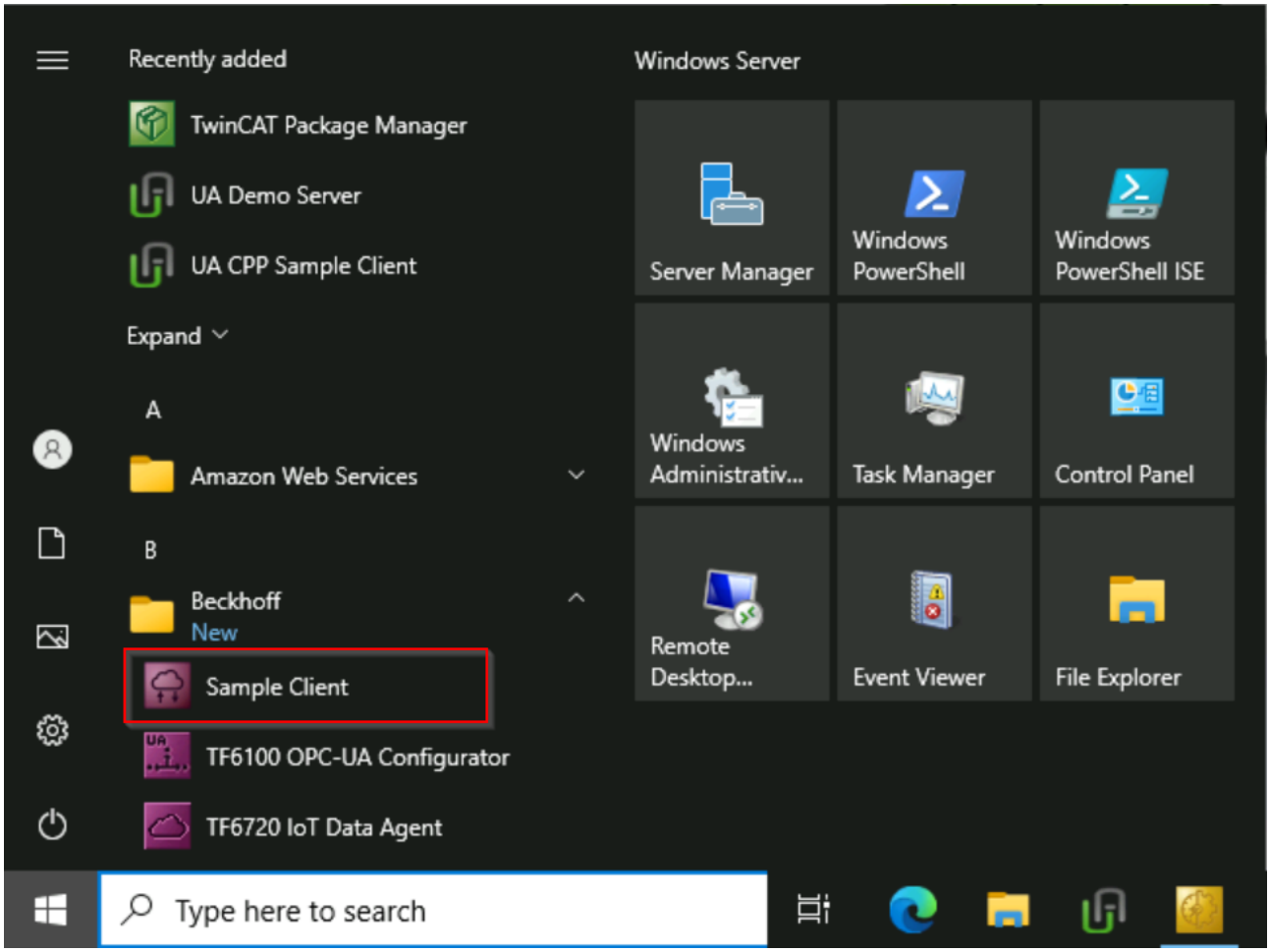
Connecting an OPC UA client

An OPC UA client uses the so-called ServerURL to connect to a server. The ServerURL contains the IP address or the host name of the device on which the server was installed. In this tutorial, we assume that the client and server are running on the same system. The client thus connects to the following ServerURL:

```
opc.tcp://localhost:4840
```

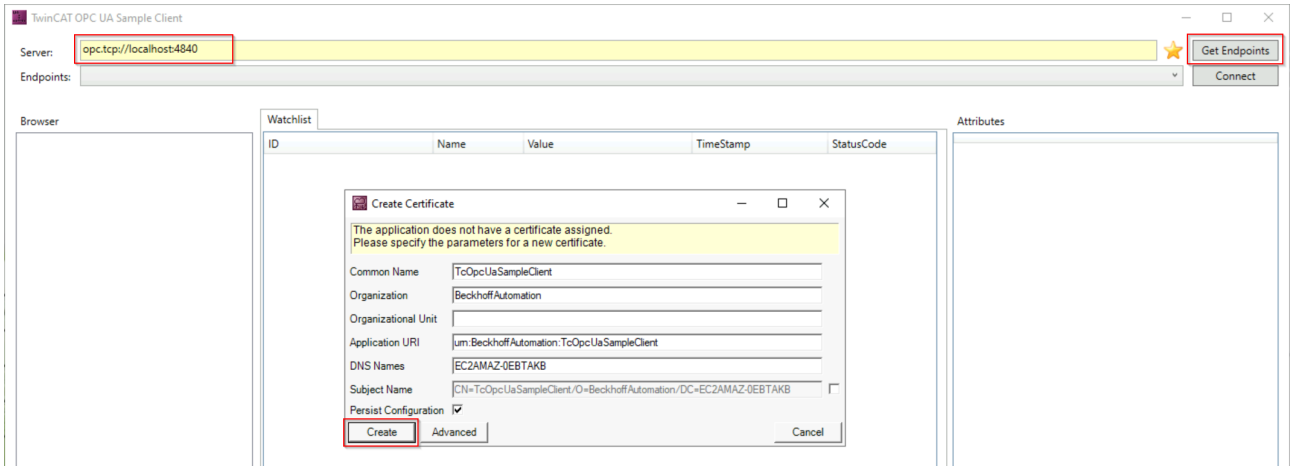
As a client, we use the TwinCAT OPC UA Sample Client, which is part of the TF6100 product package.

After installation, it can be called up via the Windows start menu.



The ServerURL to the local host is already entered by default in the TwinCAT OPC UA Sample Client.

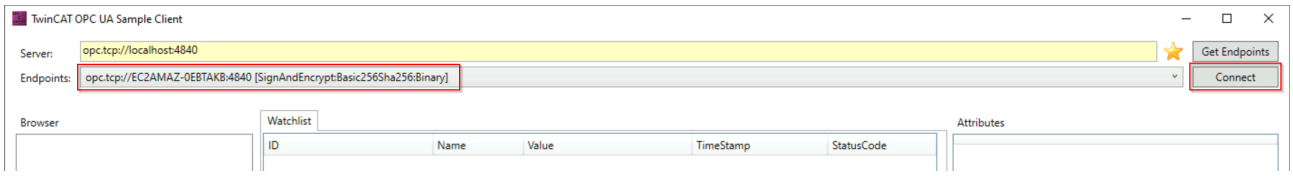
1. Click on the **Get Endpoints** button.
   ⇨ If you are using the Sample Client for the first time, a dialog box appears to accept settings for generating application certificates.



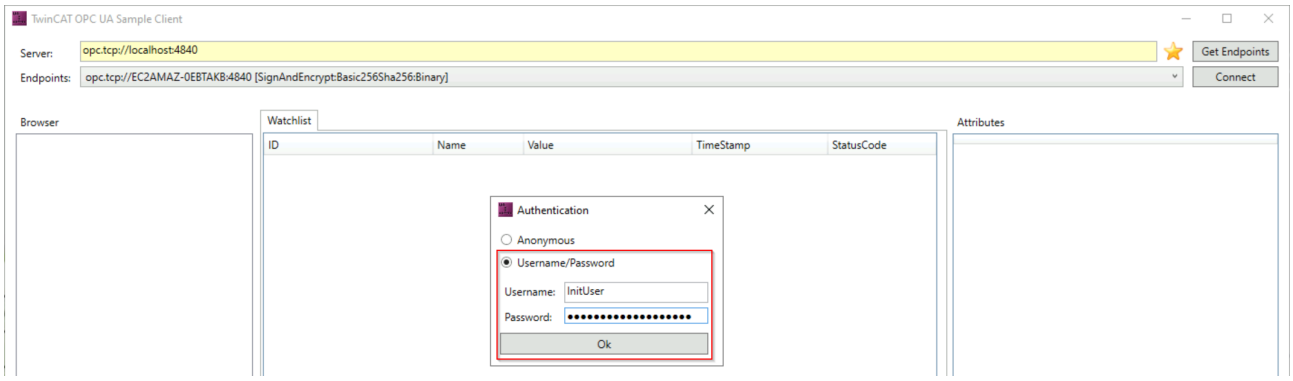
2. Confirm the dialog box with the **Create** button.

   ⇨ All connection endpoints are now read from the server and displayed.

3. Select the endpoint "SignAndEncrypt:Basic256Sha256:Binary" and click on the **Connect** button.



4. Enter the data for the user account that you configured in the first step of this documentation article for initializing the server.

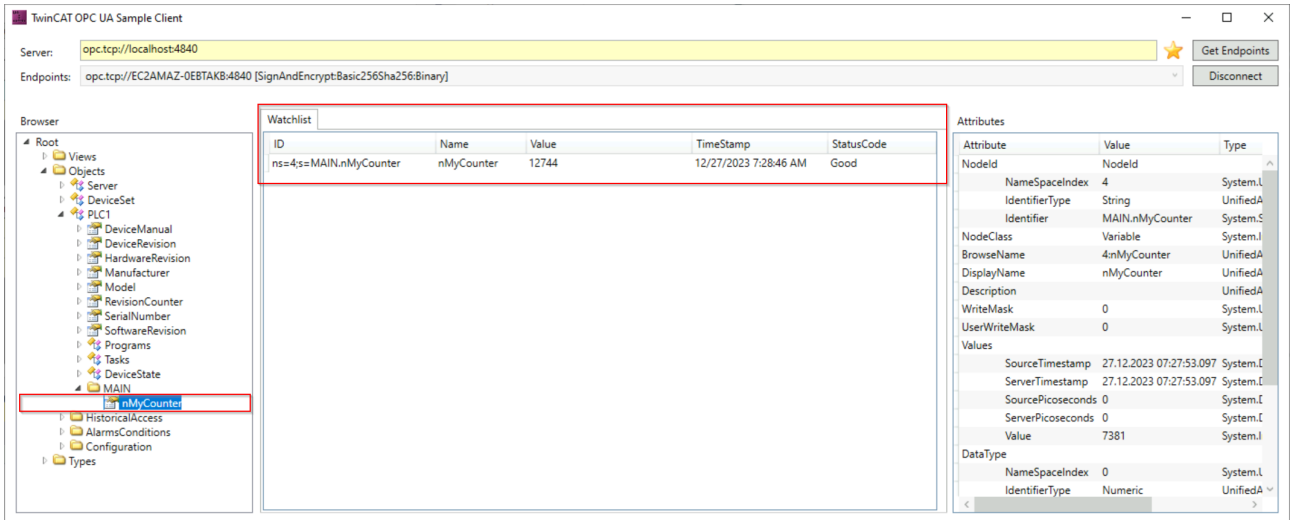


5. Click **OK**.

⇨ You are now connected to the server.
The address space of the server is displayed in a tree structure on the left-hand side of the application and you can navigate through the symbolism of the PLC program. In this example, we have enabled a PLC variable for OPC UA. These can be found at the following path:

```
Root \ Objects \ PLC1 \ MAIN \ nMyCounter
```

You can add the variable to the "Watchlist" by double-clicking on it. This means that a subscription is created for the variable and the variable value is transferred from the server to the client in the event of a value change.

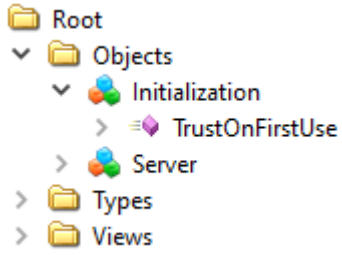


4.3 Initialization

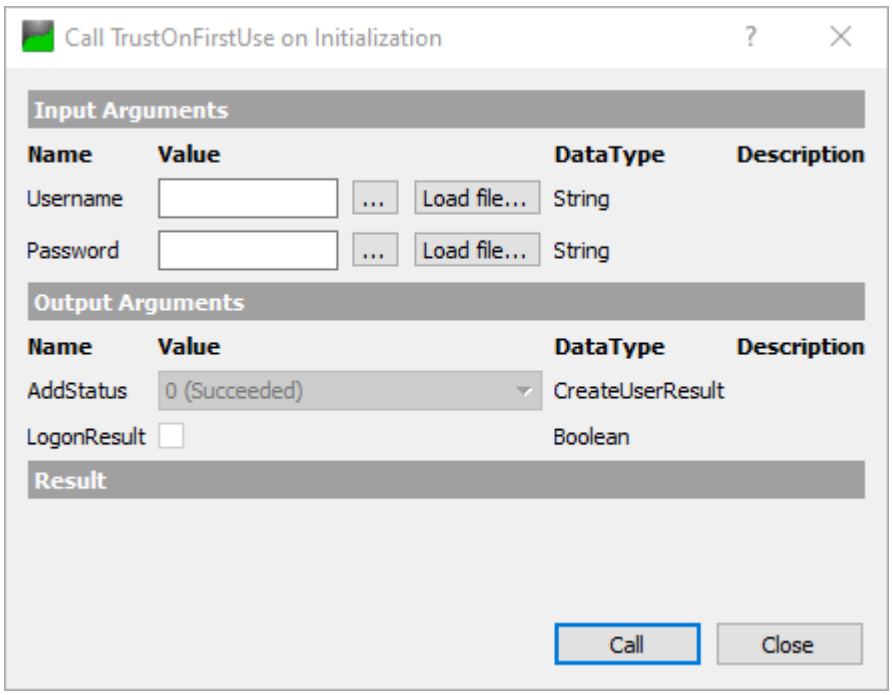
Starting with setup version 4.4.0, the TwinCAT OPC UA Server requires an initialization phase, which is based on the TOFU principle (**T**rust **O**n **F**irst **U**se). This means that the server must be actively initialized by the user so that it can be used for its various functions (Data Access, Historical Access, etc.).

By default, the server allows clients to establish an unauthenticated connection ("Anonymous"). The one-time TOFU initialization now requires the configuration of an operating system user that an OPC UA client must subsequently use to successfully log on to the server.

For this purpose, the server provides only a special initialization namespace in the uninitialized state. This namespace contains an object "Initialization" with a method "TrustOnFirstUse".



The method defines the following input/output parameters:



Parameter	Description
[in] Username	Username for the operating system user to be created. If the user already exists, the server attempts to perform a test login with the specified password and, if successful, transfers the existing user to its security configuration.
[in] Password	Password for the operating system user. **The password is not stored in the server configuration, but is only available in the user database of the operating system. Please note that the type of password may depend on any security settings of the operating system (keyword "complex passwords").**
[out] AddStatus	Indicates whether the creation of the operating system user was successful or whether the user already exists.
[out] LogonResult	Indicates whether the server was able to login to the operating system with the specified username/password combination. This is a good way to check if you have entered the wrong password if the user already exists.
[out] OPC UA Statuscode	The regular OPC UA Status Code when calling a method. If the method has been called successfully on OPC UA level, this status code returns GOOD, otherwise BAD.

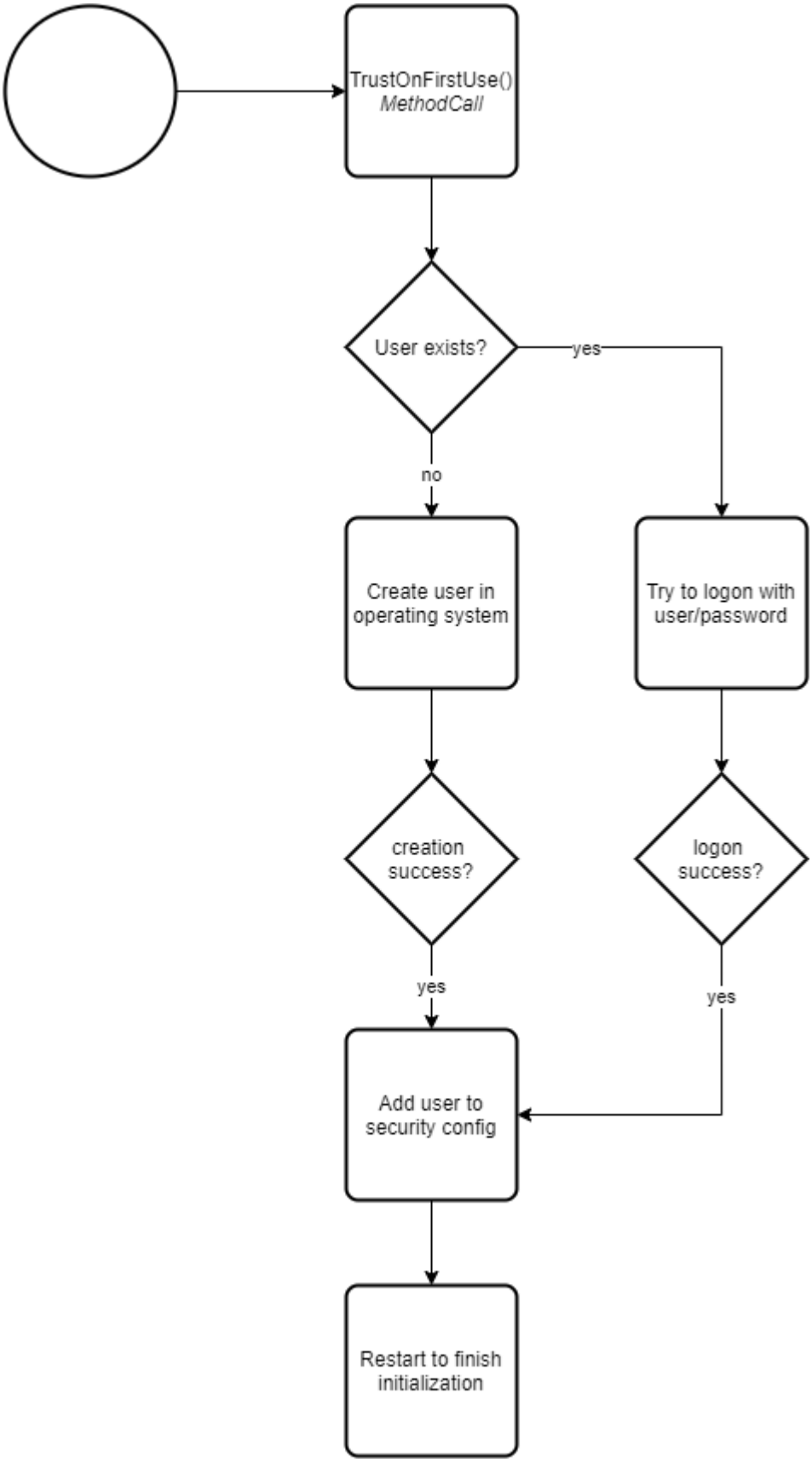
The server is initialized by calling this method. The method tries to create a user specified by the user in the lower-level operating system of the server. If this is successful, the user is automatically added to the security configuration (TcUaSecurityConfig.xml) of the server and defined as server administrator. After an automatic restart of the server at the end of the method call, an OPC UA client can then log on to the server with this user.

If a specified user already exists in the operating system, this is indicated by an output parameter (AddStatus). In this case, the server attempts to log on to the operating system with the specified password. If this logon process is successful, the user is entered in the server's security configuration and the initialization is successfully completed by an automatic restart of the server. If the logon to the operating system fails (e.g. because the wrong password was entered), this is indicated by an output parameter (LogonResult) and the initialization is not continued. This prevents you from accidentally trying to initialize the server with a wrong username/password combination and thus "locking yourself out".

Expiration of a user password

When the OPC UA server creates an operating system user, it is **not** explicitly enabled for this user that the password does not expire. Here the settings of the operating system are adopted, where the maximum password age is defined in the password policies. If the maximum password age is set to 0, passwords do not expire; otherwise they do so after the number of days specified in the operating system.

The following diagram illustrates this process once again in a highly simplified form:

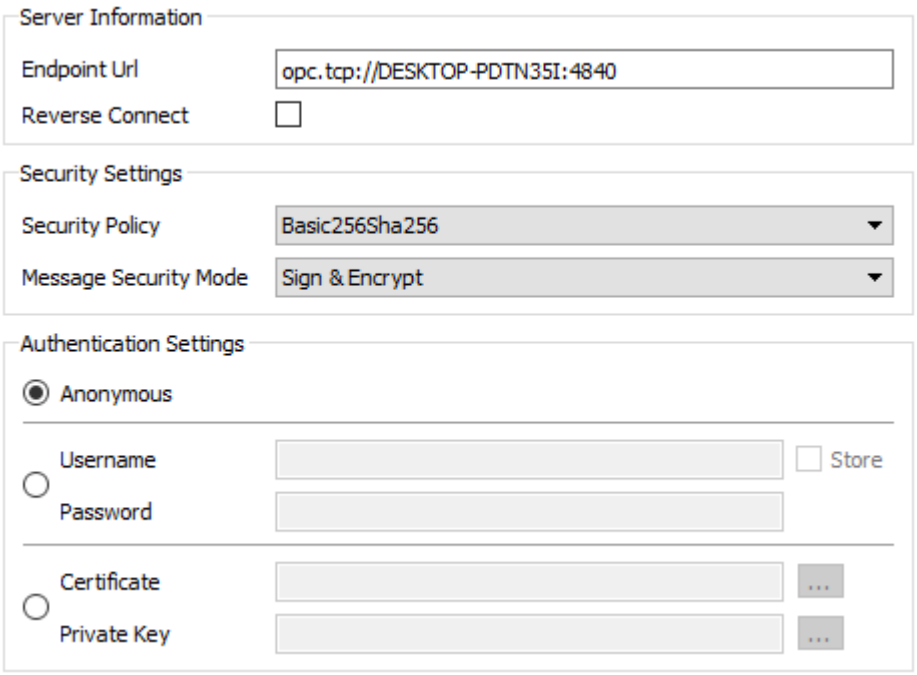


After restarting the server, an OPC UA client must use the operating system user used for initialization for authentication when establishing a connection.

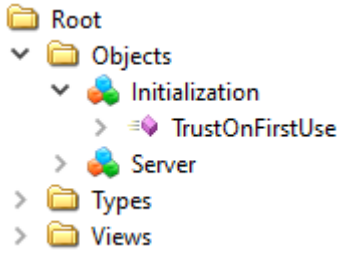
The following screenshots show the entire process using the OPC UA client "UA Expert" as an example. In this example, we assume that the user does not yet exist in the operating system and is therefore created by the server.

Step 1: OPC UA Client connects to the server for the first time

The server has been installed and UA Expert connects to the server for the first time. Anonymous access can still be used for this connection.

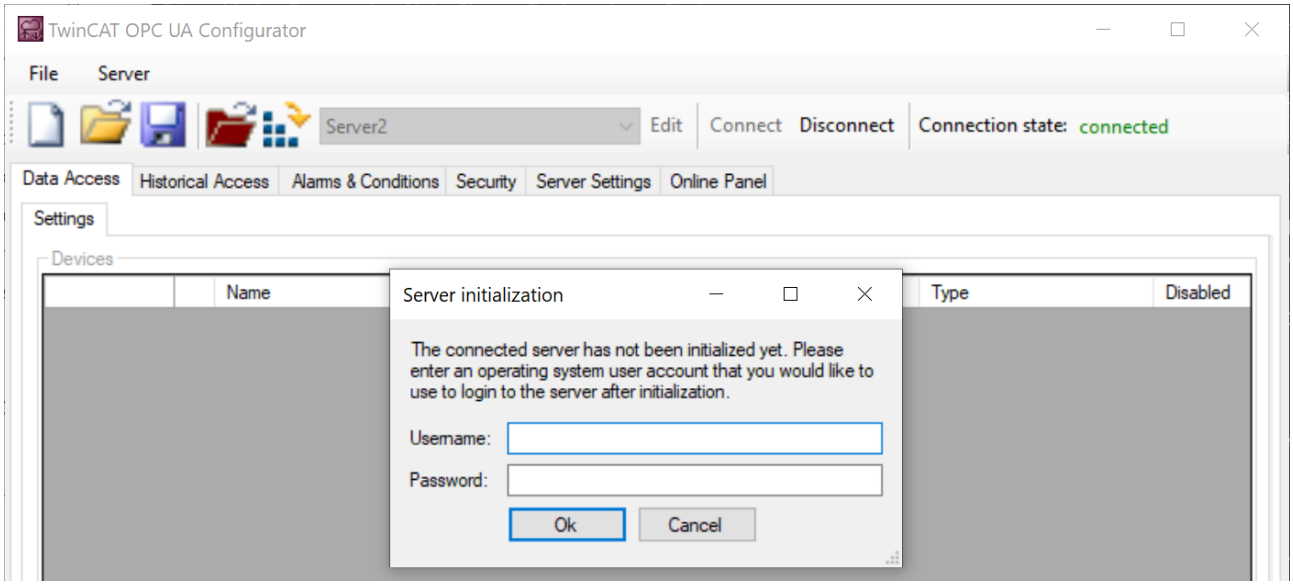


After the connection has been established, the initialization object together with the TrustOnFirstUse method can be found in the server's address space.



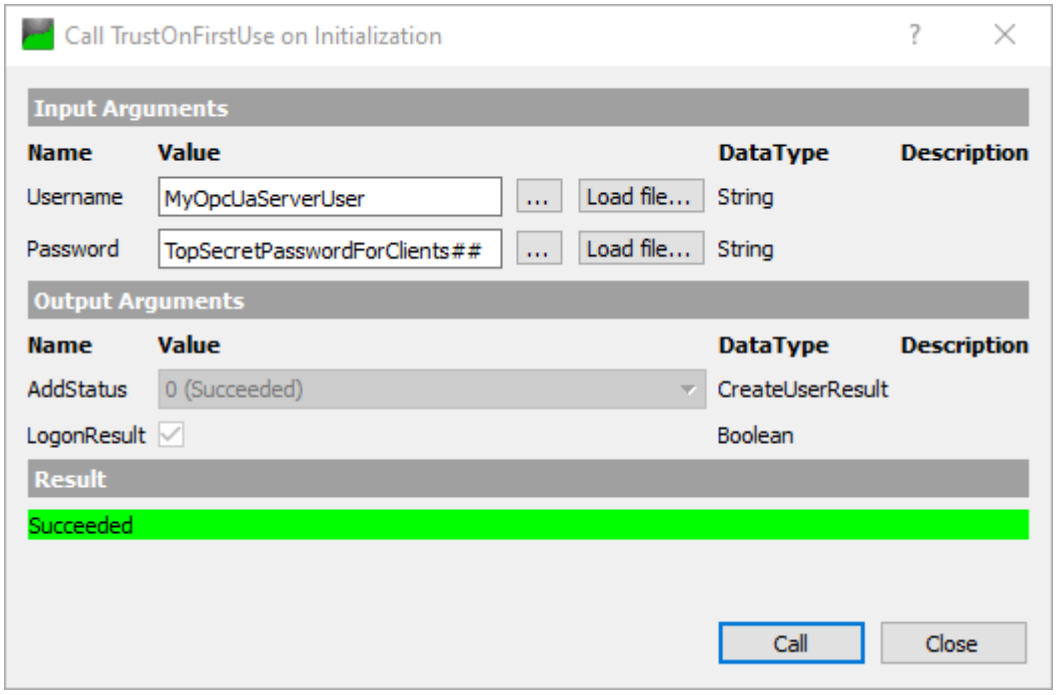
Step 2: OPC UA Client starts TrustOnFirstUse

The TrustOnFirstUse method can be called via any OPC UA client, e.g. the UA Expert. However, Beckhoff's own configuration tools also allow the use of this initialization interface. The TwinCAT OPC UA Configurator (standalone or Visual Studio integrated) automatically detects an uninitialized server when a connection is established and enables initialization via a corresponding configuration interface:



The following steps show the same process as it can be done manually e.g. in the UA Expert software:

In the UA Expert, the TrustOnFirstUse method is called to create a user and configure the server for this user. "MyOpcUaServerUser" was used as the username in this example. The password must meet the complexity requirements of the operating system, otherwise the initialization will fail. The following screenshot shows the successful call of the method.



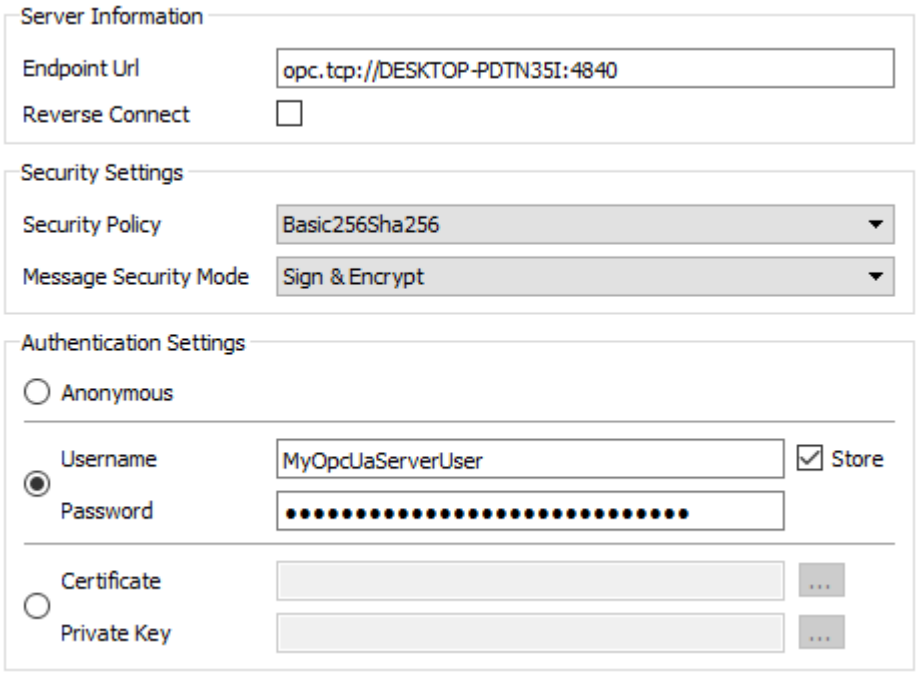
The parameter AddStatus indicates that the user was successfully created in the operating system's user database. The parameter LogonResult indicates that an initial test authentication of the server with the specified user information was successful.

The server restarts automatically after this successful method call.

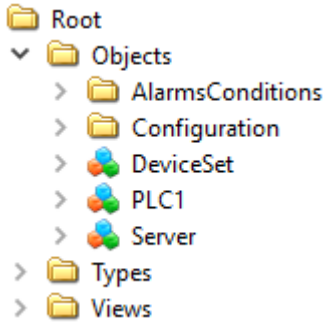
Step 3: OPC UA client logs on to the initialized server

Username/password disables anonymous access

Please note that the UA Expert cannot automatically reconnect to the server after the method call, because the anonymous access has been disabled and from now on the logon must be done using the specified username.



Once the connection has been established, the regular namespaces and objects can be found again in the server's address space and the configuration of the application can begin.



Permissions of the TOFU user

The user configured by the TOFU mechanism has full access to the server, which may not be desirable. Beckhoff therefore recommends creating an explicit user for pure data access in the next step, see Recommended steps.

Manually adding the TOFU user to the user database

In certain system configurations, the TwinCAT OPC UA Server may not have sufficient authorizations to automatically add the specified user to the user database automatically during initialization.

This occurs, for example, when Server version 5.4 is used on a TwinCAT UMRT standalone system under Microsoft Windows. In this environment, the TwinCAT OPC UA Server is executed with restricted system rights. This means that it does not have write access to the user database of the operating system, which prevents automatic creation of user accounts.

Procedure

To ensure that the configured user can be used correctly, it must be created manually in the operating system before the server is initialized.

This can be done via Windows Computer Management, for example:

- Open Computer Management
- Navigate to **Local users and groups**
- Create the user. The user does not need to have any further authorizations and does not need to be added to a user group.

The TwinCAT OPC UA Server should only be initialized after the user account has been created successfully. You will receive an error message stating "Access denied (not root)" during initialization, but the login with the created user will work (LogonResult) and the initialization is thus complete.

4.4 Recommended steps

After the initial commissioning, Beckhoff recommends that you pay attention to the following points to further configure the server and ensure a stable and secure operating environment.

Data Access

Data Access describes a function of OPC UA for displaying symbols and the corresponding access to them in the address space of the server. In the TwinCAT OPC UA Server, the configuration of data access devices is an elementary component and the basis for further functionalities. We therefore recommend that you read our chapter on Data Access [▶ 56] in the next step, which also describes how you can establish a connection with the runtime [▶ 56].

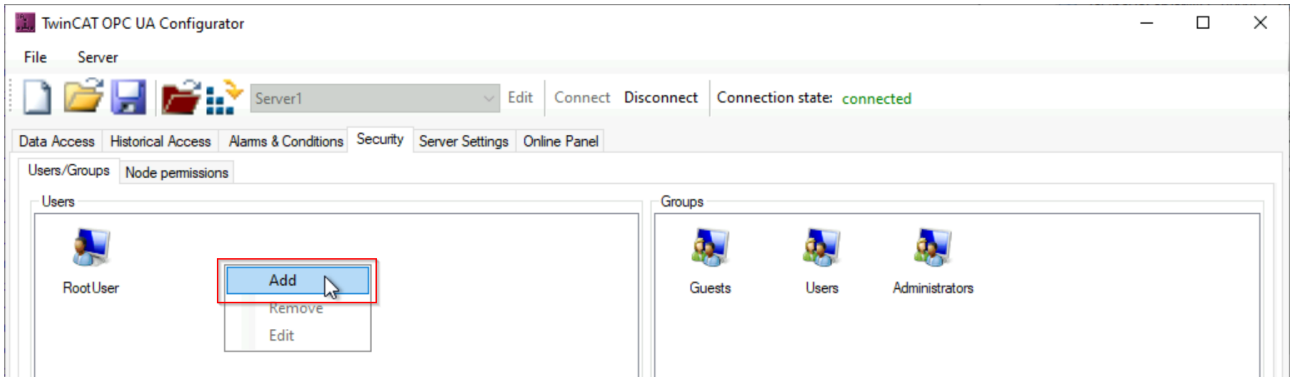
Only use secure IdentityTokens

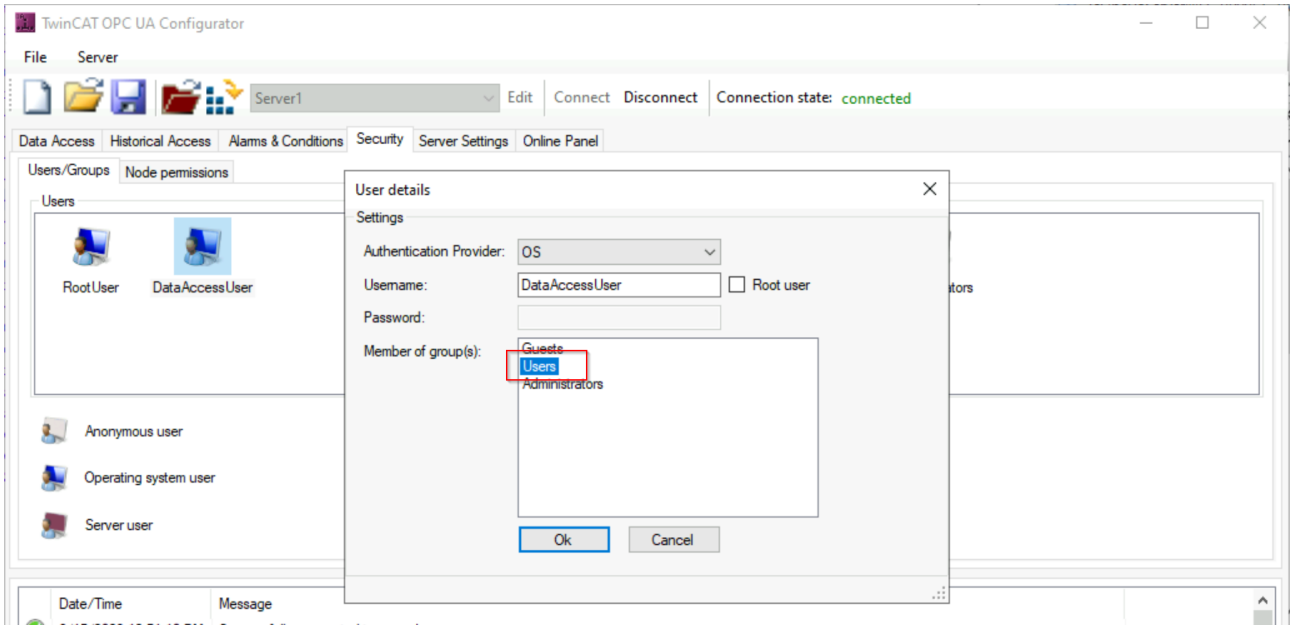
The one-time initialization of the server disables the IdentityToken "Anonymous". **For security reasons, you should leave this disabled.** The server should only be accessed by authenticated client applications, such as the username/password authentication configured by default during initialization.

Creation of a user for pure data access

The aforementioned initialization of the server configures a user for access to the server and then disables anonymous access to the server. The configured user has full access to all objects in the server namespace. In most application scenarios, this is not desired and the administrator user should be separated from the application user.

Beckhoff therefore recommends configuring an additional, dedicated user who is given the necessary permissions to access variables on a Data Access device, but who is not allowed to access the configuration namespace. This setting can be made via the configurator by adding a new user who is assigned to the "Users" group.





The newly configured user then has all the necessary permissions to access TwinCAT variables, to read the type system, but not to influence the configuration of the server. Please note that if you use the authentication provider "OS", you must also create the user in the operating system, i.e. it must exist there.

NOTICE

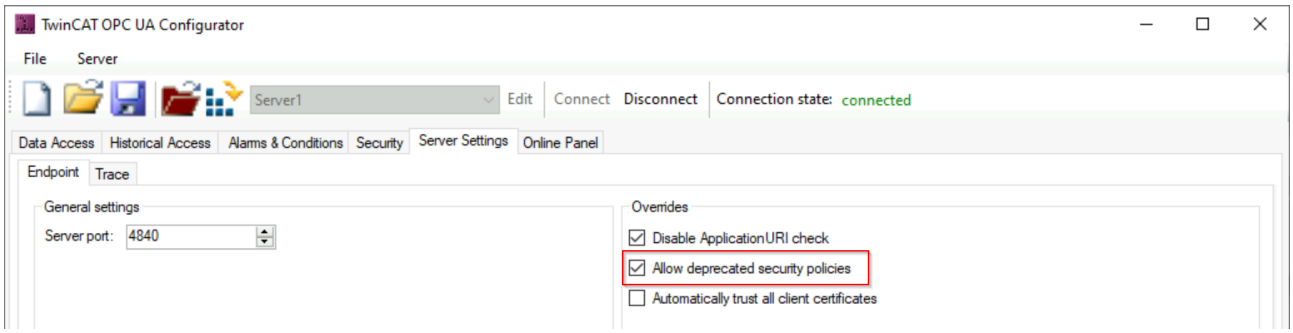
Leave insecure endpoints disabled

✓ Endpoints classified as unsafe are not offered by the TwinCAT OPC UA Server by default. These can be made available in the server via a configuration switch - Beckhoff does not recommend this!

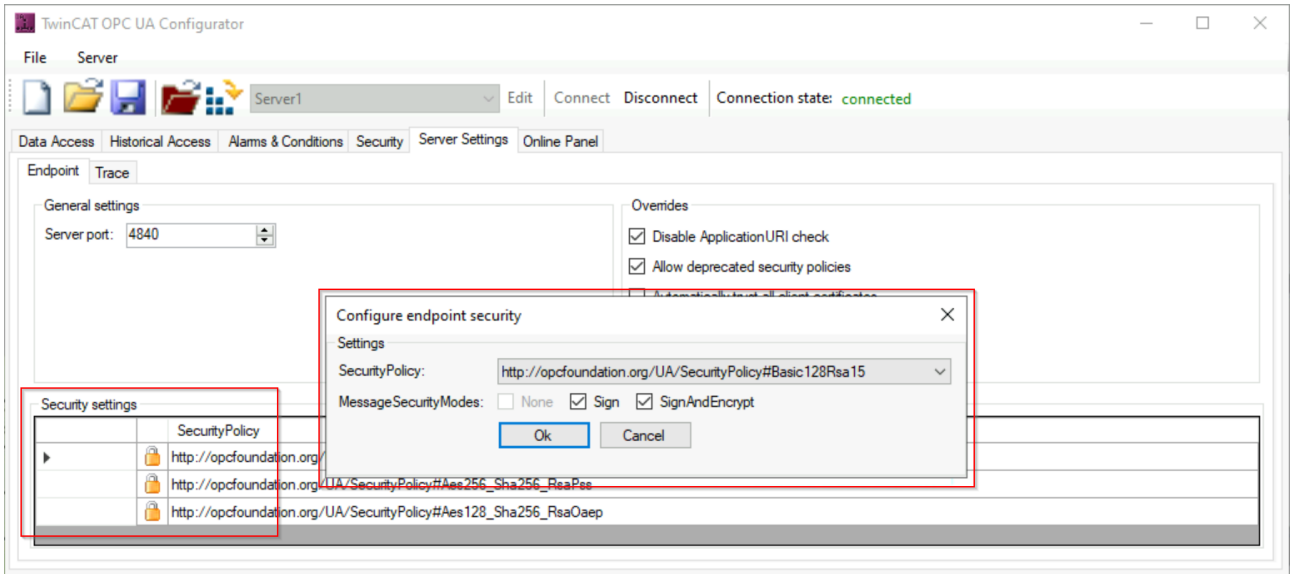
a) Only use endpoints that are currently considered secure.

b) Observe and follow the other safety-relevant recommendations in the following section.

The following screenshot shows you how to enable older server endpoints in the TwinCAT OPC UA Configurator.



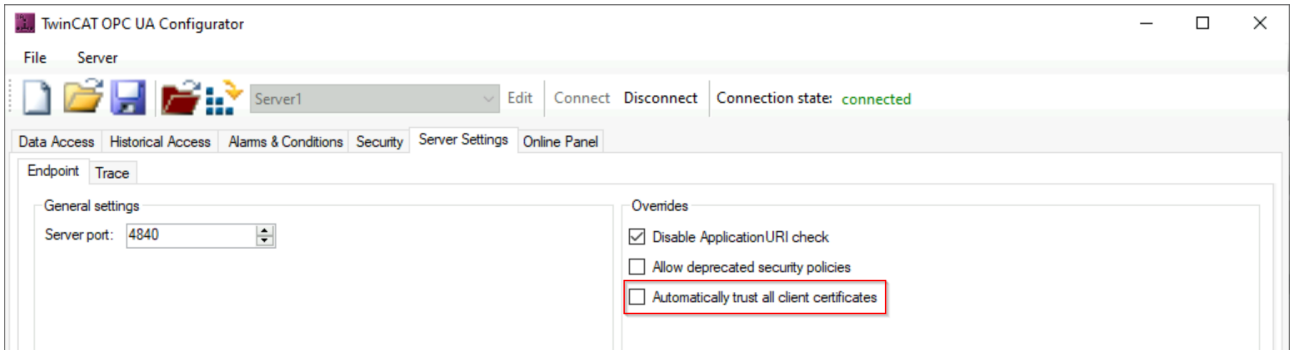
You can then add the insecure endpoints back to the server configuration, for example via the context menu in the configurator in the "Security Settings" area:



The None/None endpoint is already disabled when the server is delivered. For security reasons, Beckhoff recommends that you also leave this endpoint disabled and only allow access to the server via a secure endpoint. If required, the None/None endpoint can be added back to the server configuration using the method described above.

Disable 'AutomaticallyTrustAllClientCertificates'

By default, the server is configured for easy commissioning so that it automatically trusts all client certificates without having to manually exchange certificates on the server side. For security reasons, Beckhoff recommends disabling this setting. This setting can be made via the TwinCAT OPC UA Configurator, as shown in the following screenshot:



After disabling this setting, a trust relationship must be established between the client and server by both applications trusting each other's certificates.

4.5 Supported features

OPC UA offers a large number of features. This product may not support all features from the very beginning. Further features will be added over time through product updates. The following table shows a list of all currently supported functionalities of the current product version.

Compatibility with the OPC UA Pub/Sub specification

The current version of this product is compatible with the OPC UA Pub/Sub specification version 1.05.03.

Certification profile

The current version of this product is certified for the "Standard UA Server" profile. The Conformance Units checked in this certification are documented by the OPC Foundation.

Specification

The OPC UA specification is divided into different parts. Parts 1 to 5 define the core functionalities of OPC UA. The following table provides an overview of the various parts of the specification and indicates whether this product offers corresponding support. Below you will find additional detailed information on the supported functions from the respective part, if applicable. Please note that not all parts can be supported, as some parts of the specification are outside the product definition.

Part	Supported
OPC10000-1 Overview and Concepts	Yes
OPC10000-2 Security Model	Yes
OPC10000-3 Address Space Model	Yes
OPC10000-4 Services	Yes
OPC10000-5 Information Model	Yes
OPC10000-6 Mappings	Yes
OPC10000-7 Profiles	Yes
OPC10000-8 Data Access	Yes
OPC10000-9 Alarms & Conditions	Yes
OPC10000-10 Programs	No
OPC10000-11 Historical Access	Yes
OPC10000-12 Discovery and Global Services	Yes
OPC10000-13 Aggregates	No
OPC10000-14 Pub/Sub	No *1
OPC10000-15 Safety	No
OPC10000-16 State Machines	No
OPC10000-17 Alias Names	No
OPC10000-18 Role-Based Security	No *2
OPC10000-19 Dictionary References	No
OPC10000-20 File Transfer	Yes
OPC10000-21 Device Onboarding	No
OPC10000-22 Base Network Model	---
OPC10000-23 Common ReferenceTypes	---
OPC10000-24 Scheduler	No

*1: OPC UA Pub/Sub is mapped by the product TF6105 TC3 OPC UA Pub/Sub

*2: This product includes its own user rights implementation, which already existed before the release of Part 18 and implements a user/group-based authorization system [▶ 133].

OPC10000-4 Services

The following table provides an overview of the services (Service Sets, Service Behaviors, etc.) supported by this product as described in Part 4 of the OPC UA specification.

Feature	Supported
Attributes Read/Write	Yes
Attributes HistoryRead	Yes (Value attribute)
Attributes HistoryUpdate	Yes (Value attribute)
Subscriptions	Yes
Durable Subscriptions	No
Monitored items	Yes
Method Calls	Yes
Auditing	No
Redundancy	No

OPC10000-8 Data Access

The following table provides an overview of the Data Access functions supported by this product as described in Part 8 of the OPC UA specification.

Feature	Supported
DataItemType	Yes
AnalogItemType	Yes
DiscreteItemType	Yes *1
ArrayItemType	Yes
EUInformation	Yes *1
ComplexNumberType	Yes *1
DoubleComplexNumberType	Yes *1
AxisInformation	Yes *1
AxisScaleEnumeration	Yes *1
XVType	Yes *1
ObjectTypes	Yes (Function Blocks)
StructuredTypes	Yes
ServerTimestamp	Yes
SourceTimestamp	Yes
Quality	Yes

*1: Supported via the Nodeset import and TE6100 OPC UA Nodeset Editor.

OPC10000-9 Alarms & Conditions

The following table provides an overview of the Alarms & Conditions functions supported by this product as described in Part 9 of the OPC UA specification.

Feature	Supported
LimitAlarmType	Yes
OffNormalAlarmType	Yes
Multi-Language Alarm-/Eventtexts	Yes
TwinCAT 3 EventLogger	Yes *1
Custom EventTypes	Yes *2
Custom AlarmTypes	Yes *2

*1: Support of the TwinCAT 3 EventLogger via proprietary Alarm/EventTypes

*2: Supported via the Nodeset import and TE6100 OPC UA Nodeset Editor.

OPC10000-11 Historical Access

The following table provides an overview of the Historical Access functions supported by this product as described in Part 11 of the OPC UA specification.

Feature	Supported
HistoricalDataNodes [▶ 98]	Yes
HistoricalEventNodes	No
Historical Audit Events	No
HistoryUpdate [▶ 101]	Yes
External History Sources	No

OPC10000-20 File Transfer

This product supports the OPC UA-side exchange of configuration files, which are used for commissioning the product. A generic File Transfer [▶ 136] functionality is implemented in the OPC UA server of the Beckhoff Device Manager software.

Security

The following table provides an overview of the Security features supported by this product. Some of the features shown here are split across several parts of the specification.

Feature	Supported
SecureChannel	Yes
SecureEndpoints [▶ 128]	Yes
Application Instance Certificate	Yes
Certificate Trust Relationship	Yes
IdentityToken Anonymous	Yes
IdentityToken Username/Password	Yes
IdentityToken user certificates	Yes
User rights at Namespace level	Yes
User rights at Node level	Yes

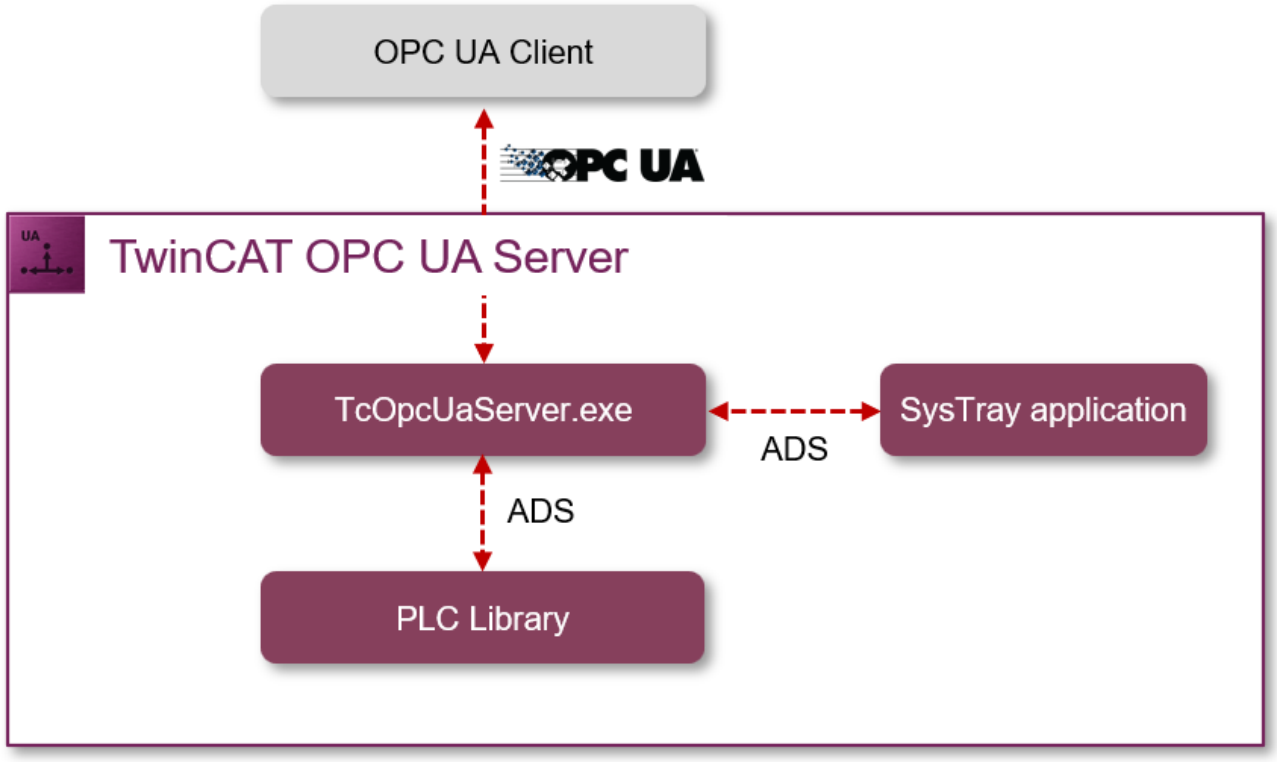
4.6 Software architecture

You do not need to know the internal software architecture of this product to use it, but it may be of interest in some cases. This is why we will briefly introduce you to them below.

The TwinCAT OPC UA Server essentially consists of the following components:

- The process in the operating system
- The application in the Windows System Tray [▶ 146]
- The PLC library Tc2_OpcUa [▶ 148]

The interaction of the individual components is described in more detail in the following diagram:



Process in the operating system

The process in the operating system (TcOpcUaServer.exe) takes care of the OPC UA protocol functions (i.e. communication with the OPC UA clients), the provision of the OPC UA address space and communication with the lower-level devices [▶ 56]. Furthermore, the process provides an ADS server interface so that the PLC library Tc2_OpcUa and the Windows System Tray application can interact with the server application.

Windows System Tray application

This application enables the triggering of a restart of the TwinCAT OPC UA Server. The corresponding function can be called up via the icon in the Windows System Tray.

PLC library

The PLC library Tc2_OpcUa enables interaction with the TwinCAT OPC UA Server both for retrieving status information and for triggering a restart.

4.7 Configurator

Two graphical user interfaces are available for simple configuration of the TwinCAT OPC UA Server as part of the TwinCAT OPC UA Configurator: an interface integrated into Visual Studio (or the XAE Shell) and a standalone tool.

Documentation for the TwinCAT OPC UA Configurator

Although the TwinCAT OPC UA Configurator is discussed in various sections of this documentation, it also has its own product documentation, which can be found in the Beckhoff information system.

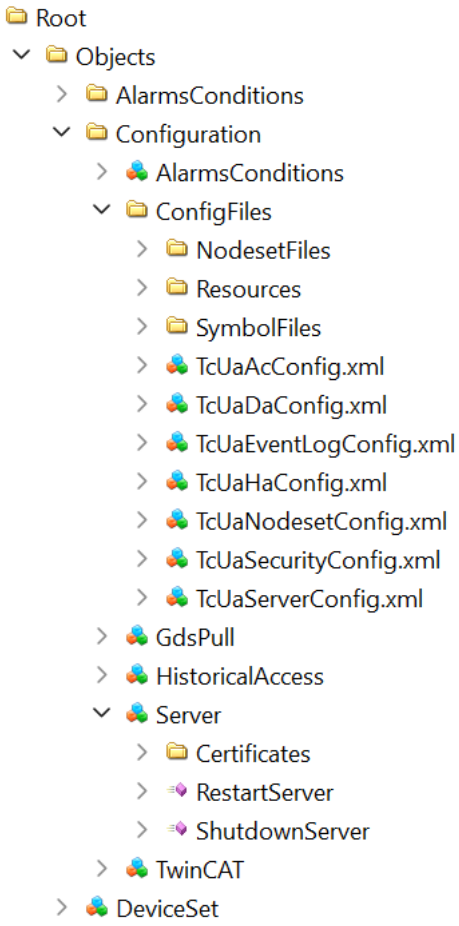
Configuration via OPC UA

The TwinCAT OPC UA Server contains a so-called configuration namespace, which enables local and remote configuration of the server via OPC UA. The TwinCAT OPC UA Configurator makes use of this interface to access the individual configuration files of the server.

The following functionalities are mapped in the configuration namespace:

- Management of the server configuration files
- Management of the certificates
- Restarting the server

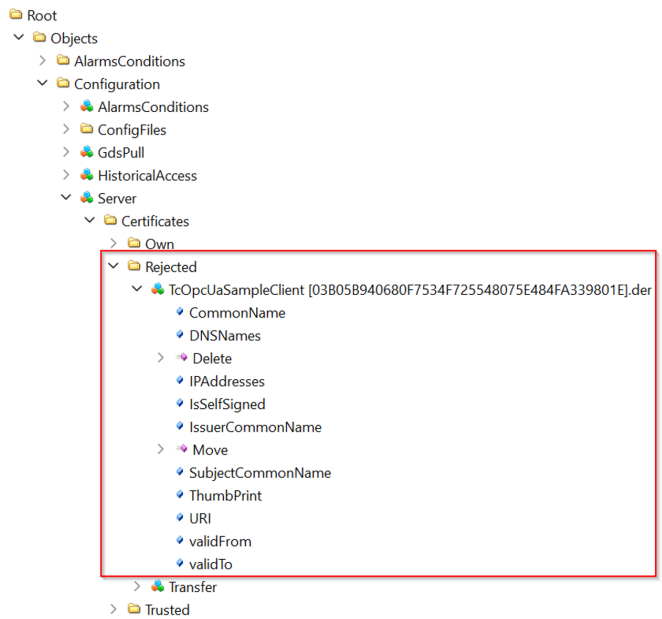
After initializing [▶ 34] the server, the user configured there has access to the configuration namespace and can be used to further configure the server. The following screenshot shows the configuration namespace from the perspective of an OPC UA client (in this example, the UA Expert from Unified Automation):

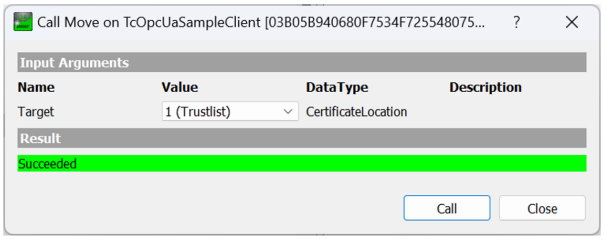


All server configuration files are available as objects of type FileType. This object type and access to it is standardized by OPC UA. Certificate files are created in the configuration namespace as CertificateType.

Management of certificates

Client certificates are divided into "Rejected" and "Trusted" certificates, which is represented by a separate folder in the namespace. A certificate can be moved between the trusted lists by calling the method `Move()` on a certificate object. In addition, various properties provide additional information about the certificates themselves for easier identification.

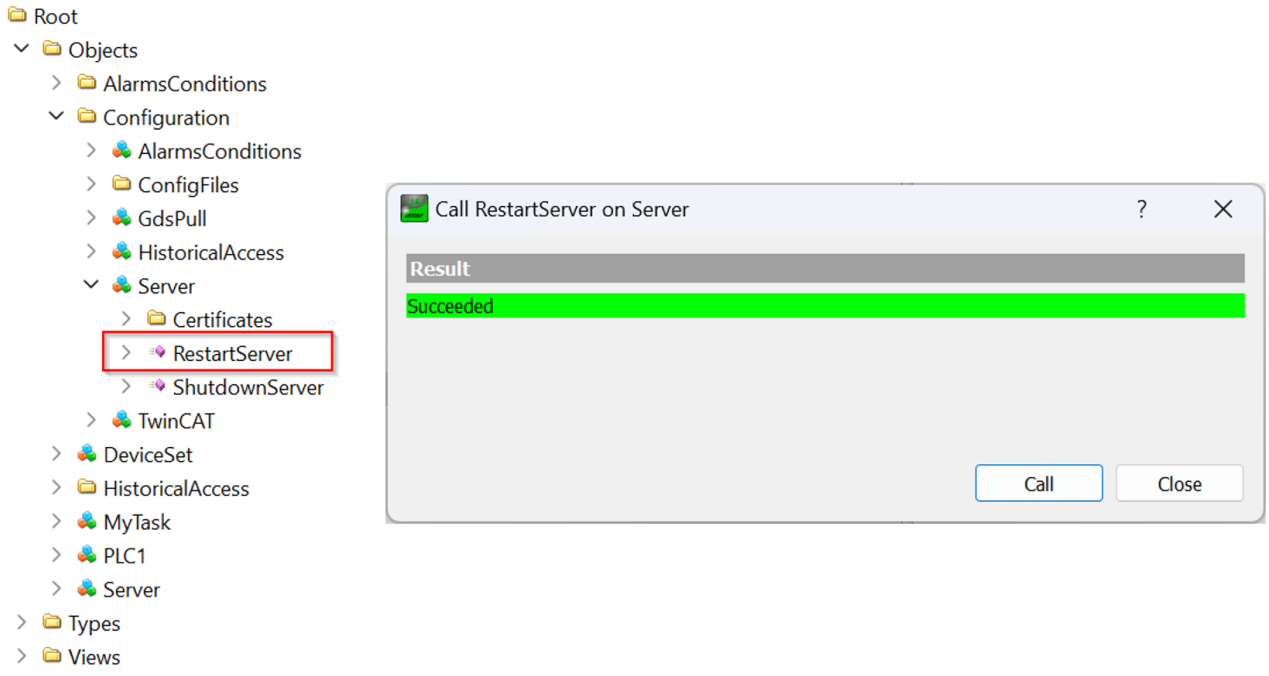




The TwinCAT OPC UA Configurator provides you with a user interface to make it easier to trust/reject certificates. Further information can be found in chapter Certificate exchange [▶ 129].

Restarting the server

The configuration namespace contains a method with which the TwinCAT OPC UA Server can be restarted without the need for a TwinCAT restart.



4.8 Optimizations

There are various ways to optimize the communication connection between OPC UA client and server or PLC. The runtime behavior, in particular the CPU and memory load of the TwinCAT OPC UA Server, can also be optimized using various parameters. This chapter presents various optimization options, in particular on the following topics:

- SamplingInterval vs. PublishingInterval
- StructuredTypes
- StructuredTypes and their member variables

The screenshots and performance values shown here represent examples under laboratory conditions, which were run on different hardware devices. Therefore, they cannot be transferred 1:1 to customer projects and only serve to illustrate certain facts.

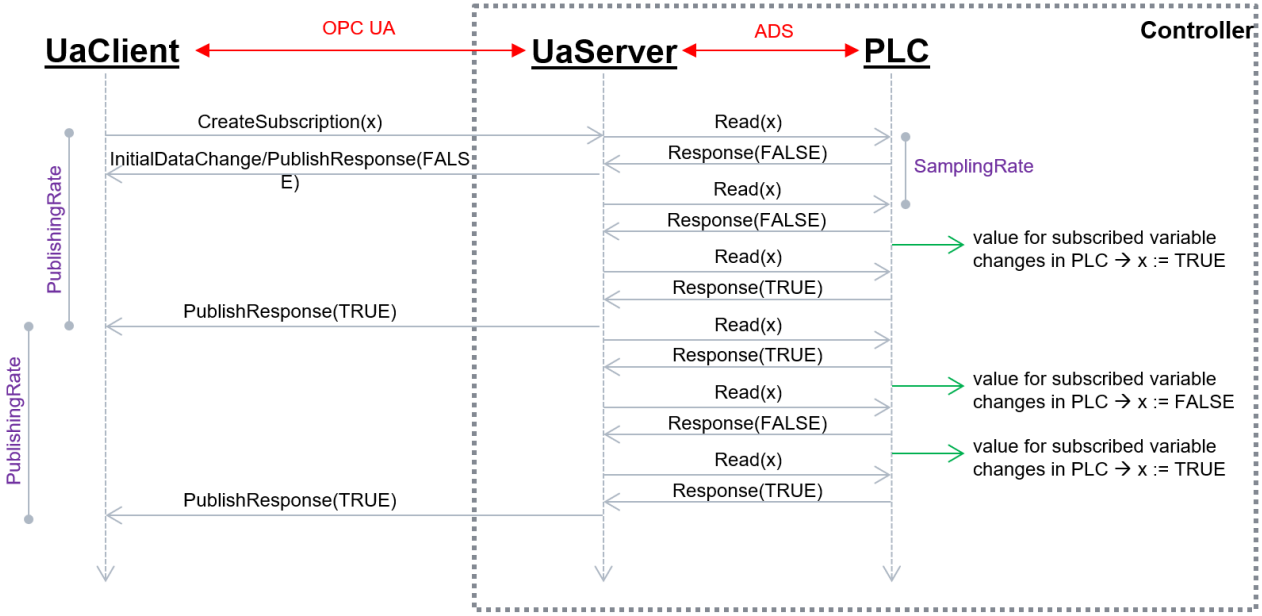
SamplingInterval vs. PublishingInterval

When creating a subscription, an OPC UA client uses various parameters for the subscription and the so-called MonitoredItems contained in it to receive notifications about variable changes. The following table explains two of these parameters, which will then be described in more detail.

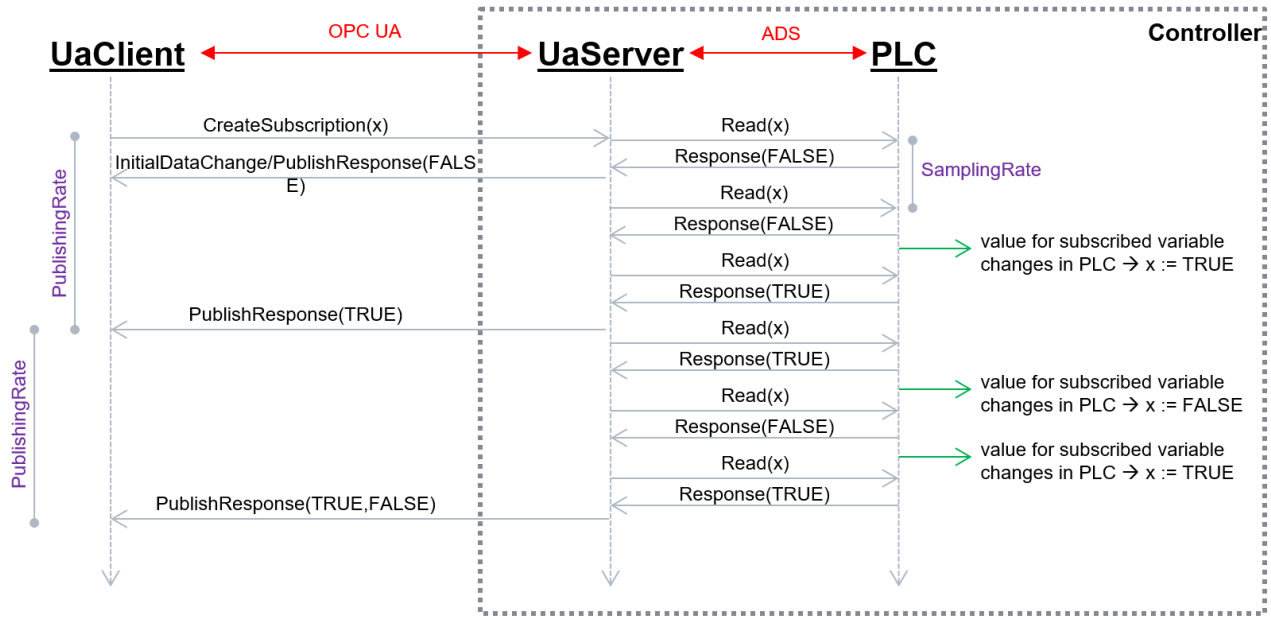
Parameter	Description
PublishingInterval	The PublishingInterval specifies the rate at which an OPC UA client is informed about value changes by the server. The PublishingInterval is described in detail in Part 4 of the OPC UA specification.
SamplingInterval	The SamplingInterval specifies the rate at which the OPC UA server should sample its underlying data source for value changes, in the case of the

Parameter	Description
	TwinCAT OPC UA Server via the ADS connection. The SamplingInterval is described in detail in Part 4 of the OPC UA specification.

The following figure illustrates the relationship between these two parameters once again. It is assumed here that the TwinCAT OPC UA Server has been installed on the PLC controller and that the OPC UA client accesses the server from an external system.



As can be seen in the figure, the situation can arise that the OPC UA client does not notice certain value changes in the PLC, e.g. if they happen "too fast" in the PLC or the sampling rate is not high enough or a variable value returns to the original value (as can be seen above). Via another parameter, the so-called QueueSize, several value changes between the PublishingIntervals can be recorded and transferred to the OPC UA client. In the above example, a QueueSize of 1 was selected, i.e. the "Last Known Value" is always transferred to the client. In the following figure, on the other hand, a QueueSize of 2 was selected, i.e. the last two known value changes are transmitted to the client.

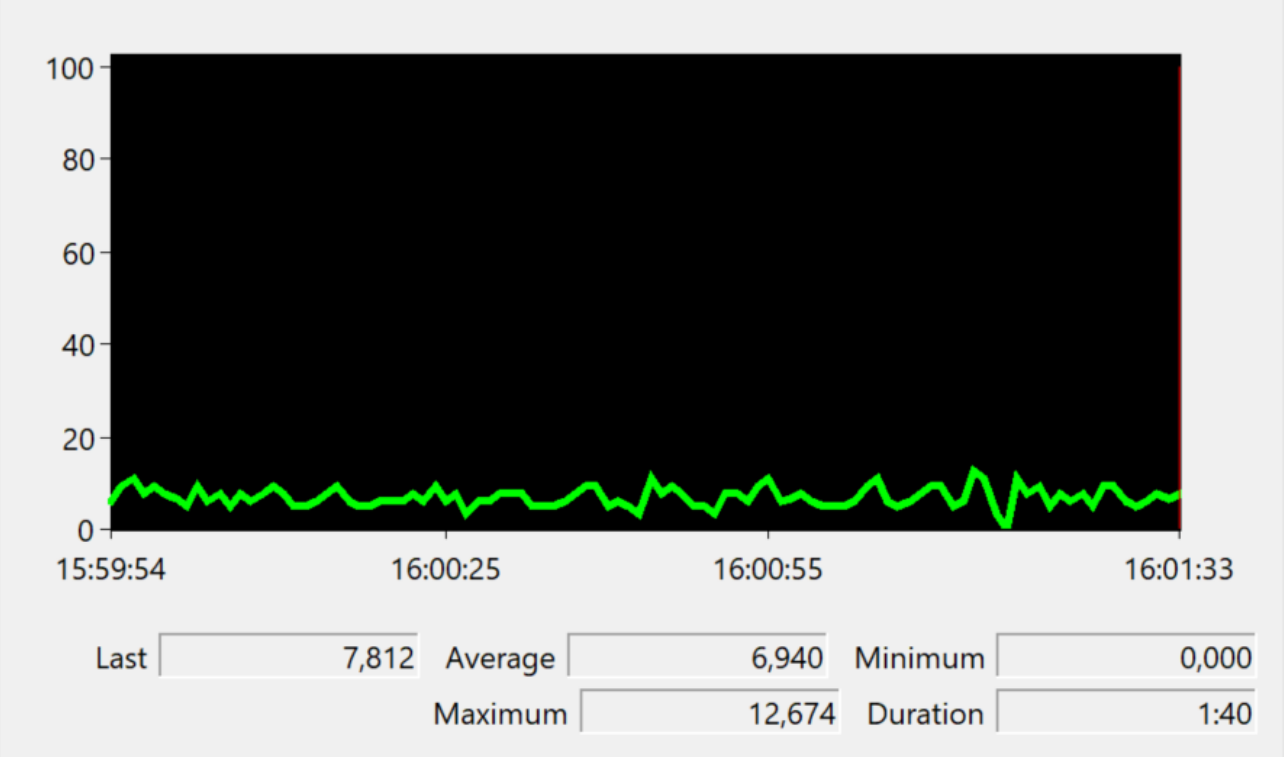


With the first PublishResponse it can be seen that only one value change is transmitted to the client, because also only one value change has taken place in the PLC. With the second PublishResponse it can be seen that two value changes have occurred in the PLC and both are also transmitted to the client.

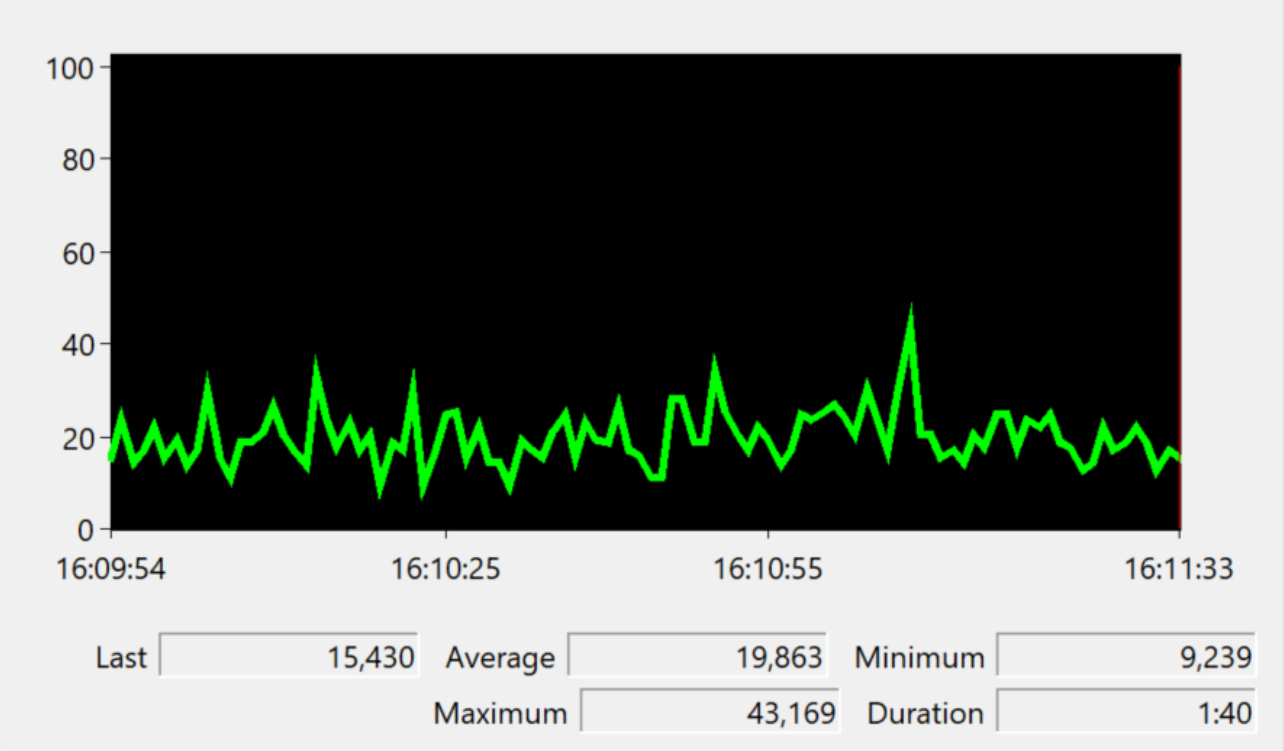
The parameters described above are settings that the client usually controls and requests from the server. And it is exactly here that many optimizations can be made, because both parameters have a strong influence on how much CPU time the OPC UA client and server need, since a corresponding amount of information has to be processed or requested.

Depending on the application scenario used, the two parameters should be set appropriately. For example, if the OPC UA client is a visualization, then fast PublishingIntervals and SamplingRates make only limited sense, since the human eye cannot process information faster than ~200 ms anyway. The use of the QueueSize should also be chosen sensibly depending on the situation. If the OPC UA client does not process any values from the queue anyway, a QueueSize of 1 is sufficient and makes sense, since correspondingly less information has to be transferred and this further optimizes the system.

In the following example an OPC UA client has created a subscription with 10,000 variables on the TwinCAT OPC UA Server. 500 ms was selected as the PublishingInterval and 250 ms as the SamplingInterval. The CPU load of the TwinCAT OPC UA Server was at an average value of about 6.9%, see the following screenshot of the Windows Performance Monitor.



Then the PublishingInterval was set to 200 ms and the SamplingInterval to 100 ms. This increased the CPU load of the TwinCAT OPC UA Server to an average value of approx. 19%.



StructuredTypes

By default, the TwinCAT OPC UA Server provides structures [▶ 82] as FolderType in its address space. The member variables are then displayed as separate nodes below the folder and can be accessed. The server also allows the use of StructuredTypes for the structure, whereby the type information of the structure is processed by the client and the structure can be read/written in a data-consistent manner. When communicating with the PLC (via the TwinCAT ADS protocol), the two variants behave fundamentally differently.

When an OPC UA client accesses individual member variables, the ADS communication may well be split into multiple ADS read/write commands, depending on the number of variables. This in turn can result in the ADS commands being processed by the PLC in different PLC cycles. Data consistency cannot therefore be guaranteed.

When an OPC UA client accesses a StructuredType, on the other hand, it is ensured that the StructuredType is processed by the PLC in a single ADS Read/Write command, thus ensuring data consistency.

When using StructuredTypes for large PLC data structures, the following should be considered:

- Depending on the structure size, the ADS read/write command or the corresponding response can become very large and require a correspondingly large amount of memory in the TwinCAT ADS router. **Therefore, please pay attention to the router memory.**
- StructuredTypes must be encoded or decoded by the TwinCAT OPC UA Server when communicating with the TwinCAT real-time environment, which requires additional CPU time.
- Depending on the size of the structure, read/write commands can take a long time, which is also heavily dependent on the time the server needs to encode/decode.

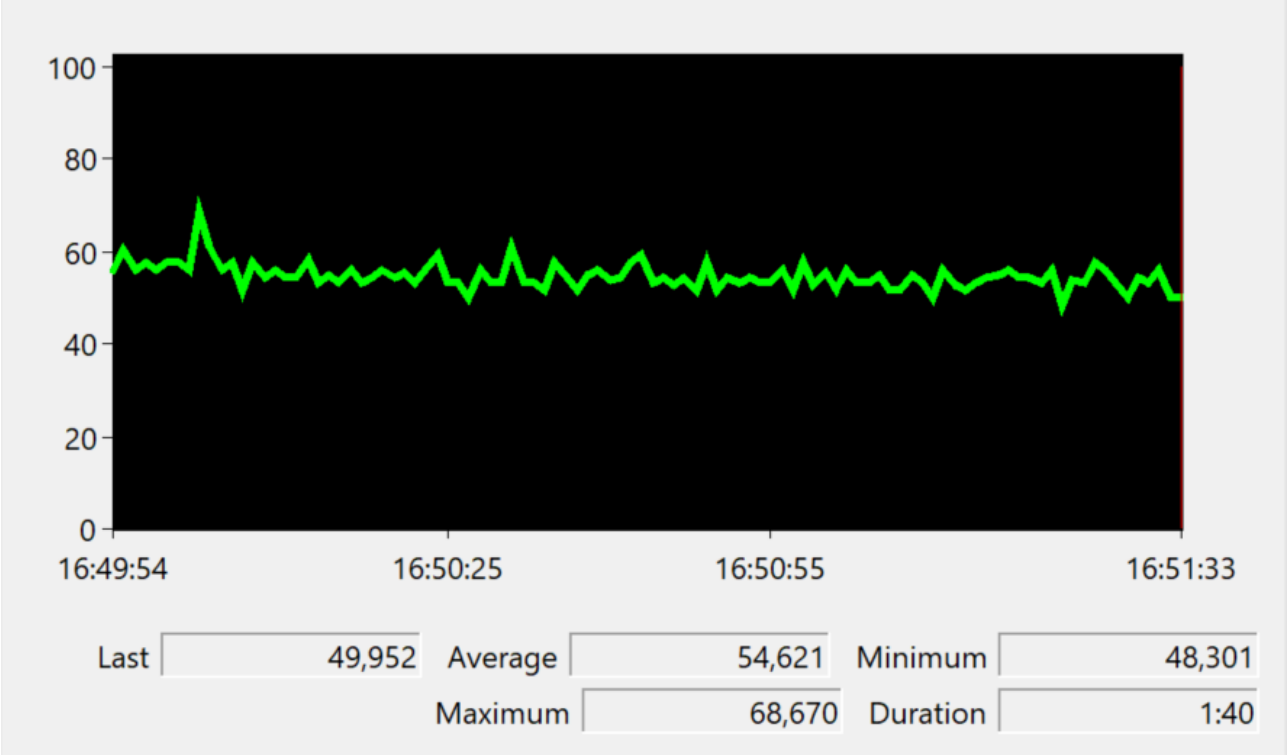
For these reasons, the maximum size of a StructuredType is limited in the delivery state of the server, but can be adjusted as required using the parameter <MaxStructureSize> in the configuration file TcUaDaConfig.xml. This parameter specifies the maximum size of a structure in bytes. If a structure exceeds <MaxStructureSize>, it is imported as FolderType, where each structure element is available as a single node. For more information, see the chapter Structures [▶ 82].

Especially in connection with the parameters for the SamplingInterval and PublishingInterval that can be set for a subscription, some optimizations can be made with StructuredTypes, which can have a major influence on the runtime behavior of the server.

In the following sample, an OPC UA client has created a subscription to a StructuredType. The lower-level PLC data structure is structured as follows:

```
TYPE ST_TEST :
STRUCT
  stComplex : ST_Complex_1;
  strString5 : STRING[5];
  eEnum : E_Enum_1;
  strString3 : STRING[3];
  arrComplex : ARRAY[0..9999] OF ST_Complex_1;
  arrDint : ARRAY[0..9999] OF DINT;
END_STRUCT
END_TYPE
```

The structure ST_Complex_1 used as member variable is about 91 bytes. In total, this is a data structure with a size of about 1 MB. 500 ms was selected as the PublishingInterval and 250 ms as the SamplingInterval. The CPU load of the TwinCAT OPC UA Server after creating the subscription was at an average value of about 54.6%, see the following screenshot of the Windows Performance Monitor.



According to the selected SamplingInterval, an ADS Read is sent every 250 ms for the lower-level ADS communication. In the corresponding response you can clearly see the size of the data structure of approx. 1 MB, i.e. every 250 ms a 1 MB data packet is transported through the TwinCAT ADS router (and must be processed accordingly by the server). The encoding/decoding process from the server requires a lot of CPU power, which explains the high CPU load.

15/06/2022 16:55:17 326 ms	R	Req	10.0.2.15.1.1 (33391)	10.0.2.15.1.1 (851)	0	0x755	12	IG 0xf005 IO 0x1a000210 Len 1000108
15/06/2022 16:55:17 327 ms	R	Res	10.0.2.15.1.1 (851)	10.0.2.15.1.1 (33391)	0	0x755	1000116	Res 0x0, Len 1000108 + 03 00 00 00 2a 00 00 00 48
15/06/2022 16:55:17 574 ms	R	Req	10.0.2.15.1.1 (33359)	10.0.2.15.1.1 (851)	0	0x852	12	IG 0xf005 IO 0x1a000210 Len 1000108
15/06/2022 16:55:17 575 ms	R	Res	10.0.2.15.1.1 (851)	10.0.2.15.1.1 (33359)	0	0x852	1000116	Res 0x0, Len 1000108 + 03 00 00 00 2a 00 00 00 48
15/06/2022 16:55:17 822 ms	R	Req	10.0.2.15.1.1 (33391)	10.0.2.15.1.1 (851)	0	0x756	12	IG 0xf005 IO 0x1a000210 Len 1000108
15/06/2022 16:55:17 823 ms	R	Res	10.0.2.15.1.1 (851)	10.0.2.15.1.1 (33391)	0	0x756	1000116	Res 0x0, Len 1000108 + 03 00 00 00 2a 00 00 00 48
15/06/2022 16:55:18 70 ms	R	Req	10.0.2.15.1.1 (33359)	10.0.2.15.1.1 (851)	0	0x853	12	IG 0xf005 IO 0x1a000210 Len 1000108
15/06/2022 16:55:18 71 ms	R	Res	10.0.2.15.1.1 (851)	10.0.2.15.1.1 (33359)	0	0x853	1000116	Res 0x0, Len 1000108 + 03 00 00 00 2a 00 00 00 48
15/06/2022 16:55:18 318 ms	R	Req	10.0.2.15.1.1 (33391)	10.0.2.15.1.1 (851)	0	0x757	12	IG 0xf005 IO 0x1a000210 Len 1000108
15/06/2022 16:55:18 319 ms	R	Res	10.0.2.15.1.1 (851)	10.0.2.15.1.1 (33391)	0	0x757	1000116	Res 0x0, Len 1000108 + 03 00 00 00 2a 00 00 00 48

StructuredTypes and their member variables

By default, the member variables of a StructuredType are represented and made available as separate nodes in the server's address space. This requires additional main memory, because the TwinCAT OPC UA Server allocates main memory for each node. An OPC UA client that works exclusively

with the StructuredType, i.e. the "root element" of a structure, does not require these additional nodes. These can be explicitly hidden using a special pragma, which reduces the server's memory load. The pragma is described in more detail in the chapter Structures [▶ 82].

4.9 Application directories

This application uses various directories to store relevant information, e.g. configuration or certificate files.

Installation directory

The base installation directory of the application is always relative to the TwinCAT installation directory on all operating systems.

```
%TcInstallDir%\Functions\TF6100-OPC-UA
```

The application is then installed in the following directory below this directory. A distinction is made here according to platform (x86/x64). As of TwinCAT Build 4026, the 64-bit variant of the server is registered and started on the system by default.

```
%TcInstallDir%\Functions\TF6100-OPC-UA\Win32\Server
%TcInstallDir%\Functions\TF6100-OPC-UA\Win64\Server
```

Base directory for PKI infrastructure

Certificate files, which are used to establish a secure communication connection, are stored in the following directory on all operating systems:

```
%ProgramData%\Beckhoff\TF6100-OPC-UA\TcOpcUaServer\PKI
```

Directory for trusted certificates

Client certificates in this directory are declared as "trusted". This path is identical on all operating systems.

```
%ProgramData%\Beckhoff\TF6100-OPC-UA\TcOpcUaServer\PKI\CA\trusted
```

Directory for rejected certificates

Client certificates in this directory are declared as "rejected". This path is identical on all operating systems.

```
%ProgramData%\Beckhoff\TF6100-OPC-UA\TcOpcUaServer\PKI\CA\rejected
```

Directory for the server certificate

The directories for the server certificate are defined as follows, whereby a distinction is made between the directory for the public key ("certs") and private key ("private").

```
%ProgramData%\Beckhoff\TF6100-OPC-UA\TcOpcUaServer\PKI\CA\own\certs
```

Log files

Log files are stored in the following directory. There is a distinction according to the operating system.

```
%ProgramData%\Beckhoff\TF6100-OPC-UA\TcOpcUaServer (Windows)
/var/log/TF6100-OPC-UA-Server (TwinCAT/BSD)
```

Configuration files

The TwinCAT OPC UA Server uses various configuration files to configure the individual functions, which are defined as listed in the table below.

File	Description
TcUaAcConfig.xml	Configuration file for the Alarms & Conditions [▶ 105] (A&C) functionality of the server. The nodes configured for A&C are saved in this file.

File	Description
TcUaDaConfig.xml	Configuration file for the Data Access [▶ 56] (DA) functionality of the server. The communication connections with the lower-level ADS devices are configured in this file.
TcUaGdsClientConfig.xml	Configuration file for the Global Discovery Server [▶ 123] configuration. The connection to a GDS is configured in this file.
TcUaHaConfig.xml	Configuration file for the Historical Access [▶ 98] (HA) configuration of the server. This file stores both the data memories used for HA and the nodes configured for this purpose.
TcUaNodesetConfig.xml	Configuration file for the Nodeset [▶ 76] import of the server. The Nodesets to be imported and their link to an ADS device are defined in this file.
TcUaSecurityConfig.xml	Configuration file for users, groups and Access Control Lists (ACL) as part of the definition of access rights [▶ 133] and authentication [▶ 131] mechanisms.
TcUaServerConfig.xml	Configuration file for various parameters of the server, e.g. configuration of the endpoints [▶ 128], Logging [▶ 145], Reverse Connect [▶ 138].

All configuration files are stored in the ProgramData directory by default. Access is usually not achieved by directly editing the XML files, but via the TwinCAT OPC UA Configurator.

```
%ProgramData%\Beckhoff\TF6100-OPC-UA\TcOpcUaServer
```

4.10 Namespaces

In OPC UA, a namespace is used to logically assign nodes and avoid naming conflicts between different information models or manufacturers. The unique identification is done via the NamespaceUri, not via the numerical NamespaceIndex.

The NamespaceIndex simply corresponds to the current position of a namespace in the server's NamespaceArray. As this assignment can change, clients should not accept a NamespaceIndex statically or save it permanently. Instead, the required index should be determined again from the NamespaceArray for each connection using the associated NamespaceUri.

The following table shows an example of the NamespaceArray in the TwinCAT OPC UA Server.

Name	Value
⌄	String Array[9]
[0]	http://opcfoundation.org/UA/
[1]	urn:MichaelKn-NB04:BeckhoffAutomation:TcOpcUaServer:1
[2]	http://opcfoundation.org/UA/DI/
[3]	http://PLCopen.org/OpcUa/IEC61131-3/
[4]	urn:BeckhoffAutomation:Ua:PLC1
[5]	urn:BeckhoffAutomation:Ua:Types:GlobalTypes
[6]	urn:BeckhoffAutomation:Ua:HA
[7]	urn:BeckhoffAutomation:Ua:AC
[8]	http://Beckhoff.com/TwinCAT/TF6100/Server/Configuration

In the TwinCAT OPC UA Server it is not possible to change the order of the namespaces or to freely define the NamespaceIndex.

4.11 Data Access

4.11.1 Overview

This chapter describes the steps required to configure the variables in the TwinCAT OPC UA Server namespace for **D**ata **A**ccess (DA).

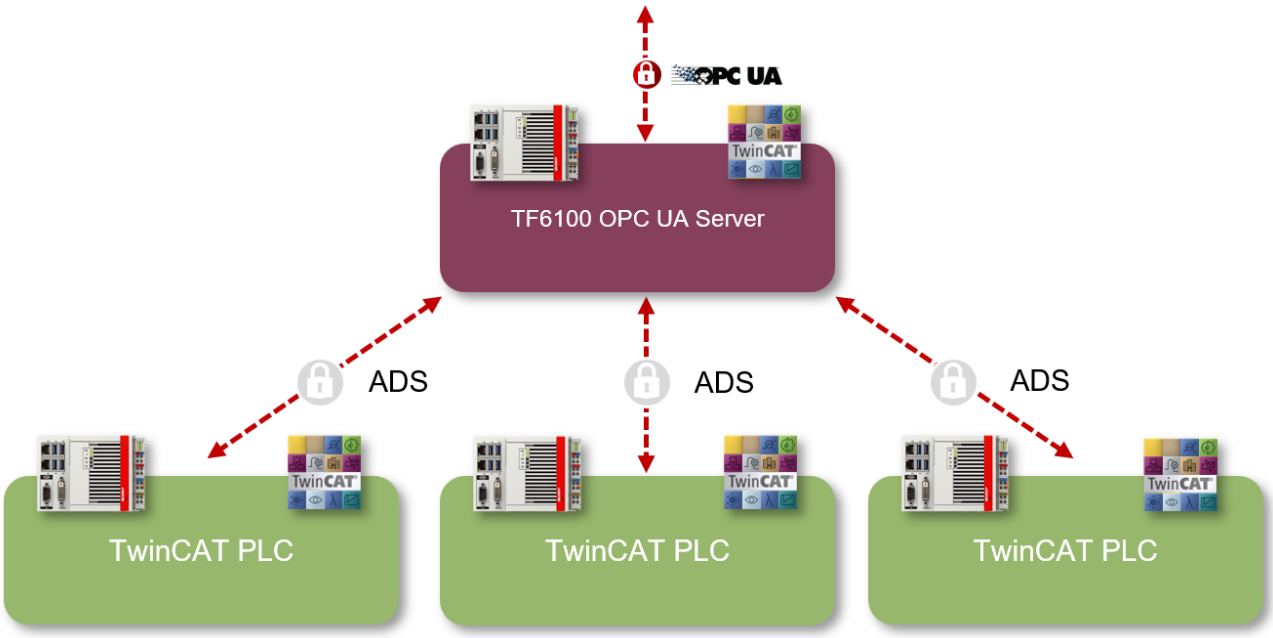
Data Access is an OPC UA function that describes the representation and use of variable values. For example, this is about the service functionalities, how an OPC UA client can access variable values.

The TwinCAT OPC UA Server can provide variables from all TwinCAT real-time environments. These include, for example, the TwinCAT PLC, TwinCAT 3 C++, TwinCAT 3 MATLAB® und Simulink® or a TwinCAT I/O task. The server can access several real-time environments and provide their symbolism in its address space.

Basis for further functions

The Connection with the runtime [▶ 56] and enabling symbols [▶ 59] for Data Access is the basis for other functions, such as Historical Access and Alarms & Conditions.
Please make sure you are familiar with the settings required here.

The different real-time environments do not necessarily have to be on the same system, but can be distributed across different controllers. In this case, an ADS route must be set up for each system. The following figure illustrates this relationship.



In the TwinCAT OPC UA Configurator, the Data Access configuration can be carried out in the **Data Access** tab. All connected real-time environments are displayed there as a separate "device". The chapter Connection with the runtime [▶ 56] describes how to add data access devices and which parameters are required for this. You can then start with Enabling symbols [▶ 59].

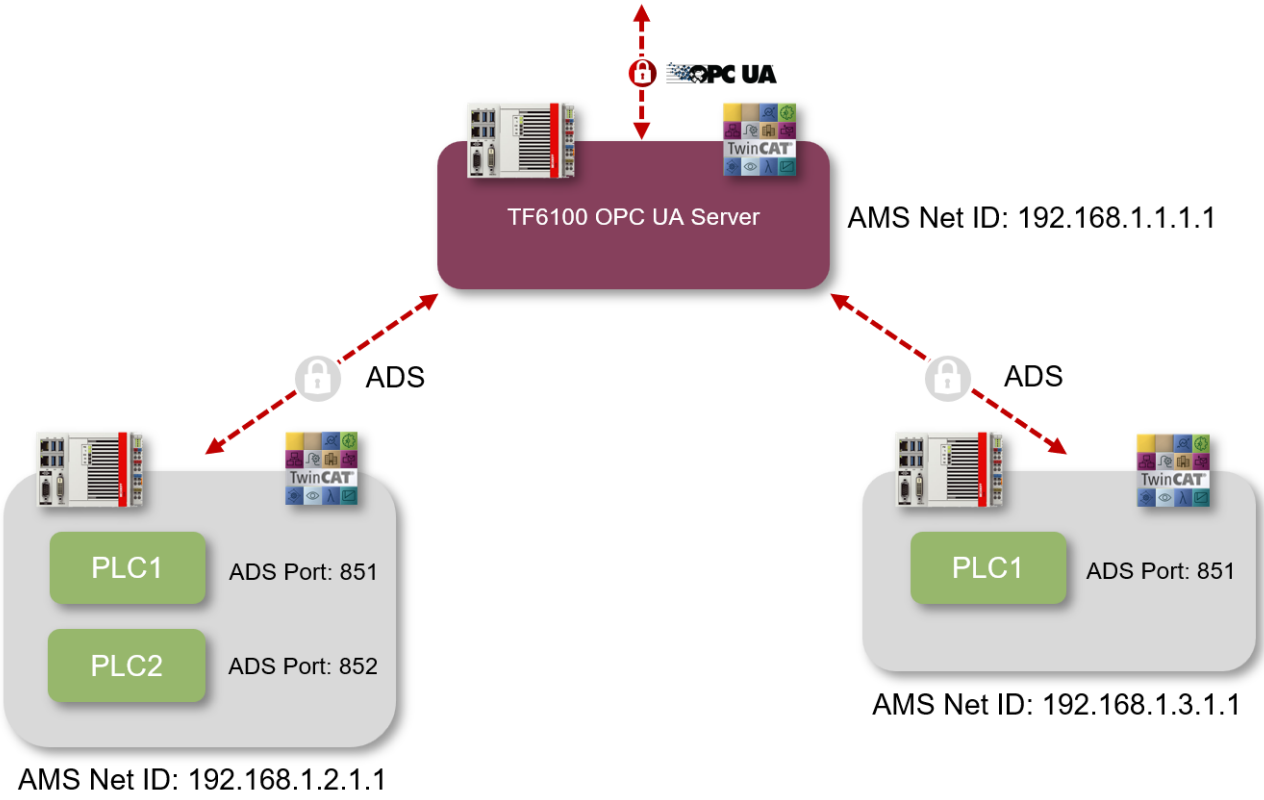
Also see about this

- Overview [▶ 98]

4.11.2 Connection with the runtime

The TwinCAT OPC UA Server can provide symbols from one or more real-time environments. These can also be located on different physical control systems. In this case, an ADS route must be established to the respective target system. The so-called "AMS Net ID" uniquely identifies a control system in the network and

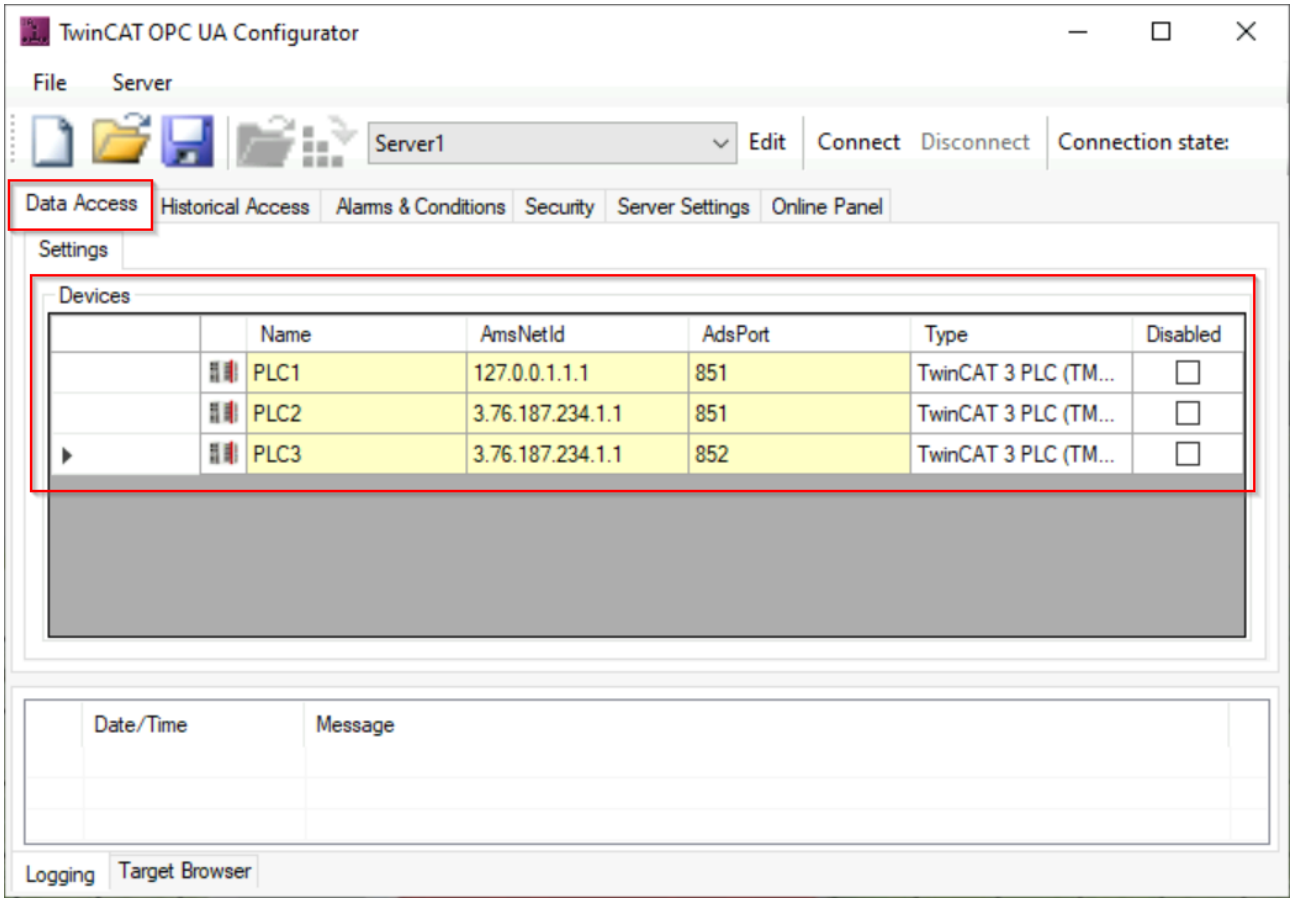
is required to establish an ADS route to the system. The ADS port, on the other hand, then identifies a specific application on this system, e.g. the TwinCAT PLC.
The following figure illustrates this relationship.



In this example, there are three control devices. The TwinCAT OPC UA Server is installed on the first device and this device is identified by the AMS Net ID 192.168.1.1.1.1. On the second device, which is identified by the AMS Net ID 192.168.1.2.1.1, two PLC runtimes are started and are each addressed by their own ADS port. Only one PLC runtime is running on the third device (192.168.1.3.1.1).

Configuration

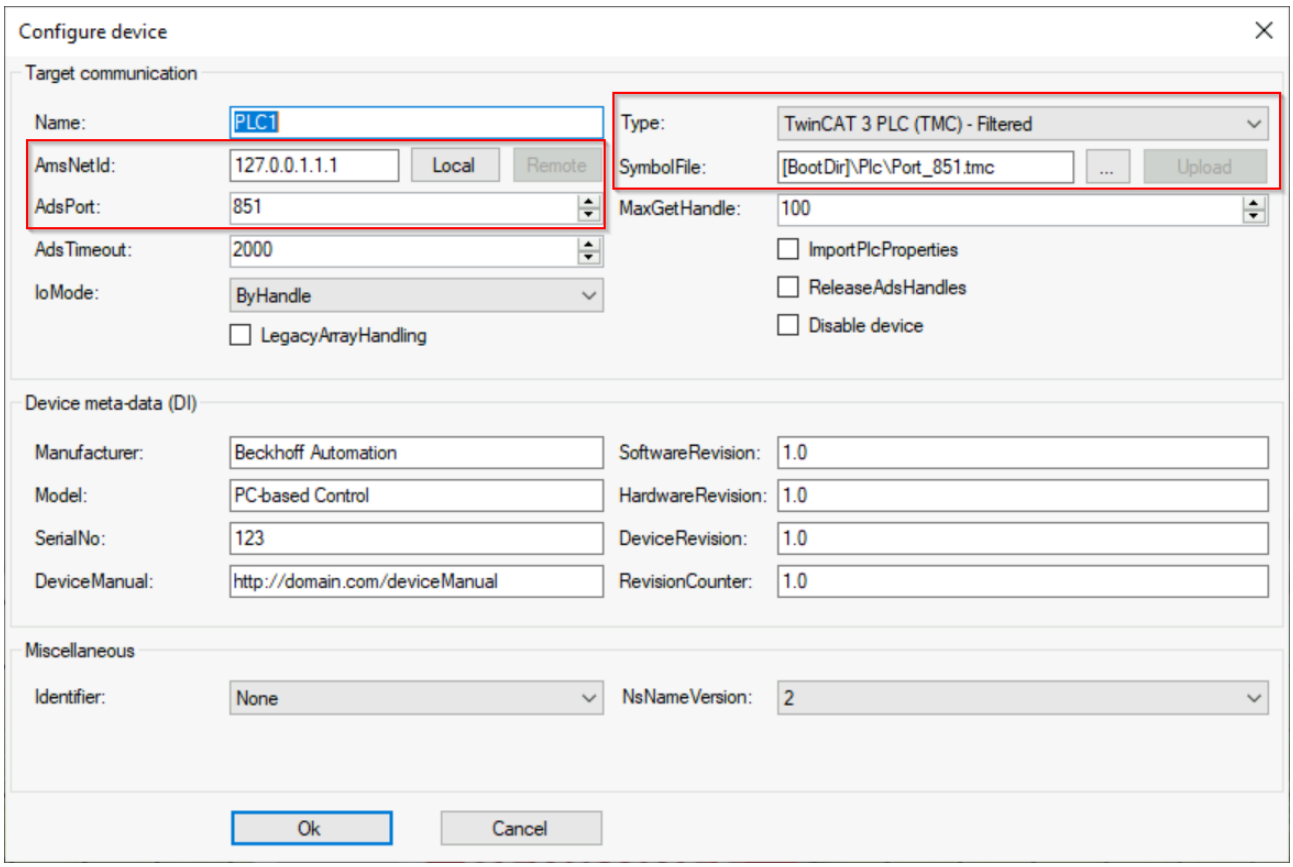
The configuration for a Data Access device can be carried out using the TwinCAT OPC UA Configurator. The **Data Access** tab gives you an overview of all configured devices and allows you to add or remove devices.



In this screenshot, for example, you can see a configuration with three Data Access devices. These have been configured as follows:

- PLC1: Local PLC runtime (127.0.0.1.1.1) on port 851
- PLC2: Remote PLC runtime (3.76.187.234.1.1) on port 851
- PLC3: Remote PLC runtime (3.76.187.234.1.1) on port 852

All parameters required for the connection with a device and its symbolism can now be configured in the properties of the device.



Depending on the real-time environment used, you can, for example, choose a symbol file here, select the ADS route to be used and enter the ADS port of the respective application. You can find more information on the various symbol files in the chapter Enabling symbols [▶ 59].

4.11.3 Enabling symbols

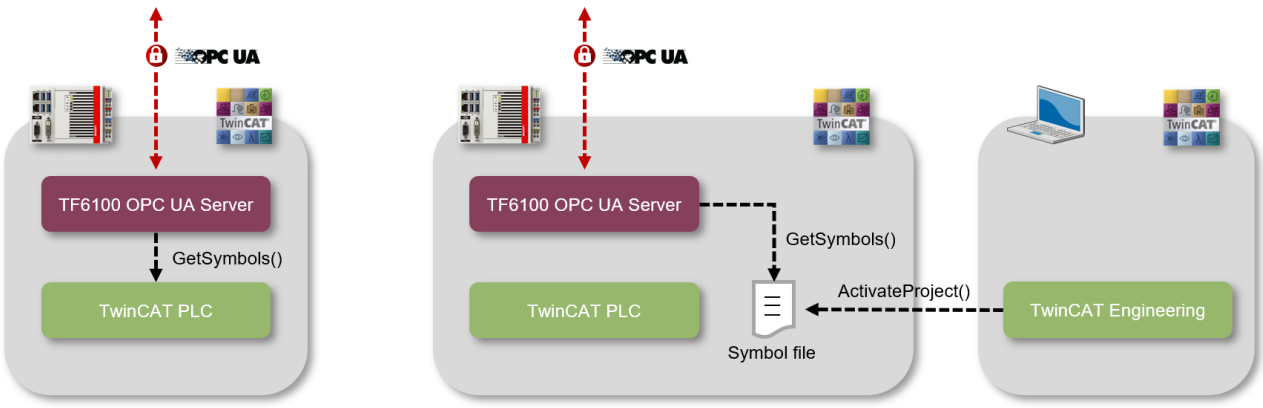
As you have already learned in the Quick Start [▶ 27] tutorial, symbols are enabled in the TwinCAT 3 PLC via a so-called pragma (also known as an "attribute"). In other real-time environments, the enabling mechanism may be different. The following chapter provides an overview.

The term "Symbol"

The term "Symbol" is used in this documentation as a substitute for variable, structure, function block instance, method, etc.

Symbols

The TwinCAT OPC UA Server receives its information via real-time variables (e.g. addresses and data types), the so-called "symbolism". Depending on the real-time environment used (e.g. TwinCAT PLC or TwinCAT 3 C++), various so-called "symbol files" are available for reading the symbolism (also known as "offline symbolism"). In addition, the symbolism can be read via TwinCAT ADS when the runtime environment is started (also referred to as "online symbolism").



Online Symbolic Offline Symbolic

The advantage of using online symbolism is that no symbol files need to be exchanged. Especially in the installation variant [▶ 19], in which the TwinCAT OPC UA Server is operated on a gateway PC, the manual copying process of the symbol files of all connected controllers is no longer necessary. The disadvantage of online symbolism is that the symbols are only available when the PLC environment is running.

The symbol file must be imported by the server to access the address information of the symbols. Alternatively, the server can also work directly with the online symbolism. The symbols are then only available in the server's address space when the real-time environment is available.

The TwinCAT OPC UA Configurator can be used to configure the server and which symbol file it should read. The corresponding path to the symbol file can be configured there as part of the data access device.

Depending on the real-time environment used, the symbol file that must be imported from the server in order to obtain address information for the existing symbols and the symbol enabling mechanism will vary. While the TwinCAT 3 PLC works with so-called pragmas to enable a symbol for OPC UA, TwinCAT 2 still uses specially formatted comments on the symbol for this purpose. TwinCAT 3 C++, on the other hand, uses the so-called TMC code generator to enable symbols for OPC UA.

Symbol files can be configured in the TwinCAT engineering interface so that they are automatically copied to the boot directory of the target device when the project is activated. The TwinCAT OPC UA Server is usually configured to read the symbol file from the boot directory.

The following table provides an overview of the various symbol files in relation to their real-time environment and the enabling mechanism.

Real-time environment	Symbol file	Enable via...	Path in the boot directory
TwinCAT 2 PLC	TPY	Comment	CurrentPlc_1.tpy
TwinCAT 3 PLC [▶ 60]	TMC	Pragma	Plc\Port_%AdsPort%.tmc
TwinCAT 3 C++ [▶ 63]	TMI	TMC code generator	Tmi\%ObjectId%.tmi
TwinCAT 3 MATLAB® und Simulink®	TMI	TMC code generator	Tmi\%ObjectId%.tmi
TwinCAT 3 I/O Task [▶ 73]	XML	Comment	CurrentConfig.xml
Online symbolism [▶ 75]	---	Pragma TMC code generator	---

For more information, see the sub-chapters for each real-time environment.

Also see about this

- MATLAB® und Simulink® [▶ 66]

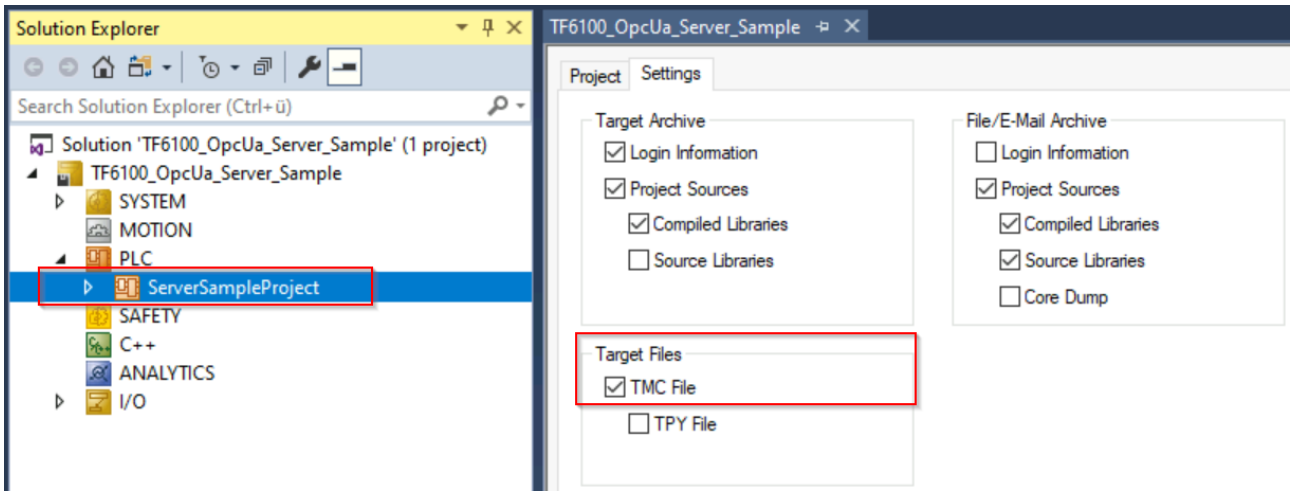
4.11.3.1 PLC

This section describes how to configure the namespace of the TwinCAT OPC UA Server in order to gain access to the symbols of a PLC project. This requires the following steps to be carried out:

- Activating the symbol file
- Enabling symbols
- Configuration of the server

Activating the symbol file

To make the symbols of a PLC project available via OPC UA, you must first activate the download of the symbol file (TMC) in the properties of the PLC project.



This means that the symbol file is automatically transferred to the corresponding target device when the project is activated and is then available there in the TwinCAT boot directory. The name of the symbol file is based, for example, on the ADS port (see below) of the PLC runtime:

```
%TwinCATInstallDir%\3.1\Boot\Plc\Port_851.tmc
```

Quick start

If the TwinCAT OPC UA Server is also installed on the same device, it always automatically reads the symbol file of the first PLC runtime in the delivery state, i.e. no further settings are required on the server. Activating the project thus transfers the symbol file to the target device and a subsequent TwinCAT restart also restarts the TwinCAT OPC UA Server. This will now find the symbol file in the boot directory during the boot process and read it. The server recognizes which symbols it should provide in its address space through the set pragmas.

Enabling symbols

In TwinCAT 3 PLC, symbols are enabled for OPC UA via so-called pragmas. Such a pragma was also used in the Quick Start [▶ 27] Tutorial, for example, to enable a variable for OPC UA. The pragma is inserted before the symbol to which it applies. The following pragma can be used to enable a variable, array, or structure for OPC UA. The TwinCAT OPC UA Server recognizes this pragma when reading the symbolism and imports the corresponding symbol into its namespace. The corresponding type information [▶ 77] is transferred and mapped to OPC UA.

```
{attribute 'OPC.UA.DA' := '1'}
nMyCounter : INT;
```

Additional pragmas may be required for individual sub-functions of data access, for example structures [▶ 82], properties [▶ 85], AnalogItemType [▶ 89], StatusCode [▶ 86], setting a description [▶ 90], Alias or the read-only [▶ 91] flag of a node. The additional pragmas are then inserted in a separate line, for example:

```
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
nMyCounter : INT;
```

(Sets the read-only flag for this variable)

The pragma for enabling a symbol is automatically inherited to all child symbols. If you want to block inheritance from a certain point, for example from a certain member of a structure, you can work with the following pragma:

```
{attribute 'OPC.UA.DA' := '0'}
stChild : ST_ChildStruct;
```

In the case of function blocks and structures, the definition location of the pragma is decisive. If the pragma is defined on an instance, only this instance (and all child elements) is enabled for OPC UA. If, on the other hand, the pragma is defined at the function block or structure definition, all instances of the function block or structure are enabled. In the following sample, the two instances fbTest1 and fbTest2 of the function block FB_BLOCK1 are to be made available via OPC UA. When an entire instance is enabled, all its symbols are also available via OPC UA. The PLC program looks like the following:

```
PROGRAM MAIN
VAR
  {attribute 'OPC.UA.DA' := '1'}
  fbTest1 : FB_BLOCK1;
  fbTest2 : FB_BLOCK1;
END_VAR
```

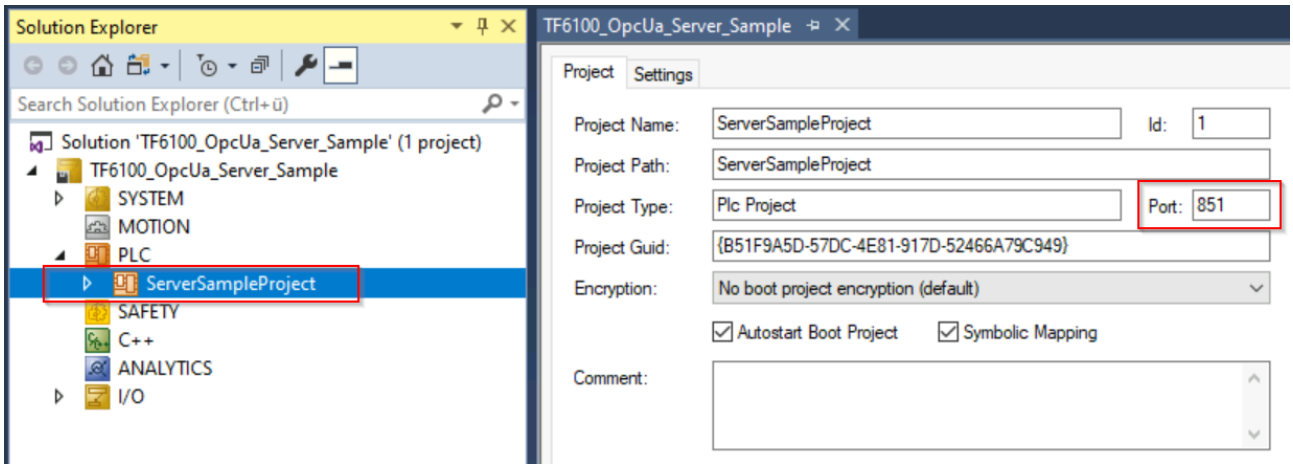
```
FUNCTION_BLOCK FB_BLOCK1
VAR_INPUT
  {attribute 'OPC.UA.DA' := '1'}
  ni1 : INT;
  ni2 : INT;
END_VAR
VAR_OUTPUT
  {attribute 'OPC.UA.DA' := '1'}
  no1 : INT;
  no2 : INT;
END_VAR
VAR
  {attribute 'OPC.UA.DA' := '1'}
  nx1 : INT;
  nx2 : INT;
END_VAR
```

The instance fbTest1 now receives the pragma, which means that all the symbols it contains are also automatically enabled for OPC UA, i.e. fbTest.ni1, fbTest.ni2, etc. The instance fbTest2, on the other hand, does not receive the pragma, but the three variables ni1, no1 and nx1 it contains are marked with the pragma. They are therefore available in all instances via OPC UA.

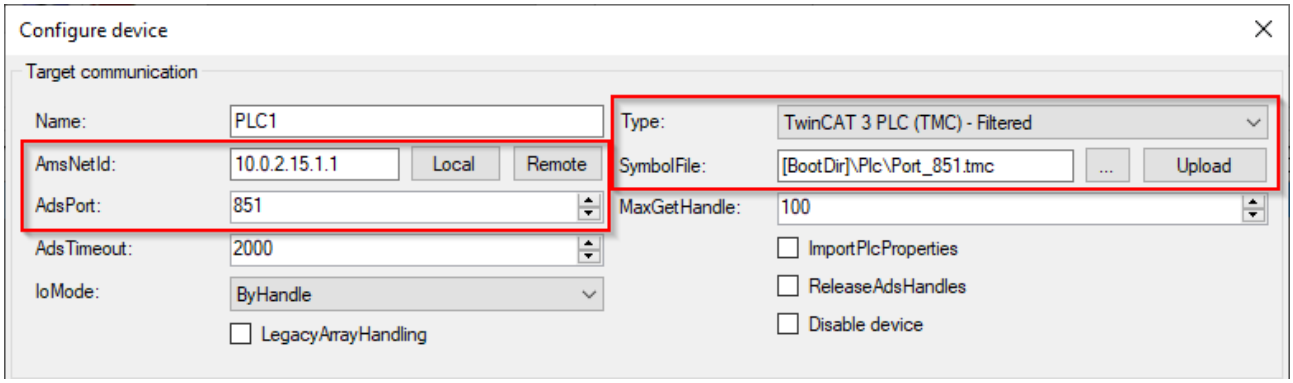
Configuration of the server

You can now use the TwinCAT OPC UA Configurator to configure the TwinCAT OPC UA Server so that it reads the generated symbol file and makes the PLC runtime available as a data access device in its address space.

To do this, add a new device in the Data Access tab of the TwinCAT OPC UA Configurator. Select the AMS Net ID of your device and enter the ADS port of the PLC runtime. You can find the ADS port in the properties of the PLC project:



Now select the entry "TwinCAT 3 PLC (TMC) - Filtered" from the selection list in the "Type" field. In the "SymbolFile" field, select the symbol file (TMC) that was created after activating the TwinCAT PLC project and should now be located in the boot directory. Alternatively, you can also use the online symbolism [▶ 75].



You can leave all other settings at their default values.

4.11.3.2 C++

This section describes how to configure the namespace of the TwinCAT OPC UA Server in order to obtain access to the symbols of a TwinCAT 3 C++ module. This requires the following steps to be carried out:

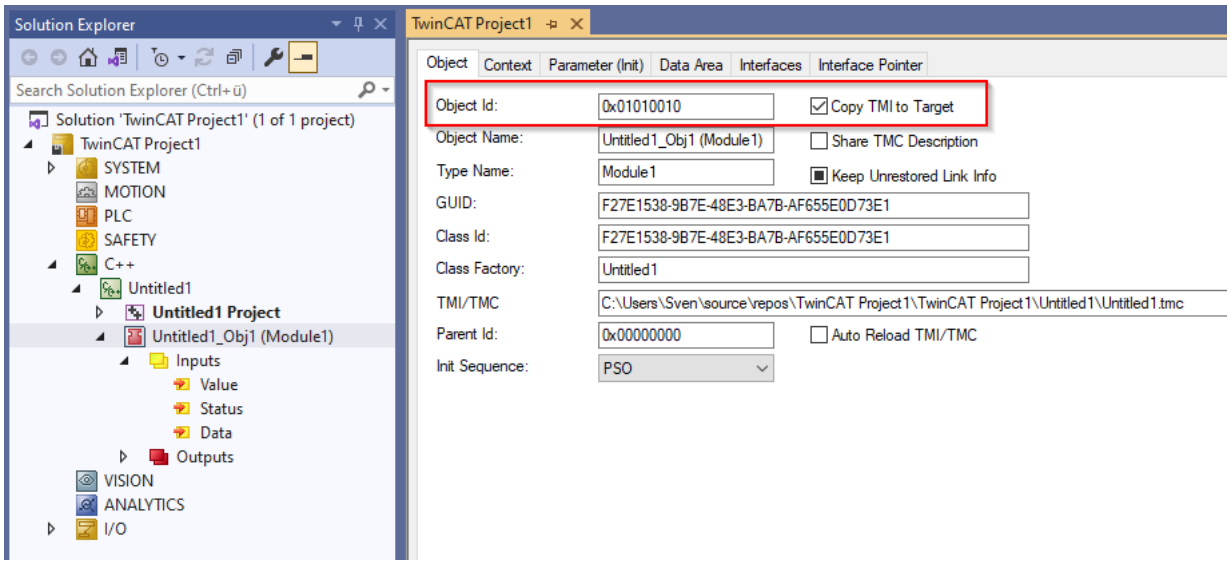
- Activating the symbol file
- Enabling symbols
- Configuration of the server

These steps are described in more detail below.

Activating the symbol file

To configure certain symbols contained in an instance of a C++ module so that they become accessible via OPC UA, you must first activate the download of the symbol file for the module instance. This copies the symbol file to the boot directory of the target device when the configuration is activated.

Activate the download of the symbol file by setting the following option in the properties of the module instance:

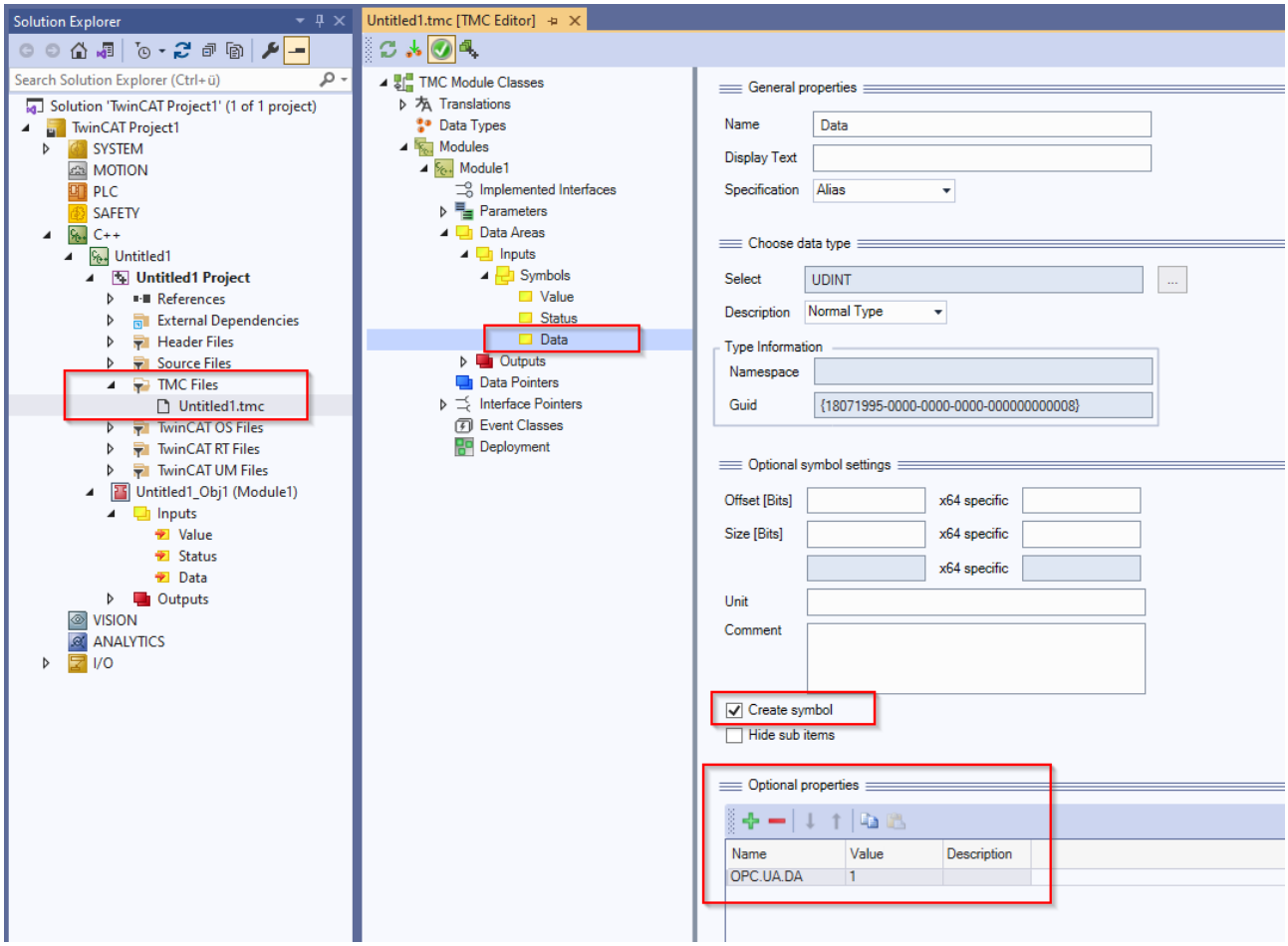


The generated symbol file (TMI) is always named after the ObjectId of the module instance and is now copied to the boot directory of the target device when activated, for example:

```
%TwinCATInstallDir%\3.1\Boot\Tmi\Obj_01010010.tmi
```

Enabling symbols

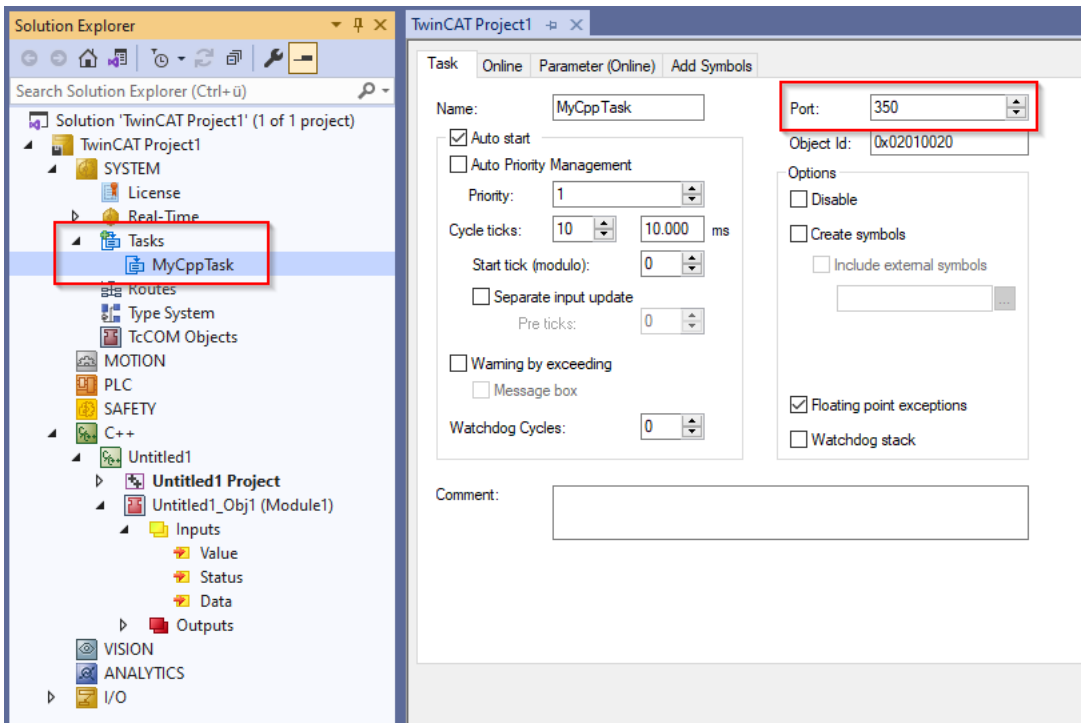
You can use the TMC Code Generator to define which symbols are to be enabled via OPC UA. Two settings are required for this. First activate the "Create Symbol" checkbox for each symbol to be enabled in the TMC Code Generator. Then add an optional property with the key "OPC.UA.DA" and the value "1".



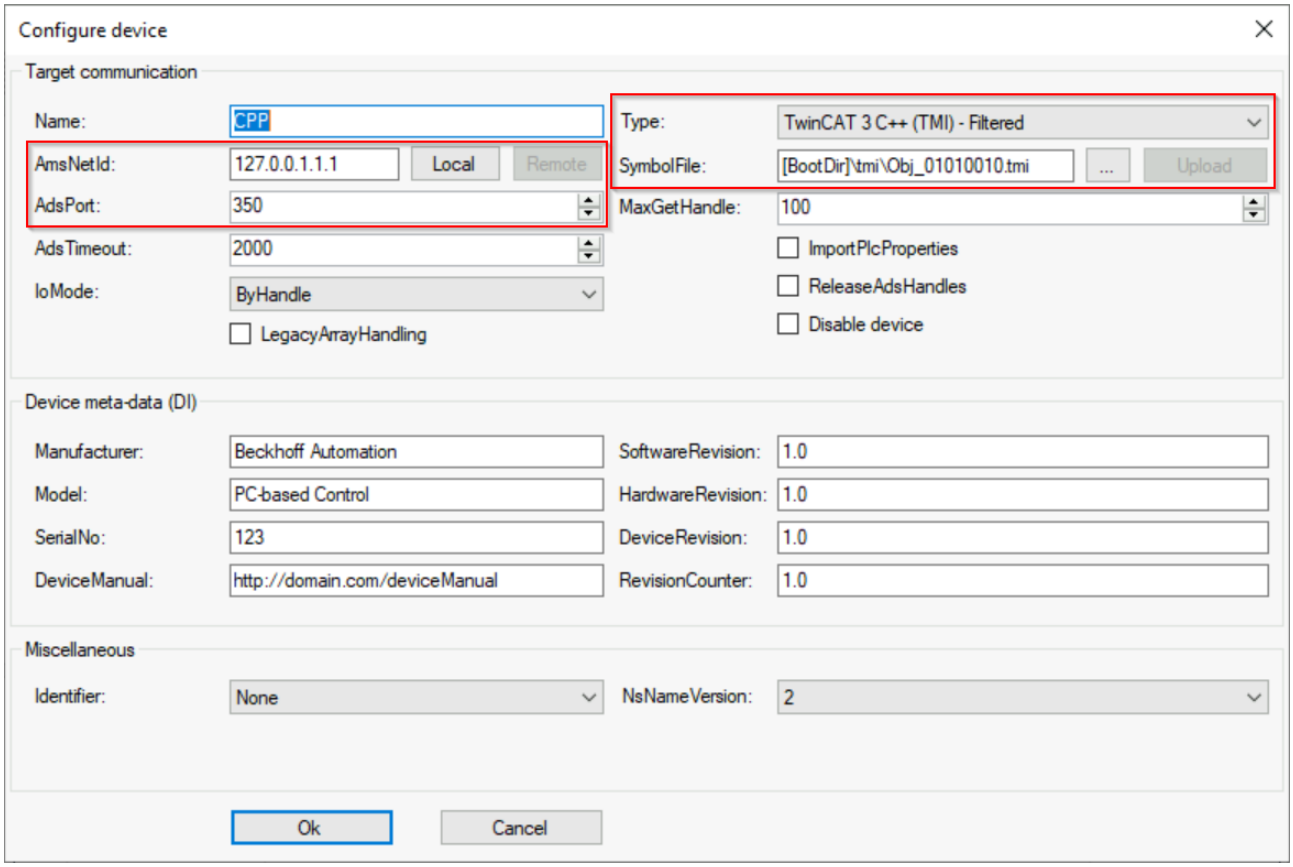
Configuration of the server

You can now use the TwinCAT OPC UA Configurator to configure the TwinCAT OPC UA Server so that it reads the generated symbol file and makes the C++ module instance available as a data access device in its address space.

To do this, add a new device in the Data Access tab of the TwinCAT OPC UA Configurator. Select the AMS Net ID of your device and enter the ADS port of the TwinCAT 3 C++ module instance. You can find the ADS port in the properties of the task connected to the C++ module instance:



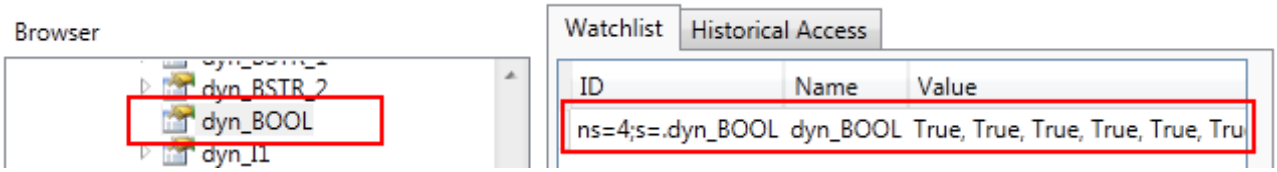
Now select the entry "TwinCAT 3 C++ (TMI) - Filtered" from the selection list in the "Type" field. In the "SymbolFile" field, select the symbol file (TMI) that was created after activating the TwinCAT 3 C++ project and should now be located in the boot directory. Alternatively, you can also use the online symbolism [▶ 75].



You can leave all other settings at their default values.

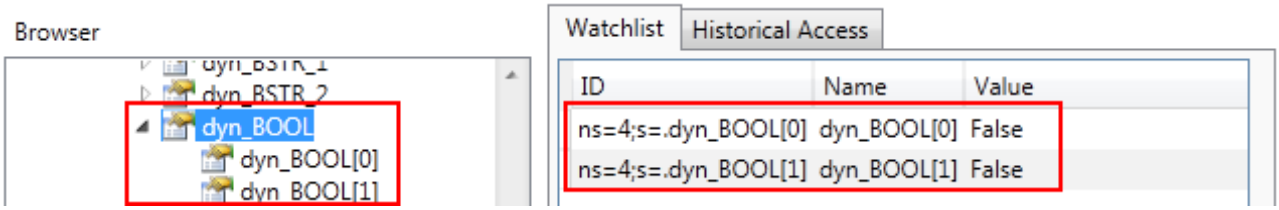
4.11.3.2.1 Arrays

By default arrays are regarded as individual nodes in the UA namespace. This means that if you define, for example, an array dyn_BOOL[10] in the PLC (and have also enabled it for OPC UA), it will subsequently appear in the UA namespace as follows:



The advantage of this approach is a considerable reduction in the complexity of the UA namespace and in memory consumption, since not every position of an array needs to be made available as an individual node in the namespace. However, modern UA Clients can continue to access the individual array positions via the so-called "RangeOffset".

In order to support older UA Clients that don't offer this feature, however, you can also make the positions of an array available as individual nodes in the UA namespace. It is illustrated as follows:



This setting is available by activating the **Legacy Array Handling** option in the UA Configurator within the respective namespace configuration.

Depending on the scope of the PLC project, the UA namespace can become significantly more complex, which in turn is reflected in an increased memory utilization of the UA Server.

Changes to the settings listed above only become active after restarting the UA Server.

4.11.3.3 MATLAB® und Simulink®

This section describes how to configure the namespace of the TwinCAT OPC UA Server in order to obtain access to the symbols of a TwinCAT 3 MATLAB® und Simulink® module. This requires the following steps to be carried out:

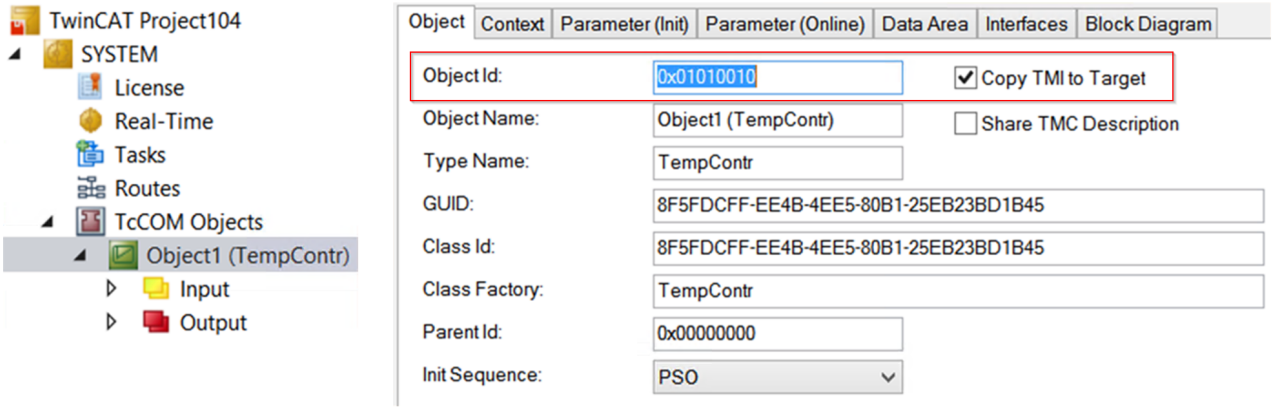
- Activating the symbol file
- Enabling symbols
- Configuration of the server

These steps are described in more detail below.

Activating the symbol file

To configure certain symbols contained in an instance of a C++ module so that they become accessible via OPC UA, you must first activate the download of the symbol file for the module instance. This copies the symbol file to the boot directory of the target device when the configuration is activated.

Activate the download of the symbol file by setting the following option in the properties of the module instance:



The generated symbol file (TMI) is always named after the ObjectId of the module instance and is now copied to the boot directory of the target device when activated, for example:

```
%TwinCATInstallDir%\3.1\Boot\Tmi\Obj_01010010.tmi
```

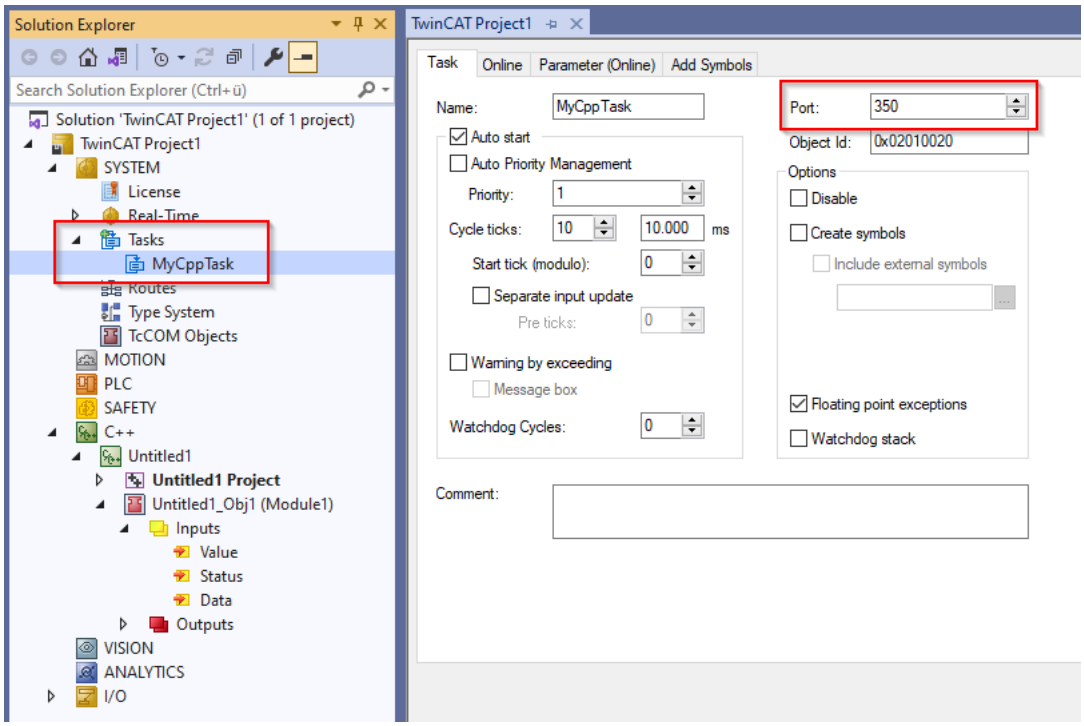
Enabling symbols

The TE1400 Target for Simulink® product offers various setting options for selecting variables that are to be enabled via OPC UA. These settings are described in the corresponding TE1400 product documentation.

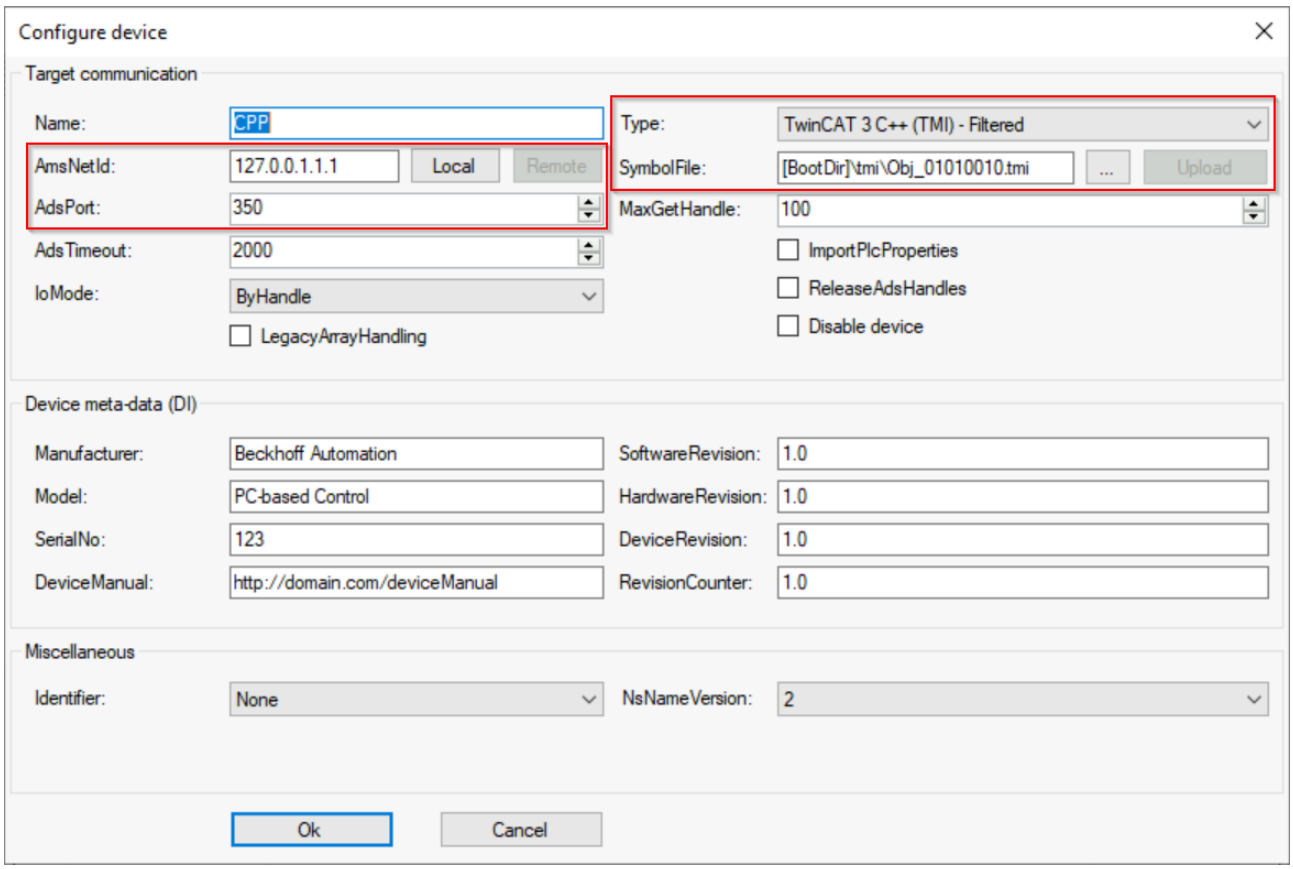
Configuration of the server

You can now use the TwinCAT OPC UA Configurator to configure the TwinCAT OPC UA Server so that it reads the generated symbol file and makes the MATLAB® und Simulink® module instance available as a data access device in its address space.

To do this, add a new device in the Data Access tab of the TwinCAT OPC UA Configurator. Select the AMS Net ID of your device and enter the ADS port of the TwinCAT 3 MATLAB® und Simulink® module instance. You can find the ADS port in the properties of the task connected to the module instance:



Now select the entry "TwinCAT 3 C++ (TMI) - Filtered" from the selection list in the "Type" field. In the "SymbolFile" field, select the symbol file (TMI) that was created after activating the TwinCAT 3 C++ project and should now be located in the boot directory. Alternatively, you can also use the online symbolism [▶ 75].



You can leave all other settings at their default values.

4.11.3.4 EtherCAT Master

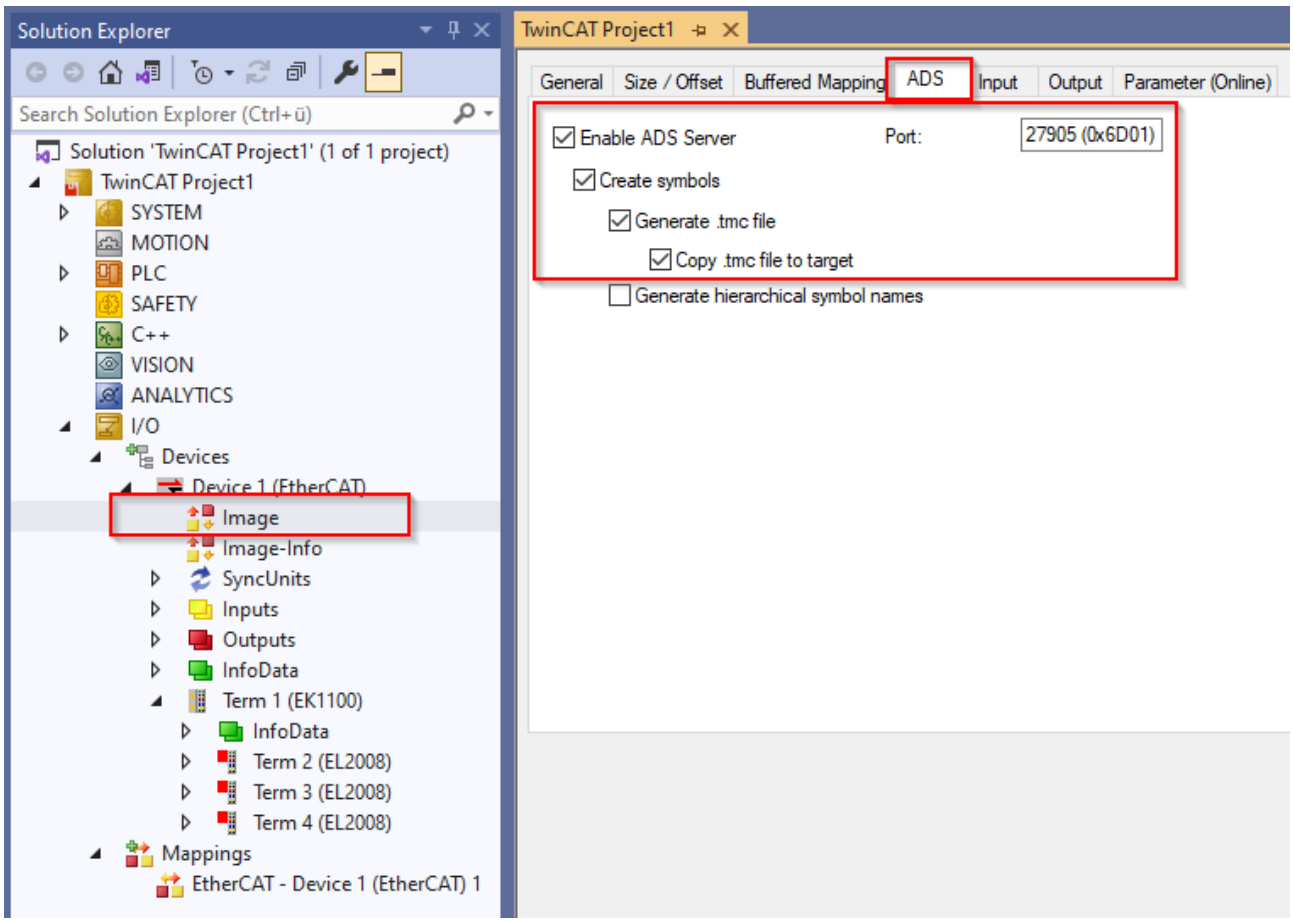
This section describes how to configure the namespace of the TwinCAT OPC UA Server in order to obtain access to the symbols of an EtherCAT master. This requires the following steps to be carried out:

- Activating the symbol file
- Configuration of the server
- [optional] Change the namespace structure

These steps are described in more detail below.

Activating the symbol file

To make the symbols of an EtherCAT master available via OPC UA, you must first activate the ADS server in the image of the EtherCAT master and the generation of the symbols. This enables access to the symbols via ADS. In the same dialog, you will also find the ADS port, which will be required later in the configuration process.



By activating the "Generate .tmc file" and "Copy .tmc file to target" parameters, a symbol file is generated and automatically downloaded to the target device when the configuration is activated. This is then located in the TwinCAT boot directory and has the ADS port in the file name for easy identification, for example:

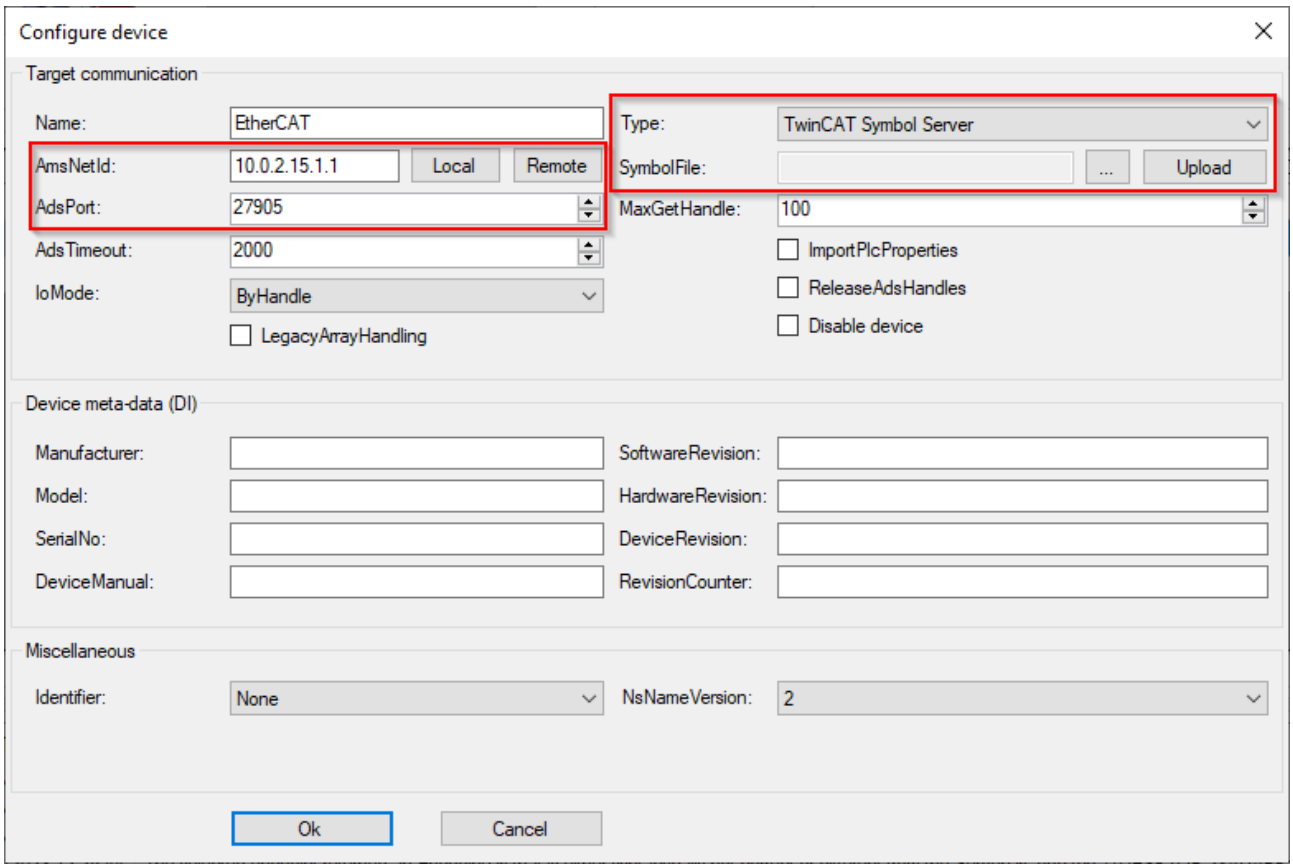
```
%TwinCATInstallDir%\3.1\Boot\Io\Port_27905.tmc
```

Configuration of the server

You can now use the TwinCAT OPC UA Configurator to configure the TwinCAT OPC UA Server so that it reads the generated symbol file and makes the EtherCAT master available as a data access device in its address space.

Add a new device in the Data Access tab of the TwinCAT OPC UA Configurator. Select the AMS Net ID of your device and enter the ADS port of the EtherCAT master image (see above).

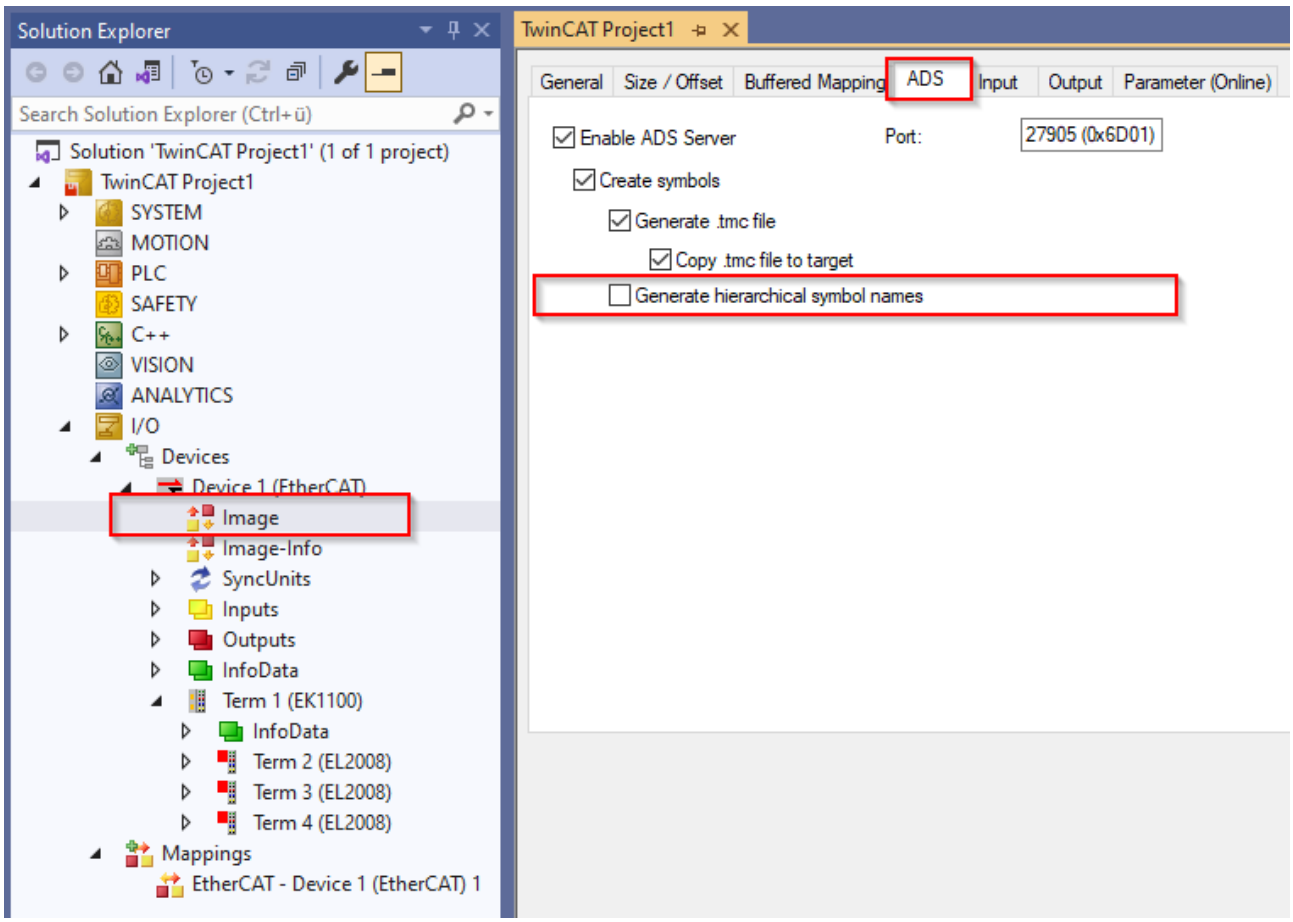
Now select the "EtherCAT Master (TMC)" entry from the selection list in the "Type" field. In the "SymbolFile" field, select the symbol file (TMC) that was created after activating the TwinCAT project and should now be located in the boot directory. Alternatively, you can also use the online symbolism [▶ 75].



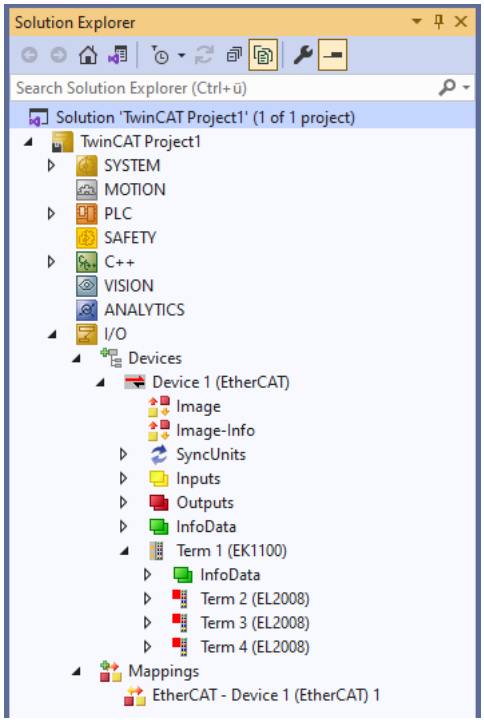
You can leave all other settings at their default values.

Changing the namespace structure

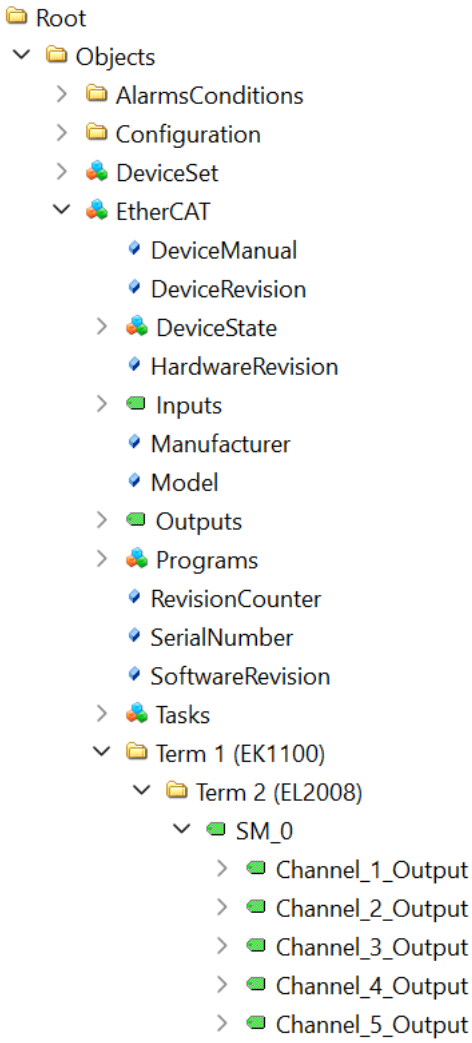
The parameter "Generate hierarchical symbol names" enables a different structure of the symbol space, which also has a direct effect on the namespace of the TwinCAT OPC UA Server.



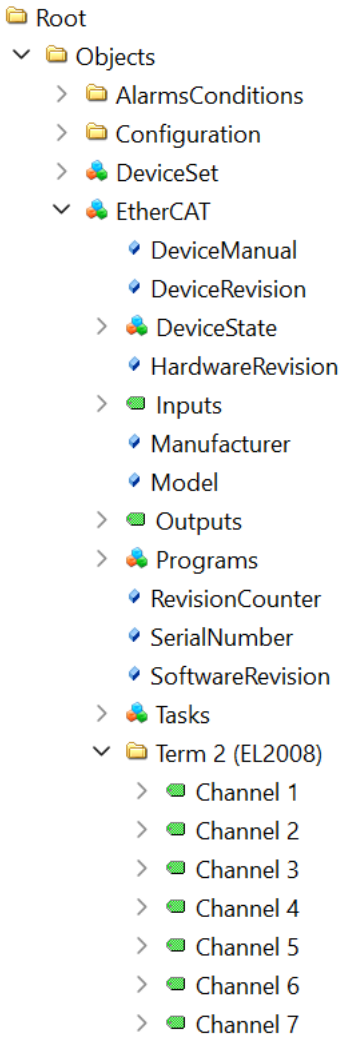
Let's take the following EtherCAT master configuration in TwinCAT XAE as an example:



If this option is activated, the namespace reflects the structure in TwinCAT XAE:



If this option is disabled, the namespace is structured "flatter":



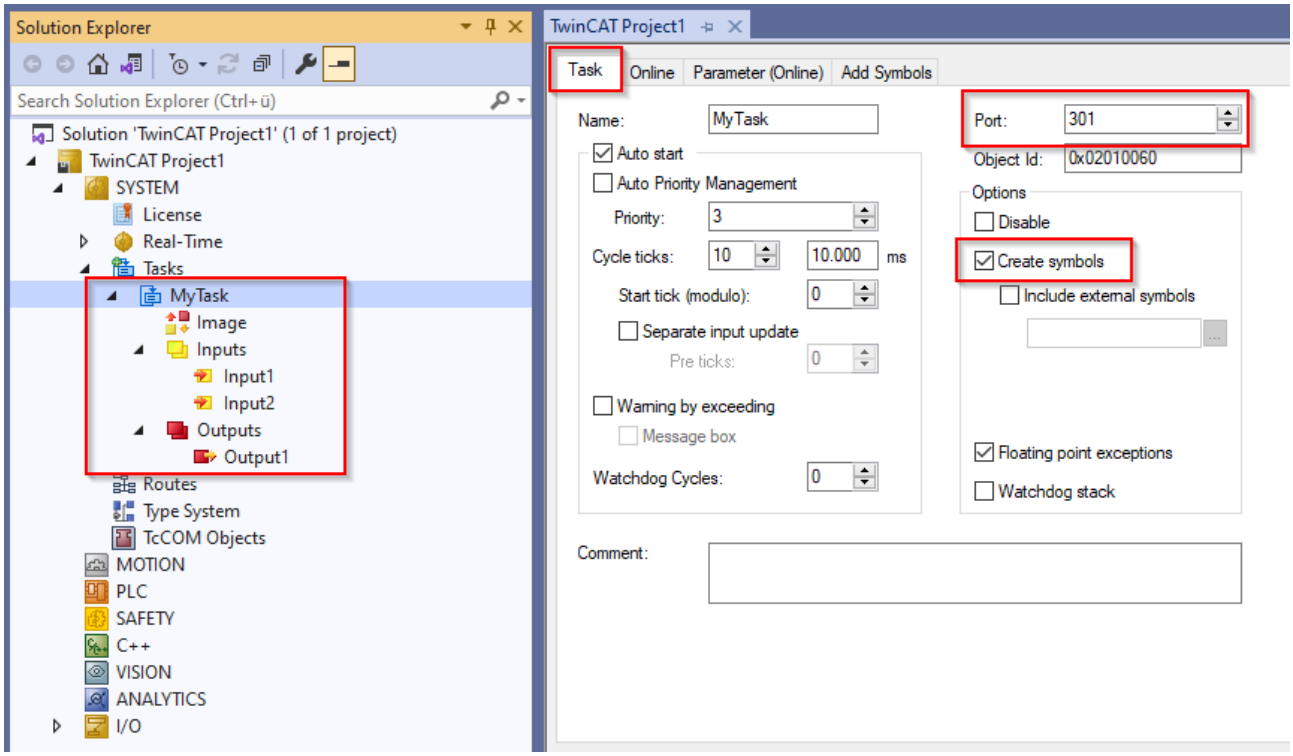
4.11.3.5 I/O task

This section describes how to configure the namespace of the TwinCAT OPC UA Server in order to gain access to the symbols of a task with process image. This requires the following steps to be carried out:

- Activating the symbolism
- Configuration of the server

Activating the symbolism

You can provide the variables of a task with a process image via OPC UA. To do this, you must enable the "Create symbols" option in the task settings. This generates symbols and makes them available via the online symbolism [▶ 75]. In the same dialog, you will also find the ADS port, which will be required later in the configuration process.

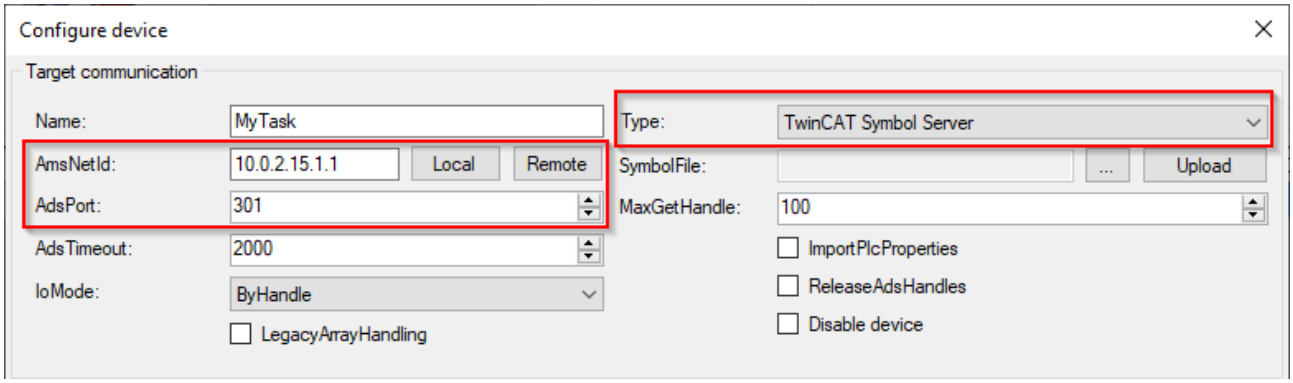


Configuration of the server

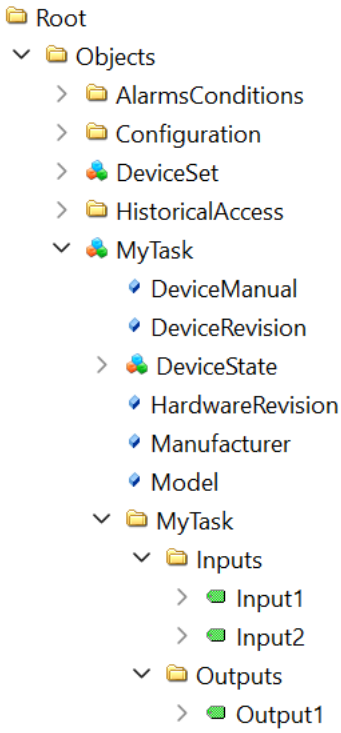
You can now use the TwinCAT OPC UA Configurator to configure the TwinCAT OPC UA Server so that it accesses the online symbolism of the task and makes them available in its address space.

Add a new device in the Data Access tab of the TwinCAT OPC UA Configurator. Select the AMS Net ID of your device and enter the ADS port of the task (see above).

In the selection list of the "Type" field, select the "TwinCAT Symbol Server" entry.



After activating the TwinCAT project, the symbols are ready and the TwinCAT OPC UA Server can read them and build up its address space accordingly.



4.11.3.6 Online symbolism

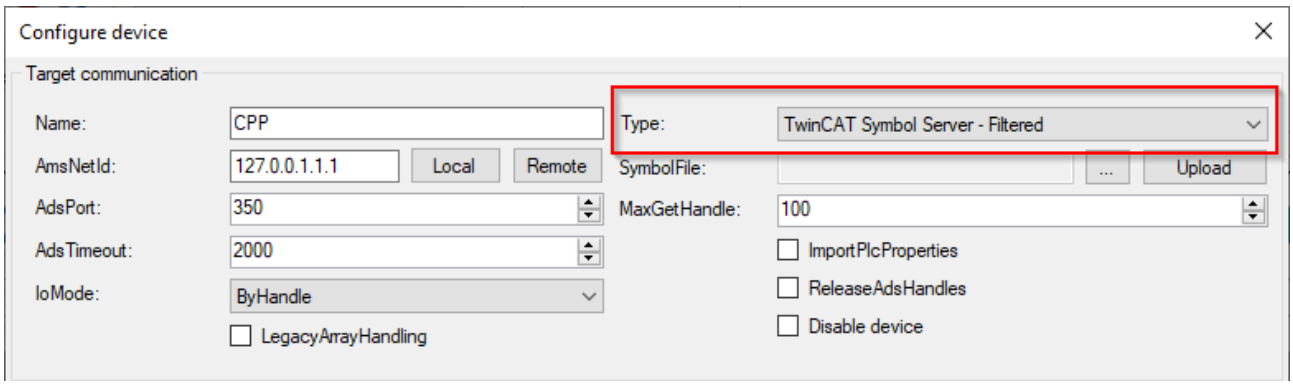
The TwinCAT OPC UA Server enables symbol information to be read via the so-called "online symbolism". This is a standardized ADS interface, which is offered by some ADS servers in TwinCAT. This interface is often referred to as "Symbol Upload". The interface enables an ADS client to read all available symbols from the ADS server and process them further if necessary. In addition to the address information of the symbols, further information is also provided, such as data type descriptions and optional attributes.

The optional attributes can be used to explicitly enable symbols or to make further settings for this purpose - the handling is identical to the use of symbol files and runs, for example, via pragmas in the PLC or the TMC Code Generator for TwinCAT 3 C++. This means that you can use all the enables that you have set there both via the symbol file import and the online symbols. Please note that not all real-time environments allow the explicit enabling of symbols. In this case, all symbols are always provided. The following table provides an overview of some ADS servers that provide the online symbolism interface and whether explicit symbol enabling is available.

Type	ADS Port	Symbol enable
TwinCAT 2 PLC	801, 811, 821, ...	Yes
TwinCAT 3 PLC	851, 852, 853, ...	Yes
TwinCAT 3 C++ / MATLAB® und Simulink®	350, 351, 352, ...	Yes
TwinCAT 3 EtherCAT Master	27905, 27906, ...	No (all symbols are enabled)
TwinCAT 3 I/O Task with Image	301, 302, 303, ...	No (all symbols are enabled)

In the TwinCAT OPC UA Configurator, the online symbol mechanism is available in two forms when configuring a Data Access device:

- TwinCAT Symbol Server: All symbols are imported
- TwinCAT Symbol Server - Filtered: Only enabled symbols are imported



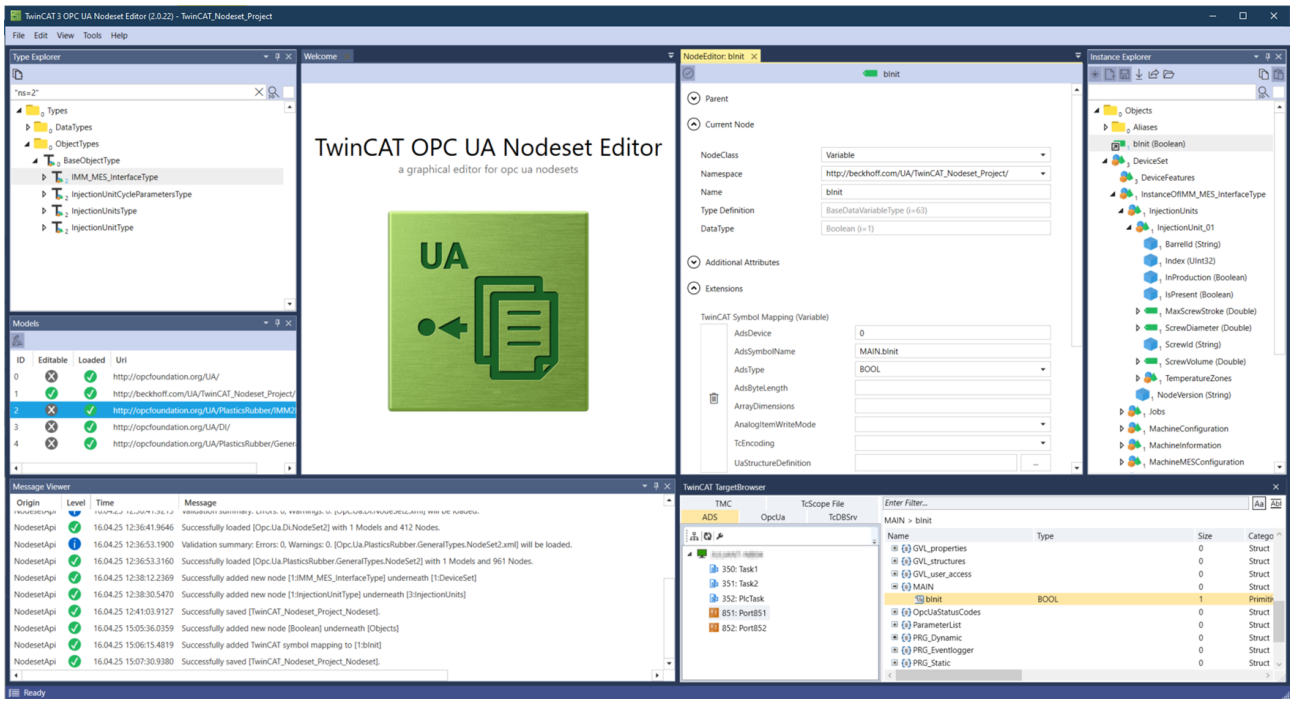
In the structure of its namespace, the TwinCAT OPC UA Server is based on the namespace of the ADS symbolism. As the data is available here, it is also provided via OPC UA.

4.11.4 Nodesets

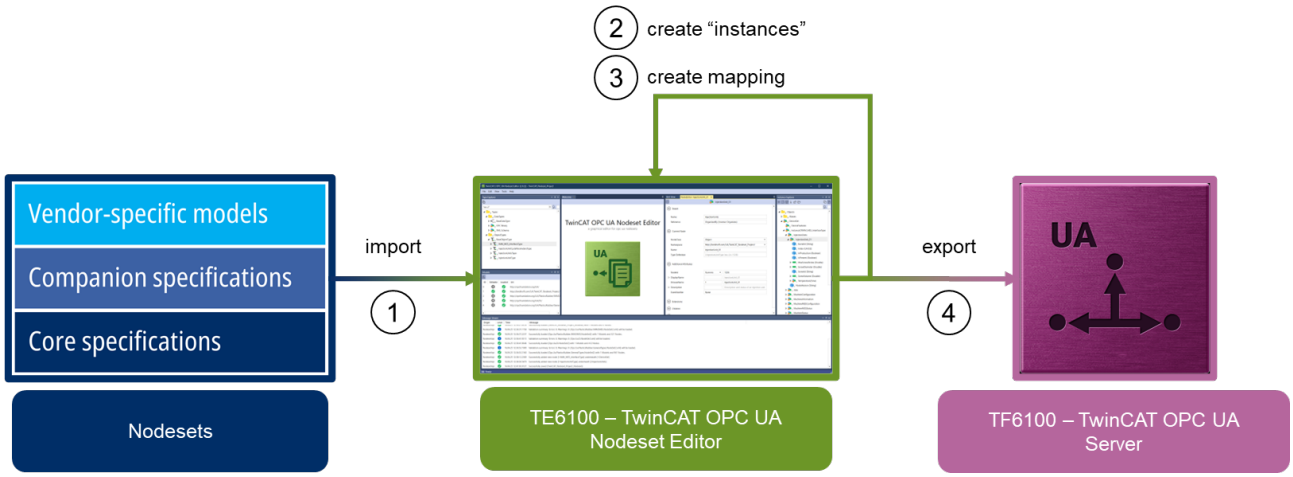
The TwinCAT OPC UA Server allows importing of OPC UA Nodeset files. A nodeset can define data types and instances, for example as part of a companion specification, which are available within the nodeset in an XML structure (standardized by the OPC Foundation).

Using the TE6100 OPC UA Nodeset Editor software, nodeset files can be created, modified and linked with symbols from a TwinCAT real-time environment. The nodeset can then be exported to the TwinCAT OPC UA Server and used there.

Further information on this topic can be found in the TE6100 OPC UA Nodeset Editor product documentation



The following screenshot shows an example of the workflow for using both TwinCAT functions.



4.11.5 Data types

The IEC 61131-3 data types are mapped to OPC UA data types in accordance with the PLCopen mapping specification. The following table shows the specified mapping.

PLC	OPC UA
BOOL	Boolean
SINT	SByte
INT	Int16
DINT	Int32
LINT	Int64
USINT	Byte
UINT	UInt16
UDINT	UInt32
ULINT	UInt64
REAL	Float
LREAL	Double
TIME [▶ 77]	Int64
LTIME [▶ 77]	Int64
STRING [▶ 78]	String
WSTRING [▶ 78]	String

4.11.5.1 TIME/LTIME

TIME/LTIME

For the TIME and LTIME data types, there are special features to observe with regard to mapping between PLC and OPC UA data types, see the following section.

TIME and LTIME

In the PLCopen mapping of IEC 61131-3, both TIME and LTIME are mapped to the Int64 data type. The data areas of the IEC61131-3 data types UDINT and ULINT, on the other hand, correspond to UInt32 and UInt64.

Data type	Lower limit	Upper limit	Mapping in time
TIME (UDINT)	0	4294967295	49d17h2m47s295ms
LTIME (ULINT)	0	18446744073709551615	213503d23h34m33s709ms551us615ns
Int64	-9223372036854775808	9223372036854775807	-

The behavior of the server when handling these two data types is explained in the following list:

TIME

- The possible value range can be seen in the table.
- For all other values (<0 and >4294967295), the server returns the status code **BadOutOfRange** when written by an OPC UA client.

LTIME

- The possible value range can also be seen in the table.
- For all values <0, the server returns the status code **BadOutOfRange** when written by an OPC UA client.
- Values >9223372036854775807 cannot be mapped with the value range of Int64 and therefore cannot be written via OPC UA. However, they can be written in the PLC. In this case, the value of the affected OPC UA variables in the server becomes **null**. In addition, the StatusCode of the variable is set to **BadOutOfRange**.

4.11.5.2 Strings

According to the PLCopen mapping, the PLC data types STRING and WSTRING are both mapped to the OPC UA data types String. The difference lies in the internal storage:

- STRING: Characters are 1 byte long.
- WSTRING: Characters are 2 bytes long.

Both data types are explained below. For more details, please refer to the TwinCAT 3 PLC documentation.

STRING

The STRING data type can be interpreted in TwinCAT as either Latin-1 or UTF-8.
The interpretation is controlled via the {attribute 'TcEncoding'} attribute:

- Latin-1 is used without an attribute.
- UTF-8 is used with attribute.

Examples:

```
// Standard: Latin-1
sMyStringText   : STRING := 'This is a STRING';

// UTF-8 from STRING
{attribute 'TcEncoding' := 'UTF-8'}
sMyUTF8Text1    : STRING  := sLiteral_TO_UTF8('The dinner costs 30 €.');

// UFT-8 from WSTRING
{attribute 'TcEncoding' := 'UTF-8'}
sMyUTF8Text2    : STRING  := wsLiteral_TO_UTF8("The dinner costs 30 €.");

// UFT-8 simple STRING
{attribute 'TcEncoding' := 'UTF-8'}
sMyUTF8Text3    : STRING  := 'Hello World.';
```

▾ Value	
SourceTimestamp	1/21/2026 1:25:24.973 AM
SourcePicoseconds	0
ServerTimestamp	1/21/2026 1:25:24.973 AM
ServerPicoseconds	0
StatusCode	Good (0x00000000)
Value	The dinner costs 30 €.
▾ DataType	STRING
NamespaceIndex	3
IdentifierType	Numeric
Identifier	3013

WSTRING

The WSTRING data type follows the UCS2 coding (2 bytes per character). Initialization always takes place with double quotation marks.

Sample:

```
wsMyWSTRING : WSTRING := "This is a WSTRING.";
```

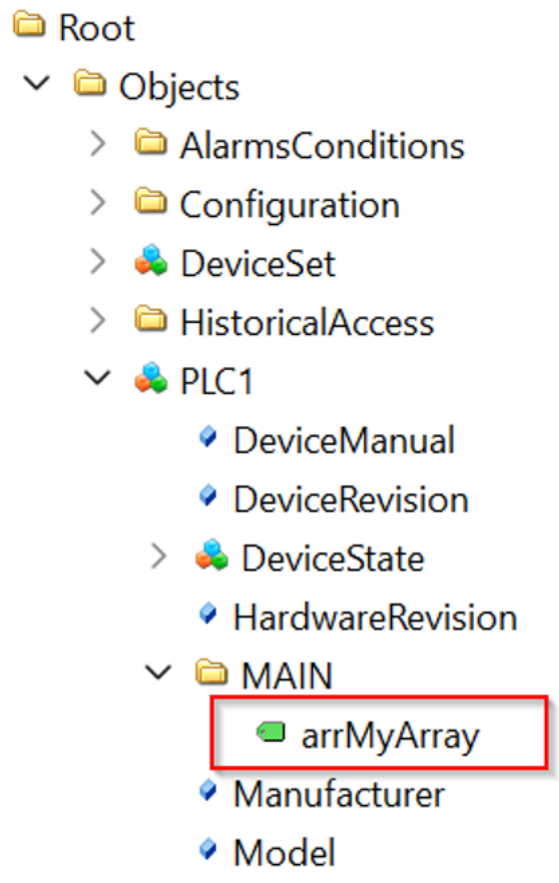
Value	
SourceTimestamp	1/21/2026 1:29:20.945 AM
SourcePicoseconds	0
ServerTimestamp	1/21/2026 1:29:20.945 AM
ServerPicoseconds	0
StatusCode	Good (0x00000000)
Value	This is a WSTRING.
DataType	String
NamespaceIndex	0
IdentifierType	Numeric
Identifier	12 [String]

4.11.6 Arrays

By default, arrays are mapped as individual nodes in the server's address space. An OPC UA client still has the option of accessing individual elements of the array.

The following array in the PLC would thus be displayed as a single node, with the array limits being defined via the node attributes.

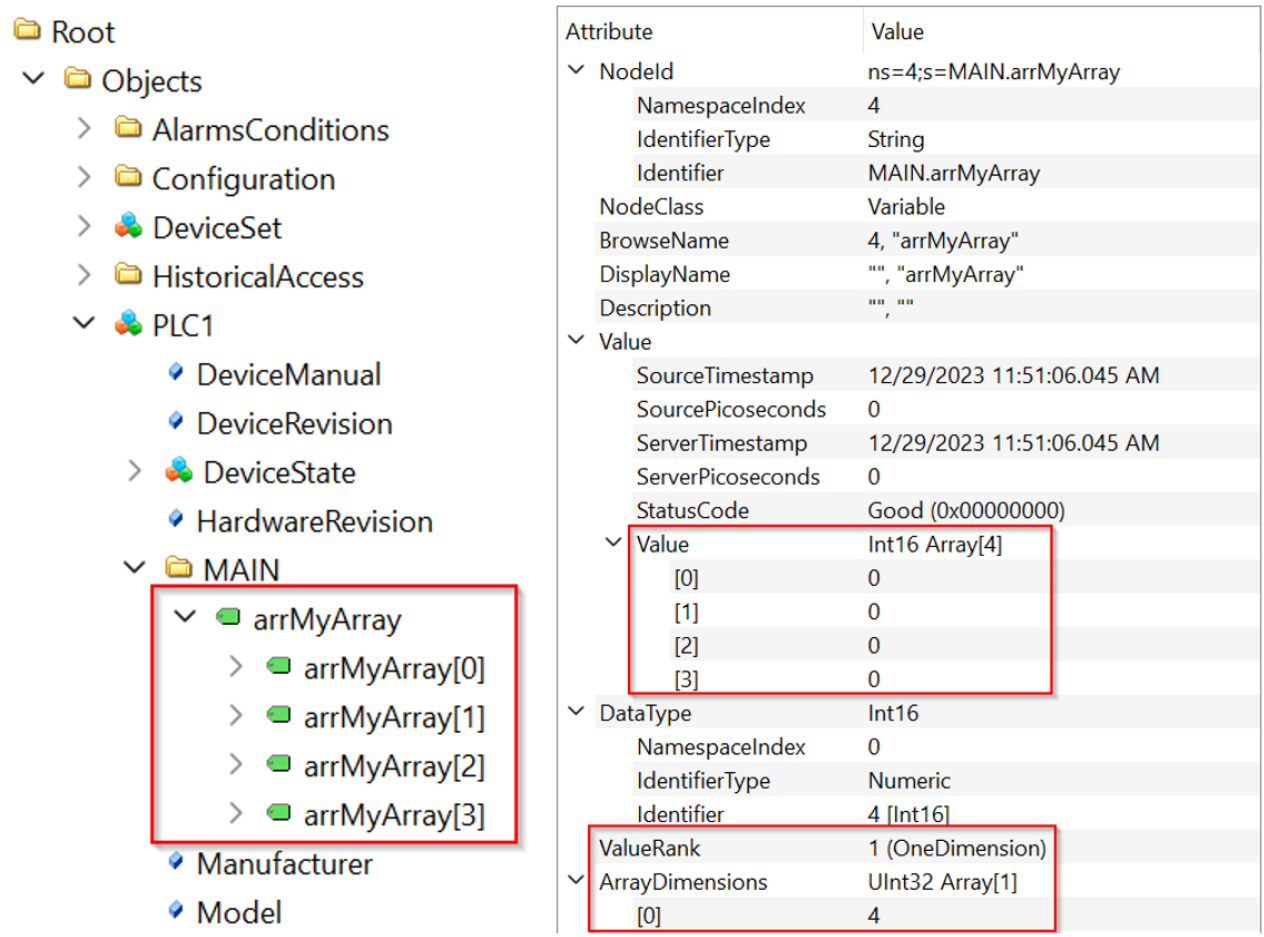
```
{attribute 'OPC.UA.DA' := '1'}
arrMyArray : ARRAY[0..3] OF INT;
```



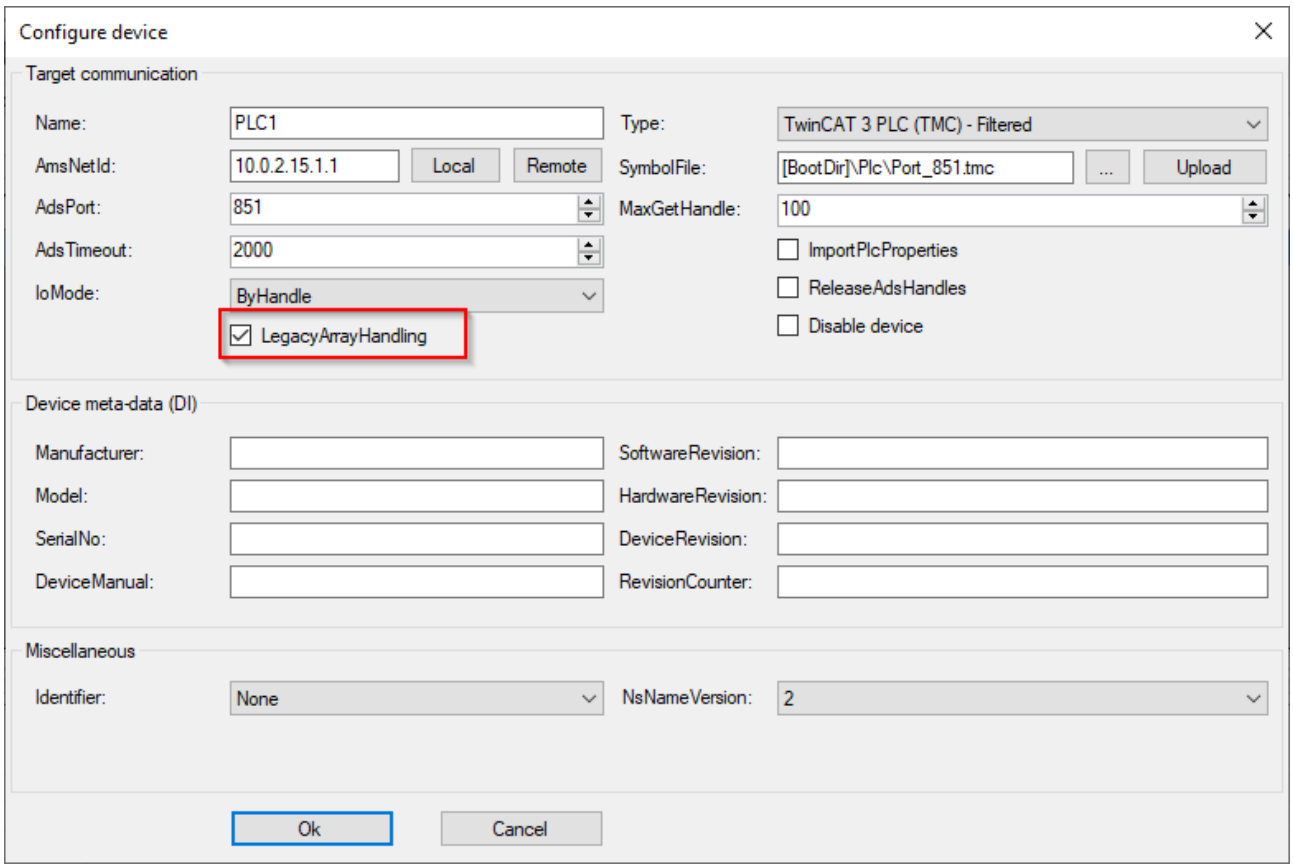
Attribute	Value
NodeId	ns=4;s=MAIN.arrMyArray
NamespaceIndex	4
IdentifierType	String
Identifier	MAIN.arrMyArray
NodeClass	Variable
BrowseName	4, "arrMyArray"
DisplayName	"", "arrMyArray"
Description	"", ""
Value	
SourceTimestamp	12/29/2023 11:51:06.045 AM
SourcePicoseconds	0
ServerTimestamp	12/29/2023 11:51:06.045 AM
ServerPicoseconds	0
StatusCode	Good (0x00000000)
Value	Int16 Array[4]
[0]	0
[1]	0
[2]	0
[3]	0
DataType	Int16
NamespaceIndex	0
IdentifierType	Numeric
Identifier	4 [Int16]
ValueRank	1 (OneDimension)
ArrayDimensions	UInt32 Array[1]
[0]	4

The advantage is a considerable reduction in the complexity of the server address space and memory consumption, since not every position of an array needs to be made available as an individual node in the namespace.

Older OPC UA clients may not yet support this function. Therefore, a special configuration switch enables all individual elements of an array to be provided as individual nodes in the namespace. This is as follows:



In addition to the root element of the array, all individual elements are also available as separate nodes. This configuration switch is available by activating the "LegacyArrayHandling" option on a data access device in the TwinCAT OPC UA Configurator.



Increased storage requirements

Activating this configuration switch significantly increases the address space of the server. Depending on the scope of the project and especially when using nested, complex arrays, this results in increased memory load of the TwinCAT OPC UA Server.
Make sure that your hardware device has sufficient main memory.

4.11.7 Enums

Enumerations in OPC UA always have the data type Int32. However, the IEC 61131-3 standard allows the definition of larger data types than Int32. To ensure that these enumerations are handled properly, the TwinCAT OPC UA Server offers the configuration option *<ImportBigEnumsNumeric>*, which can be enabled in its Data Access configuration file.

This option is set to FALSE by default. This means that a status code exception BadOutOfRange is triggered if the enumeration value is outside the Int32 range.

If the option is set to TRUE, enumerations with data types greater than Int32 are treated as regular variables with this particular data type.

Let's assume we have the following enumeration definitions in our PLC code:

```
TYPE E_Enum_Normal :
(
  enum_member_0 := 0,
  enum_member_1 := 1,
  enum_member_2 := 2
);
END_TYPE

TYPE E_Enum_NotSoBig :
(
  enum_member_0 := 0,
  enum_member_1 := 1,
  enum_member_2 := 2
) UINT;
END_TYPE

TYPE E_Enum_VeryBig :
```

```
(
  enum_member_0 := 0,
  enum_member_1 := 1,
  enum_member_2 := 2
) LINT;
END_TYPE
```

If the above configuration option is activated, instances of E_Enum_VeryBig are treated as regular variables of data type Int64 in the server namespace, while instances of E_Enum_Normal and E_Enum_NotSoBig are treated as OPC UA enumeration (with data type Int32):

Data Access View

#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Status
1	TcOpcUaServer...	NS4|String|MAI...	eEnumVeryBig	0	Int64	09:05:56.115	09:05:56.115	Good
2	TcOpcUaServer...	NS4|String|MAI...	eEnumNormal	0 (enum_member_0)	Int32	09:05:59.115	09:05:59.115	Good
3	TcOpcUaServer...	NS4|String|MAI...	eEnumNotSoBig	0 (enum_member_0)	Int32	09:07:25.594	09:07:25.594	Good

4.11.8 Structures

Requirements

This functionality is only available for Data Access [▶ 56] devices based on TwinCAT 3 and the import of TMC symbol files [▶ 59].

The TwinCAT OPC UA Server enables the use of so-called StructuredTypes for structures from the TwinCAT 3 PLC. StructuredTypes allow structures to be read or written in a data-consistent manner and the type description of the structure to be made available to the client.

You can use a special pragma to define a structure as a StructuredType and make it available. If this pragma is not set, the structure is not loaded into the server's address space as a StructuredType, but is displayed as a FolderType. The member variables of the structure are displayed as separate nodes that can be accessed.

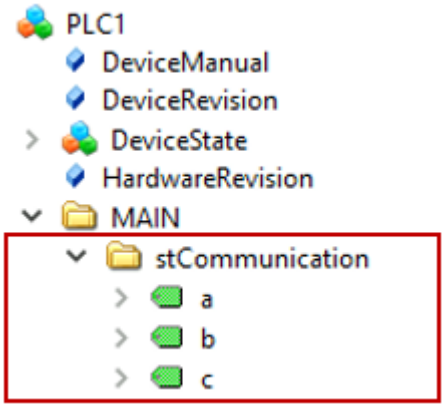
The following sample of a structure in the TwinCAT 3 PLC is given:

```
TYPE ST_Communication :
STRUCT
  a : INT;
  b : INT;
  c : INT;
END_STRUCT
END_TYPE
```

This structure is now instantiated in the MAIN program and enabled for OPC UA via a pragma, as described in the chapter Enabling (PLC) symbols [▶ 60].

```
PROGRAM MAIN
VAR
  {attribute 'OPC.UA.DA' := '1'}
  stCommunication : ST_Communication;
END_VAR
```

The structure instance is now provided in the server as follows by default:

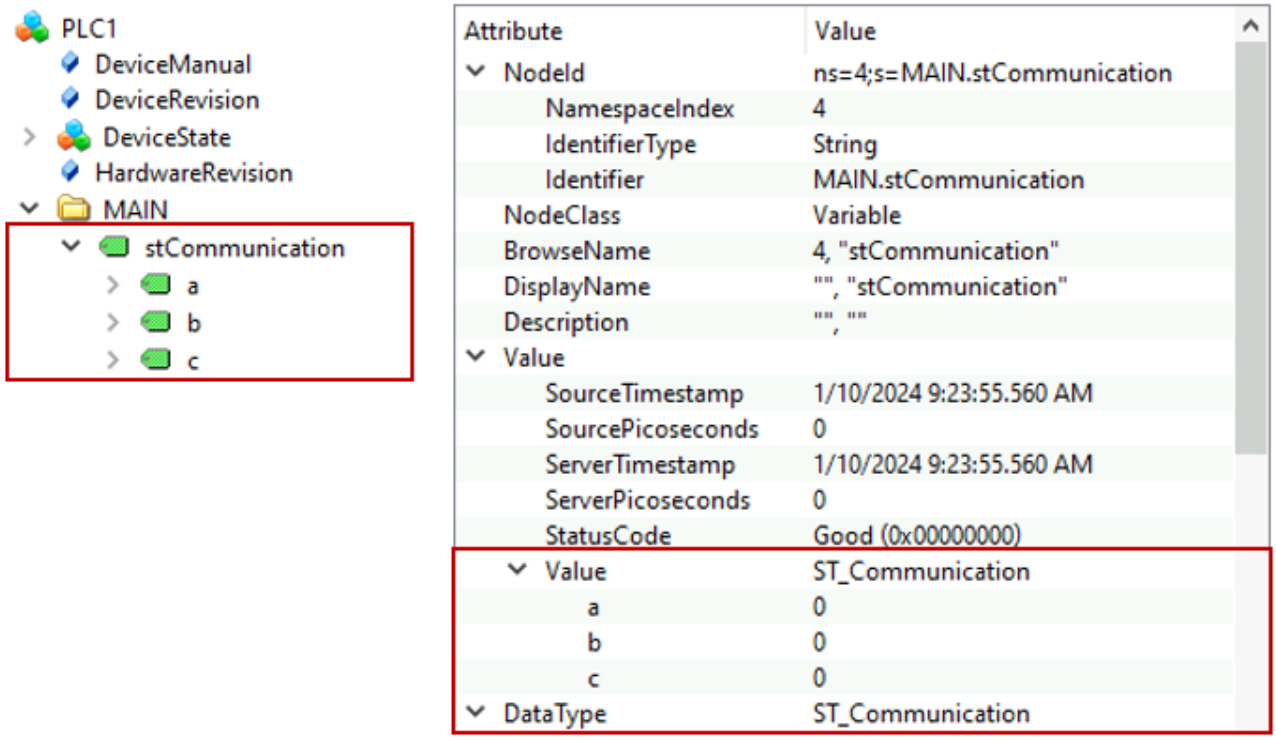


The individual member variables of the structure can now be accessed, but not the root element of the structure. As a result, data-consistent access to the entire structure may not be possible.

Using another pragma, this structure instance is now defined as a StructuredType.

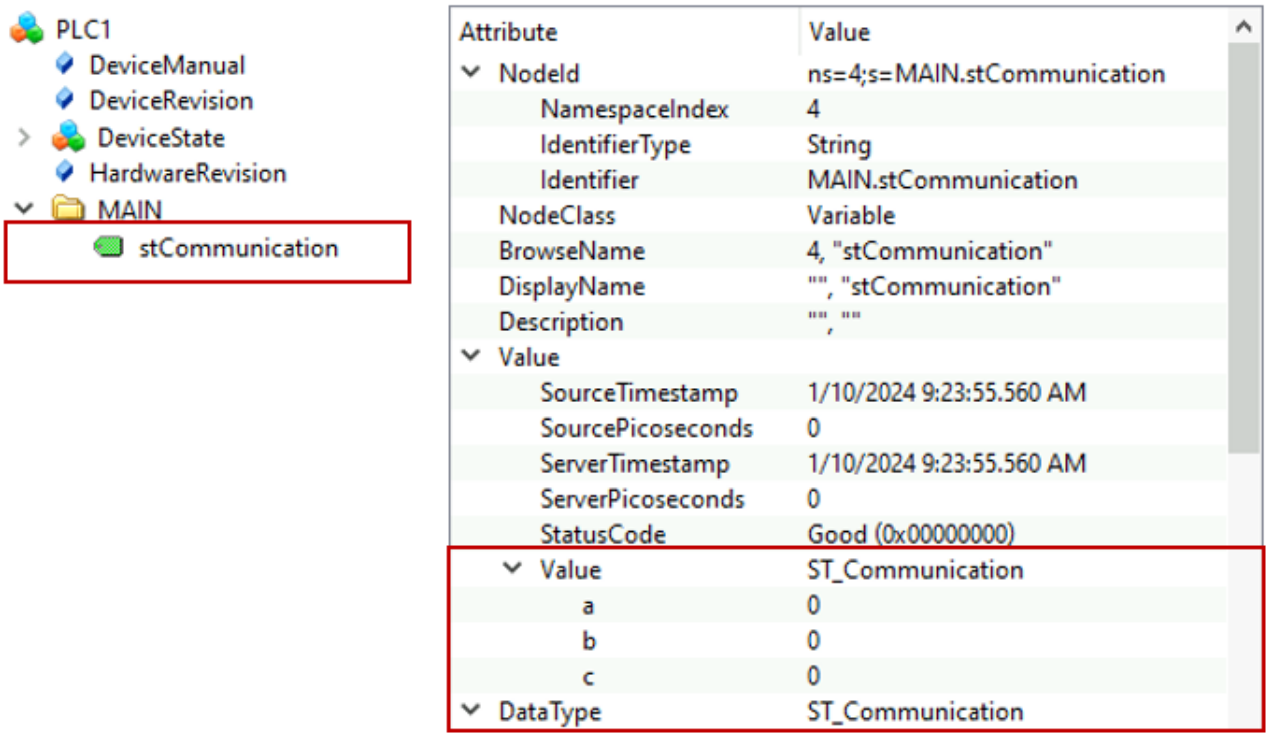
```
PROGRAM MAIN
VAR
  {attribute 'OPC.UA.DA' := '1'}
  {attribute 'OPC.UA.DA.StructuredType' := '1'}
  stCommunication : ST_Communication;
END_VAR
```

The instance is then provided in the TwinCAT OPC UA Server as follows:



When the root element is read, the structure information is provided in a data-consistent manner via OPC UA and can be processed by a client. In addition, the member variables are also displayed as separate nodes and can be accessed. You can deactivate this via a special attribute. The member variables can then only be accessed via the StructuredType, as shown in the following sample:

```
PROGRAM MAIN
VAR
  {attribute 'OPC.UA.DA' := '2'}
  {attribute 'OPC.UA.DA.StructuredType' := '1'}
  stCommunication : ST_Communication;
END_VAR
```



This is primarily a memory optimization of the server, as also described in the chapter Optimizations [▶ 49], because main memory must be allocated for each node in the address space.

You do not necessarily have to place the pragmas at every instance of a structure, but can also do this once at the structure definition. In this case, all instances of this structure are automatically enabled for OPC UA as StructuredType.

```
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.StructuredType' := '1'}
TYPE ST_Communication :
STRUCT
  a : INT;
  b : INT;
  c : INT;
END_STRUCT
END_TYPE
```

In this case, you can also explicitly deactivate the StructuredType definition for certain instances by using the following attribute:

```
PROGRAM MAIN
VAR
  {attribute 'OPC.UA.DA.StructuredType' := '0'}
  stCommunication : ST_Communication;
END_VAR
```

StructuredType for function blocks

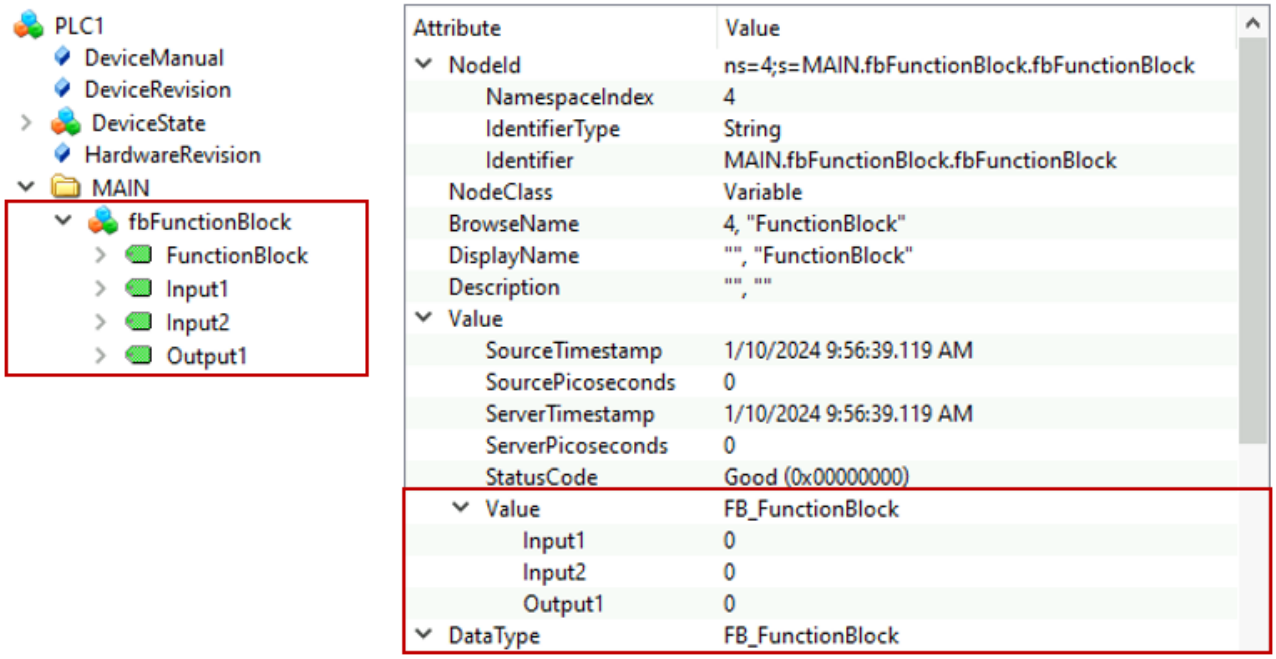
You can also use StructuredTypes with PLC function blocks. In this case, a function block instance receives a child node called "FunctionBlock", which represents the entire function block as a StructuredType. Consider the following sample:

```
FUNCTION_BLOCK FB_FunctionBlock
VAR_INPUT
  Input1 : INT;
  Input2 : LREAL;
END_VAR
VAR_OUTPUT
  Output1 : LREAL;
END_VAR
```

An instance of the function block is created in the MAIN program and both enabled for OPC UA and defined as a StructuredType via pragmas.

```
PROGRAM MAIN
VAR
  {attribute 'OPC.UA.DA' := '1'}
  {attribute 'OPC.UA.DA.StructuredType' := '1'}
  fbFunctionBlock : FB_FunctionBlock;
END_VAR
```

The TwinCAT OPC UA then provides the function block instance as follows:



The definition locations of pragmas mentioned above for structures also apply to function blocks. You can also explicitly hide the member variables of a function block here.

Restrictions

The following restrictions apply when using StructuredTypes.

Pointers and references

If pointer and reference types are used in the structure, they cannot be converted into a StructuredType. The OPC UA Server then illustrates these structures as regular FolderTypes with the corresponding member variables.

Maximum size of the structure

The maximum size of a structure is limited to 16 kB by default. If the size of the structure exceeds this limit, structure instances are displayed as FolderType. You can increase this limit if necessary. The background is described in more detail in the chapter Optimizations [▶ 49]. Each STRUCT constantly exchanges data with the basic ADS device, i.e. a large ADS message is sent with each read/write command of a StructuredType. To prevent the ADS router from being flooded with large messages, the maximum size is limited.

4.11.9 Properties

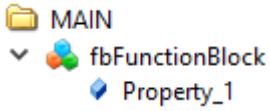
Requirements

This functionality is only available for Data Access [▶ 56] devices based on TwinCAT 3 and the import of TMC [▶ 59] symbol files, as well as the online symbolism.

The display of PLC properties in the server namespace has been supported since TwinCAT 3.1 Build 4024. All you need to do is set two special PLC attributes on the Property so that the Property is imported into the namespace and represented there as a UA Property. Sample:

```
{attribute 'OPC.UA.DA.Property' := '1'}
{attribute 'monitoring' := 'call'}
PROPERTY Property_1 : BOOL
```

The function block on which the Property is defined must contain the normal PLC attribute for enabling symbols.



In the configuration file TcUaDaConfig.xml the handling of PLC properties can be defined globally. The "ImportPlcProperties" flag is used for this purpose.

Value	Description
false	Displays all properties in the OPC UA namespace for which both the OPC.UA.DA.Property and the monitoring attribute have been set.
true	Displays all properties in the OPC UA namespace where the monitoring attribute has been set.

4.11.10 StatusCode

Requirements

This functionality is only available for Data Access [▶ 56] devices based on TwinCAT 3 and the import of TMC symbol files [▶ 59].

New implementation in setup version 5.x

The functionality for the TwinCAT OPC UA Server in setup version 5.x is described below. This is followed by a description [▶ 88] of the functionality for the TwinCAT OPC UA Server in setup version 4.x..

The TwinCAT OPC UA Server enables the adaptation of the OPC UA Status Code for a specific PLC symbol. Carry out the following steps to be able to change the StatusCode of a symbol at runtime.

Creating a structure

To ensure data consistency, the concept is based on a structure (see Structured Types). Each symbol for which the StatusCode is to be changed must be packaged in a structure.

Create a new structure and add the following pragma before the structure definition. The structure contains the symbol itself and a variable of data type DINT, which represents the StatusCode in decimal form.

```
{attribute 'OPC.UA.DA.StructuredType' := '1'}
{attribute 'OPC.UA.DA.StatusAndValue' := '1'}
TYPE ST_StatusCodeSimple :
STRUCT
  value : REAL;
  status : DINT := -2147155968; // equals 'BadCommunicationError'
END_STRUCT
END_TYPE
```

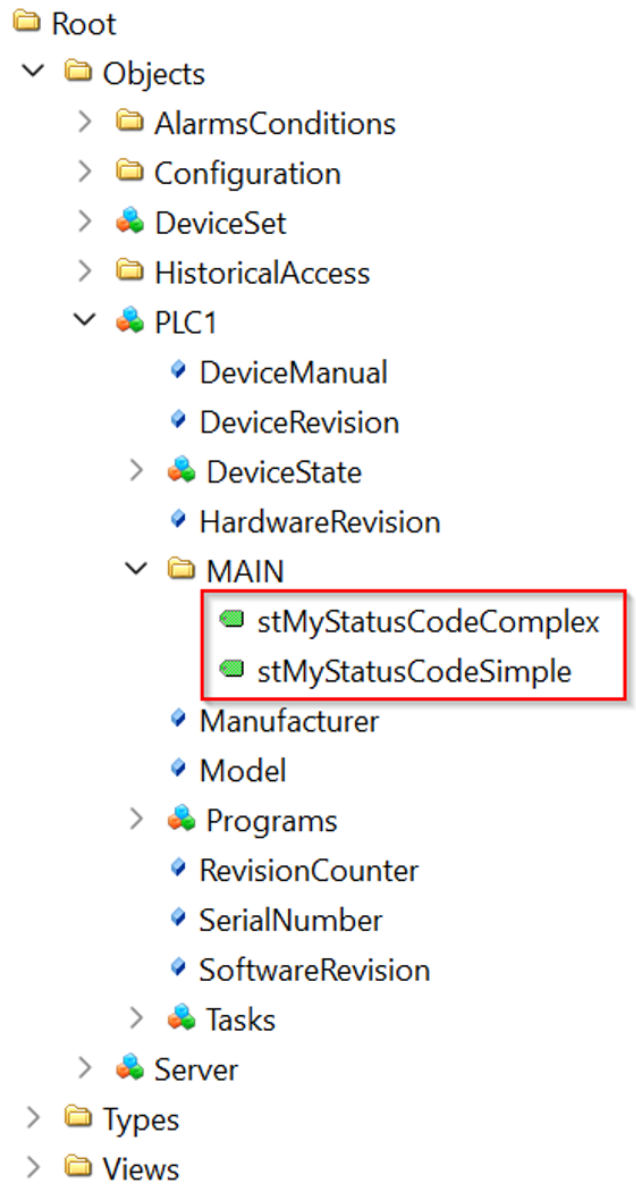
You can also use complex symbols, for example:

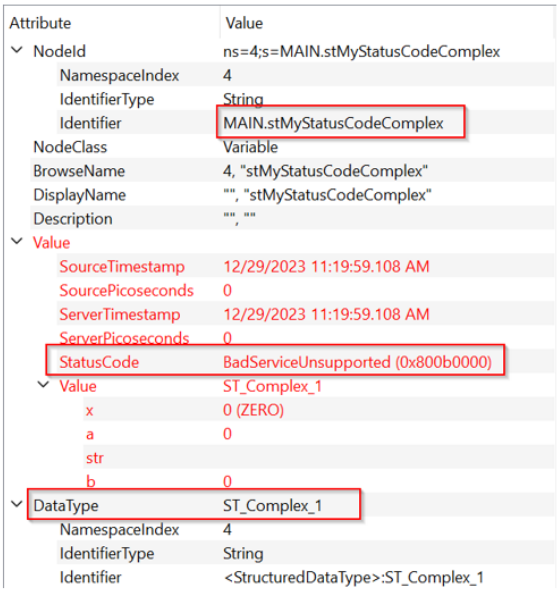
```
{attribute 'OPC.UA.DA.StructuredType' := '1'}
{attribute 'OPC.UA.DA.StatusAndValue' := '1'}
TYPE ST_StatusCodeComplex :
STRUCT
  {attribute 'OPC.UA.DA.StructuredType' := '1'}
  value : ST_Complex_1;
  status : DINT := -2146762752; // equals 'BadServiceUnsupported'
END_STRUCT
END_TYPE
```

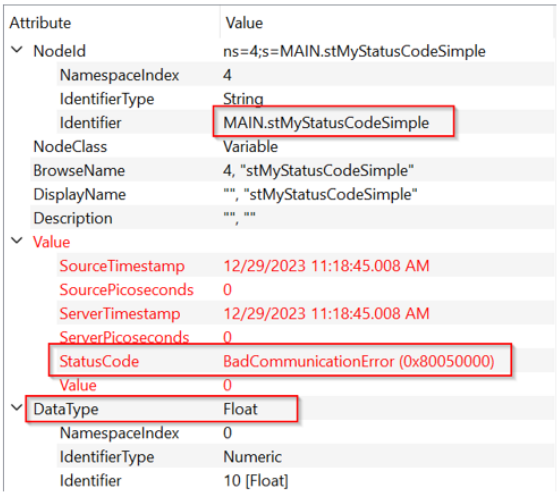
Now create an instance of this structure, e.g. in the MAIN program, and add the pragma to enable the instance via OPC UA.

```
PROGRAM MAIN
VAR
  {attribute 'OPC.UA.DA' := '1'}
  stStatusCodeSimple : ST_StatusCodeSimple;
  {attribute 'OPC.UA.DA' := '1'}
  stStatusCodeComplex : ST_StatusCodeComplex;
END_VAR
```

The defined structure instances are now provided with the data type of the member variables defined as "value" in the TwinCAT OPC UA Server.

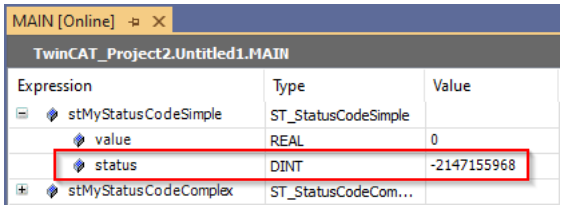






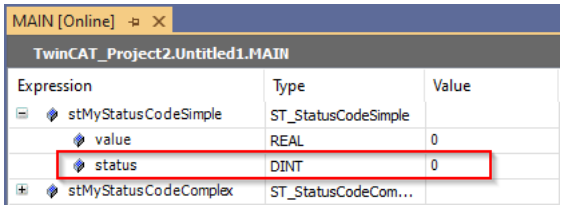
Changing the StatusCode at runtime

To change the StatusCode at runtime, simply edit the value of the member variable "status". If you set this to "-2147155968", as in the sample above, the StatusCode changes to "BadCommunicationError".



#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	TcOpcUaServer...	NS4|String|MAIN.stMyStatusCodeSimple	stMyStatusCodeSimple	0	Float	11:24:44.038 AM	11:24:44.038 AM	BadCommunicationError

However, if you set the value to "0", the StatusCode changes to "Good".

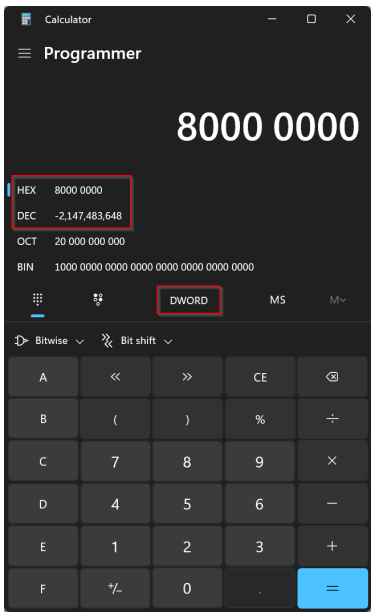


#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	TcOpcUaServer...	NS4|String|MAIN.stMyStatusCodeSimple	stMyStatusCodeSimple	0	Float	11:26:08.539 AM	11:26:08.539 AM	Good

The decimal value used here is defined by the OPC UA specification, here the current version of the StatusCode mapping. The table below lists some common StatusCodes and their numerical representation.

StatusCode	Hex	Decimal
Good	0x00000000	0
Uncertain	0x40000000	1073741824
Bad	0x80000000	-2147483648
BadUnexpectedError	0x80010000	-2147418112
BadInternalError	0x80020000	-2147352576
BadCommunicationError	0x80050000	-2147155968
BadTimeout	0x800A0000	-2146828288
BadServiceNotSupported	0x800B0000	-2146762752

When calculating the decimal representation of a StatusCode on the basis of its hexadecimal representation, please note that your computer is set to DWORD, for example the Windows Calculator ("Programmer" view).

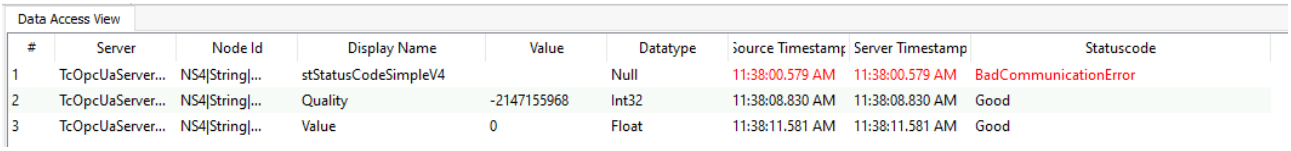


Use in Setup version 4.x

In the "older" implementation in Setup version 4.x, the variable is not displayed as a single variable on the OPC UA side, but as a structure in the same way as on the PLC side. There are also differences in the configuration of the attributes.

```
{attribute 'OPC.UA.DA.StructuredType' := '1'}
{attribute 'OPC.UA.DA.STATUS' := 'Quality'}
TYPE ST_StatusCodeSimple_V4 :
STRUCT
  Value : REAL;
  Quality : DINT := -2147155968; // equals 'BadCommunicationError'
END_STRUCT
END_TYPE
```

The StatusCode is only made available at the structure variable via OPC UA. The Value and Quality values contain the value and the value of the status code.



4.11.11 AnalogItemType

Requirements

This functionality is only available for Data Access [▶ 56] devices based on TwinCAT 3 and the import of TMC symbol files [▶ 59].

AnalogItemTypes are part of the OPC UA specification and allow meta information such as units to be attached to a variable. You can define these items of meta information in the form of PLC attributes in the TwinCAT 3 PLC.

The following parameters can be set:

- EngineeringUnits: Units defined by the OPC UA specification
- EURange: Maximum value range of the variables
- InstrumentRange: Maximum value range of the variables
- WriteBehavior: Behavior if the value range is exceeded during a write operation.

The following sample shows how the fillLevel variable is configured as an AnalogItemType. The following parameters are set:

- Unit: 20529 ("Percent", defined in the OPC UA specification)
- Max. value range: 0 to 100
- Normal value range: 10 to 90
- Write behavior: 1 (Clamping)

```
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.AnalogItemType' := '1'}
{attribute 'OPC.UA.DA.AnalogItemType.EngineeringUnits' := '20529'}
{attribute 'OPC.UA.DA.AnalogItemType.EURange' := '0:100'}
{attribute 'OPC.UA.DA.AnalogItemType.InstrumentRange' := '10:90'}
{attribute 'OPC.UA.DA.AnalogItemType.WriteBehavior' := '1'}
fillLevel : UINT;
```

EngineeringUnits can be configured using the IDs specified in OPC UA (Part 8 of the OPC UA specification). The IDs are based on the widely used and accepted "Codes for Units of Measurement (Recommendation N.20)" published by the "United Nations Center for Trade Facilitation and Electronic Business". CommonCode, which specifies the three-digit alphanumeric ID, is converted by OPC UA according to specification into an Int32 value and referenced (extract from OPC UA specification v1.02, pseudo-code):

```
Int32 unitId = 0;
Int32 c;
for (i=0; i<=3;i++)
{
  c = CommonCode[i];
  if (c == 0)
    break; // end of Common Code
  unitId = unitId << 8; // shift left
  unitId = unitId | c; // OR operation
}
```

Write behavior

When writing an AnalogItemType variable, you can define how the OPC UA Server should handle the new value in relation to the value range. The following options are available:

- 0: All values are allowed and are accepted during a write operation.
- 1: The value to be written is truncated according to the value range.

- 2: The value to be written is rejected if it exceeds the value range.

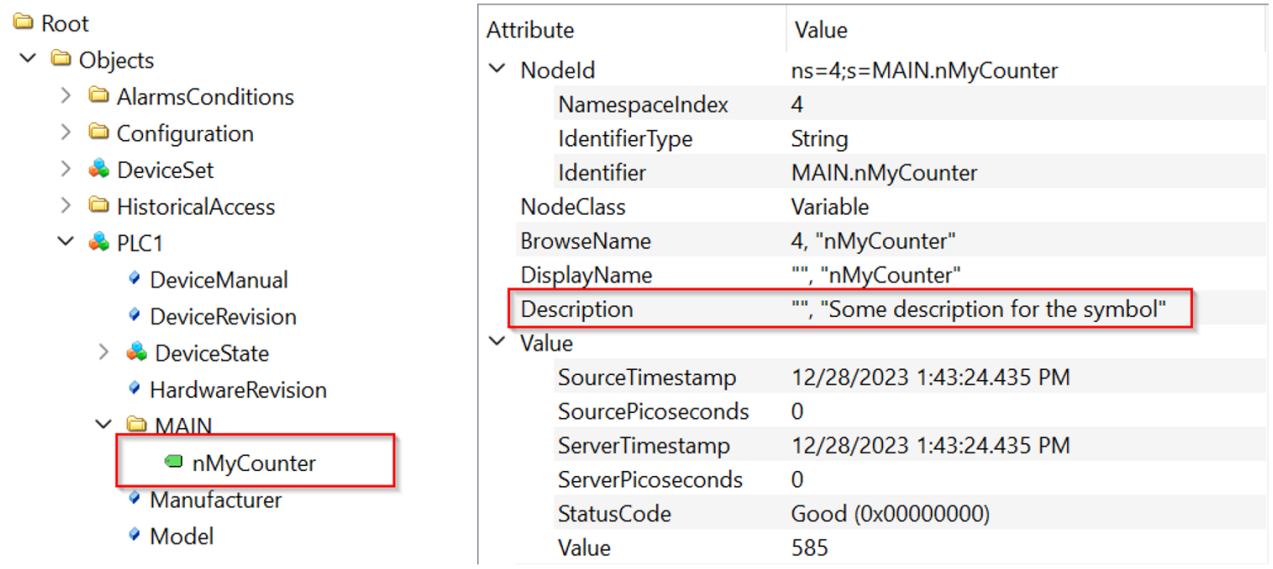
4.11.12 Description

Requirements

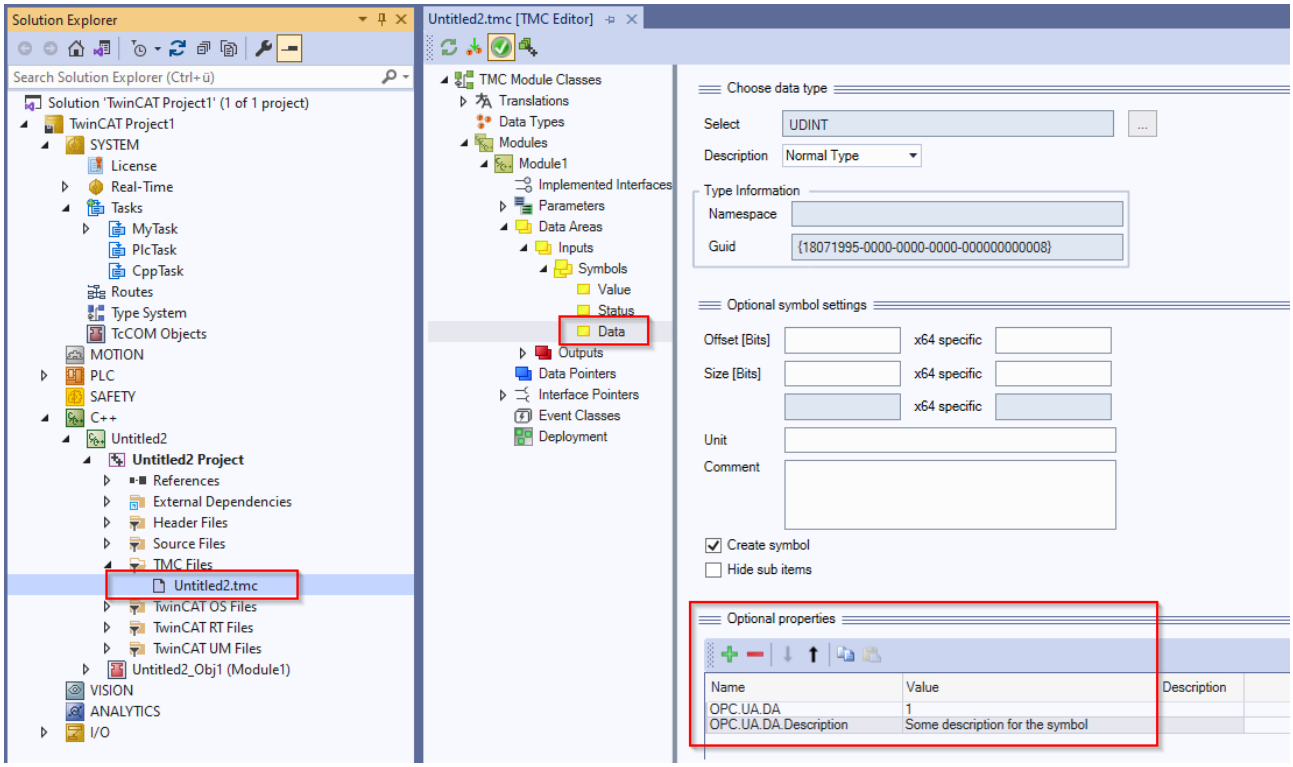
This functionality is only available for Data Access [▶ 56] devices based on TwinCAT 3 and the import of TMC and TMI [▶ 59] symbol files, as well as the online symbolism.

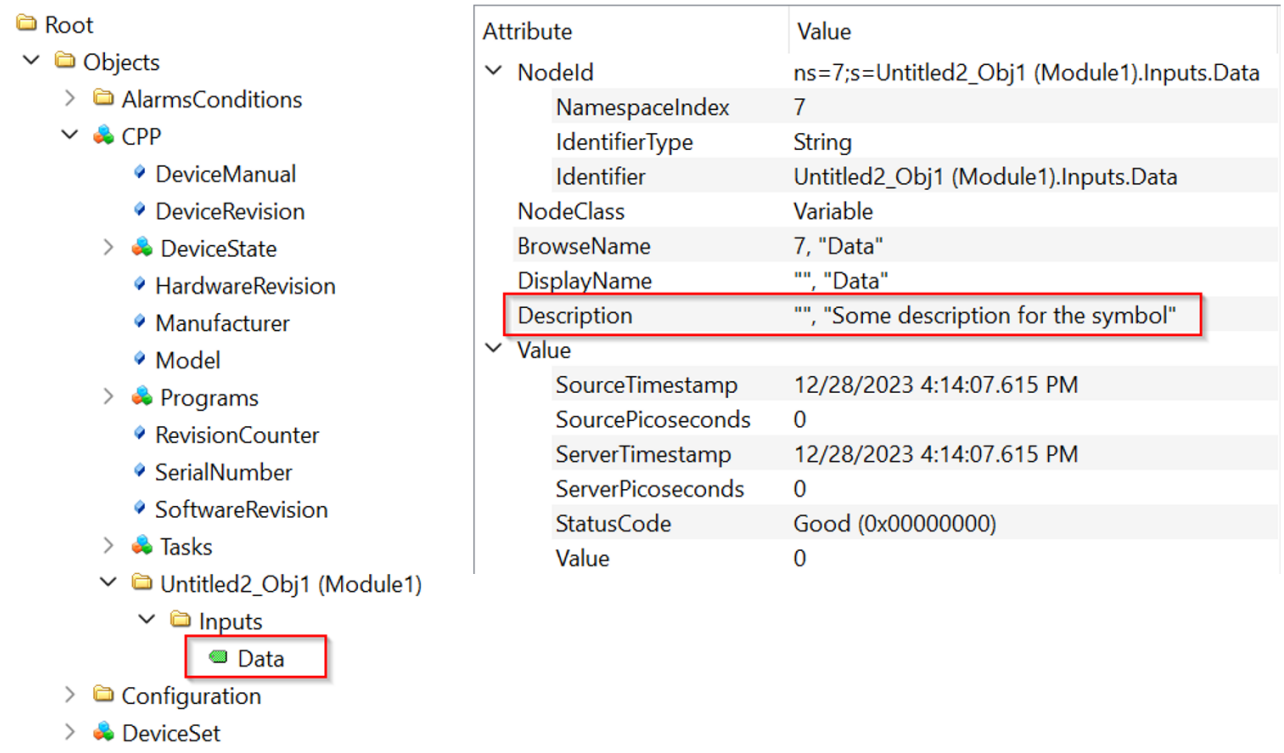
The following TwinCAT 3 PLC [▶ 60] pragma can be used to set the OPC UA attribute "Description" for a symbol.

```
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Description' := 'Some description for the symbol'}
nMyCounter : INT;
```



In the case of TwinCAT 3 C++ [▶ 63], you can also use this pragma in the TMC Code Generator to set a description for the node. Please note that this feature is only available under TwinCAT 3 C++ when using online symbolism [▶ 75].





4.11.13 ReadOnly

Requirements

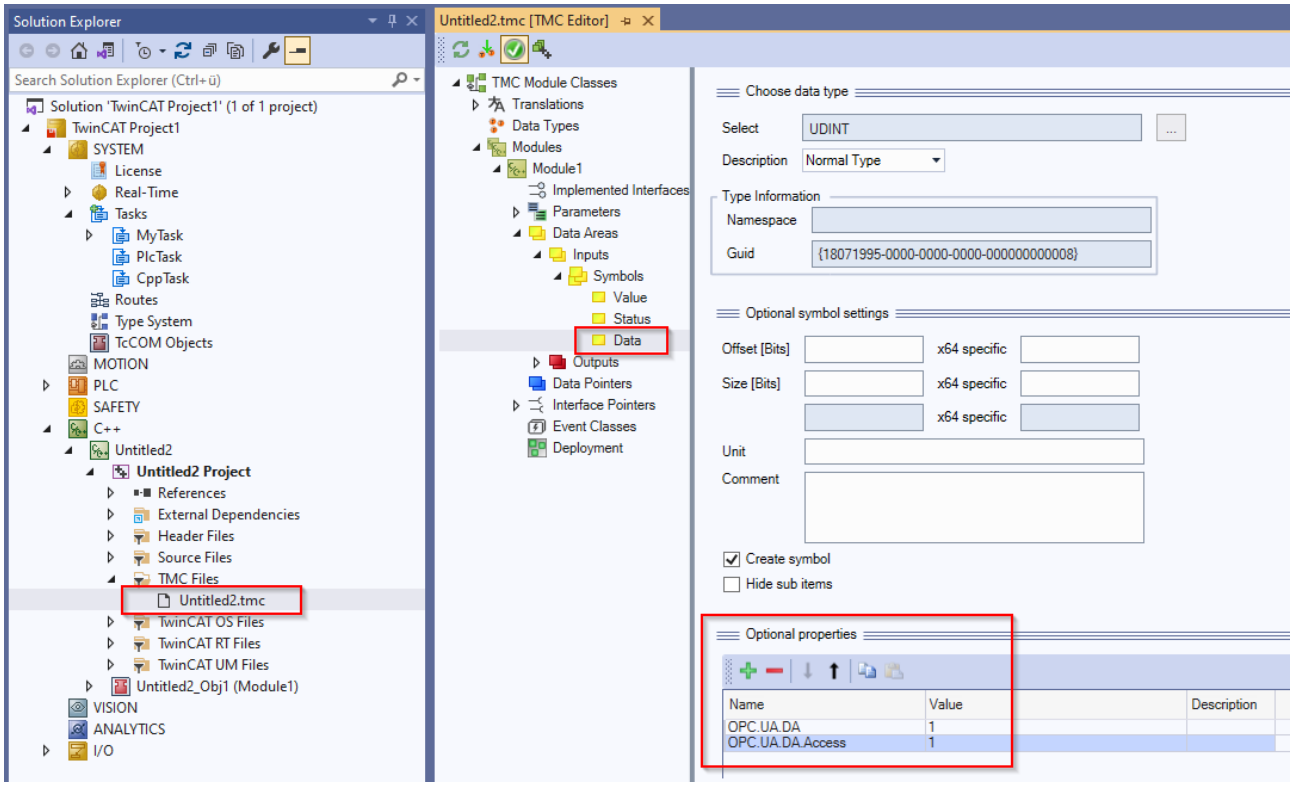
This functionality is only available for Data Access [▶ 56] devices based on TwinCAT 3 and the import of TMC symbol files [▶ 59].

The following pragma can be used to set a symbol read-only. The server then acknowledges write operations by an OPC UA client with the StatusCode BadNotWriteable.

```
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
nMyCounter : INT;
```

Timestamp	Source	Server	Message
12/28/2023 4:21:21.953 PM	DA Plugin	TcOpcUaServer@DESKTOP-28AUAPK	Write to node 'NS7|String|Untitled2_Obj1 (Module1).Inputs.Data' failed [ret = BadNotWritable]

In the case of TwinCAT 3 C++ [▶ 63], you can also use this pragma in the TMC Code Generator to mark a node as read-only. Please note that this feature is only available under TwinCAT 3 C++ when using online symbols [▶ 75].



4.11.14 Pointers and references

Pointer

Pointer variables (e.g. POINTER TO) are generally not represented by the server in the namespace. If a pointer variable is located in a structure and this structure has been configured as Structured Data Type, the structure will not be displayed as Structured Data Type but as FolderType.

References

Reference variables (REFERENCE TO) are represented as single variables by the server in the namespace and can be read without restrictions. If a reference is inside a structure, this structure can no longer be made available as StructuredTypes in the server, but only as FolderType. However, access to the individual reference variables within the structure works.

4.11.15 Type system

Requirements

This functionality is only available for Data Access [▶ 56] devices based on TwinCAT 3 and the import of TMC [▶ 59] symbol files, as well as the online symbolism.

One of the biggest advantages of OPC UA is the meta-model, which can be used to provide base types as well as to extend the type system with custom models. The same mechanism is used to represent real objects (nodes) so that OPC UA clients can determine an object type.

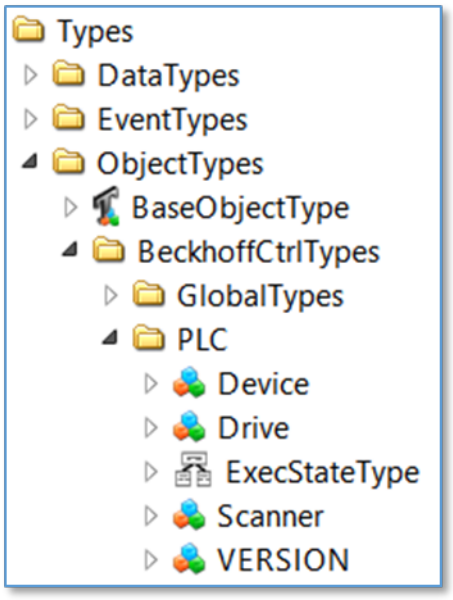
The OPC UA server publishes type information from the IEC61131 world in its namespace. This includes not only base types such as BOOL, INT, DINT or REAL, but also extended type information such as the current class (function block) or structure that represents an object.

Type information

Type information is part of the UA namespace. The OPC UA server extends the base type information as follows:

- Local type information that is only valid for one runtime is stored in the same namespace as the runtime symbols.
- Global type information that can be valid for different runtimes is stored in a separate global namespace.

The type system is also virtually available and can be viewed in the Types area of the OPC UA server:

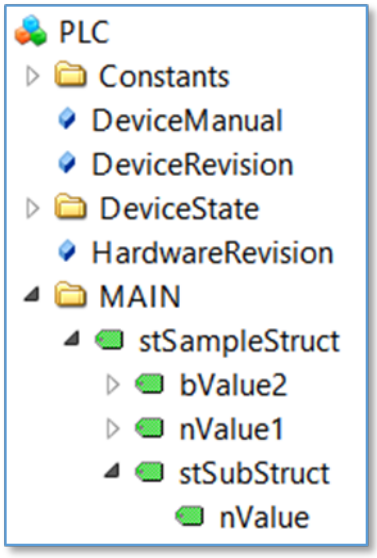


Every non-standard data type is entered in the BeckhoffCtrlTypes area.

Basics

Assuming the TwinCAT 3 PLC consists of a PLC program with different STRUCTs. Each STRUCT is represented as a node in a UA namespace, with each element of the structure as a subordinate node.

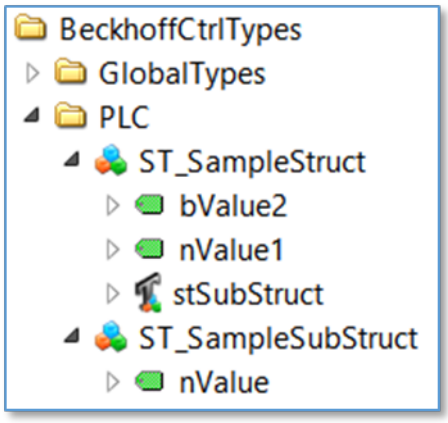
In this sample the STRUCT stSampleStruct consists of three subordinate elements: one variable nValue1 of the type INT, one variable bValue2 of the type BOOL and a further STRUCT stSubStruct, which contains only one subordinate element (variable nValue of the type INT).



The structure itself is a regular variable in the UA namespace and has the data type ByteString. The clients can therefore simply be connected to the root element (the structure itself), and its values can be read/ written by interpreting the ByteString. To simplify the interpretation of each subordinate element, the type system contains more information about the structure itself, primarily in the instance reference:

Reference	Target DisplayName
HasTypeDe...	ST_SampleStruct
HasCompo...	nValue1
HasCompo...	bValue2
HasCompo...	stSubStruct

In addition, the type system contains more information about ST_SampleStruct:



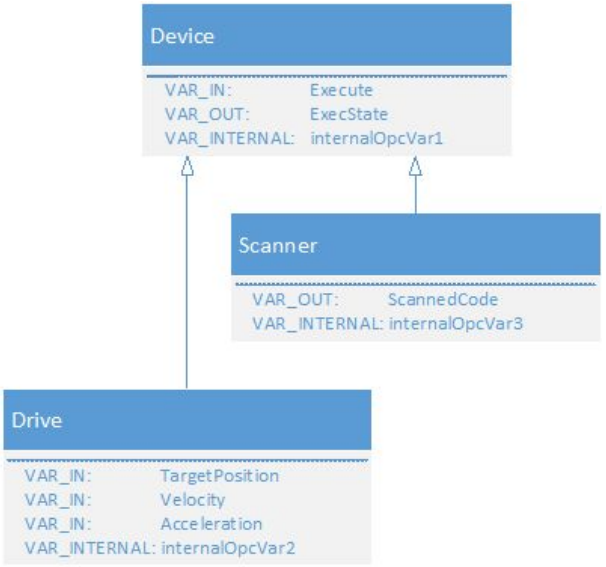
And in the references of ST_SampleStruct:

Reference	Target DisplayName
HasTypeDe...	DataItemType
HasCompo...	nValue1
HasCompo...	bValue2
HasCompo...	stSubStruct

Overall, the type system can offer very useful information if a Client wants to interpret the structure further.

Object-orientated extensions

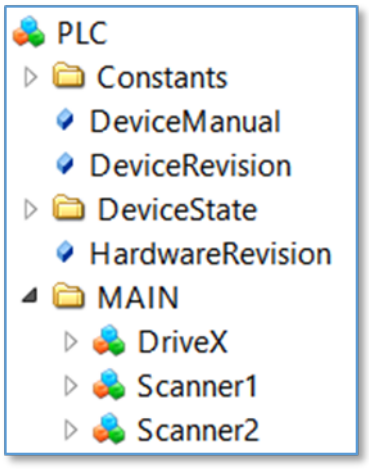
Assuming the TwinCAT 3 PLC runtime contains a PLC program whose structure can be visualized as follows:



The two function blocks Scanner and Drive are derived from the base class Device by using object-oriented extensions of IEC61131-3. The program MAIN now contains the following declarations:

```
PROGRAM MAIN
VAR
  {attribute 'OPC.UA.DA':='1'}
  Scanner1 : Scanner;
  {attribute 'OPC.UA.DA':='1'}
  Scanner2 : Scanner;
  {attribute 'OPC.UA.DA':='1'}
  DriveX : Drive;
END_VAR
```

All three objects are of type Device, but also of their special data type. The OPC UA server imports the objects as follows:



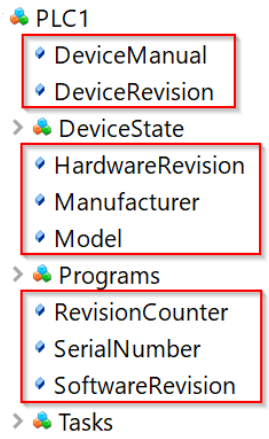
The base data type can now be determined in the reference of each object, e.g. object Scanner1:

Reference	Target DisplayName
HasTypeDe...	Scanner
HasInputVar	Execute
HasOutputV...	ExecState
HasLocalVar	internalOpcVar1
HasOutputV...	ScannedCode
HasLocalVar	internalOpcVar3

According to the basic IEC61131 program, the object Scanner1 is of the data type Scanner and also shows which variables are contained and what type the variable is: input, output or internal (local) variable. The diagram above shows that not only the variables of the actual function block are displayed here, but also the derived variables of the base class. The entire IEC61131-3 inheritance chain is represented in the UA namespace.

4.11.16 DI Components

Each PLC namespace on the server contains a number of nodes that can be used to specify static meta-information about the PLC. This optional information can be specified in the TcUaDaConfig.xml for each namespace.



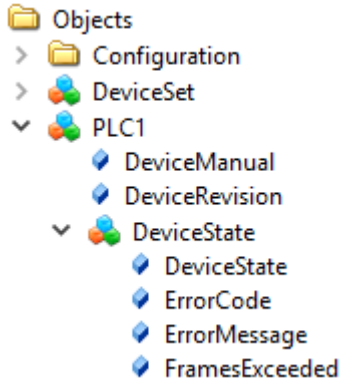
The following table provides more information about these nodes. The concrete value assignments of the individual nodes can be application-specific to a large extent, therefore only example values are used in the delivery state of the server. The server itself makes these nodes available in its namespace, but does not independently change their value assignments.

Node	Description
DeviceManual	Allows you to specify an address where the device manual can be found, e.g. a path in the file system or a web address.
DeviceRevision	Contains the revision level of a hardware component or the entire device.
HardwareRevision	Contains the revision level of the hardware.
Manufacturer	Contains the name of the device manufacturer, usually as FQDN (Fully Qualified Domain Name), e.g. beckhoff.com.
Model	Includes the name of the "product" (if applicable).
RevisionCounter	May include a counter of how many times the configuration of the device has been updated.
SerialNumber	Unique serial number of the device, as assigned by the device manufacturer.

Node	Description
SoftwareRevision	Contains the version or revision level of the software component, the firmware of a hardware component or even the firmware of the device.

4.11.17 DeviceState

Each namespace in the TwinCAT OPC UA Server contains a DeviceState object.



This object indicates the state of the lower-level ADS device by means of various properties.

```
typedef enum
{
  UADEV_NOTINIT = 0x0100,
  UADEV_STARTING = 0x0110,
  UADEV_CONNECTED = 0x0120,
  UADEV_SHUTDOWN = 0x0130,
  UADEV_ERROR = 0xF000
}UaDeviceState;
```

If the device is in an ERROR state, the ErrorCode Property returns the following values:

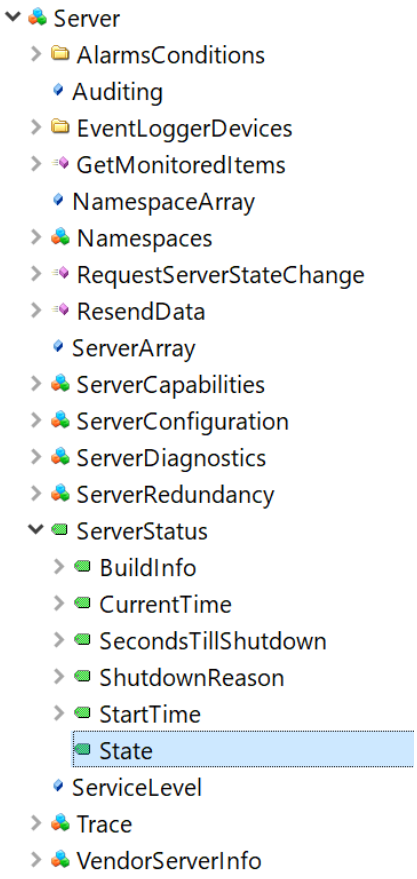
```
#define UA_DEVSTATE_INVALID_STATE 0x80EB0010
#define UA_DEVSTATE_CREATE_NS_FAILED 0x80EB0011
#define UA_DEVSTATE_LOAD_NS_FAILED 0x80EB0012
#define UA_DEVSTATE_INVALID_IO_SETTING 0x80EB0100
```

A corresponding error message is displayed in the ErrorMessage Property.

4.11.18 ServerState

The ServerState variable in the server namespace indicates the current state of the server. The following table gives an overview of the possible variable values.

Variable value	Description
Running	The server has been successfully booted.
Failed	A problem was found in one of the server configuration files, e.g. an invalid configuration in TcUaSecurityConfig.xml.
NoConfiguration	The server has not yet been initialized.
Suspended	The server has not yet been completely booted, i.e. not all functions may be available yet.



4.12 Historical Access

4.12.1 Overview

This chapter describes the steps required for configuring the variables in the namespace of the TwinCAT OPC UA Server for **H**istorical **A**ccess (HA).

Historical Access is an OPC UA function in which (historical) variable values are stored in a memory (e.g. a file or a database) and made accessible to clients via a standardized interface. Here you can configure how the TwinCAT OPC UA Server reads and saves the variable values.

Requirements

This configuration can be carried out for variables from any runtime system [▶ 56] (e.g. TwinCAT PLC or TwinCAT 3 C++, ...). As a prerequisite, the respective variable must first be enabled for OPC UA. You can find out how to do this in our Quick Start [▶ 27] tutorial.

Memory types

The individual memory types are configured as so-called "HistoryAdapters" in the TwinCAT OPC UA Server. This is done via the TwinCAT OPC UA Server configurator. Each HistoryAdapter (except for the adapter type "Main memory" (Volatile)) can be used multiple times, e.g. if the historical values for individual variables are to be stored in different data memories.

Sampling of the variable values

The TwinCAT OPC UA Server can receive variable values in two different ways:

1. Using its own sampling engine and
2. From an OPC UA client (via the so-called "HistoryUpdate" function)

The sampling rate at which the variable is to be sampled from the lower-level real-time system and stored in the memory is configured for each variable in the separate sampling engine. The time at which the server read the variable value from the real-time is used as the timestamp.

With the HistoryUpdate function, on the other hand, the server receives both the variable value and the timestamp from a connected OPC UA client and therefore does not perform its own sampling.

4.12.2 Supported functions

The following prerequisites apply to the use of Historical Access:

- The runtime system whose symbols are to be stored for Historical Access must be configured for Data Access (and the respective variables must be enabled).
- In order to use an SQL Compact database as a storage medium, SQL Compact Runtime 3.5 SP2 must be installed on the computer on which the OPC UA Server is running.
- SQL Compact databases are also supported under Windows CE.
- SQL Server databases are not supported under Windows CE.
- The following MS SQL Server versions are supported: 2017, 2019
- The following recommendations apply when using the respective memory type:

  Main memory: number of entries < 5000

  File system: number of entries < 10000

  Database: number of entries >= 10000

Memory types

The following memory types are supported:

Memory type	TF6100 Server Setup version	Operating systems	Description
Main memory	4.x and 5.x	Windows, Windows CE, TwinCAT/BSD	Saves the values in the RAM of the device on which the OPC UA Server is running. No further parameters are required. After restarting the device, the stored values are no longer available. The storage medium is therefore not persistent. The above requirements and recommendations apply.
File system	4.x	Windows, Windows CE	Saves the values in several files, the location of which can be specified. Each symbol configured for this storage medium is assigned its own file in this directory. In addition, there is a backup copy of the file for each symbol, which is created when the buffer size is reached. The contents of the data file are then saved as a backup copy, and a new file is created. The above requirements and recommendations apply.
SQL Compact database	4.x	Windows, Windows CE	Saves the values to an SQL Compact database, the location of which can be

Memory type	TF6100 Server Setup version	Operating systems	Description
			specified. The above requirements and recommendations apply.
MS SQL Server database	4.x	Windows	Saves the values in an SQL Server database that is referenced with various parameters. The above requirements and recommendations apply.
TwinCAT Analytics [▶ 102]	5.x	Windows, TwinCAT/BSD	Saves the values in a file format that corresponds to the TwinCAT Analytics storage format. The data can therefore be used further with the TwinCAT Analytics Toolchain. This format has replaced the old file-based storage format since TF6100 version 5.x (see above).

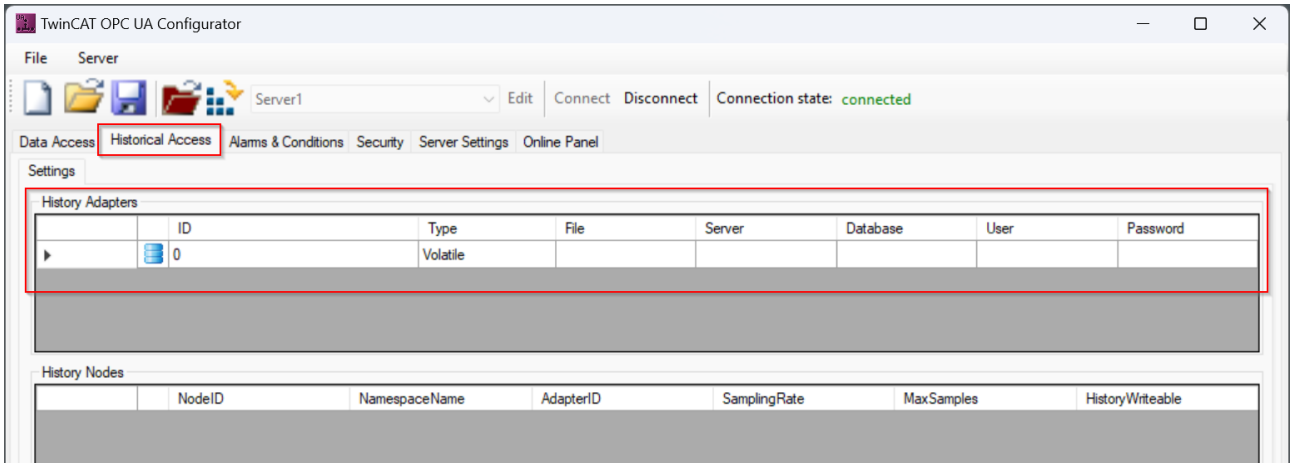
4.12.3 Configuration

You can use the TwinCAT OPC UA Configurator to configure the Historical Access functionality.
To configure a symbol for Historical Access, you must perform three steps:

1. The symbol must have been enabled for Data Access (see chapter Enabling symbols [▶ 59]).
2. A memory type must be created in which the symbol values are to be saved.
3. Configuration parameters must be defined for the symbol.

Creating a memory type

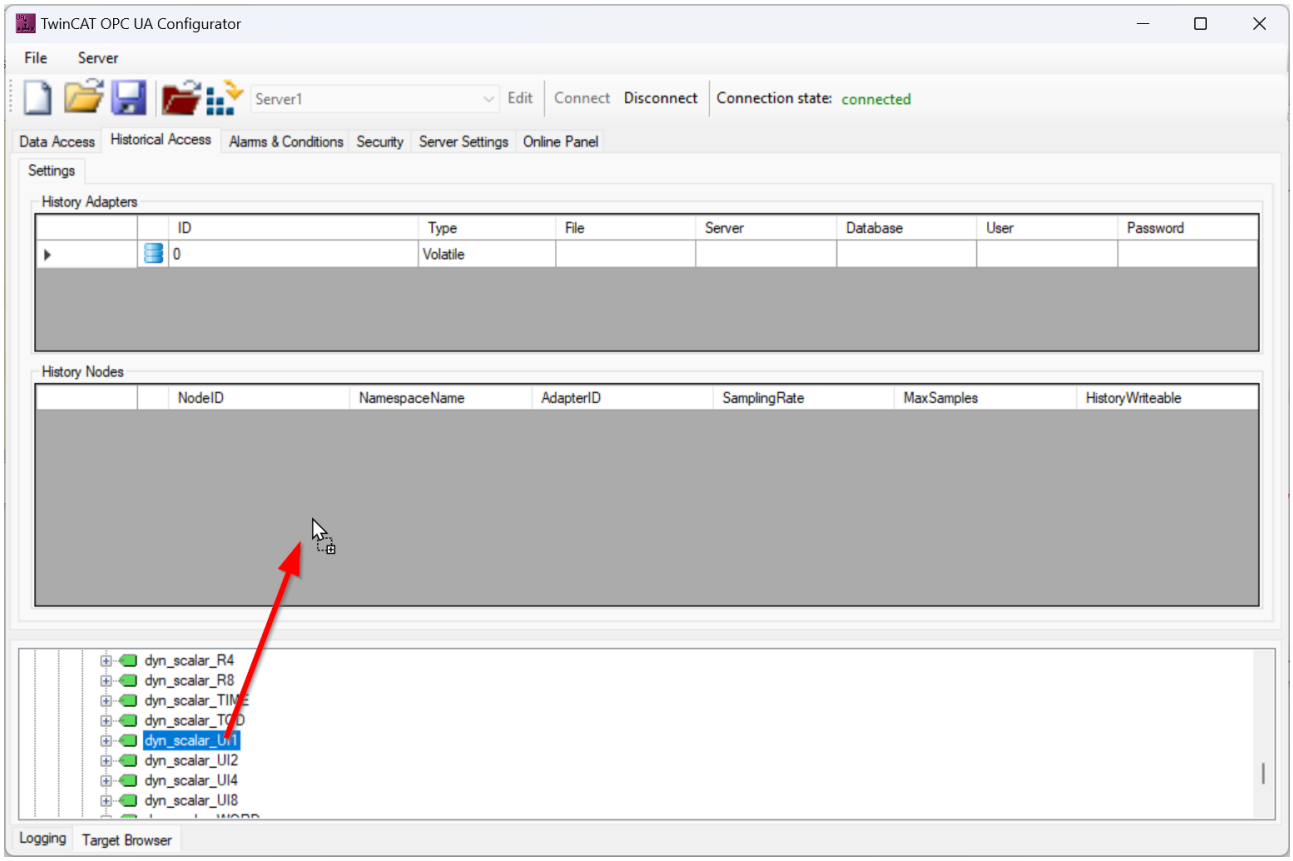
In the standalone version of the TwinCAT OPC UA Configurator you can configure the memory types (so-called "History Adapters") via the **Historical Access** tab. In the following screenshot, for example, the memory type "Main memory" ("Volatile") has been configured.



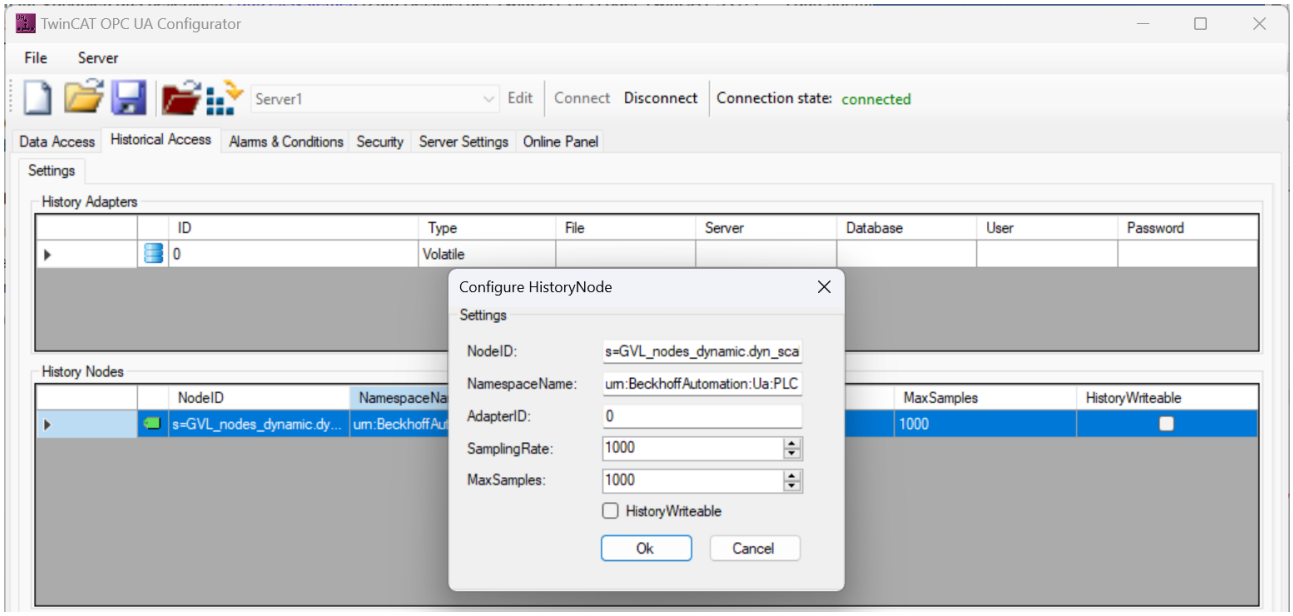
Additional memory types can be created or existing memory types can be edited or removed from the configuration via the context menu.

Configuring a symbol

In the "History Nodes" section of the same dialog, you can configure the symbols and link them to a memory type. You can either create a symbol manually (assuming you know its "address" or NodeID), or simply drag and drop an existing symbol into the configuration area using the integrated Target Browser.



The symbol is then added with default configuration values. You can modify these configuration values by double-clicking on the symbol entry. Here you can also create the link to a memory type, which is referenced via its ID.



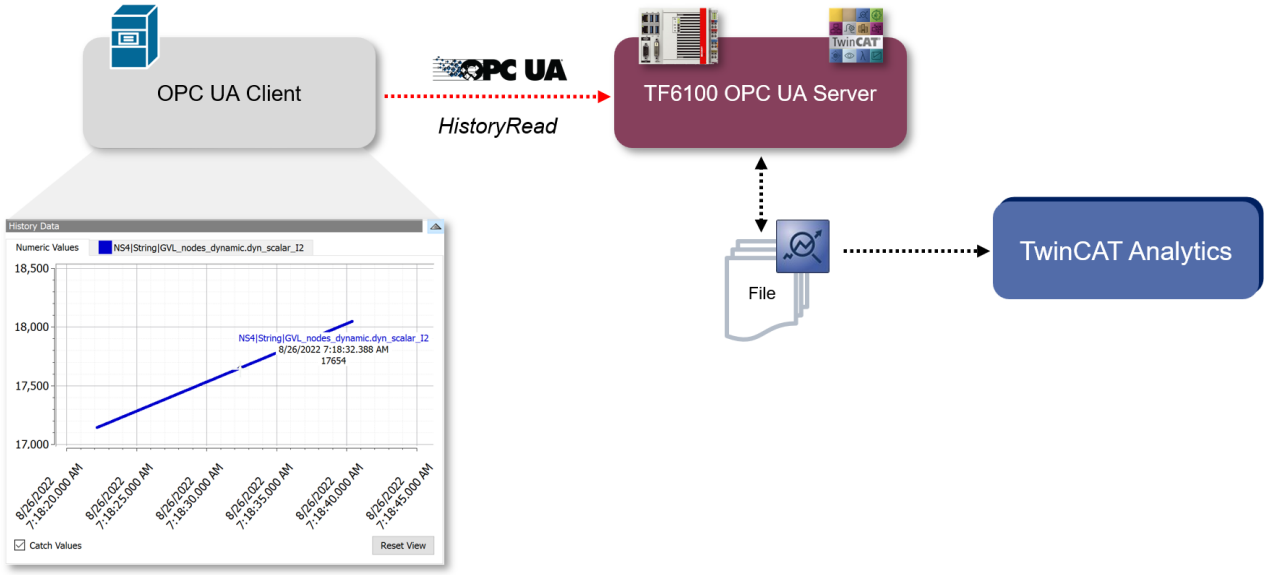
4.12.4 HistoryUpdate

With the HistoryUpdate function, the server does not independently sample variable values and save them for Historical Access. Instead, the server receives both the variable values and the timestamps from an OPC UA client.

A variable can be enabled for this function by setting the attribute `HistoryWriteable="true"` in the TwinCAT OPC UA Configurator. In this case, the server no longer performs its own sampling for this variable and "waits" for an OPC UA client to provide the values plus timestamp. Such a call can be made, for example, by the TwinCAT OPC UA Client via the function block UA_HistoryUpdate.

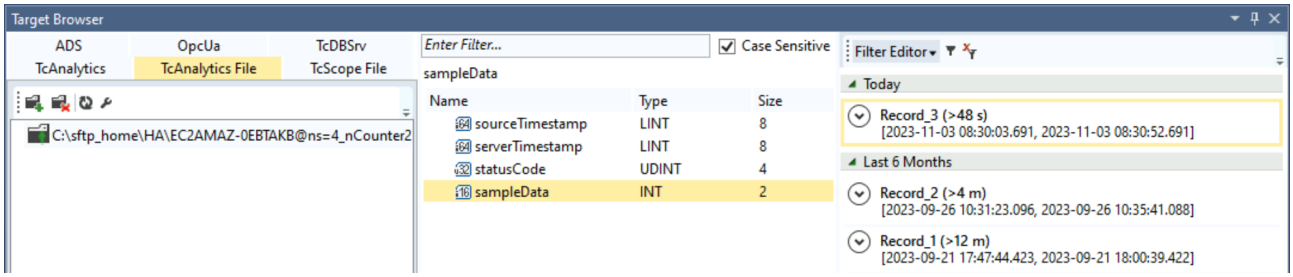
4.12.5 TwinCAT Analytics

When using the "TwinCAT Analytics" memory type, a binary file is written as memory, the format of which corresponds to the TwinCAT Analytics data format. Although this has no relevance for the OPC UA client (as it accesses the historical data via OPC UA), it does open up interesting possibilities for further use of the historical data, as you can read the binary file into the TwinCAT Analytics Toolchain and process it there. The following diagram illustrates this relationship.



TwinCAT Target Browser

If you use the TwinCAT Analytics storage format in your Historical Access configuration, you can also get the stored values via the TwinCAT Analytics Toolchain, for example the TwinCAT Target Browser. To do this, add the directory that you have defined in your configuration for saving the historical values in the **TcAnalytics File** tab of the TwinCAT Target Browser. The saved values are then displayed as records in the usual TwinCAT Analytics form and can be processed further.



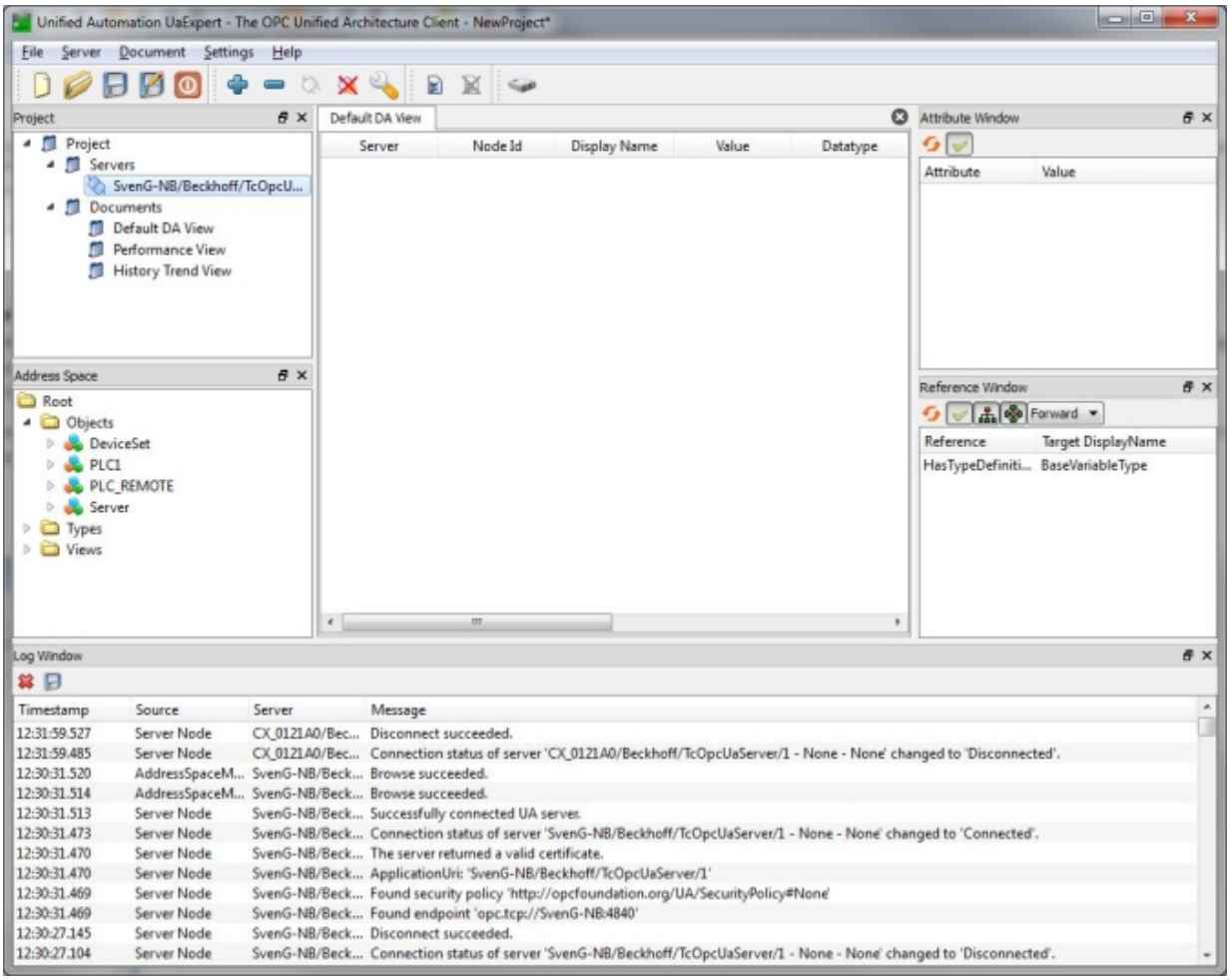
4.12.6 Access to historical data

You can use any OPC UA client that supports the OPC UA Historical Access function to get historical data. In the following example, the "UA Expert" software from Unified Automation is used to read historical data from the TwinCAT OPC UA Server and display it visually.

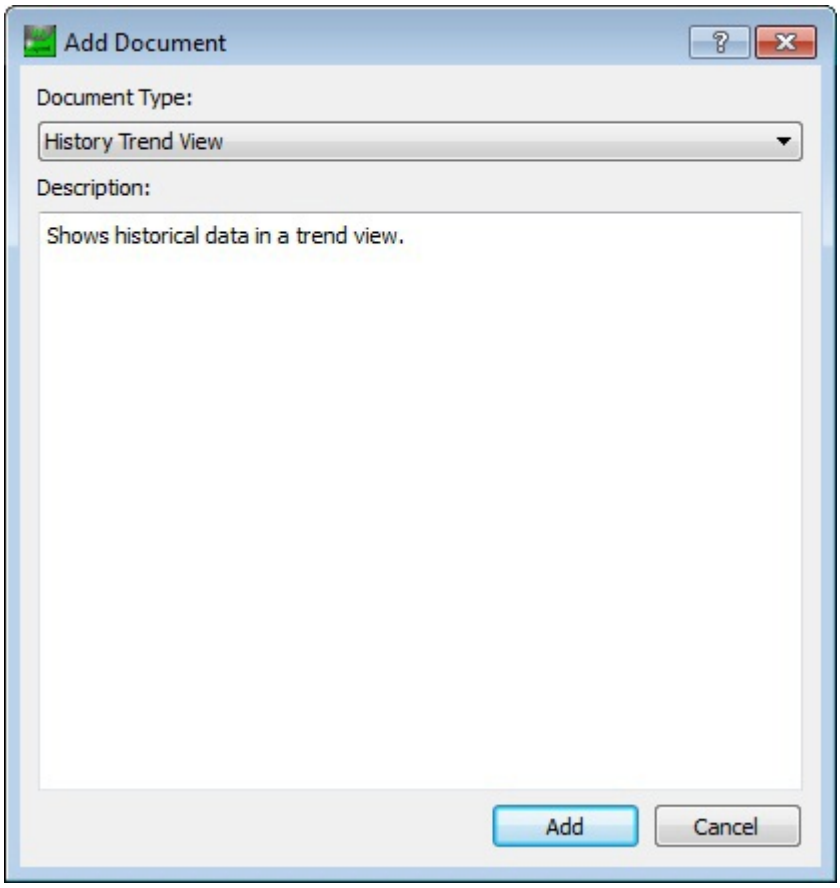
Displaying Historical Access values in an OPC UA client

The following step-by-step instructions describe how to configure the UA Expert software in order to access historical data.

1. Start the UA Expert software and connect to the OPC UA Server.

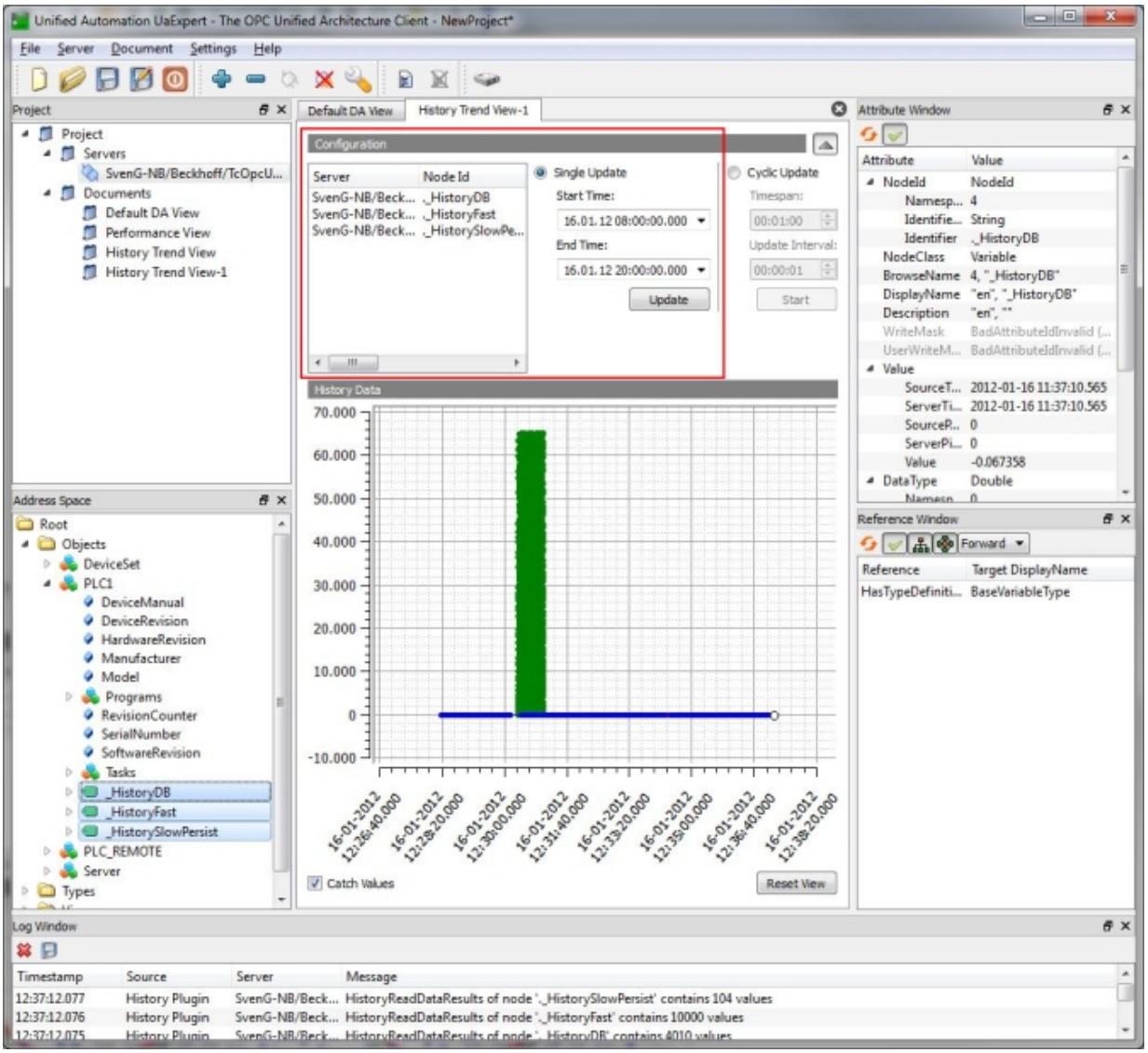


2. Add a new **History Trend View**.



3. Browse the PLC1 namespace and use drag & drop to add the PLC variables _HistoryDB, _HistoryDBcompact, _HistoryFast and _HistorySlowPersist.

⇨ You can now use the **Start Time** and **End Time** controls to specify the desired time period for which the symbol values are to be displayed, or you can start a **Cyclic Update** for these variables if necessary.



4.13 Alarms and Conditions

4.13.1 Overview

This chapter describes the configuration steps for using OPC UA **A**larms and **C**onditions (A&C) on the TwinCAT OPC UA Server. OPC UA A&C describes a model for monitoring symbol values and issuing alarms and events when values change or threshold values of a symbol are exceeded.

Requirements

This configuration can be carried out for variables from any runtime system [▶ 56] (e.g. TwinCAT PLC or TwinCAT 3 C++, ...). As a prerequisite, the respective variable must first be enabled for OPC UA. You can find out how to do this in our Quick Start [▶ 27] tutorial.

4.13.2 Supported functions

The following table lists the supported standard AlarmTypes of the Alarms & Conditions function of the TwinCAT OPC UA Server.

AlarmType	Description
LimitAlarmType	Allows you to define different threshold values for a symbol. If a threshold value is exceeded, the alarm is triggered with a configurable alarm text and severity level.
OffNormalAlarmType	Enables the definition of “normal” and “non-normal” threshold values for a symbol. If the value of the symbol deviates from the defined value, the alarm is triggered.

BOOL usage with OffNormalAlarmType

If a BOOL is to be used for the OffNormalAlarm type, the notation *true* and *false* must be used. If capitalized, the TwinCAT OPC UA Server assumes a STRING and no longer interprets the value as a BOOL.

In addition to the Alarms & Conditions function, the TwinCAT OPC UA Server also offers the option to connect to the TwinCAT EventLogger [▶ 117] and forwarding the alarms or events received there accordingly as an OPC UA Alarm or Event.

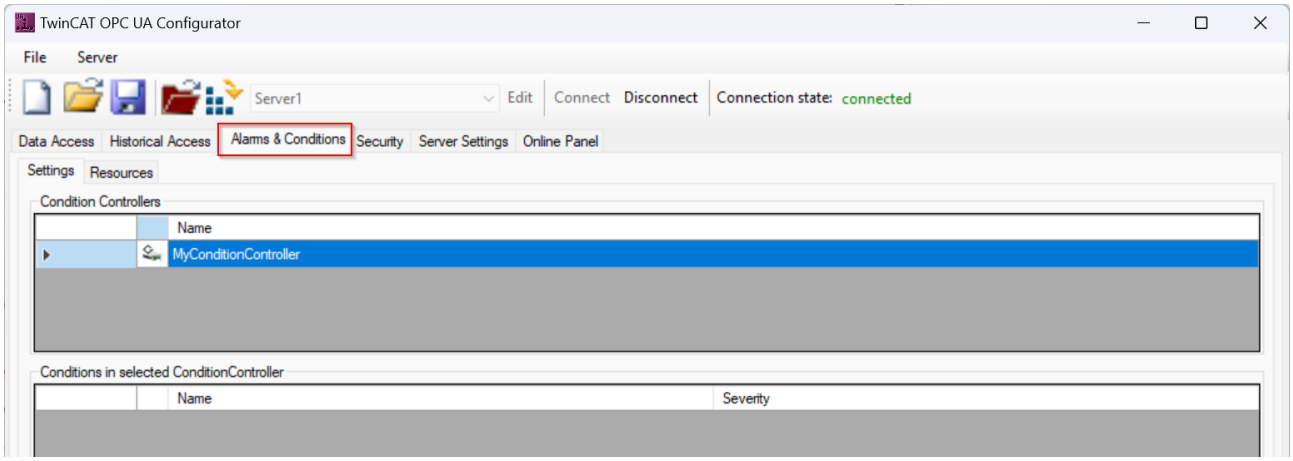
4.13.3 Configuration

You can use the TwinCAT OPC UA Configurator to configure the Alarms & Conditions functionality. To configure a symbol for Alarms & Conditions, you must perform two steps:

1. The symbol must have been enabled for Data Access (see chapter Enabling symbols [▶ 59]).
2. A ConditionController must be created that allows alarms and events to be grouped.
3. Alarm and event texts must be created.
4. Configuration parameters must be defined for the symbol.

Creating a ConditionController

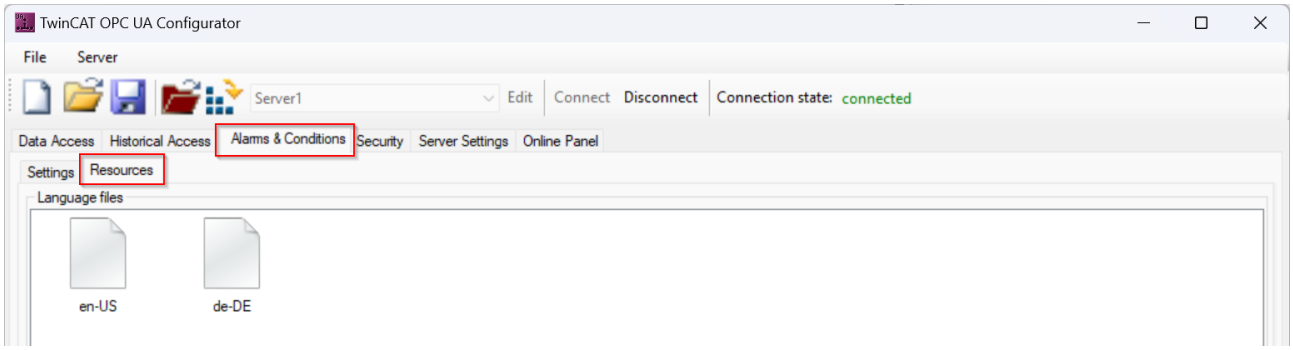
In the stand-alone version of the TwinCAT OPC UA Configurator, you can configure the ConditionController via the **Alarms & Conditions** tab. In the following screenshot, for example, a ConditionController with the name "MyConditionController" has been configured.



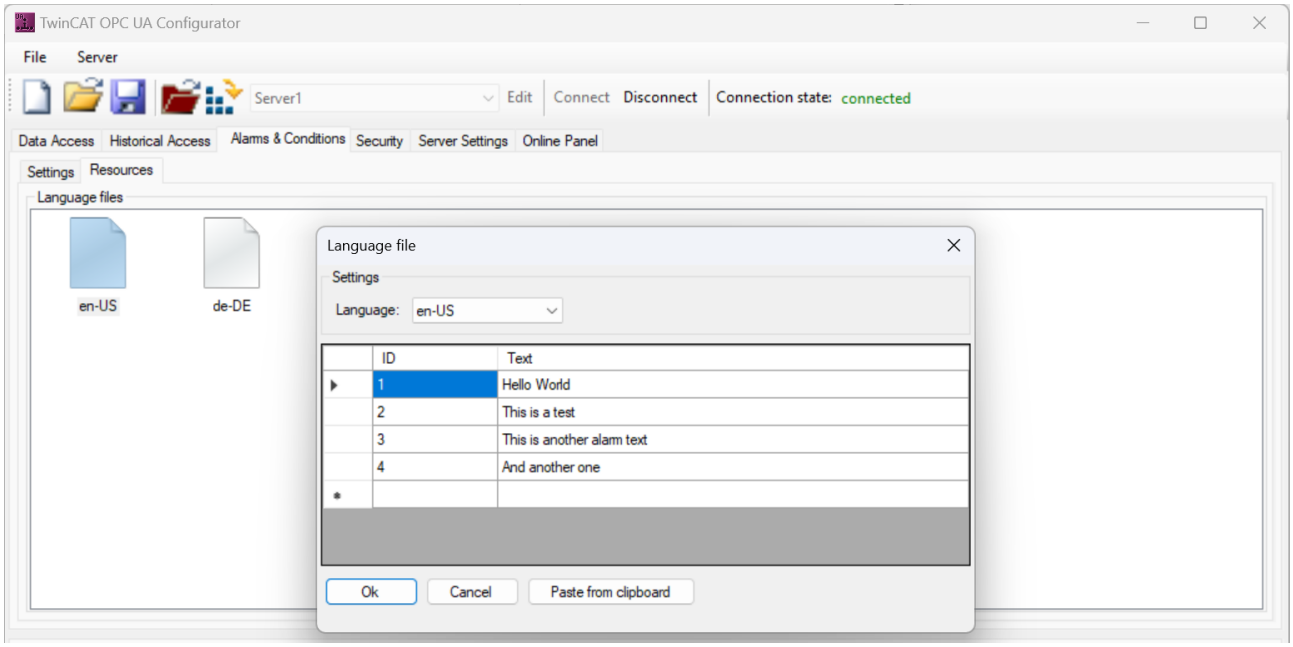
Additional ConditionControllers can be created or existing ConditionControllers can be edited or removed from the configuration via the context menu.

Creating alarm and event texts

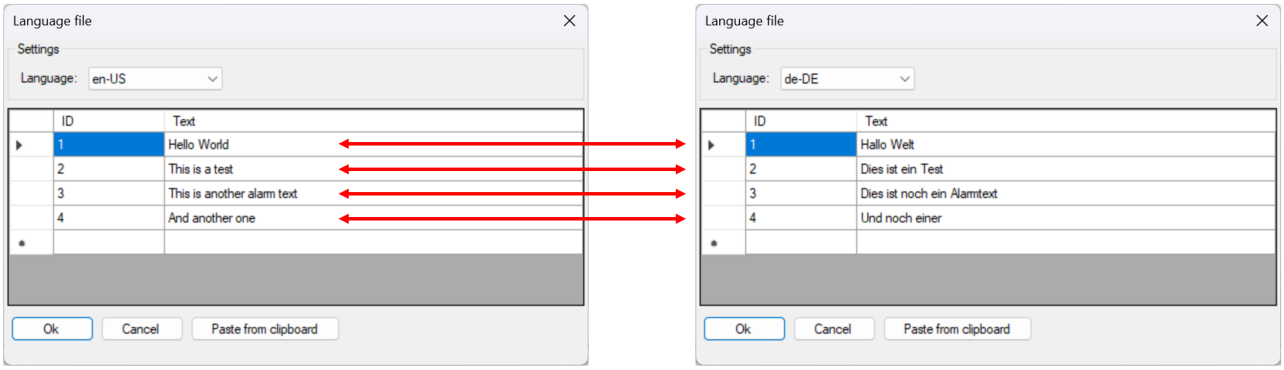
The texts that are to be used when reading an alarm or event can be stored as so-called "resources" in the TwinCAT OPC UA Server. These texts can be defined in several languages, whereby the language is identified by the respective country code. In the TwinCAT OPC UA Configurator, these resources are added via the **Resources** tab in the Alarms & Conditions configuration.



You can add, delete or edit other languages as a resource via the context menu. In the screenshot above, two languages have been configured as resources: de-DE (German/Germany) and en-US (English/USA). The texts are maintained in tabular form within a language and can also be copied and pasted from Microsoft Excel.



The configured texts are assigned across languages via their ID, for example:

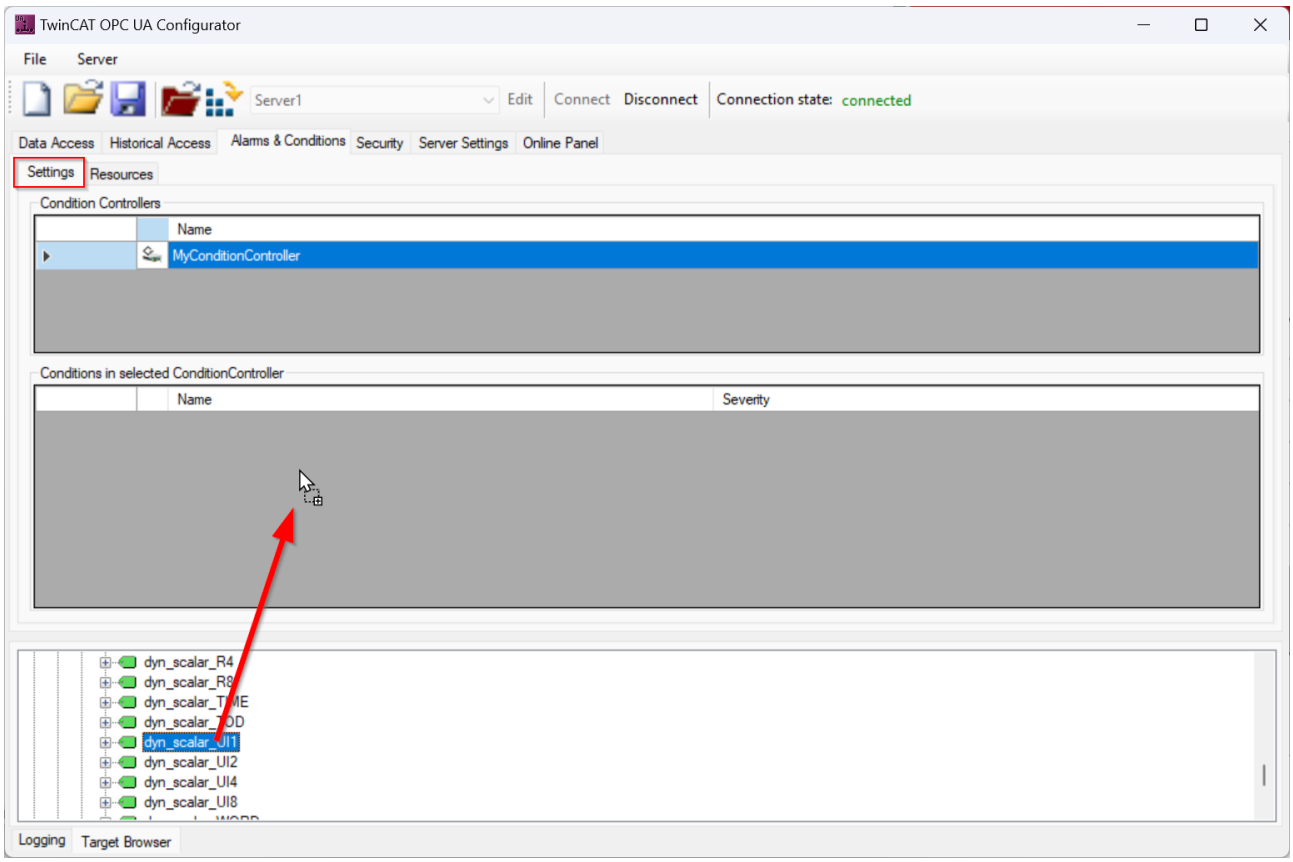


Use of the national language for an Alarm/Event

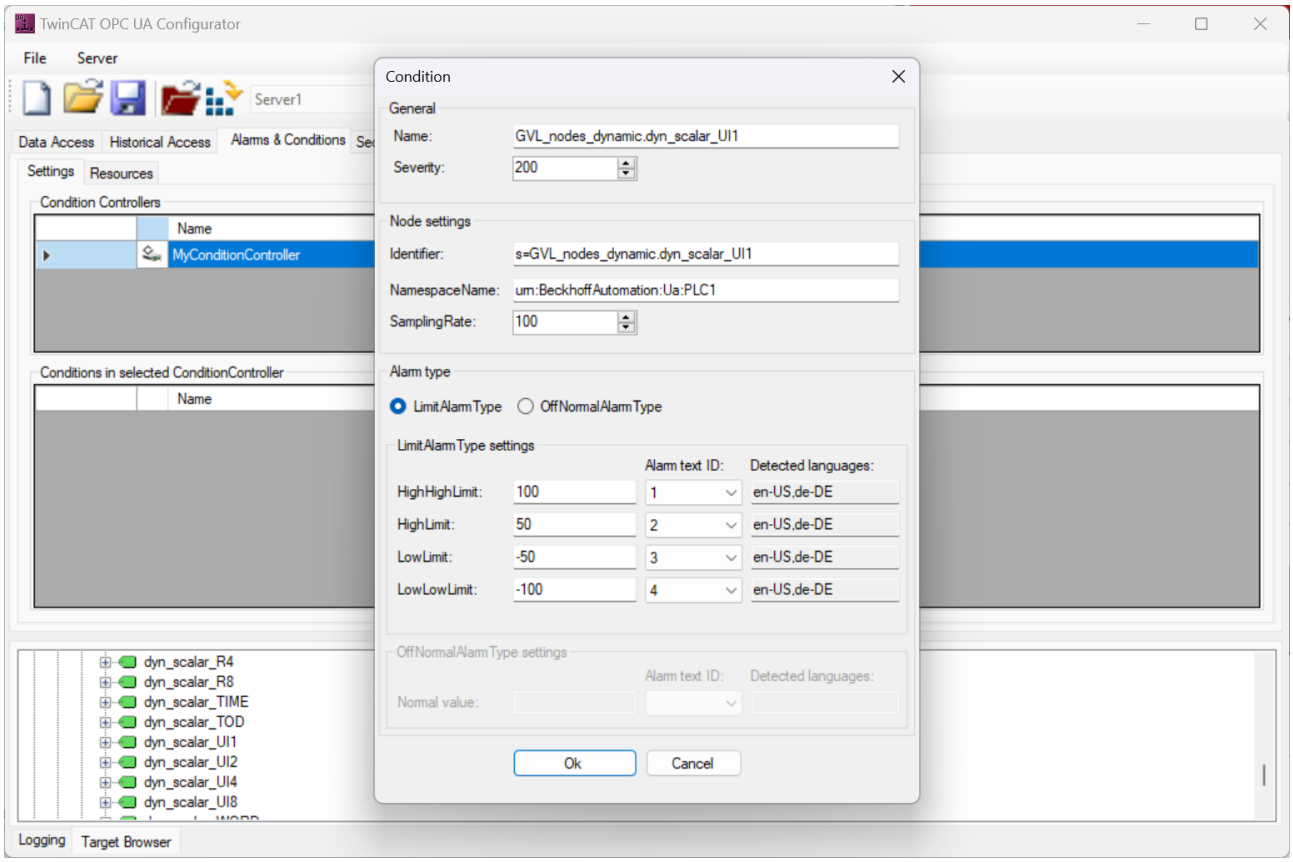
The language text used when triggering an alarm or event depends on the country code used to connect an OPC UA client to the server.

Configuring a symbol

Once the alarm and event texts have been configured, you can also use them to configure a symbol. The symbol configuration is carried out in the **Settings** tab. You can configure an icon manually or simply drag and drop it onto the configuration area using the integrated Target Browser.



The symbol is then added to the selected ConditionController. In the following configuration dialog, you can define the alarm type to be configured as well as the threshold values and also select the language texts.

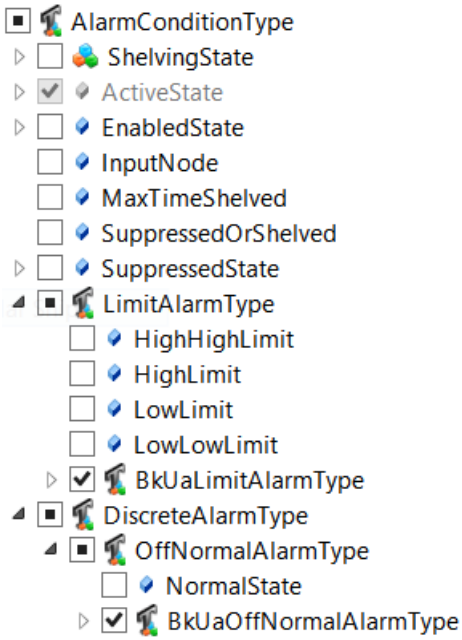


4.13.4 Additional application data

If you would like to attach additional data from your application to an alarm, you can configure this using the following special AlarmTypes:

AlarmType	Derived from
BkUaLimitAlarmType	LimitAlarmType
BkUaOffNormalAlarmType	OffNormalAlarmType

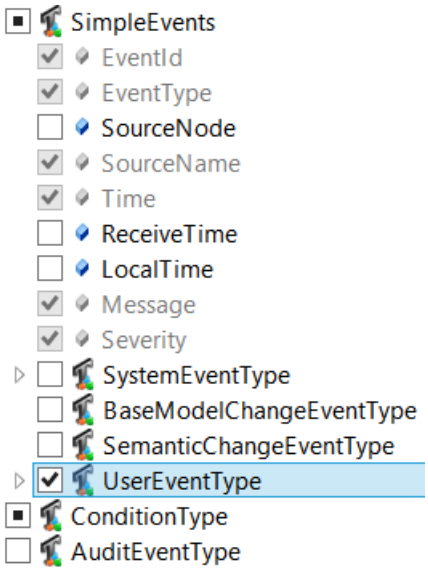
Additional fields are defined in these AlarmTypes, which you can fill with values from your application. If an OPC UA client is to be able to use these additional values, it must subscribe to and interpret the corresponding AlarmTypes.



The OPC UA client then receives the additional application data in the fields BkUaEventData and BkUaEventValue of the incoming alarm, for example:

ConditionId	NodeId
NamespaceIndex	5
IdentifierType	String
Identifier	A&C|ConditionController1.CustomStruct
5:BkUaEventData	NodeId
SourceTimestamp	19.10.2015 15:42:20.461
SourcePicoseconds	0
ServerTimestamp	19.10.2015 15:42:20.461
ServerPicoseconds	0
StatusCode	Good
Value	ST_SomeStruct
Data1	1
Data2	2
Data3	3
5:BkUaEventValue	NodeId
SourceTimestamp	19.10.2015 15:42:20.461
SourcePicoseconds	0
ServerTimestamp	19.10.2015 15:42:20.461
ServerPicoseconds	0
StatusCode	Good
Value	12

The user-defined EventFields are appended as "UserEventData". This data can be received by OPC UA clients that are logged on to the SimpleEventType "UserEventType".



To use this function, you must define a structure in the PLC that contains both the symbol value to be monitored and the additional values that are to be sent when the alarm is triggered. This structure must be defined as follows:

```
TYPE ST_CustomStruct :
STRUCT
  value : INT;
  data  : ST_SomeStruct;
END_STRUCT
END_TYPE

TYPE ST_SomeStruct :
STRUCT
```

```
  Data1 : INT;
  Data2 : REAL;
  Data3 : LREAL;
END_STRUCT
END_TYPE
```

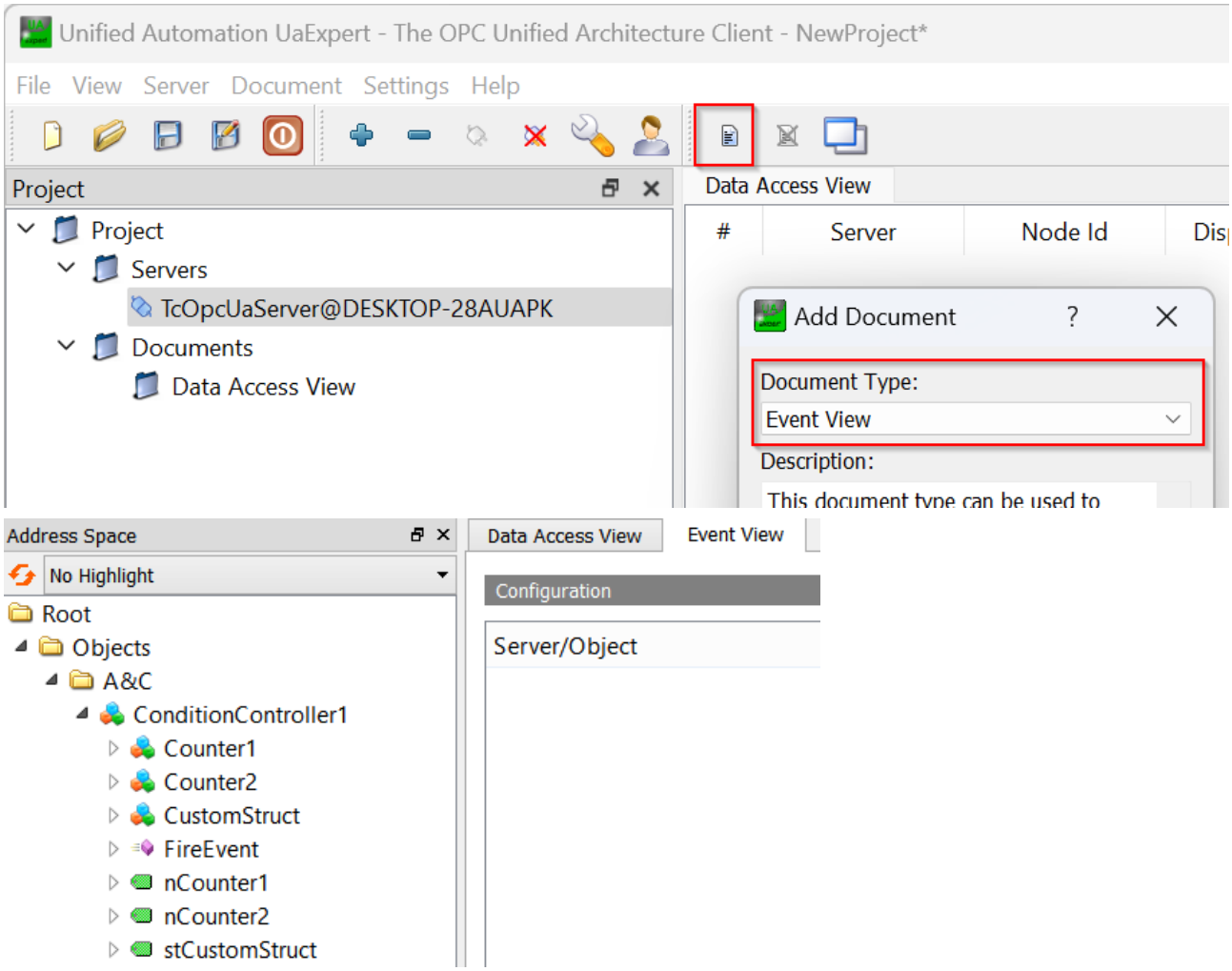
The instance of the structure ST_CustomStruct is then enabled as a symbol for Data Access [▶ 59]. The structure must also be enabled as a StructuredType [▶ 82].

4.13.5 Access to alarms and events

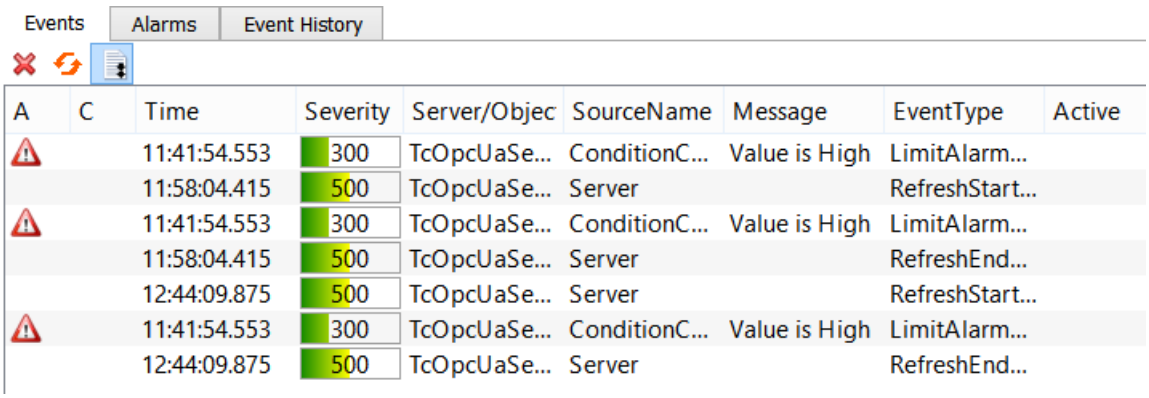
To receive alarms and events, an OPC UA client on the TwinCAT OPC UA Server must log in to one of the configured ConditionControllers. The client then receives the alarms and events for symbols that have been configured in this ConditionController.

The following example shows how you can use the UA Expert software from Unified Automation to subscribe to a ConditionController in order to receive alarms and events from it.

After you have started the UA Expert and established a connection to the TwinCAT OPC UA Server, add a new document view of the type "Event View" to your working area.



In the address space of the server, you will find all configured ConditionControllers below the "A&C" node. You can now drag and drop a ConditionController from the "Event View" working area to subscribe to it. The alarms and events for this ConditionController are then displayed in the Event Window.



4.14 Method calls

4.14.1 Overview

Two different concepts are available in the PLC for calling OPC UA methods:

1. RPC methods [▶ 115] and
2. Job methods [▶ 112]

The fundamental difference between the two concepts is the context of processing in the PLC and the implementation in the PLC code.
The following table illustrates this relationship.

Type	Processing	PLC code
RPC method	Within a PLC cycle	OPC UA method is represented by a PLC method.
Job method	Over several PLC cycles	OPC UA method is represented by a PLC function block.

Generally speaking, one can define: Job methods are the method of choice if a method call takes a "long" time, i.e. can take several PLC cycles. The processing of RPC methods, on the other hand, must be "fast", otherwise there is a risk of cycle overruns. **So Job methods are usually selected.** Programming in the PLC is somewhat more complex, but the decoupling from the PLC cycle is a great advantage that justifies this effort.

4.14.2 Job methods

The concept of Job methods has a fundamental difference compared to regular method calls: the OPC UA methods are no longer mapped 1:1 to a PLC method, but instead to a function block with a specific signature. This also allows method calls to be realized that take longer than one cycle from the perspective of a PLC application.

Requirements

This functionality is only available for Data Access [▶ 56] devices based on TwinCAT 3 and the import of TMC [▶ 59] symbol files, as well as the online symbolism.

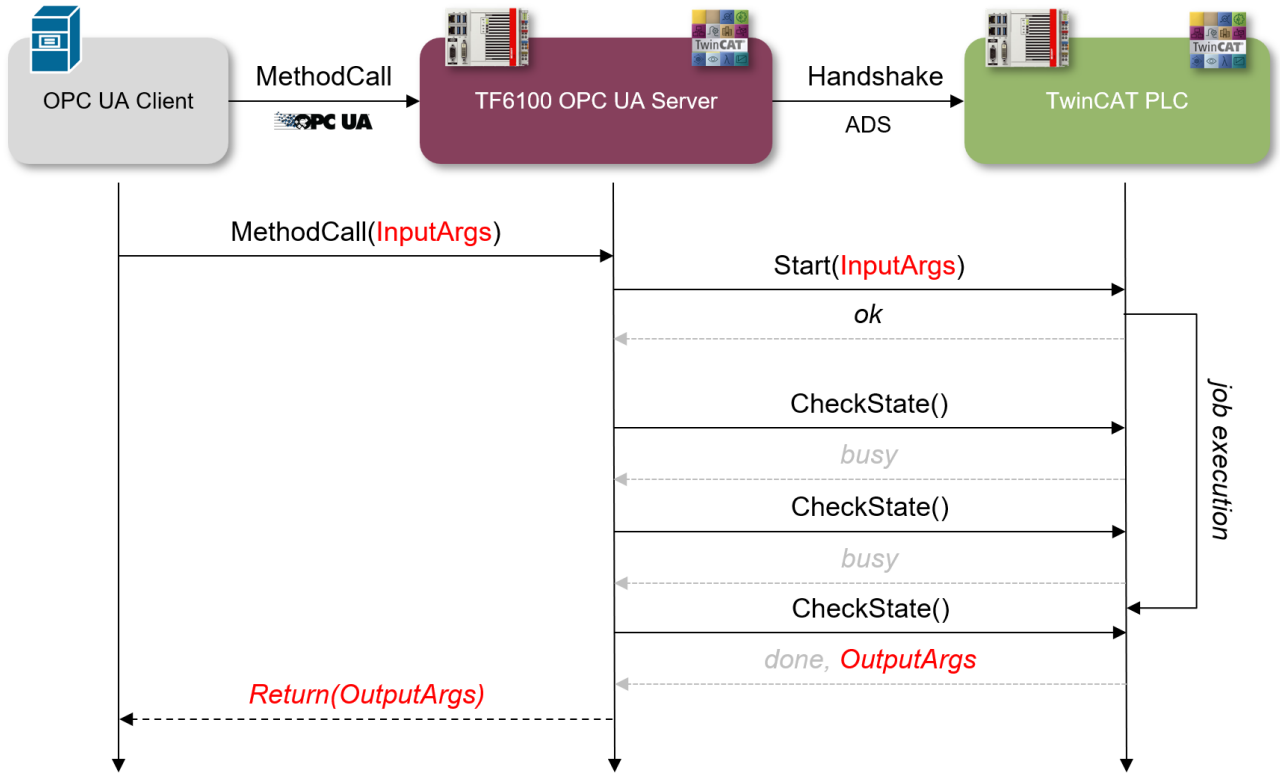
The PLC-side structure of such a Job method is defined as follows: there is a function block which is defined as a Job method via a PLC attribute. The function block then contains various PLC methods that are accessed by the TwinCAT OPC UA Server in the form of a handshake mechanism in order to be able to provide them as OPC UA methods.

Method	Description
Start	Is called by the server as soon as an OPC UA client calls the OPC UA method. Contains the input parameters of the OPC UA method call as VAR_INPUT.

Method	Description
	The HRESULT return value of the method can be used to directly return an OPC UA Status Code in its decimal representation, e.g. “0” for the Status Code “Good”. The numerical value of a Status Code defined in the OPC UA specification is used as the value. A definition of all available Status Codes can be viewed here: http://www.opcfoundation.org/UA/schemas/StatusCode.csv Typically, this method returns the value “0” (Good). However, the PLC developer can also decide to validate the input parameters, for example. In the event of an error, the OPC UA method call could then fail, e.g. with the return value “2158690304□” (BadInvalidArguments).
CheckState	Is called cyclically by the server to check whether the job is still being processed or not. As long as the job is still being processed, this method returns the value “Busy”, otherwise “Done”. Output parameters for the OPC UA method are declared here as VAR_OUTPUT. The HRESULT return value of the method can be used to directly return an OPC UA Status Code in its decimal representation, e.g. “0” for the Status Code “Good”. The numerical value of a Status Code defined in the OPC UA specification is used as the value. A definition of all available Status Codes can be viewed here: http://www.opcfoundation.org/UA/schemas/StatusCode.csv If the job is processed successfully, this method typically returns the value “0” (Status Code “Good”). However, the PLC developer can also decide that the OPC UA method call should fail in the event of an error, e.g. with the return value “2151415808” (BadOutOfRange) or the more general “2147483648” (Bad).
Abort	This method is called by the server if a job has to be aborted, for example if the server is shut down or restarted. The PLC developer then has the opportunity to clean up his PLC code accordingly.

Workflow

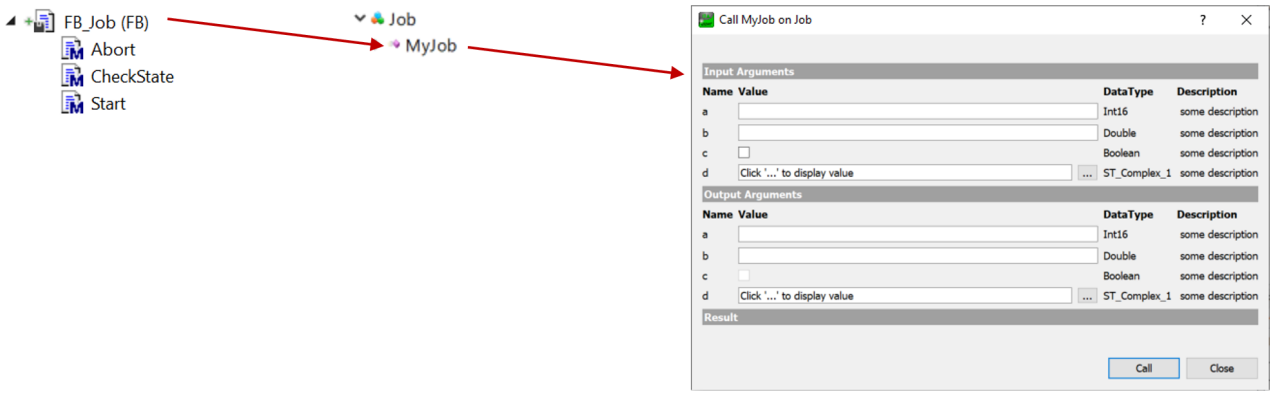
The handshake mechanism between server and PLC can be represented in simplified form as follows:



Example

The following example is also available in executable form at Examples. This part of the documentation is intended to explain the basic concepts of how the system works. The example “simulates” a Job method call that takes about four seconds to be processed in the PLC. This was realized with the help of a timer, which was declared and used in the function block part of the job. The State Machine of the function block delays the completion of the job (the CheckState handshake method returns “busy”) and the job is not completed until the timer has expired (the CheckState handshake method returns “done”).

In this example, a function block called FB_Job was created for the OPC UA method with the name “MyJob”, which has the required signature mentioned above.



The function block declaration contains the attribute OPC.UA.DA.JobMethod and the name of the OPC UA method to be used as its value.

```
{attribute 'OPC.UA.DA.JobMethod' := 'MyJob'}
FUNCTION_BLOCK FB_Job
VAR
END_VAR
```

The three methods Abort, CheckState and Start contain the attribute TcRpcEnable in their declaration (so that the methods can also be called via ADS). Example:

```
{attribute 'TcRpcEnable' := '1'}
METHOD Start : HRESULT

{attribute 'TcRpcEnable' := '1'}
METHOD Abort : HRESULT

{attribute 'TcRpcEnable' := '1'}
METHOD CheckState : HRESULT
```

The input parameters of the OPC UA method are declared as VAR_INPUT in the PLC method Start(). Comments after the variables are used as a description of the respective OPC UA input parameter. Example:

```
{attribute 'TcRpcEnable' := '1'}
METHOD Start : HRESULT
VAR_INPUT
  a : INT; // some description
  b : LREAL; // some description
  c : BOOL; // some description
  d : ST_Complex_1; // some description
END_VAR
VAR_OUTPUT
  hdl : UDINT; // handle, can be used for concurrent calls
END_VAR
```

The output parameters of the OPC UA method are declared as VAR_OUTPUT in the PLC method CheckState(). Comments after the variables are used as a description of the respective OPC UA output parameter. Example:

```
{attribute 'TcRpcEnable' := '1'}
METHOD CheckState : HRESULT
VAR_INPUT
  hdl : UDINT; // handle, can be used for concurrent calls
END_VAR
VAR_OUTPUT
  bBusy : BOOL; // required
```

```
  a : INT; // some description
  b : LREAL; // some description
  c : BOOL; // some description
  d : ST_Complex_1; // some description
END_VAR
```

(In this example, the values of the input parameters are applied 1:1 to the output parameters for illustrative purposes. The input and output parameters are therefore identical.)

Simultaneous method calls

Simultaneous method calls can be managed via the PLC application. For this purpose, a handle (output variable hdl) can be returned to the server via a return value of the Start() method, which is then used as an input variable for all further calls to the CheckState() method. This enables the PLC application to determine an execution context for the respective method call. In summary, the workflow is as follows:

- The Start() method determines the execution context and returns a handle to the server via the output variable “hdl”.
- The server now uses this handle for the cyclic CheckState() calls and lets the PLC application know which execution context is involved.

4.14.3 RPC methods

Method calls are a fundamental part of the OPC UA specification. With the introduction of these functionalities into the PLC world, TwinCAT 3 offers the possibility of efficiently executing RPC calls in the IEC61131 world and thus reduces the classic handshake patterns for communication between devices.

With the concept of RPC methods, the TwinCAT OPC UA Server imports a method from the TwinCAT PLC/C++ directly as an OPC UA method.

Requirements

This functionality is only available for Data Access [▶ 56] devices based on TwinCAT 3 and the import of TMC and TMI [▶ 59] symbol files, as well as the online symbolism.

An RPC method call is handled at OPC UA level as follows:

- If an RPC method has been executed successfully, the server returns the status code "Good" as feedback for the OPC UA method call.
- If the RPC method could not be called, the server returns an error message in the "Bad_***" format, depending on the error that occurred.
- If the RPC method was called successfully but the response could not be read in the server, the status code "OpcUa_GoodPostActionFailed" is returned from the server.

Processing context

The TwinCAT PLC/C++ method is executed within the real-time context and therefore falls within the processing context of a real-time task. The TwinCAT developer must therefore take precautions so that the execution time of the method matches the task cycle time.

Methods in TwinCAT 3 PLC

Methods in the IEC61131 world are always configured below a function block. At high-level language level, the function block can be regarded as the surrounding class of the method. You have to declare the method itself with a special PLC attribute so that the TwinCAT system knows that the method is to be activated for a remote method call.

```
{attribute 'TcRpcEnable':='1'}
METHOD M_Sum : INT
VAR_INPUT
  a : INT;
  b : INT;
END_VAR
```

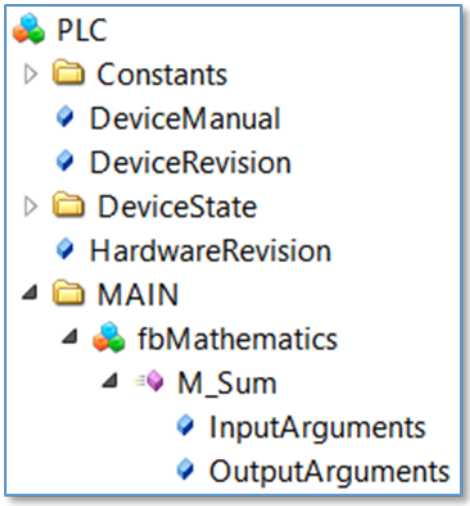
Depending on the import mode used, you also have to enable the surrounding function block for the OPC UA access. This can be done by using the normal PLC attributes for OPC UA access.

Note that you only need to use the PLC attributes if the filtered mode is used to enable symbols from the PLC to be accessed via OPC UA. This is the default setting of the OPC UA Server.

Sample:

The method M_Sum is located in the function block FB_Mathematics. The declaration of the function block instance uses the PLC attribute that has enabled the function block and thus the method for OPC UA access.

```
PROGRAM MAIN
VAR
  {attribute 'OPC.UA.DA':='1'}
  fbMathematics : FB_Mathematics;
END_VAR
```



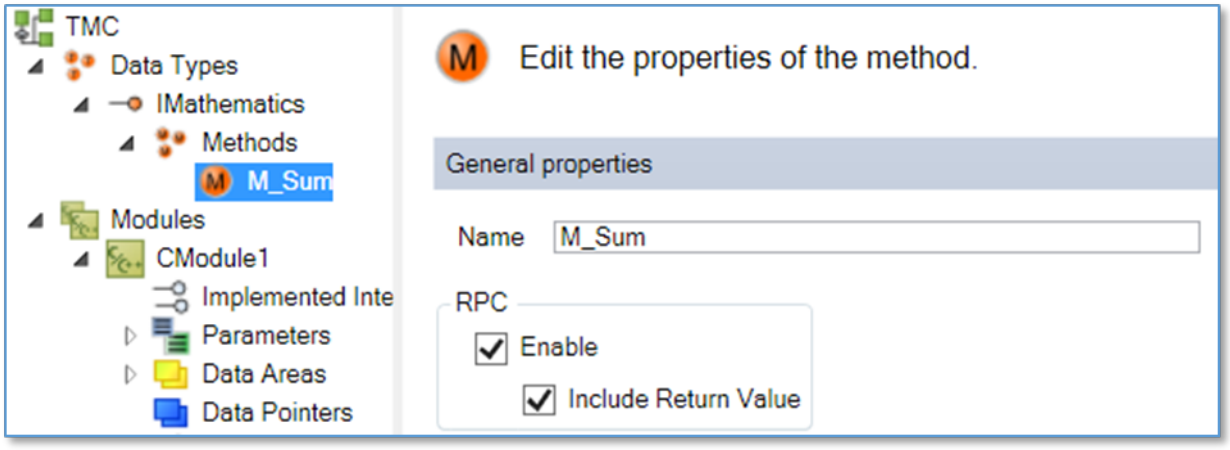
Pointer variables as VAR_IN_OUT

Pointer variables defined as VAR_IN_OUT are not handled by the PLC or the TwinCAT OPC UA Server. A corresponding Trace event is written to the server trace.

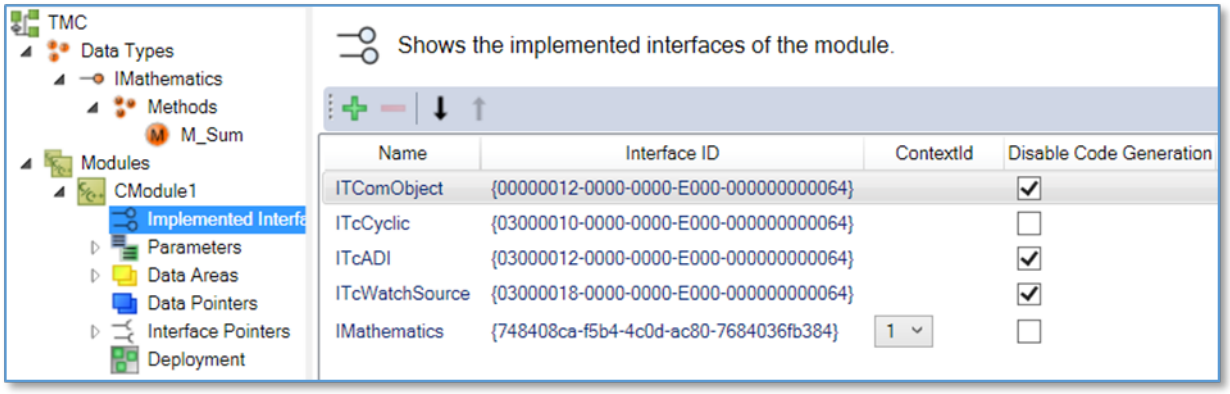
08:47:37.677Z|1|11A0* Error when importing method 'METH_PArray': VAR_IN_OUT pointer variables are not allowed!

Methods in TwinCAT 3 C++

TwinCAT modules could implement interfaces with predefined methods (see TcCOM modules). The method itself must be enabled for RPC calls during its definition (see TwinCAT Module Class Wizard documentation) so that the OPC UA Server knows that it is ready for execution.

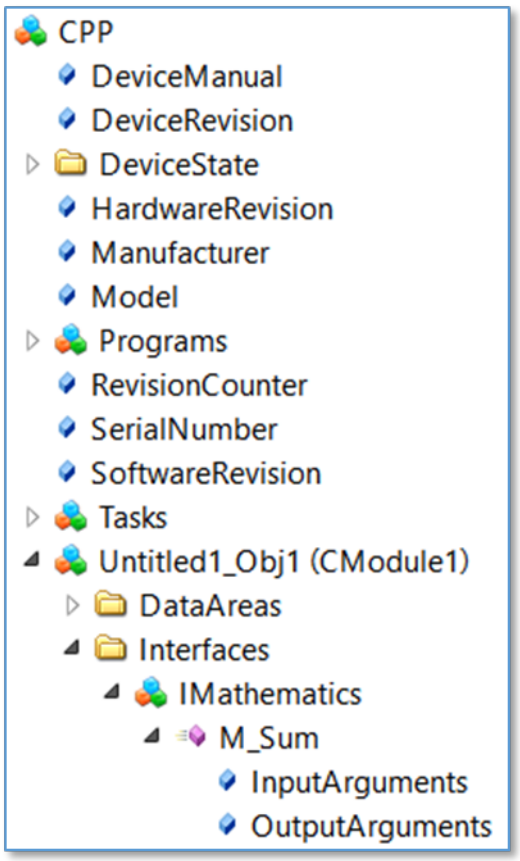


So that the return value of the method is available, the corresponding option **Include Return Value** must be activated. Note that the interface under which the method was created must be implemented.



Depending on the import mode used, you have to declare the method for the access via OPC UA. This can be done by using the TMC Code editor and the usual OPC UA attributes as optional properties.





4.15 TwinCAT EventLogger

4.15.1 Overview

The TwinCAT OPC UA Server enables integration of the TwinCAT EventLogger for sending alarms and events. An alarm/event is converted by the TwinCAT EventLogger into an OPC UA alarm or event. EventLogger messages from more than one TwinCAT device can be configured in the TwinCAT OPC UA Server. Each device is identified by its AMS Net ID.

The integration of the TwinCAT EventLogger is not available for the TwinCAT OPC UA Server under Windows CE. However, you can install the TwinCAT OPC UA Server on a non-CE device and then remotely receive TwinCAT EventLogger messages from a Windows CE device.

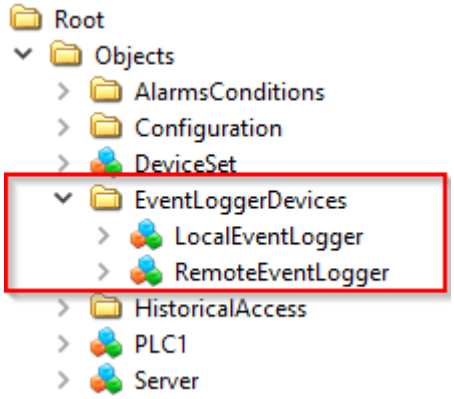
The integration of the TwinCAT EventLogger is only available in connection with the TwinCAT 3 EventLogger. The TwinCAT 2 EventLogger is not supported.

4.15.2 Configuration

To configure the server for a connection to the EventLogger, you have to create a new configuration file named "TcUaEventLogConfig.xml" in the server directory. This file contains a list of TwinCAT EventLogger devices, which are identified by their AMS NetID. The configuration has the following structure:

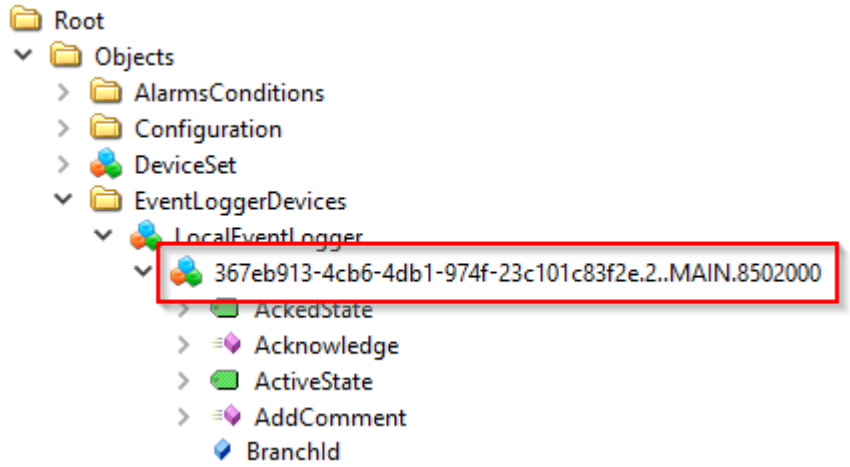
```
<TcUaEventLogConfig>
  <EventLoggerDevice Name="LocalEventLogger" AmsAddr="127.0.0.1.1.1"/>
  <EventLoggerDevice Name="RemoteEventLogger" AmsAddr="192.168.0.56.1.1"/>
</TcUaEventLogConfig>
```

The individual device entries are then displayed in the "EventLoggerDevices" folder in the server's namespace.



An OPC UA client can now subscribe to the corresponding object in order to receive events and/or alarms from the respective TwinCAT EventLogger device.

In contrast to an event, an alarm is additionally displayed as a child element by the respective EventLogger device.

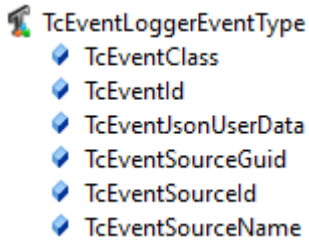


4.15.3 Access to alarms and events

When subscribing to the object and receiving alarms or events, a client must take into account that special object types are used. The respective types are defined as follows:

TcEventLoggerEventType

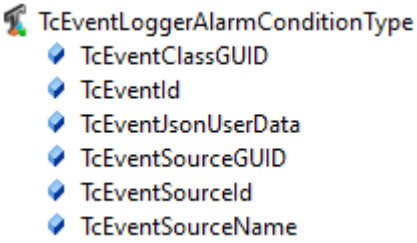
The TcEventLoggerEventType is derived from the BaseEventType and extends it with TwinCAT EventLogger-specific properties:




```
NodeID: i=4200
NamespaceName: urn:BeckhoffAutomation:Ua:Types:GlobalTypes
```

TcEventLoggerAlarmConditionType

The TcEventLoggerAlarmConditionType is derived from the AlarmConditionType and extends it with TwinCAT EventLogger-specific properties:




```
NodeID: i=4000
NamespaceName: urn:BeckhoffAutomation:Ua:Types:GlobalTypes
```

Sample

The following sample is based on the standard code sample of the TwinCAT EventLogger, which can be obtained from the Beckhoff Information System. This sample contains code snippets for the PLC, which can be used to fire an event as well as an alarm.

Step 1: Configuration of the server

TwinCAT OPC UA Server and the PLC are now running locally on the same system in this sample. Accordingly, the TcEventLogConfig.xml was also created:

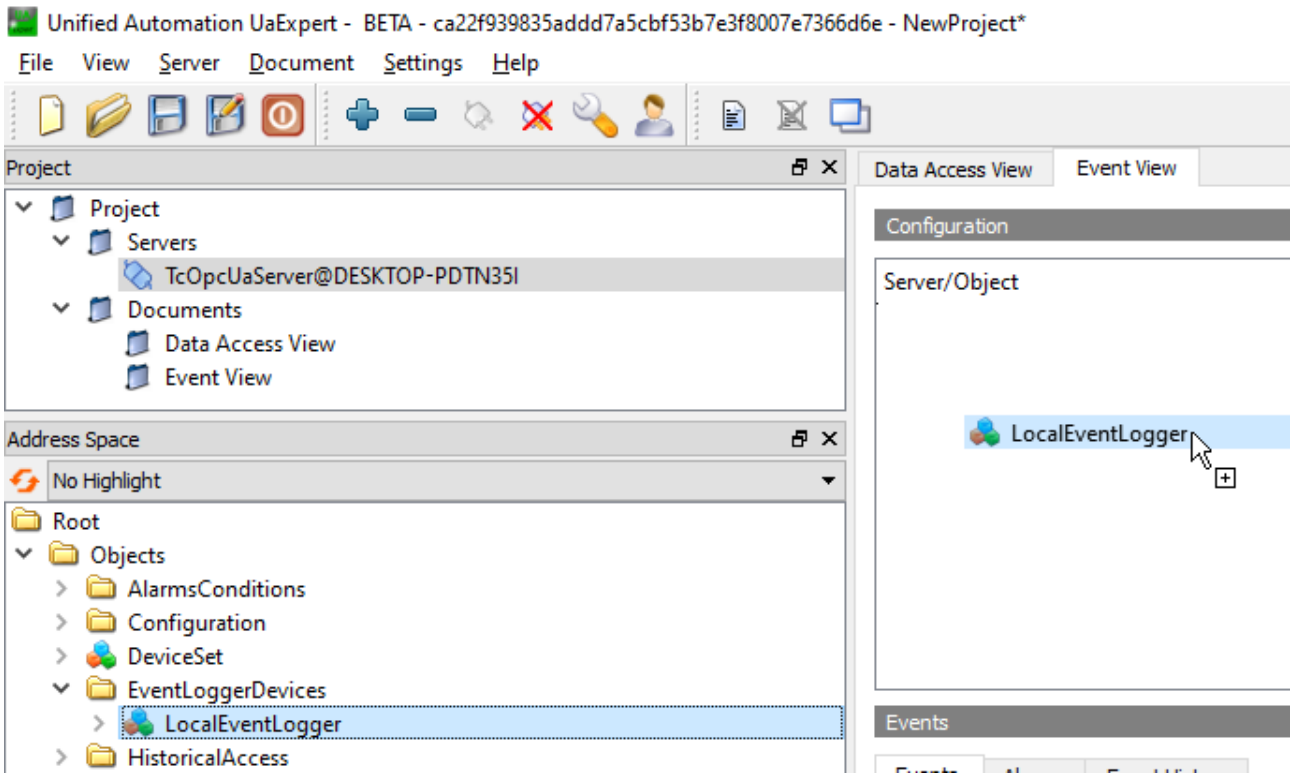
```
<TcUaEventLogConfig>
  <EventLoggerDevice Name="LocalEventLogger" AmsAddr="127.0.0.1.1.1"/>
</TcUaEventLogConfig>
```

Step 2: Activating the TwinCAT EventLogger sample

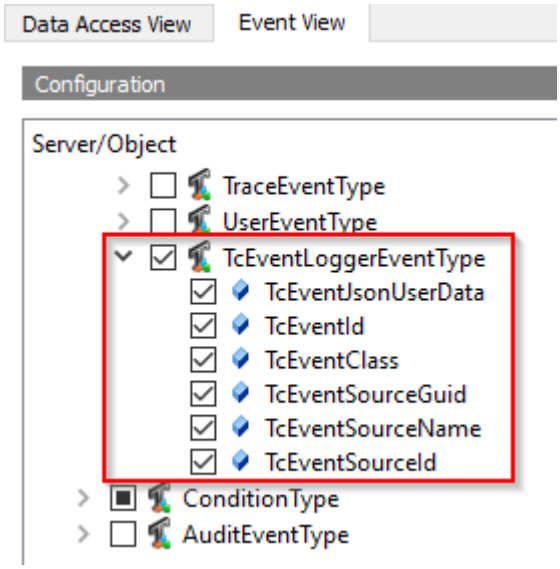
Before enabling the TwinCAT EventLogger sample, we first disabled the automatic firing of an event or alarm. In the present sample version this is done by initializing bSend and bAlmRaise with the value FALSE. The project is then activated and executed in the local PLC runtime.

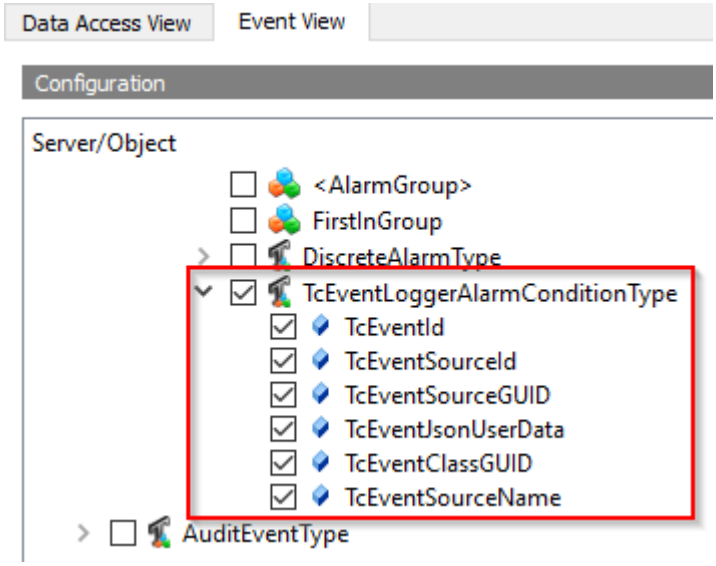
Step 3: Connecting an OPC UA Client to the Server

The UA Expert was selected as the OPC UA client. After establishing a connection with the server, you now add a new "Event View" document in UA Expert. Then drag and drop the configured TwinCAT EventLogger device into the Event View.



In order to receive the TwinCAT EventLogger-specific properties, the UA Expert must filter the corresponding Event or AlarmType. You can find the TcEventLoggerEventType or TcEventLoggerAlarmConditionType in the type list of the Event View. Sample:

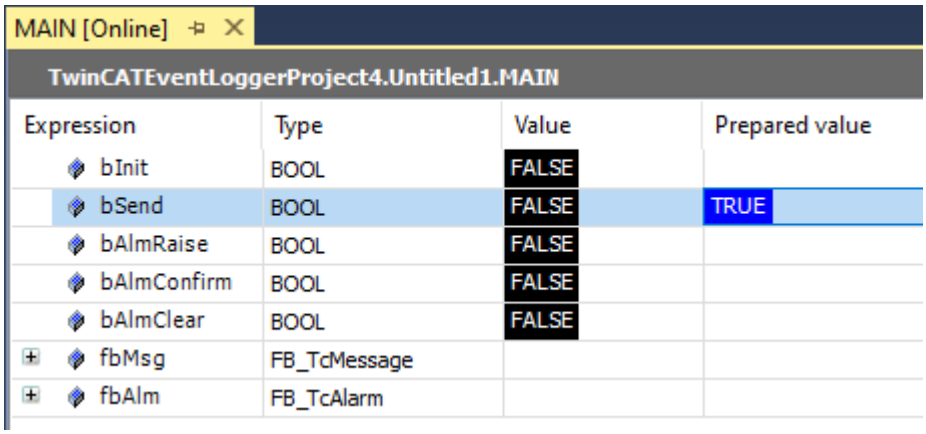


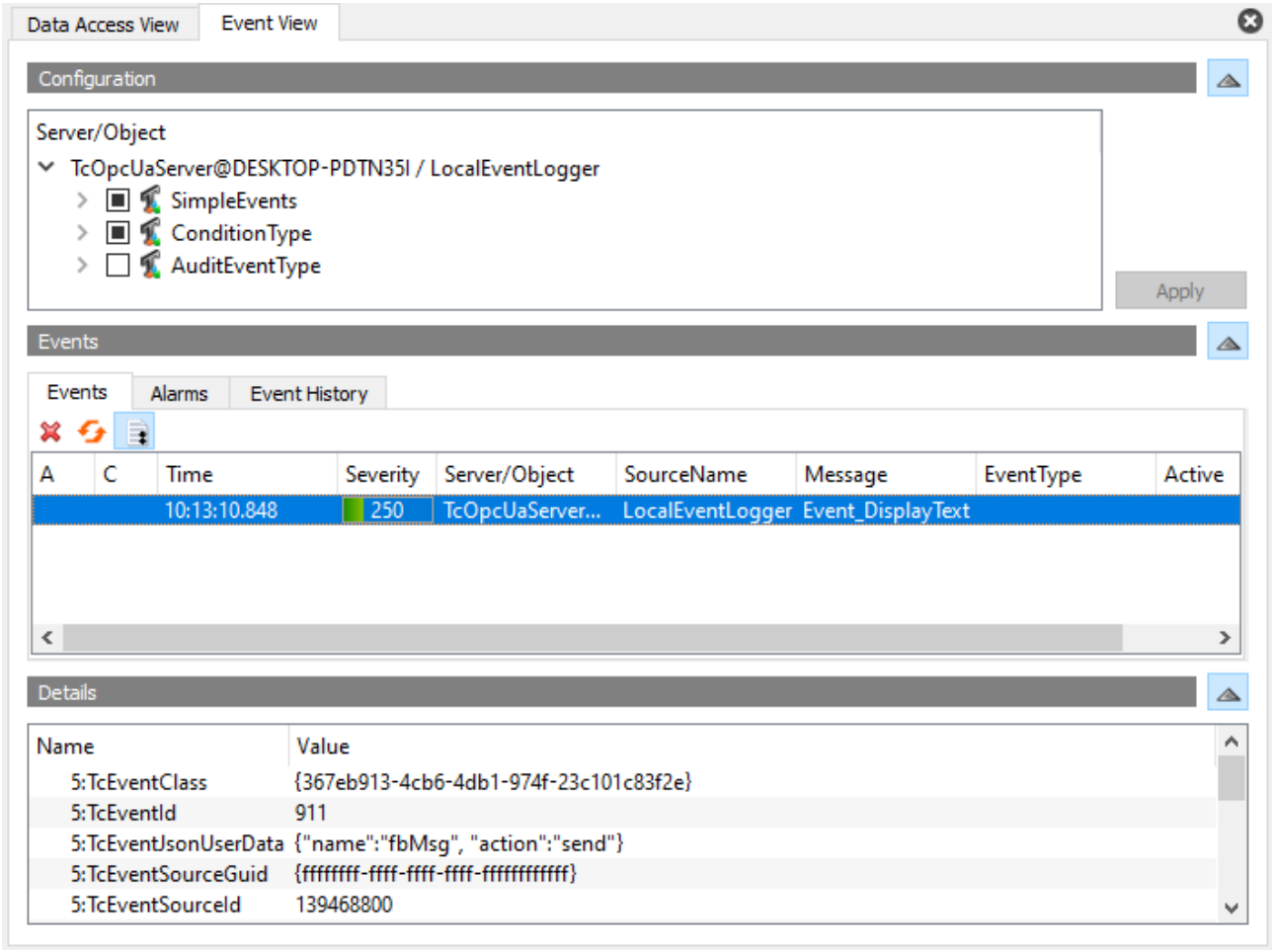


By clicking on the **Apply** button, these filters are applied.

Step 4: Firing an event

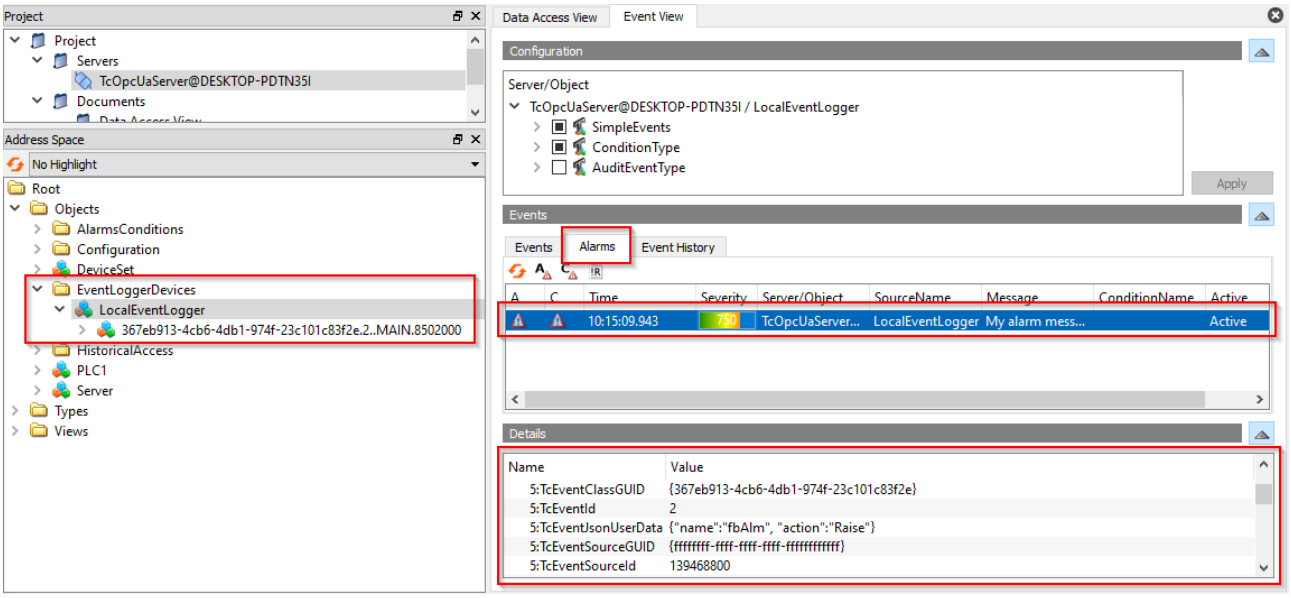
In the TwinCAT EventLogger Sample, we set the variable bSend to the value TRUE in order to fire an event. The UA Expert receives this event automatically and displays it in the Event View together with the TwinCAT EventLogger-specific properties.





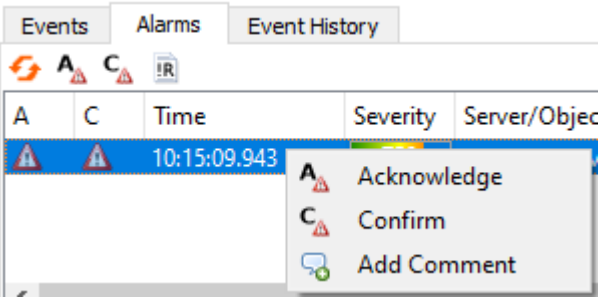
Step 5: Firing an alarm

In the TwinCAT EventLogger sample, we set the variable bAlmRaise to the value TRUE in order to fire an alarm. The UA Expert receives this alarm automatically and displays it in the Event View together with the TwinCAT EventLogger-specific properties. In addition, the alarm is displayed as a separate object in the namespace below the EventLogger device.



Step 6: Acknowledging an alarm

In addition to receiving an alarm, it can also be acknowledged. The acknowledgement is also reported by the server to the TwinCAT EventLogger. In UA Expert, an alarm can be acknowledged via the context menu in the Event View.



Acknowledge and Confirm

Please note that only a Confirm is reported back to the TwinCAT EventLogger. The information about an acknowledge remains in the server, since the TwinCAT EventLogger currently only provides for the concept of a Confirm.

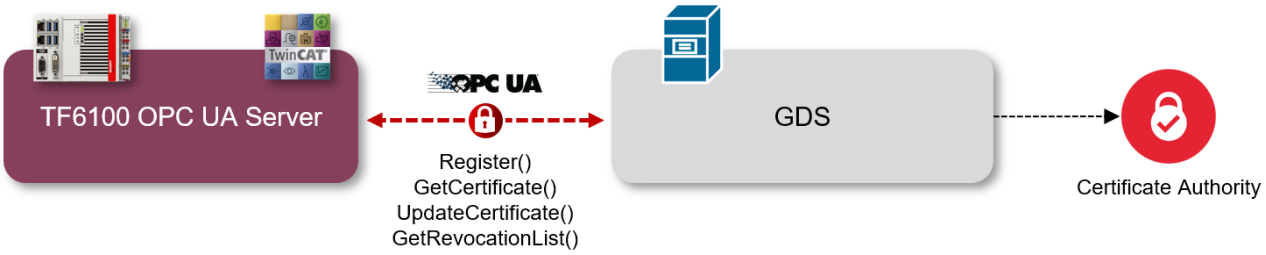
After a Confirm the corresponding variable eConfirmationState is set in the TwinCAT EventLogger instance of type FB_TcAlarm.

bRaised	BOOL	TRUE
eConfirmationState	TCEVENTCONFIRMA...	Confirmed

4.16 Global Discovery Server

4.16.1 Overview

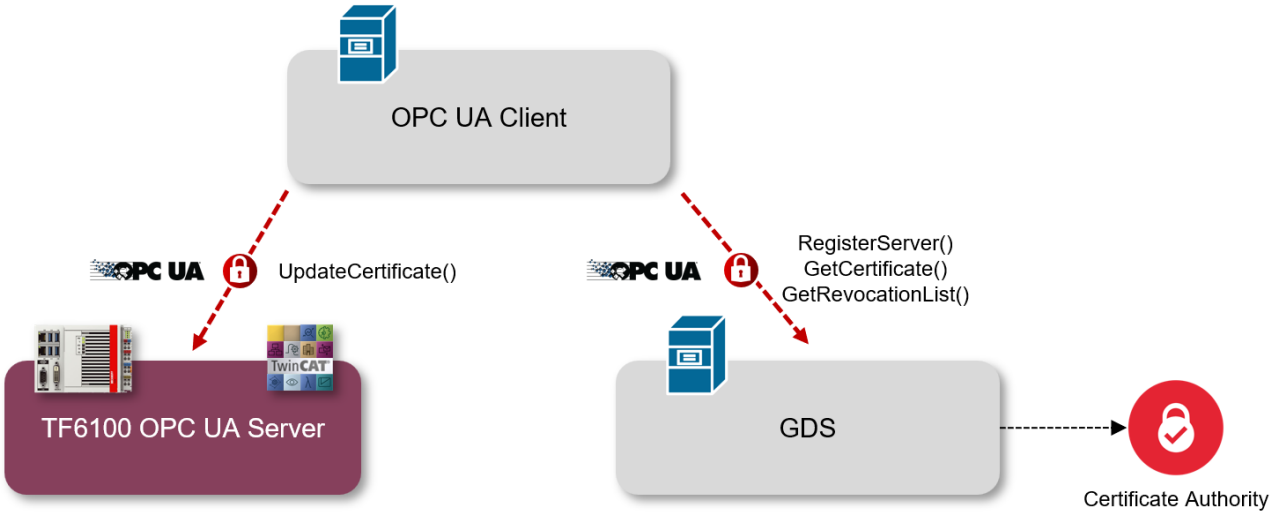
The TwinCAT OPC UA Server enables registration with a **G**lobal **D**iscovery **S**erver (GDS). A GDS provides a standardized, OPC UA-based interface for registering OPC UA applications and issuing application certificates as well as the associated revocation list information. A GDS has a connection to a **C**ertificate **A**uthority (CA) for this purpose. Applications are registered and certificates are issued (and updated) using a standardized OPC UA information model.



When accessing the GDS, the TwinCAT OPC UA Server supports the two models Push [▶ 123] and Pull [▶ 124].

4.16.2 Push

In this model, the server includes a standardized interface that an OPC UA client (*that supports the push model*) can use to connect to a Global Discovery Server on behalf of the server, register the server application there and request a server certificate including the current **C**ertificate **R**evocation **L**ist (CRL).



As a prerequisite for using this functionality, the OPC UA client must authenticate itself on the server with a user account that has administrator rights.

OPC UA Client

Any client that supports the Push model can be used as an OPC UA client. Various OPC UA toolkit manufacturers offer corresponding software packages. Alternatively, the OPC Foundation also provides a GDS Sample Client on Github.

GDS Server

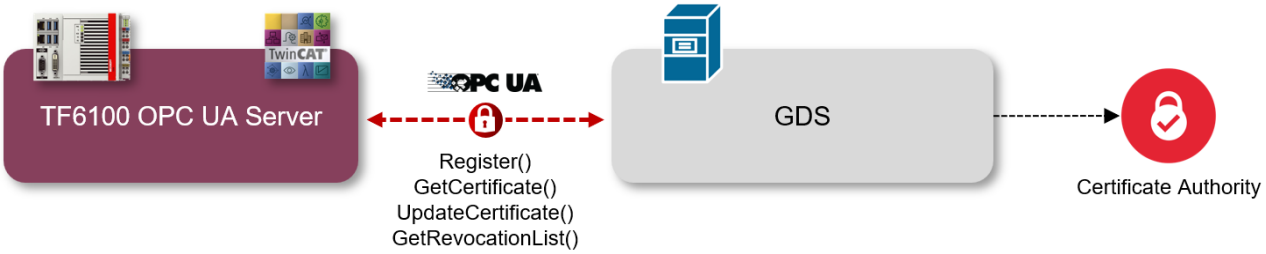
Any GDS can be used as a Global Discovery Service. Various OPC UA toolkit manufacturers offer corresponding software packages. Alternatively, the OPC Foundation also provides a GDS Sample Server on Github.

Configuration in the server

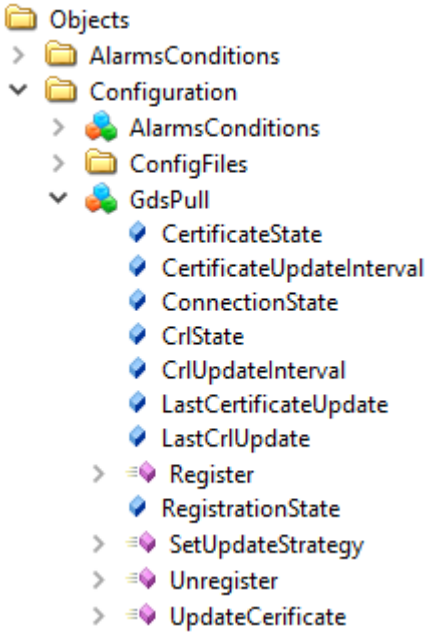
No further special configuration steps are required in the TwinCAT OPC UA Server to use this functionality. The corresponding interface is activated by default and can be used by a user with administrator rights. The user configured during initialization [▶ 34] has all the necessary permissions for this.

4.16.3 Pull

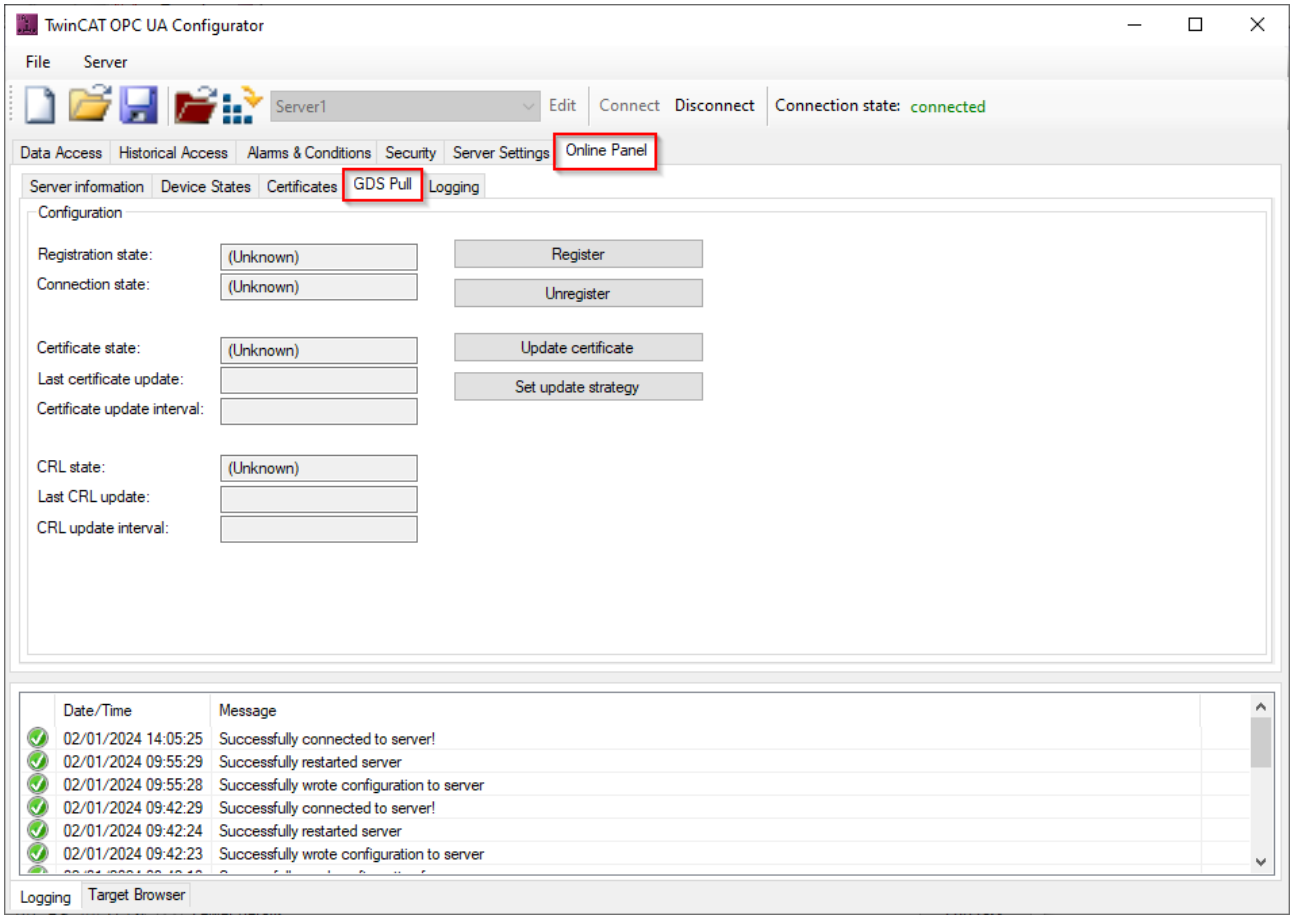
In this model, the server independently connects to the Global Discovery Server, registers there as a server application and obtains a suitable certificate.



The TwinCAT OPC UA Server offers an option to enable and configure the GDS pull functions via its configuration namespace.



Alternatively, the TwinCAT OPC UA Configurator can also be used to configure this function in the server. A corresponding user interface is available for this purpose.



Configuration via OPC UA Client

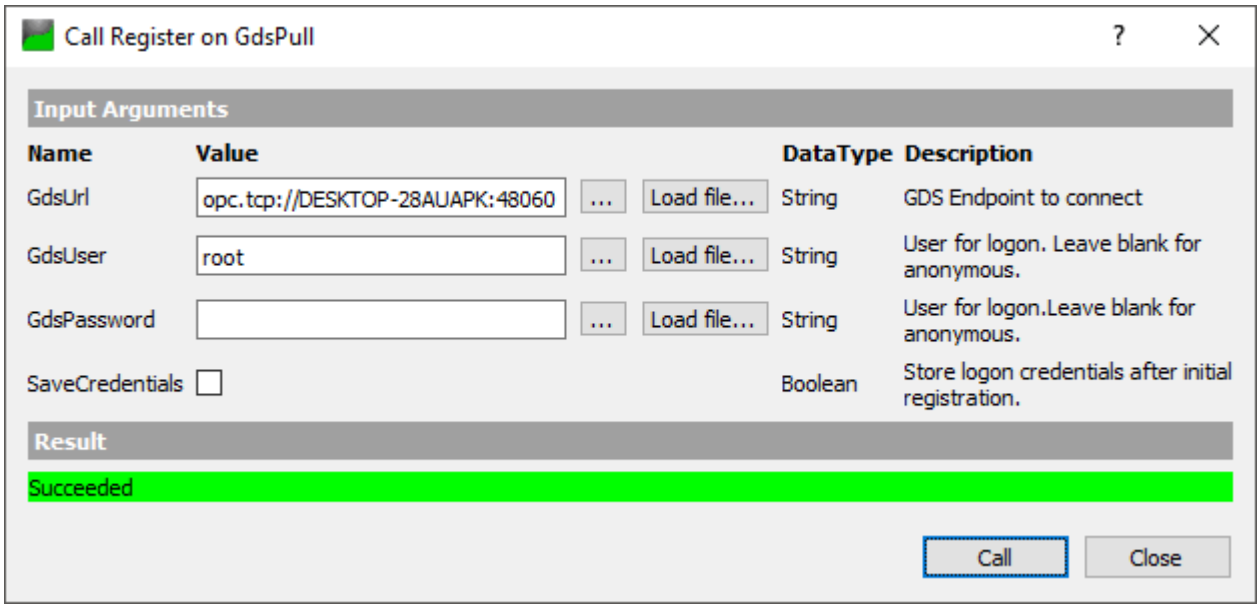
The following section describes the configuration via the address space of the server using a generic OPC UA client. The UA Expert from Unified Automation is used as the client software.

In the first step, the TwinCAT OPC UA Server must be registered as an application at the GDS. This is done using the Register() method. By registering with the GDS, a server certificate is automatically requested for the server application. Depending on the implementation of the GDS application, such a certificate is issued either automatically or after manual approval by an administrator. The variables RegistrationState and

CertificateState can be used to check whether the server has already been registered with a Global Discovery Service and has received a certificate from it. The variable CrlState indicates the status of the Certificate Revocation List and whether it could be obtained from the GDS.

#	Server	Node Id	Display Name	Value	Datatype	ource Timestamp	Server Timestamp	Statuscode
1	TcOpcUaServer...	NS13|Numeric|...	RegistrationState	5 (Registered)	Int32	2:06:47.020 PM	2:07:42.248 PM	Good
2	TcOpcUaServer...	NS13|Numeric|...	CertificateState	10 (Certificate updated)	Int32	2:06:47.020 PM	2:11:07.041 PM	Good
3	TcOpcUaServer...	NS13|Numeric|...	CrlState	12 (Crl updated)	Int32	2:06:47.020 PM	2:11:10.329 PM	Good

The method expects the following input parameters:



Input parameter	Meaning
GdsUrl	The URL of the Global Discovery Service in the format opc.tcp://hostnameOrIpAddress:port
GdsUser	Username of a GDS user with the right to register new applications.
GdsPassword	User password
SaveCredentials	Saves the user password in a configuration file of the TwinCAT OPC UA Server. For security reasons, this setting is not recommended. It was developed exclusively for Global Discovery Services, which require mandatory username/password authentication. Username/password authentication is usually only required once when registering the application. The issued server certificate is then used for all subsequent connections to the GDS.

Re-initialization of the server endpoints

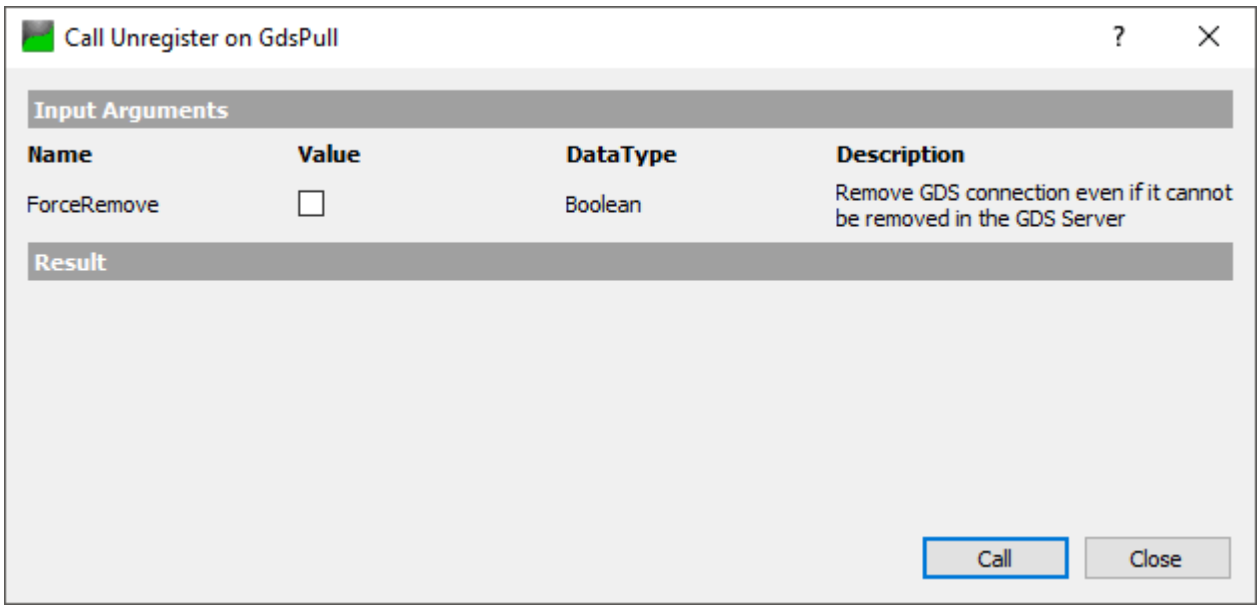
After registering the server application with the Global Discovery Service and obtaining a server certificate, the server reinitializes its endpoints once, causing connected clients to momentarily lose connectivity.

After the server application has been registered with a Global Discovery Service, a new file named "TcUaGdsClientConfig.xml" is created in the installation directory of the TwinCAT OPC UA Server. It contains the connection information of the configured GDS and the registration information obtained from there, plus timestamp information for the server application, e.g., when the certificate and CRL were last updated.

Unregistration at the Global Discovery Service

The Unregister() method can be used to unregister the TwinCAT OPC UA Server application with the GDS. Successful execution of the method causes the server application to be unregistered on the GDS and the contents of the TcUaGdsClientConfig.xml file to be deleted.

The method expects the following input parameters:



Input parameter	Meaning
ForceRemove	If the connection to the Global Discovery Service is no longer available, the connection to the GDS can be removed by setting this input parameter. The TwinCAT OPC UA Server removes the GDS from its configuration.

The issued server certificate and the CRL remain valid after the TwinCAT OPC UA Server has been decoupled from the GDS. If you want to delete them and run the server with a self-signed certificate, you have to remove the corresponding files in the PKI directory of the server and restart the server. The server then creates another self-signed certificate.

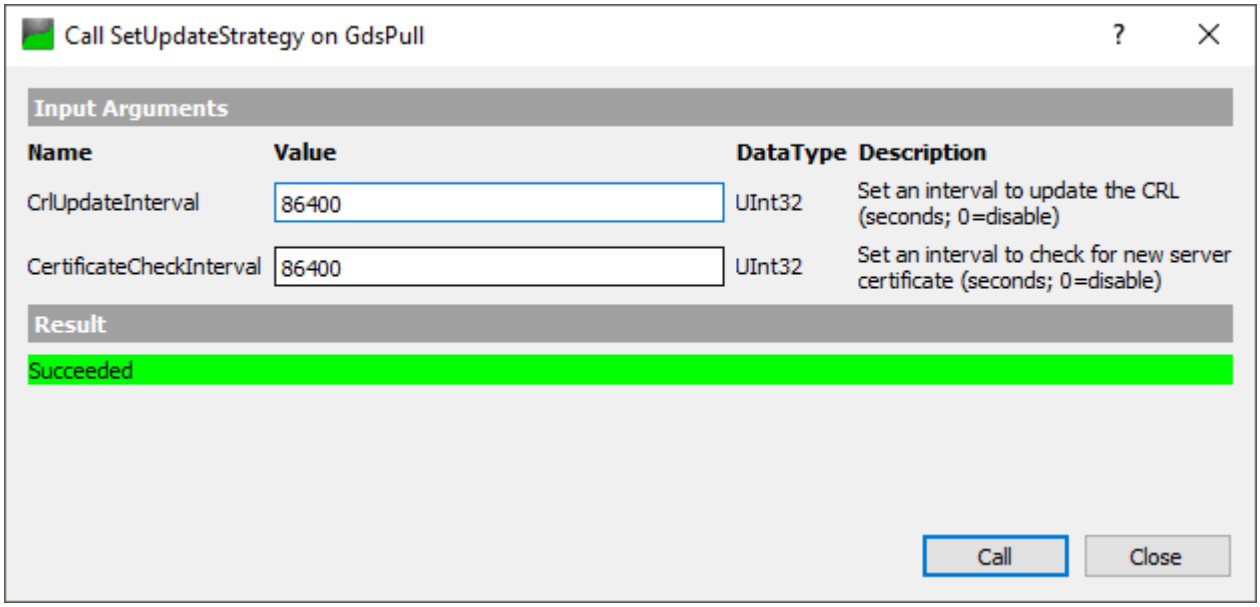
Updating the server certificate

An update of the server certificate can be requested from the GDS outside the regular update interval by executing the method UpdateCertificate(). The method does not expect any further input parameters.

Setting the update intervals for server certificate and CRL

The update intervals for the server certificate and the certificate revocation list can be set by executing the SetUpdateStrategy() method.

The method expects the following input parameters:



Input parameter	Meaning
CrlUpdateInterval	Sets the update interval for the certificate revocation list (seconds).
CertificateCheckInterval	Sets the update interval for the server certificate (seconds).

4.17 Security

4.17.1 Overview

One of the reasons OPC UA has been so successful as a communication technology is because of its integrated security mechanisms. Data communication based on OPC UA can be secured on two layers: transport *and* application layer. When connecting to the server, the client first selects an endpoint, which specifies the security functions to be used.

Endpoints

A server offers the client a list of different endpoints [▶ 128] to which the client can connect. An endpoint describes, among other things, which security functions (e.g. Message Security mode, Security Policy and available Identity Tokens) the communication connection via this endpoint should fulfill. For example, an endpoint may require signing and encryption of data packets (transport layer), as well as additional authentication of the client based on username/password (application layer).

Transport layer

A communication connection based on OPC UA can be secured at the transport layer. This is done through the use of client/server certificates and a mutual trust relationship between client and server application. Here, the client must trust the server certificate and vice versa in order for a communication connection to be established. This requires a mutual certificate exchange [▶ 129].

Application layer

In addition to the transport layer, a communication connection can also be secured at the application layer. For this purpose, various authentication mechanisms [▶ 131] are available, which are offered by the server endpoint.

4.17.2 Endpoints

The TwinCAT OPC UA Server makes various endpoints available for OPC UA Clients via the default port 4840/tcp. The endpoints define the connection type between client and server and whether it should be secured or unsecured.

Standard port

Note that the standard port 4840 may be used by other OPC UA servers, such as the **L**ocal **D**iscovery **S**erver (LDS) from the OPC Foundation, which is used by some vendors with OPC UA software packages.

Relationship of trust

Note that in order to use the secure endpoints, a trust relationship must be established between server and client, which is usually done via their certificates. Here is how to configure such a trust relationship on the server side [▶ 129].

Deprecated endpoints

Please note that the security profiles currently available in the endpoints may be classified as potentially insecure over time and will be replaced by newer ones. In this case, an update of the TwinCAT OPC UA Server is recommended. A configuration switch (<AllowDeprecatedSecurityPolicies>) can be used to reactivate security policies that are deprecated and classified as insecure.
For security reasons, Beckhoff recommends to leave this configuration switch disabled.

List of endpoints

The following list summarizes the endpoints of the TwinCAT OPC UA Server. This includes endpoints that have already been discontinued. By default, the TwinCAT OPC UA Server only offers endpoints that are currently considered secure. Since setup version 4.3.28, the unencrypted endpoint has also been disabled by default for security and certification reasons.

Security profile	Security mode	Short description
None	None	No encryption or signing of messages is carried out at this endpoint. Authentication, on the other hand, is possible.
Basic128Rsa15 (deprecated)	Sign Sign & Encrypt	This endpoint has been classified as deprecated from a security perspective and is disabled by default. If necessary, the endpoint can be enabled again.
Basic256 (deprecated)	Sign Sign & Encrypt	This endpoint has been classified as deprecated from a security perspective and is disabled by default. If necessary, the endpoint can be enabled again.
Basic256Sha256	Sign Sign & Encrypt	Endpoint currently present in the server for secure signing and encryption. Additional authentication is possible.
Aes256_Sha256_RsaPss	Sign Sign & Encrypt	Endpoint currently present in the server for secure signing and encryption. Additional authentication is possible.
Aes256_Sha256_RsaOaep	Sign Sign & Encrypt	Endpoint currently present in the server for secure signing and encryption. Additional authentication is possible.

All endpoints in the list can be enabled or disabled via the server configuration. In the following figure, all endpoints are enabled.



4.17.3 Certificate exchange

In order to establish a secure communication connection at the transport layer via a secure endpoint [▶ 128], mutual trust must be established between the client and the server. By default, both the TwinCAT OPC UA Server and the TwinCAT OPC UA Client generate a machine-specific, self-signed key pair consisting of a public and a private key when they are started for the first time, with the corresponding certificate having a validity period.

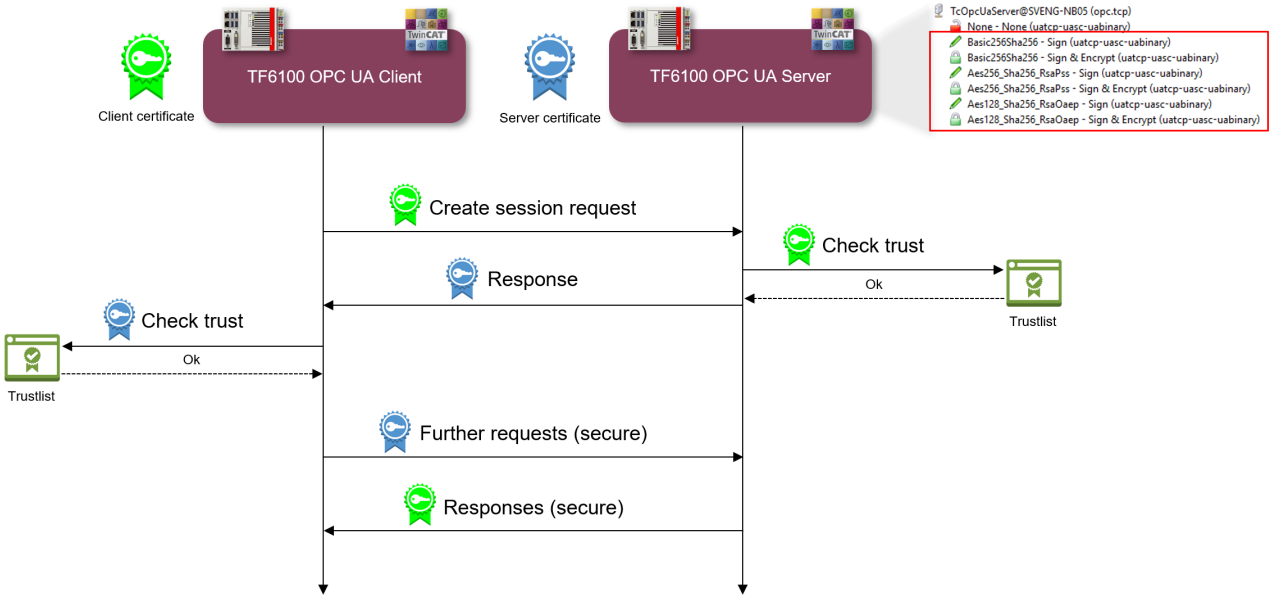
Validity period of the certificate

The self-signed certificate has a default validity period of 20 years. This is a server-specific setting that can be adjusted as necessary.

Instead of using the self-signed certificate, you can also have a certificate issued by any certification authority and use it. However, we recommend using a Global Discovery Server [▶ 123] to reduce the administrative workload.

To establish a trust relationship between an OPC UA Client and the TwinCAT OPC UA Server, you need the public key of the client certificate. The server must trust this certificate. The server manages the trust settings for client certificates in a subdirectory of the application directory [▶ 54].

The following diagram illustrates the relationship between the client and server certificate when establishing a secure communication connection:



The client transmits its public key with the CreateSession request. The server then has the option of checking the trust relationship. If the server trusts the client, it transmits its own public key in its response. The client therefore also has the option of checking the trust relationship with the server.

If mutual trust is ensured, the communication connection is initiated. The server's public key is used to encrypt a request from the client to the server. The response from the server to the client is then encrypted with the client's public key. Both communication participants have the option of decrypting the received message with their private key.

Messages are signed in reverse: a message is signed with the sender's private key. Since the recipient recognizes the sender's public key, the signature can be verified.

Configure trust relationship via file system

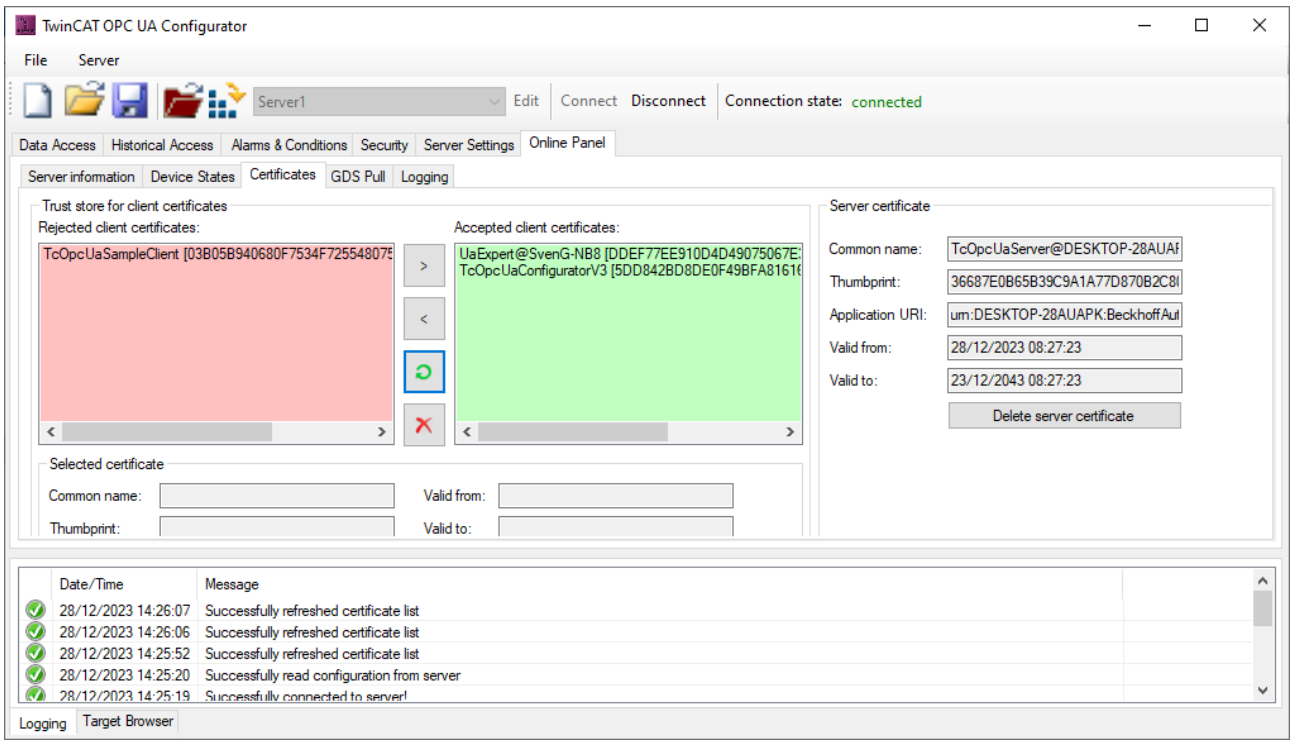
By moving client certificates between the trusted/rejected directories, the trust settings can be adjusted accordingly. The public key of a client certificate is automatically stored in the directory for rejected certificates the first time the client attempts to connect to a secure endpoint. By subsequently moving the public key to the directory for trusted certificates, the client is trusted at the next connection attempt and can connect.

AutomaticallyTrustAllClientCertificates

If this configuration option is enabled on the server, it automatically trusts all client certificates. In this case, they will not be listed in any of the above directories.

Configure the trust relationship using the configurator

You can also adjust the trust settings via Configurator [▶ 47]. The TwinCAT OPC UA Configurator includes a graphical user interface for configuring the trust settings.



4.17.4 Authentication

An OPC UA client application can authenticate itself to the TwinCAT OPC UA Server via various IdentityTokens. The following IdentityTokens are supported:

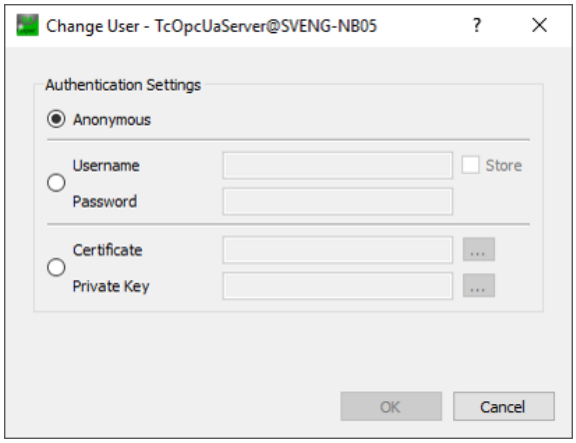
- Anonymous
- Username/Password
- User certificate

Delivery state

The "Anonymous" IdentityToken is enabled when the server is delivered; however, the server requires a one-time initialization to get started. This IdentityToken is then disabled and client applications must authenticate themselves with a username on the server.

Anonymous

This type of authentication allows any OPC UA client to connect to the server application. It is not necessary to specify a user identity, which means that there are no options for defining access rights on the server. Beckhoff recommends disabling this authentication type after commissioning the server. This can be done via the TwinCAT OPC UA Configurator. Below you will find an example screenshot from the OPC UA client application "UA Expert":



Username/Password

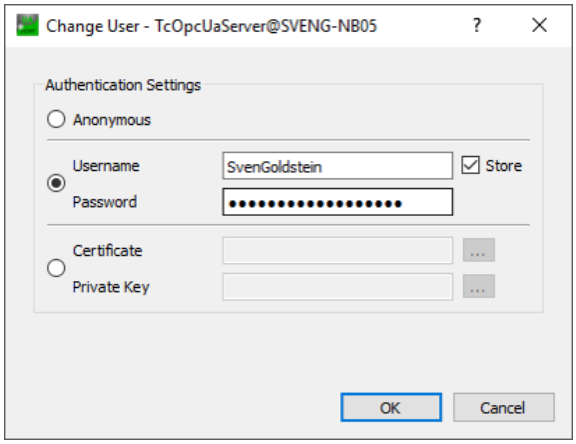
This type of authentication uses a username/password combination to authenticate the client to the server application. On the server, access rights can then be defined for the respective user identity. The user identity can be defined on different levels:

- User identity is defined in the server
- User identity comes from the lower-level operating system (e.g. a local Windows user)
- User identity comes from the Active Directory (e.g. if the industrial PC is part of a Windows domain)

Recommendation when using UserIdentityTokens

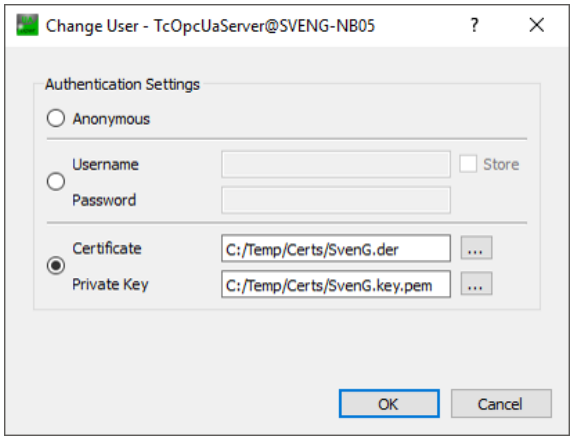
If UserIdentityTokens are to be used to authenticate client applications, Beckhoff recommends against the user identity being defined in the OPC UA server. Instead, the user account should be defined in the operating system or, if applicable, Active Directory.

Below you will find an example screenshot from the OPC UA client application "UA Expert":



User certificate

This type of authentication uses a certificate to authenticate to the server application. The handling of user certificates on the server side is identical to the use of certificates on the transport layer, i.e. the server must trust the (user) certificate before the client can successfully authenticate itself to the server with the certificate. A separate directory ("pkiuser") for the administration of user certificates is available in the server for this purpose. Below you will find an example screenshot from the OPC UA client application "UA Expert":



In addition, an X.509 user must be created via the TwinCAT OPC UA Configurator in order to use this authentication method. The name of the user must match the Common Name of the certificate.

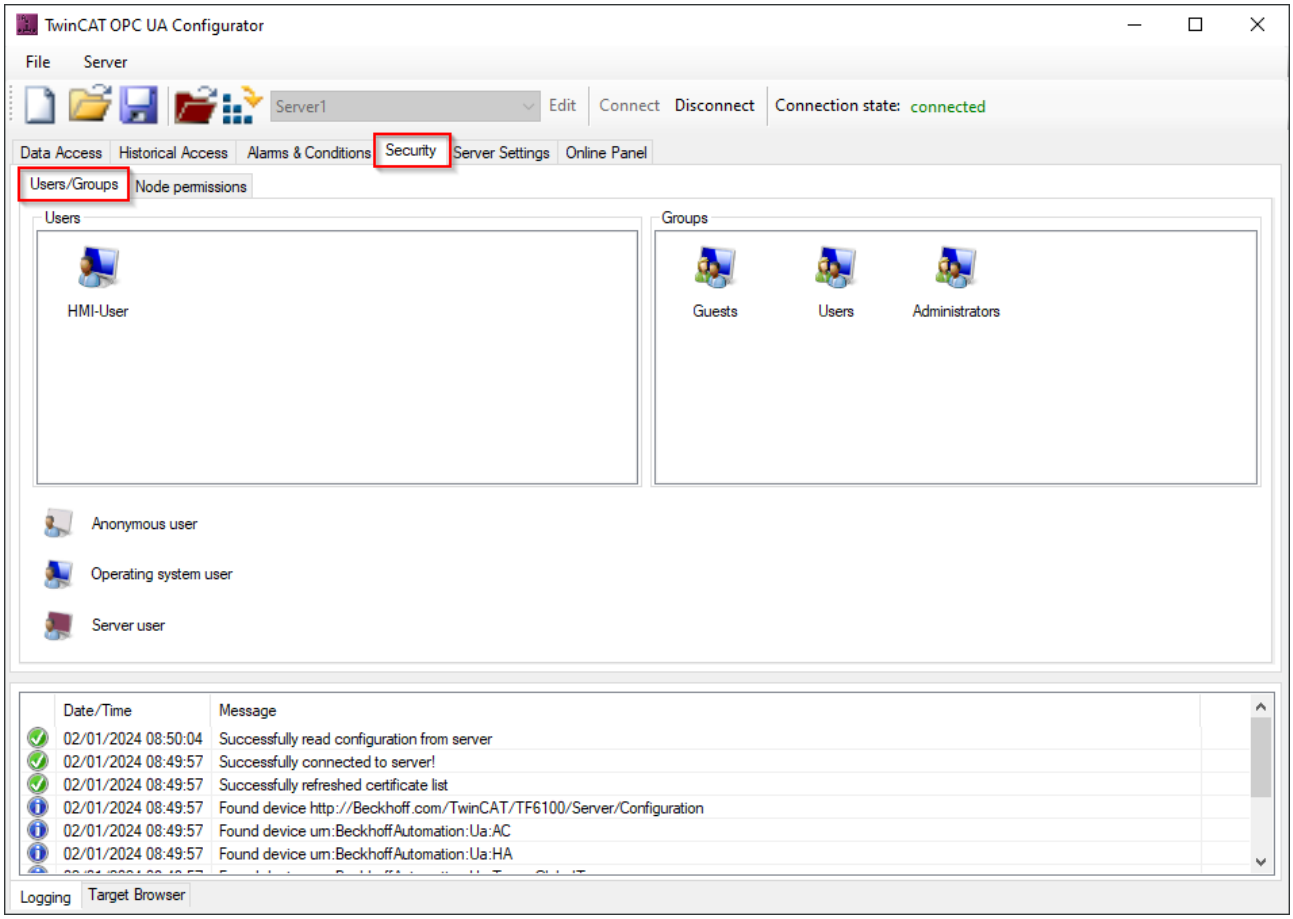
NOTICE

Authentication and server certificate

When using the unencrypted endpoint in combination with authentication, the TwinCAT OPC UA Client still requires the public key from the OPC UA Server certificate in order to encrypt the password during transmission. To this end the certificate must be trusted in the TwinCAT OPC UA Client (see Certificate exchange [▶ 129]).

Configuration

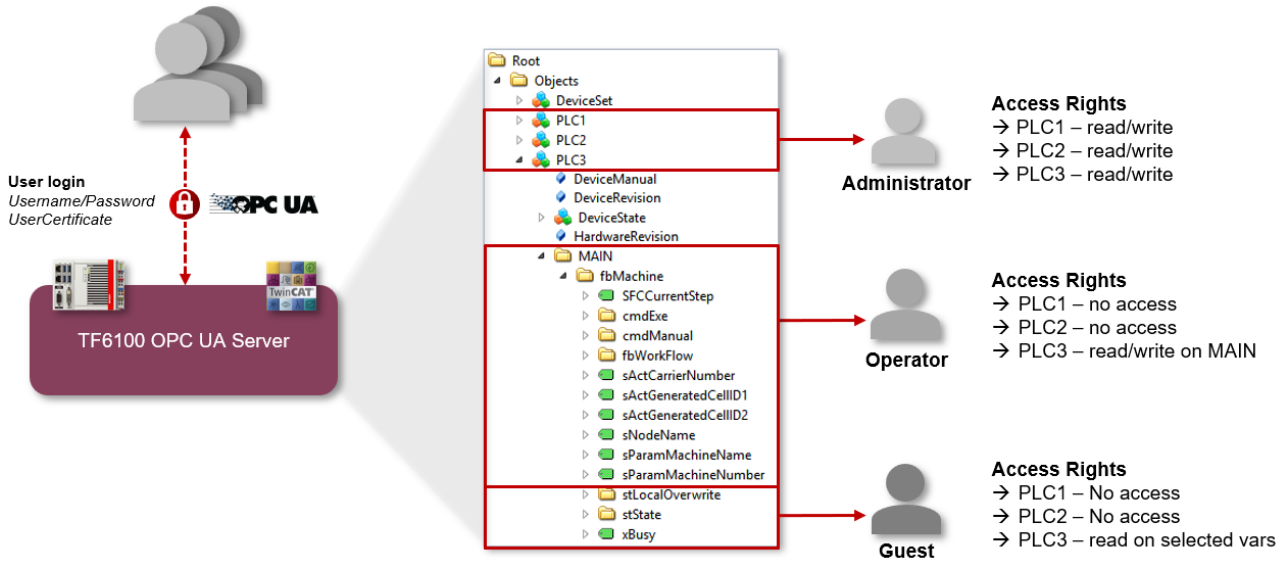
The IdentityToken is usually configured via the TwinCAT OPC UA Configurator. A graphical user interface is available here to enable IdentityToken, for example in the standalone configurator:



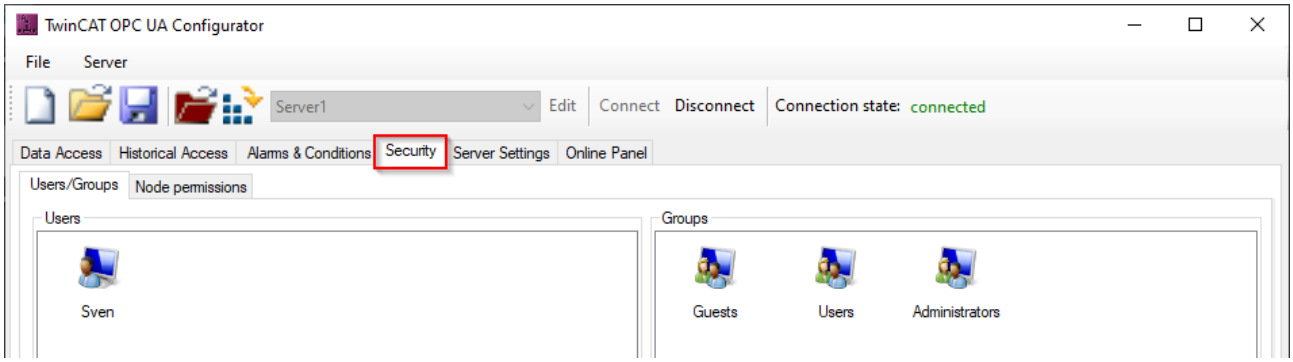
4.17.5 Access rights

The TwinCAT OPC UA Server enables the configuration of access rights for specific authenticated user identities [▶ 131]. These access rights can be configured for entire namespaces as well as for individual nodes. This allows both access to ADS devices (e.g. to different PLC runtimes) and individual symbols to be set with fine granularity.

These security settings are available for all Data-Access [▶ 56] devices that can be represented in the server namespace.

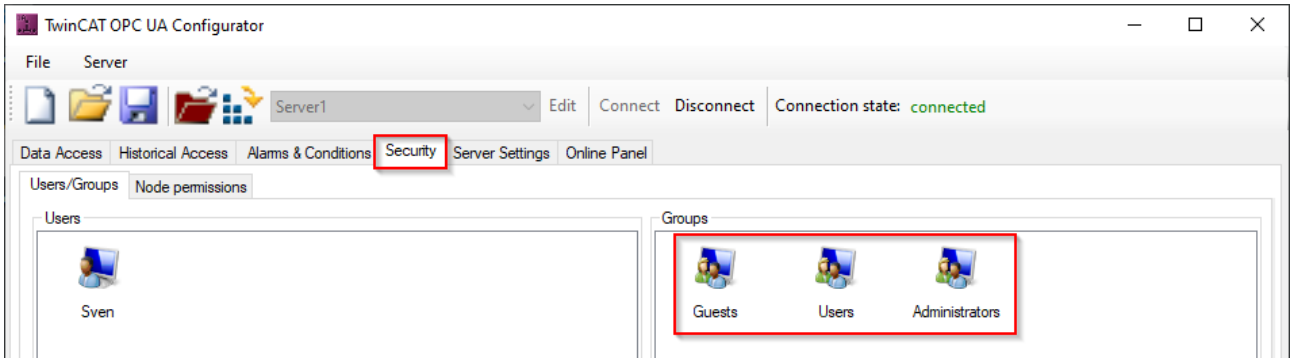


This functionality is configured via the TwinCAT OPC UA Configurator. In the standalone version of the configurator, the corresponding configuration interface can be found under the **Security** tab.



Configuring access to namespaces

The configuration of user access to individual namespaces is always based on the configured user groups. You can manage the corresponding access rights for individual namespaces in the user group settings. Some groups come preconfigured and you can use their configuration parameters as a guide.



The following table provides an overview of the permissions that are defined for each of the user groups as they are delivered.

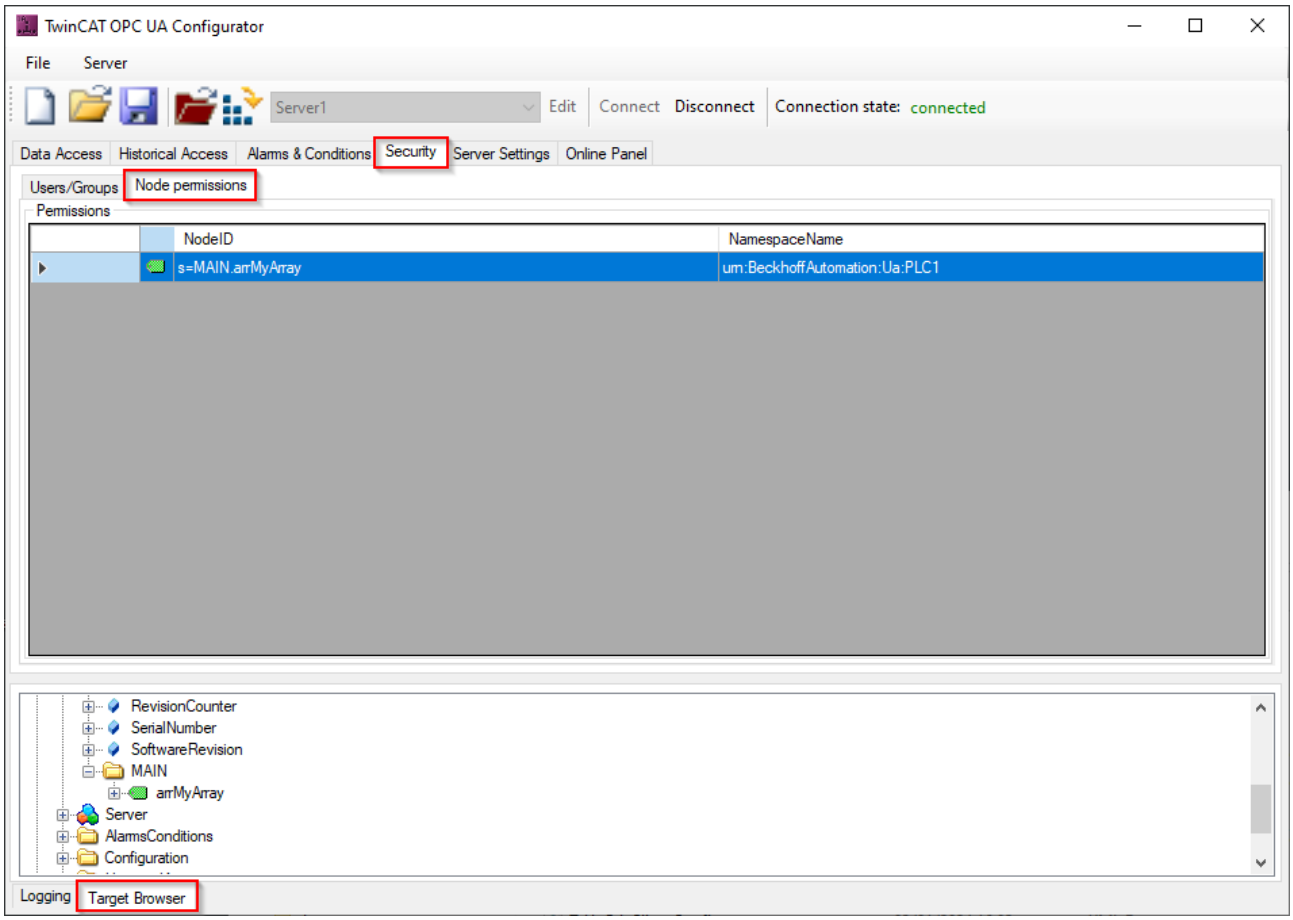
User group	Description
Administrators	Predefined user group for server administrators. This user group has full access to all namespaces.
Guests	Predefined user group for guest users. This user group has limited access to the server and only to the default namespace "0" with the permissions ReadAttribute, ReadValue and Browse.

User group	Description
Users	Predefined user group for normal users. This user group has extended access rights to all namespaces, in particular full access to the namespace of the preconfigured Data Access [▶ 56] device.

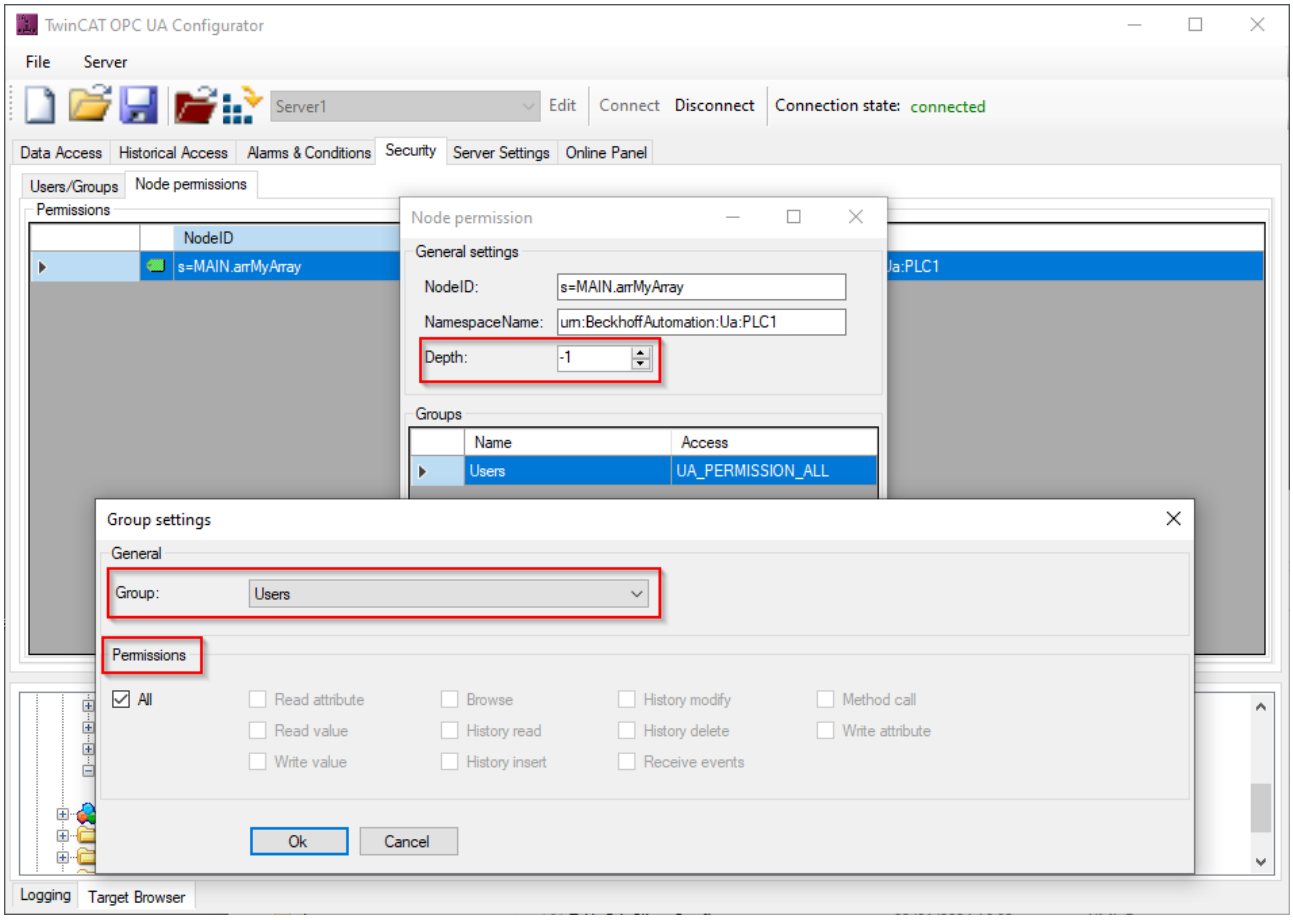
By adding users to the user groups, they automatically inherit the corresponding permissions from the group.

Configuring access at the node level

The **Node permissions** tab can be used to configure extended and very fine-grained permissions at node level. Sub-elements can inherit the permissions. You can use the Target Browser to transfer nodes to be configured from the server to the configuration using drag and drop.



The individual permissions on the node can be linked to a configured user group.

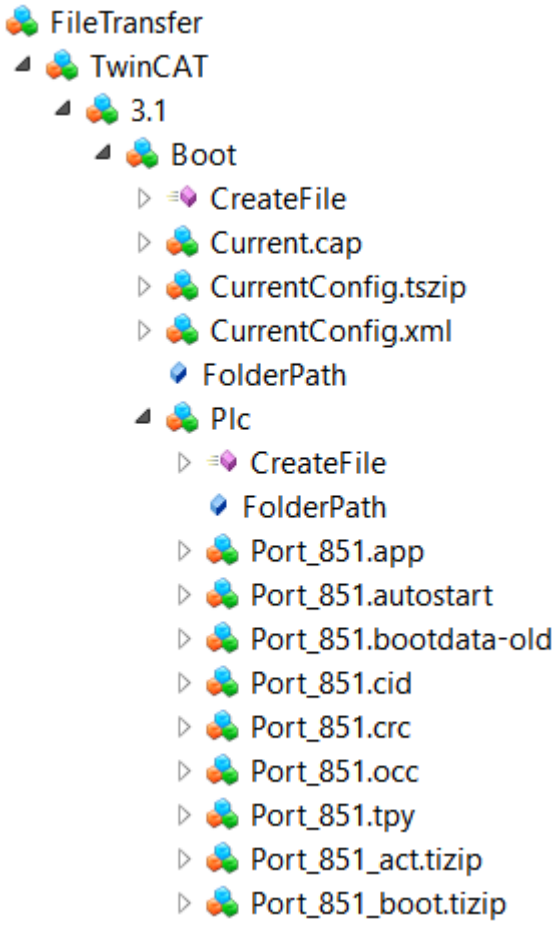


4.18 File Transfer

4.18.1 Overview

The OPC UA specification contains a special data type for transferring files. This special ObjectType called "FileType" describes the information model for the data transmission. Files can be modeled as simple variables in OPC UA with ByteStrings. FileType is a file with methods for accessing the file. The OPC UA specification provides further information about FileType and the structure and handling of the underlying methods and properties for accessing a file in the OPC UA namespace.

Beckhoff has implemented a generic way to load files and folders from a local hard disk into the OPC UA namespace. Each file is represented by a FileType and allows read and write operations for this file. In addition, each folder contains a CreateFile() method to create new files on the hard disk and a separate FolderPath to specify the actual path to the folder on the OPC UA Server.



FileTransfer in the OPC UA Server Device Manager

Only the OPC UA Server of the Beckhoff Device Manager (IPC diagnostics) has this function. The TwinCAT OPC UA Server also provides some parts of this file transfer. However, the general function that enables disclosure of all files and folders is only available in the OPC UA Server, which is part of the device manager that is automatically available on every Beckhoff Industrial PC or Embedded PC. See the Device manager documentation for more information.

4.18.2 Configuration

FileType objects are created in a separate namespace called "FileTransfer". An XML file (*files.xml*) is used to configure this namespace and to select the files and folders available via OPC UA. The file must be located in the same directory as the executable file of the OPC UA server. The system must be restarted in order to activate the configuration. The XML file contains information about the folder path and a search mask that defines which files are published in the OPC UA namespace:

```
<Files>
  <FolderObject DisplayName="TwinCAT">
    <FolderObject DisplayName="3.1">
      <FolderObject DisplayName="Boot" Path="c:/TwinCAT/3.1/Boot" Search="*.*" >
        <FolderObject DisplayName="Plc" Path="c:/TwinCAT/3.1/Boot/Plc" Search="*.*" ></FolderObject>
        <FolderObject DisplayName="Tmi" Path="c:/TwinCAT/3.1/Boot/Tmi" Search="*.*" ></FolderObject>
      </FolderObject>
    </FolderObject>
  </FolderObject>
</Files>
```

Reading a file with an OPC UA client

General file handling is described in Appendix C of the OPC UA specification.
Reading a file via OPC UA can therefore be divided into the following steps:

- Calling the Open method of a file. This method returns a file handle that must be saved for later access. The mode defines whether the file is read or written to (see File modes).

- Determining the file size with the property "Size". In this way, the entire file can be read when the Read method is called.
- Calling the Read method. Inserting the file handle and file size as inputs. Selecting the destination folder in which the file contents are to be saved AFTER the method call.
- Calling the Close method to enable the file handle.

File modes

The following table shows all available file modes:

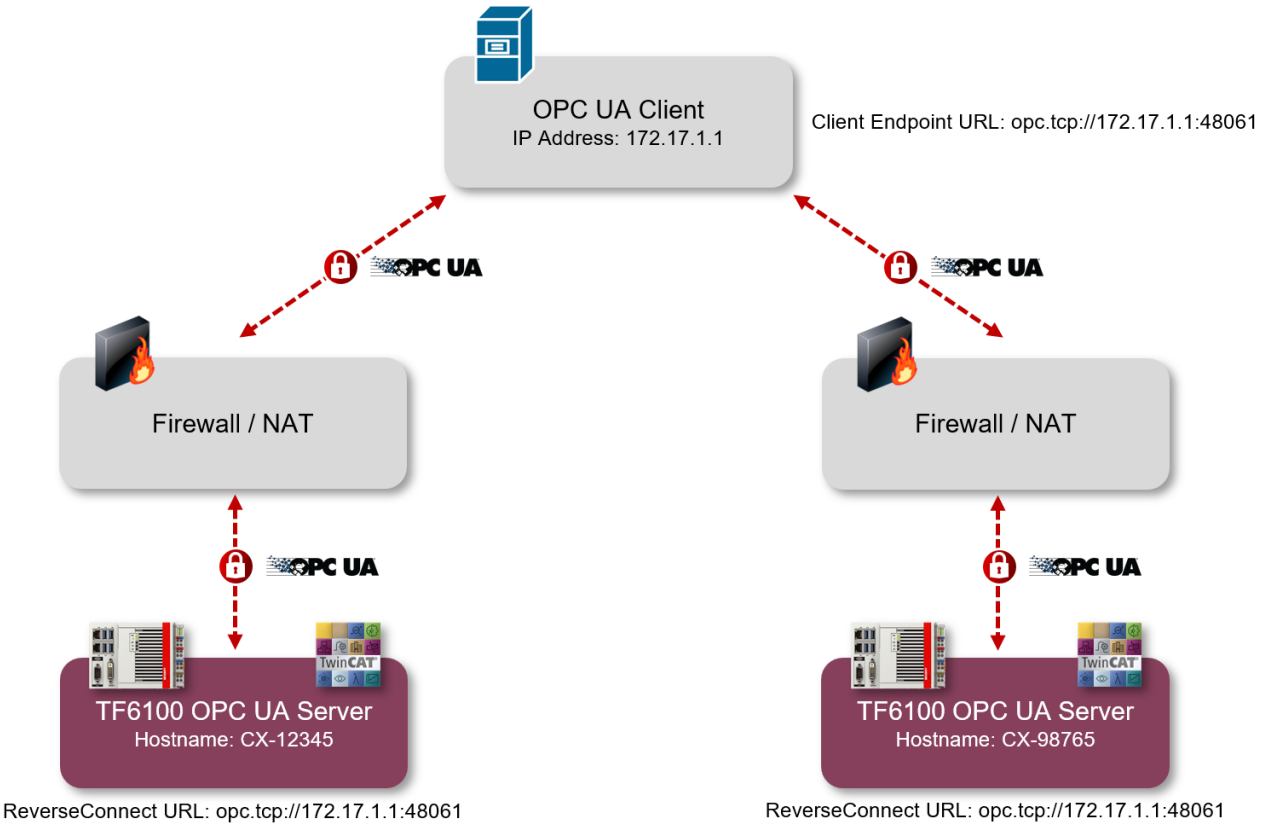
Field	Bit	Description
Read	1	The file is opened for reading. If this bit is not set, Read cannot be executed.
Write	4	The file is opened for writing. If this bit is not set, Write cannot be executed.
EraseExisting	6	The existing file contents are deleted, and an empty file is made available.
Append	10	The file is opened and positioned at the end, otherwise it is moved to the beginning. This position can be changed with SetPosition.

General behavior

The number of files opened in parallel is in principle unlimited and is subject only to any restrictions of the underlying operating system. However, files are subject to a 60 seconds timeout. After this timeout, open files are not automatically closed immediately. Instead, they are marked as "to close". If the corresponding FileHandle is used for a read/write operation during this time, the timeout is reset and the FileHandle remains valid. If an Open operation is performed on the same file during this time, the old FileHandle is enabled. If an OPC UA Client disconnects from the server and still has files open, all FileHandles belonging to this session will be closed automatically.

4.19 Reverse Connect

The TwinCAT OPC UA Server supports the ReverseConnect function of OPC UA to establish a backward communication connection from the server to the client. To activate this function, a list of client addresses must be stored in the server. Then the server establishes an OPC UA TCP connection for each client in the list. This type of connection setup is often used when an OPC UA client needs to establish connections with servers that are located behind a firewall or NAT device. The following figure illustrates this relationship.



In this example, there is an OPC UA client that needs to connect to two servers, each of which is behind a firewall. The firewall blocks all incoming communication traffic and does not open any ports in the internal network. The client is now configured for ReverseConnect and opens its own network port under the client endpoint URL opc.tcp://172.17.1.1:48061. Each of the lower-level servers has also been set up for ReverseConnect and has entered the client endpoint URL as the ReverseConnect URL. The server now opens and maintains a TCP connection to the client using this URL. The actual OPC UA client communication with the server is tunneled through this TCP connection. From the firewall's point of view, this is an "outgoing" communication due to the initially established TCP connection. Only the outgoing communication port (48061 in this example) needs to be enabled in the firewall.

Client compatibility

Please note that the OPC UA client must also support this function and be accessible via its Client Endpoint URL.

Configuration in the server

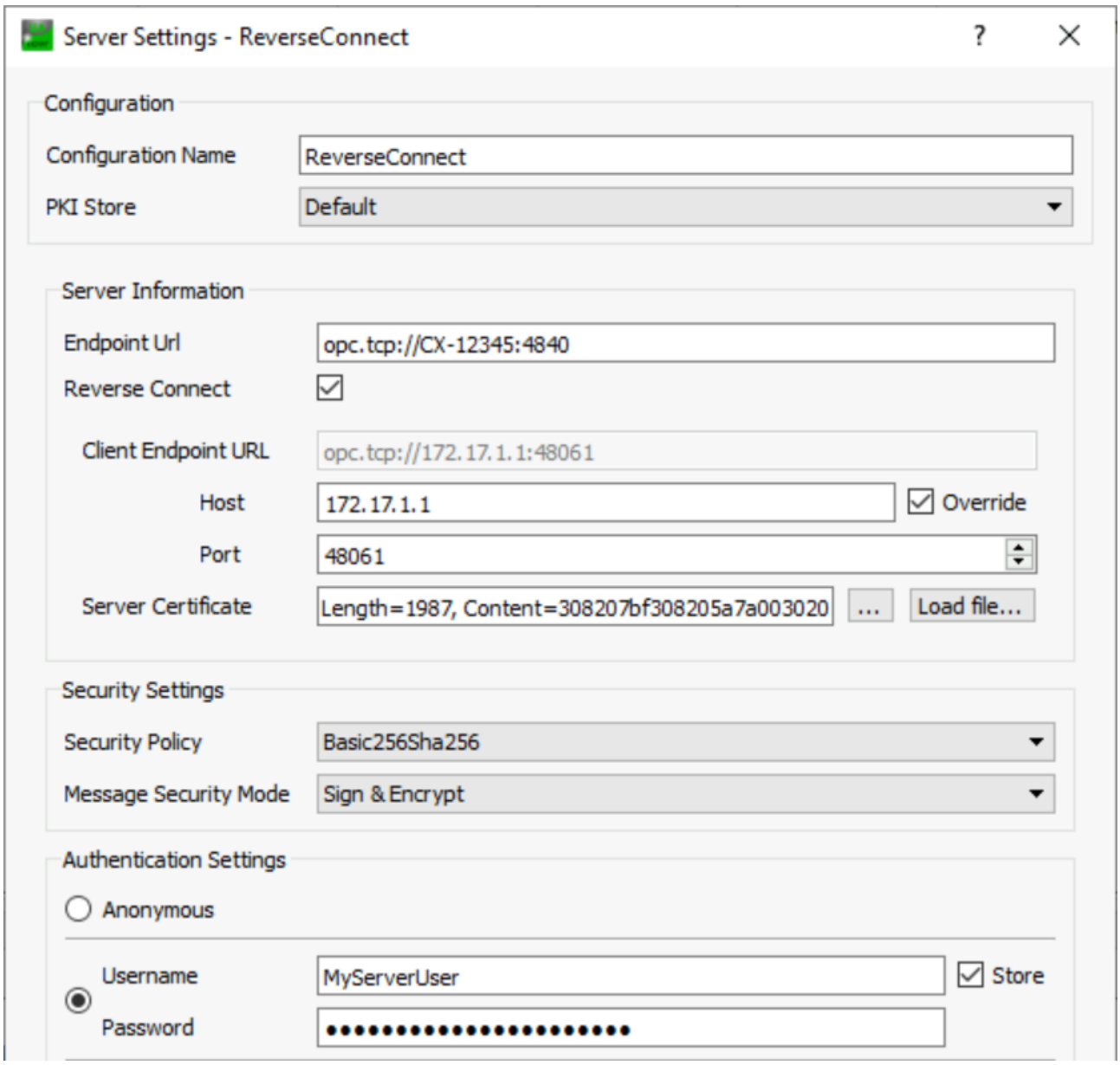
A list of OPC UA Clients can be configured in the TcUaServerConfig.xml within the <UaEndpoint>. The Client Endpoint URL under which the respective clients can be reached is entered here.

```
<ReverseConnect>
  <Url>opc.tcp://172.17.1.1:48061</Url>
</ReverseConnect>
```

Configuration in the OPC UA Client

The following screenshot shows an example of the configuration of a ReverseConnect connection in the OPC UA client software "UA Expert" from Unified Automation. ReverseConnect is enabled in the connection settings and the client endpoint URL is entered under which the client can be reached by the server. In the EndpointURL field, the Server Endpoint URL is entered which the client should use as soon as a ReverseConnect TCP connection has been established by the server. Referring to our figure above, opc.tcp://CX-12345:4840 or opc.tcp://CX-98765:4840 is entered here, for example. The settings for Security Settings, Message Security Mode and the authentication parameters then also apply to this connection.

Please note that you may still need to import the server certificate. This is the server's public key, which is stored in the corresponding application directory [▶ 54].



Communication history

The following Wireshark® Trace shows an example of a connection setup based on ReverseConnect. The only difference to the above figure is that the client used in this recording was configured for the Client Endpoint URL opc.tcp://172.30.3.86:48061. The server is located behind a NAT device, which in turn has the IP address 178.200.200.59.

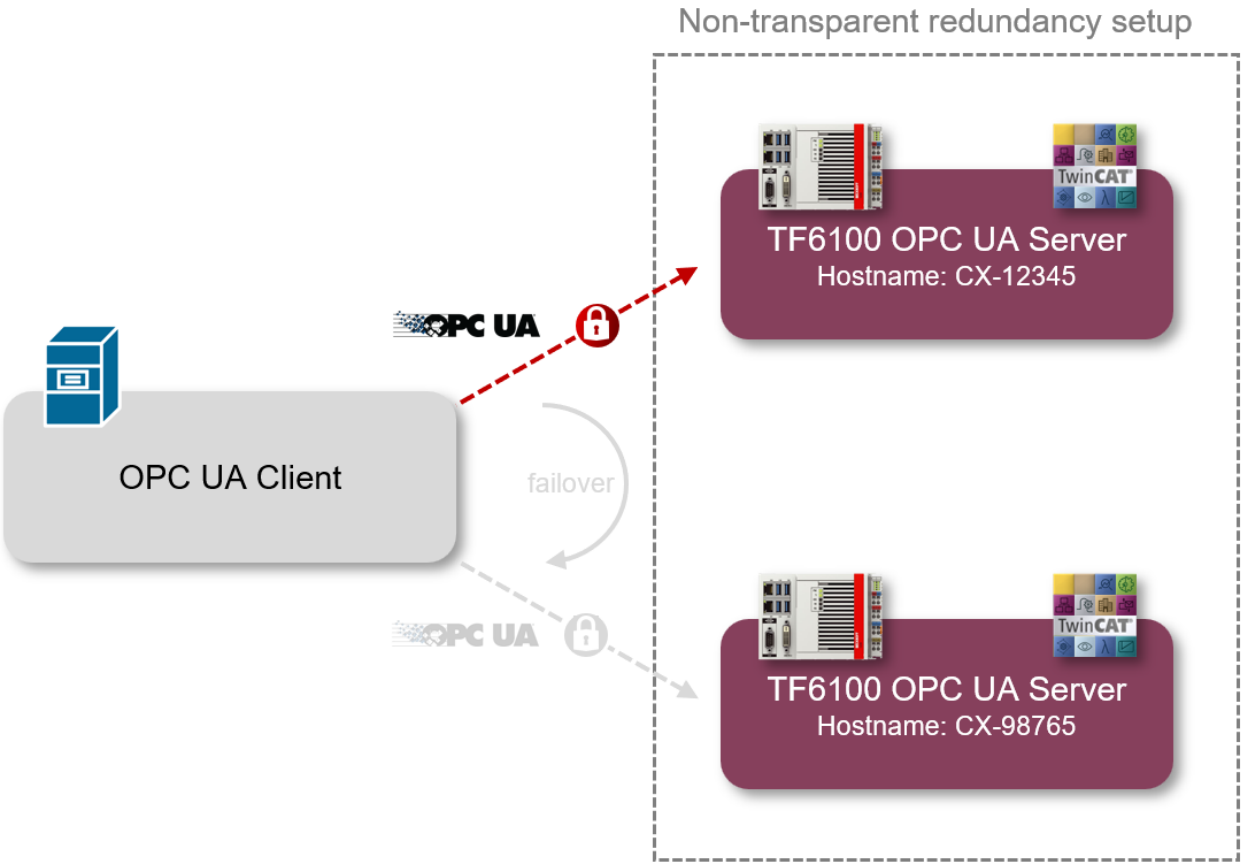
No.	Time	Source	Destination	Protocol	Length	Info
608	5.228616	178.200.200.59	172.30.3.86	OpcUa	154	Reverse Hello message
609	5.228927	172.30.3.86	178.200.200.59	OpcUa	116	Hello message
610	5.249617	178.200.200.59	172.30.3.86	OpcUa	82	Acknowledge message
612	5.344295	172.30.3.86	178.200.200.59	OpcUa	3039	OpenSecureChannel message: ServiceId 0
620	5.541799	178.200.200.59	172.30.3.86	OpcUa	246	OpenSecureChannel message: ServiceId 0
626	5.868707	172.30.3.86	178.200.200.59	OpcUa	2246	UA Secure Conversation Message: ServiceId 0
646	5.992669	178.200.200.59	172.30.3.86	OpcUa	1486	UA Secure Conversation Message: ServiceId 0

When the TCP connection is established from the server to the client, a so-called "Reverse Hello" message is sent. In this, the server informs the client under which Server Endpoint URL it can be reached. This is the same Server Endpoint URL that you configured in the client (see above). The client uses this Server Endpoint URL for the further connection to the server.

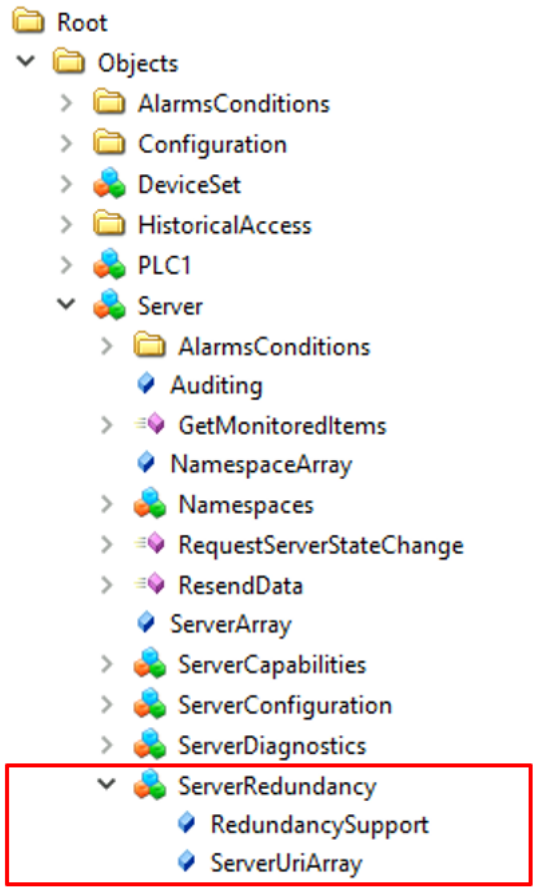
4.20 Redundancy

The OPC UA specification defines two types of server redundancy: transparent and non-transparent redundancy. With transparent redundancy, the transfer of functions from one server to another is transparent for the client, i.e. the client does not know that a failover process has occurred and therefore has no control over the failover behavior. In addition, the client does not need to take any action to continue sending or receiving data. With non-transparent redundancy, the failover process from one server to another and the actions to continue sending or receiving data are performed by the client. The client must know the redundancy configuration of the server and perform the necessary actions in order to benefit from the server redundancy in the event of a failover. The non-transparent redundancy defines various failover modes which, among other things, specify how many servers are actively involved in the communication or are available as purely passive systems in the event of a failover.

The TwinCAT OPC UA Server supports non-transparent redundancy with the operation modes "cold", "warm" and "hot". The differences in these three operation modes are explained below.



For non-transparent redundancy, OPC UA provides the data structures that allow the client to recognize which servers are available in the redundant server group, as well as server information that tells the client which failover modes the server supports. Based on this information, the client can determine what measures it needs to take to achieve failover. For this purpose, the ServerRedundancy object exists in the server's address space and indicates the redundancy mode supported by the server and the redundancy set, i.e., all servers that are part of the redundancy configuration.



#	Server	Node Id	Display Name	Value	Datatype
1	TcOpcUaServer...	NS0|Numeric|...	RedundancySupport	1 (Cold)	Int32
2	TcOpcUaServer...	NS0|Numeric|...	ServerUriArray	{'urn:EC2AMAZ-R1LR85K:BeckhoffAutomation:TcOpcUaServer:...	String

Failover modes

The following table lists all supported failover modes for non-transparent redundancy.

Mode	Description
Cold	Only one server can be active at a time in Cold Failover mode. This can mean that redundant servers are not available (not switched on) or are available but not running (the PC is running but the application is not started).
Warm	In Warm Failover mode, the backup servers can be active but not make a connection to the actual data points (typically a system where the underlying devices are limited to a single connection). The underlying devices, such as the PLC, may have limited resources that only allow a single server connection. Therefore, only one server is able to retrieve data.
Hot	In Hot Failover mode, all servers are switched on and ready for operation. In scenarios where servers retrieve data from a downstream device, such as a PLC, one or more servers are active and connected in parallel to the downstream device(s). These servers have only minimal knowledge of the other servers in their group and work independently. If a server fails or a serious problem occurs, its ServiceLevel drops (see below). After recovery, the server returns to the Redundant Server Set with an appropriate ServiceLevel to indicate that it is available.

ServiceLevel

The ServiceLevel provides a client with information about the state of a server and its ability to deliver data. The ServiceLevel is a node of the data type byte with a value range from 0 to 255. The TwinCAT OPC UA Server currently sets the ServiceLevel statically to the value 255. It is also possible to influence the ServiceLevel of a TwinCAT OPC UA Server via an ADS interface in the server. Further information can be found in the subchapter ServiceLevel [▶ 144].

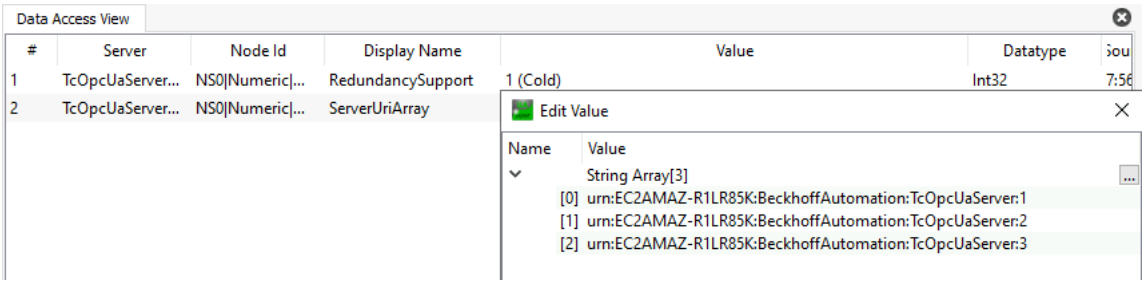
Configuration

The server redundancy can be configured using the TwinCAT OPC UA Configurator. Two steps are necessary for this:

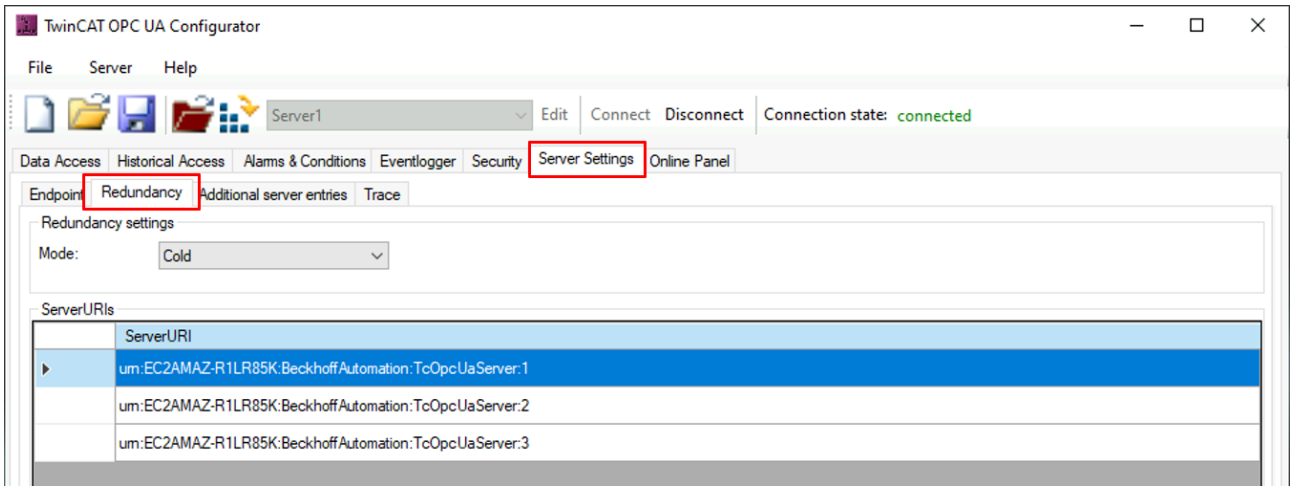
- Configuration of redundancy
- Adding the servers from the redundancy set as AdditionalServerEntry

Configuration of redundancy

When configuring server redundancy, you define which Failover mode the server provides and which other servers should be part of the redundancy configuration. These servers are identified via their ServerURI. The Failover mode and a list of ServerURIs are then stored by the server in its ServerRedundancy object (see above).



These settings can be configured via the TwinCAT OPC UA Configurator in the "Server Settings" tab, for example:



As the ServerURI itself does not yet contain any connection information (e.g. the server or discovery URL), this information must be made available to the client elsewhere. This is done via the so-called AdditionalServerEntry configuration. This allows a client to retrieve all necessary information from the server with a FindServers call and identify the connection information for a specific ServerURI. This configuration step is explained below.

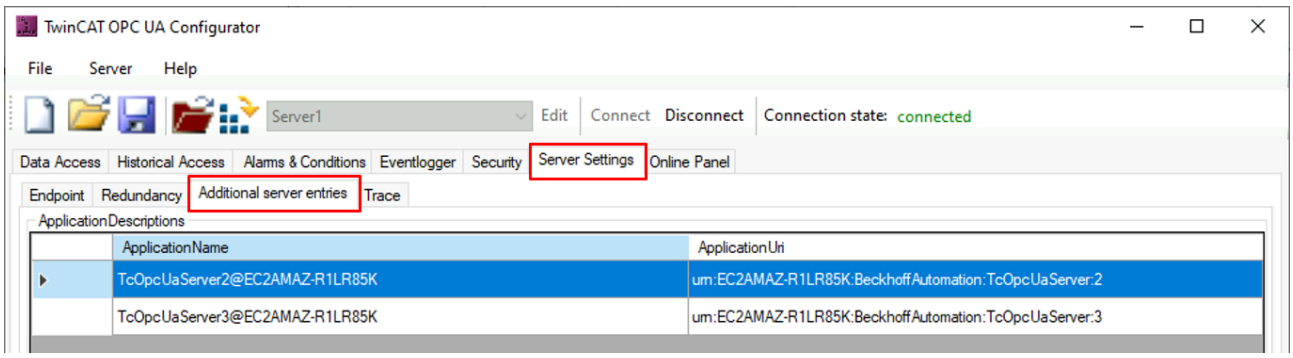
Adding the servers from the redundancy set as AdditionalServerEntry

With the AdditionalServerEntry configuration, information from other TwinCAT OPC UA Servers is added to the FindServers call of a server. This allows a client to locate all servers from the redundancy configuration and connect to them. The following screenshot shows an exemplary FindServers call on the local TwinCAT OPC UA Server. The call returns three TwinCAT OPC UA Servers, which are also part of a redundancy configuration.

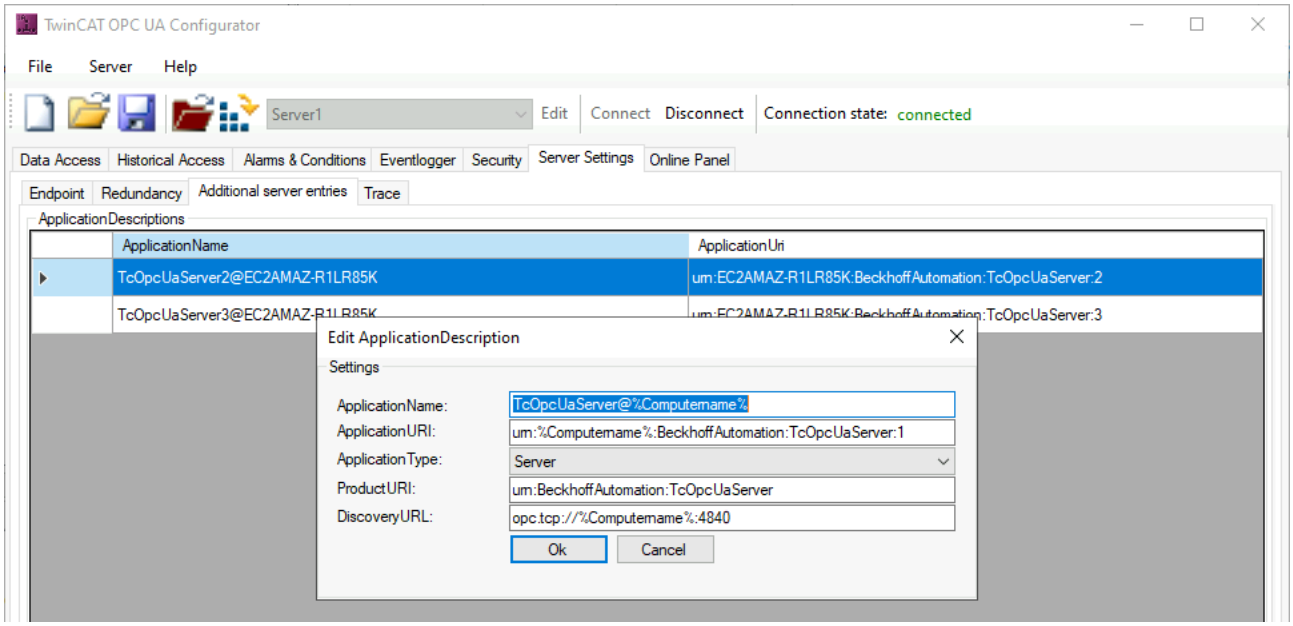


The client uses the ServerURI from the redundancy configuration to identify a server.

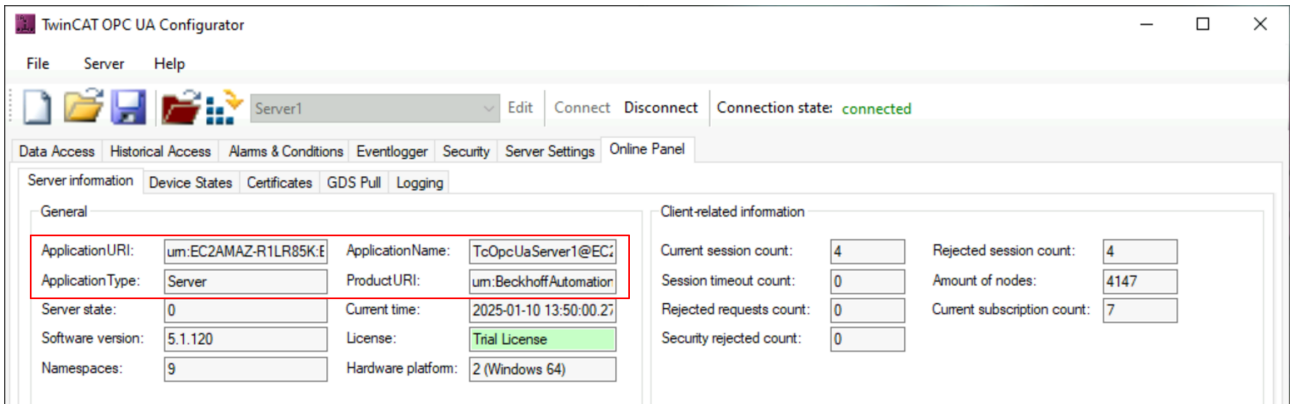
The AdditionalServerEntries can be configured via the TwinCAT OPC UA Configurator in the **Server Settings** tab, e.g.:



When adding a new entry, certain fields are already filled in with a template. Please adjust these entries according to your configuration.



The specific values for ProductUri, ApplicationName, ApplicationUri and ApplicationType can be found in the OnlinePanel of the TwinCAT OPC UA Configurator:



The DiscoveryURL corresponds to the ServerURL that you usually use to connect to the respective server.

Also see about this

- DeviceState [▶ 97]

4.20.1 ServiceLevel

The ServiceLevel provides a client with information about the state of a server and its ability to deliver data. The ServiceLevel is a node of the byte data type with a value range from 0 to 255. A high value stands for high availability or quality of service, a low value for limited availability.

Clients can use the ServiceLevel to evaluate the current state of a server or to compare servers in redundant or distributed environments. The ServiceLevel is to be understood as a general status indicator and does not replace a detailed diagnosis of individual error causes.

The TwinCAT OPC UA Server currently sets the ServiceLevel statically to the value 255. It is also possible to influence the ServiceLevel of a TwinCAT OPC UA server via an ADS interface in the server. The function can be used either with or without a so-called watchdog.

With watchdog monitoring

The written ServiceLevel must be written again cyclically within the configured timeframe. If there is no update, a configured fallback value is automatically set. The function block FB_OpcUaServer_SetServiceLevel [▶ 159] is used for that.

In the context of the watchdog, the two configurable values are:

Setting	Meaning
ServiceLevelWatchdogTimeout	This is the amount of time in ms after which the watchdog is triggered if the ServiceLevel has not been submitted to the server again.
ServiceLevelWatchdogFail	This is the value to which the ServiceLevel is set when the watchdog encounters a timeout.

Without watchdog monitoring

The written ServiceLevel is set once by the function block FB_OpcUaServer_SetServiceLevel [▶ 159]. Any active watchdog is deactivated. The value is retained until it is changed by another write access. If the watchdog is to be reactivated, the method must be executed again with watchdog monitoring.

Also see about this

- FB_OpcUaServer_SetServiceLevel [▶ 159]

4.21 Logging

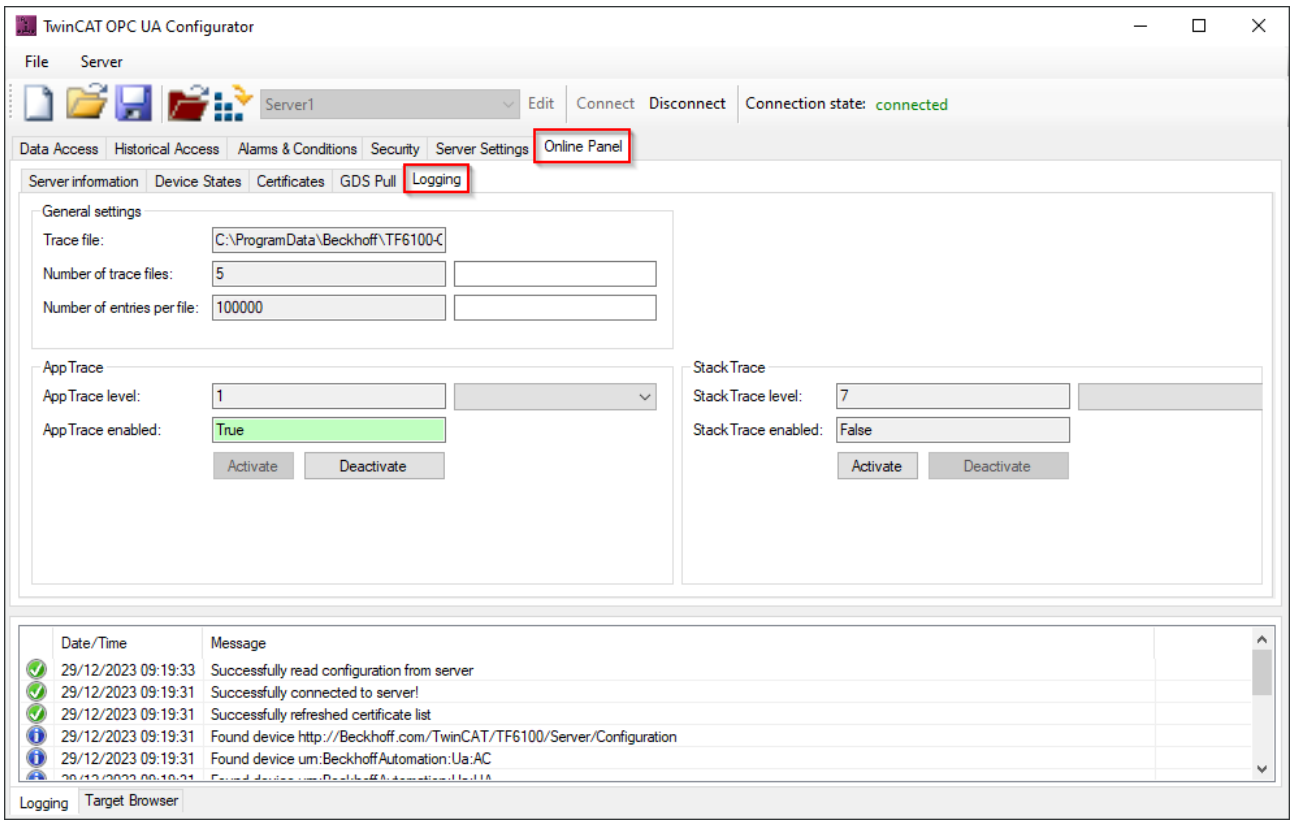
You can activate a log file in the server for extended diagnostics, in which various information is then recorded on the basis of different log levels.

Influence of logging on the operating behavior

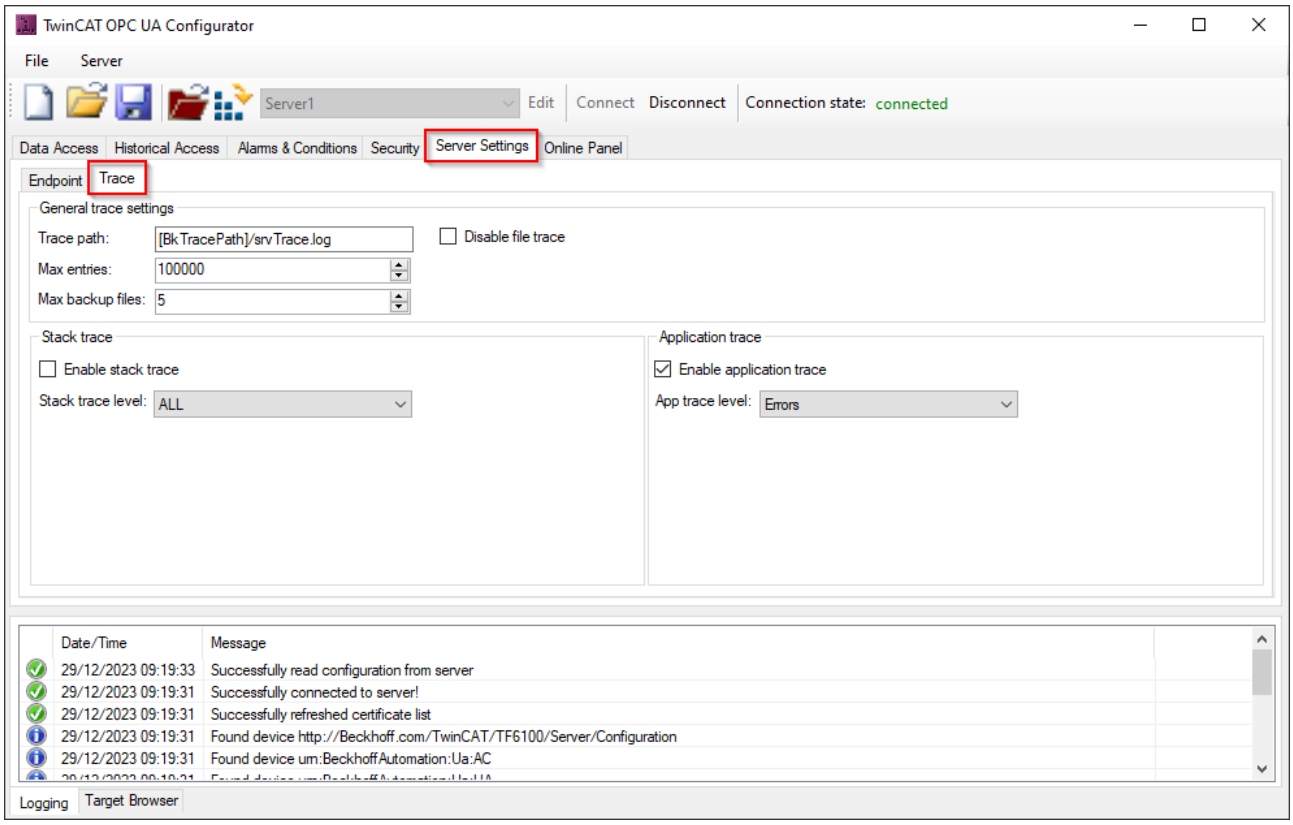
Please note that enabling the log file can have a negative impact on the speed and operating behavior of the TwinCAT OPC UA Server.

The default path for the log files created depends on the operating system used and is described in more detail in the chapter Application directories [▶ 54].

Logging is usually enabled/disabled via the TwinCAT OPC UA Configurator. You can enable/disable logging both online on the server and offline using the corresponding configuration file, which means that logging is only enabled after a server restart. The following screenshot shows the configuration interface for online logging:

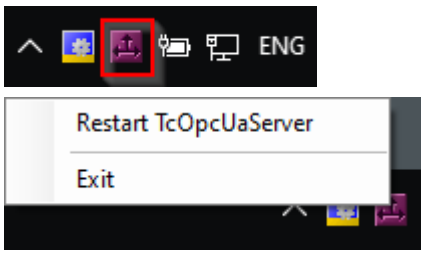


The following screenshot shows the configuration interface for offline logging:



4.22 System Tray

The TwinCAT OPC UA Server contains an application that can be called up as an icon in the Windows system tray. This application enables the triggering of a server restart. The corresponding function can be called via the context menu.


ENG
Restart TcOpcUaServer
Exit

5 PLC API

5.1 Tc2_OpcUa

5.1.1 Data types

5.1.1.1 ST_OpcUAServerInfo

ST_OpcUAServerInfo contains session information of a TwinCAT OPC UA Server.

Syntax

```
TYPE ST_OpcUAServerInfo :
STRUCT
    nReserved : UDINT;
    nCummulatedSessionCount       : UDINT;
    nCurrentSessionCount          : UDINT;
    nRejectedSessionCount         : UDINT;
    nSecurityRejectedSessionCount : UDINT;
    nSessionTimeoutCount          : UDINT;
    nCurrentSubscriptionCount     : UDINT;
    nRejectedRequestCount         : UDINT;
    nSecurityRejectedRequestCount : UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
nReserved	UDINT	Placeholder.
nCummulatedSessionCount	UDINT	Total number of client sessions since the server was started.
nCurrentSessionCount	UDINT	Total number of current client sessions.
nRejectedSessionCount	UDINT	Total number of sessions rejected by the server.
nSecurityRejectedSessionCount	UDINT	Total number of sessions rejected by the server for security reasons (example: incorrect combination of user name and password).
nSessionTimeoutCount	UDINT	Total number of sessions that had a timeout.
nCurrentSubscriptionCount	UDINT	Total number of current subscriptions in the server.
nRejectedRequestCount	UDINT	Total number of failed requests.
nSecurityRejectedRequestCount	UDINT	Total number of failed requests for security reasons.

5.1.1.2 E_OpcUAServerOption

E_OpcUAServerOption determines which command is to be sent to the TwinCAT OPC UA Server.

Syntax

```
TYPE E_OpcUAServerOption
(
    eOPCUAServerOption_None,
    eOPCUAServerOption_Restart,
    eOPCUAServerOption_Shutdown,
    eOPCUAServerOption_RefreshCfg,
    eOPCUAServerOption_ServerInfo
);
END_TYPE
```

Parameter

Name	Description
eOPCUAServerOption_None	Initial state of the enumeration.
eOPCUAServerOption_Restart	This option triggers a restart of the OPC UA interface of the server.
eOPCUAServerOption_Shutdown	This option triggers the shutdown of the OPC UA interface of the server. As the restart option above works via OPC UA, it is no longer available after using this option until a complete server restart.
eOPCUAServerOption_RefreshCfg	This option currently has no function.
eOPCUAServerOption_ServerInfo	This option queries the server information contained in ST_OpcUAServerInfo [▶ 148].

5.1.1.3 E_OpcUAServerStatus

E_OpcUAServerStatus represents the runtime status of a TwinCAT OPC UA Server.

Syntax

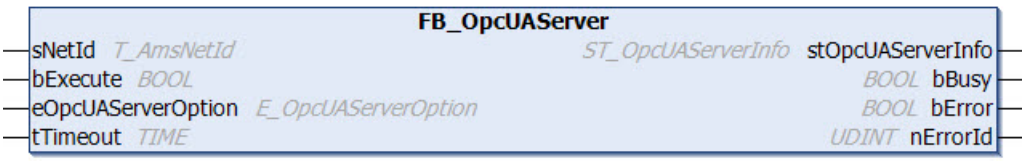
```
TYPE E_OpcUAServerStatus
(
    eOPCUAServerStatus_None,
    eOPCUAServerStatus_Alive,
    eOPCUAServerStatus_NotResponding
);
END_TYPE
```

Parameter

Name	Description
eOPCUAServerStatus_None	Initial state of the enumeration.
eOPCUAServerStatus_Alive	The ADS interface of the TwinCAT OPC UA Server is accessible.
eOPCUAServerStatus_NotResponding	The ADS interface of the TwinCAT OPC UA Server is not accessible.

5.1.2 Function blocks

5.1.2.1 FB_OpcUAServer



The function block enables status information to be read out and a TwinCAT OPC UA Server to be restarted.

Syntax

Definition:

```
FUNCTION_BLOCK FB_OpcUAServer
VAR_INPUT
    sNetId             : T_AmsNetId;
    bExecute           : BOOL;
    eOpcUAServerOption : E_OpcUAServerOption;
    tTimeout           : TIME;
END_VAR
```

```
VAR_OUTPUT
    stOpcUAServerInfo : ST_OpcUAServerInfo;
    bBusy             : BOOL;
    bError            : BOOL;
    nErrorId          : UDINT;
END_VAR
```

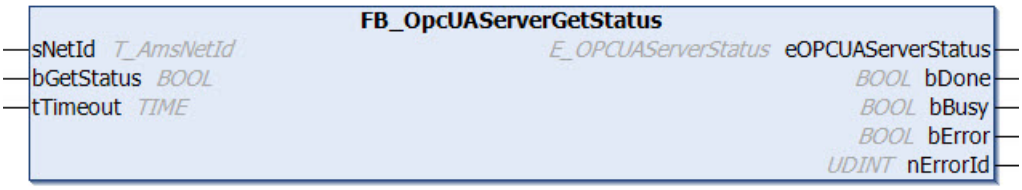
Inputs

Name	Type	Description
sNetId	T_AmsNetId	AmsNetId of the system on which the TwinCAT OPC UA Server runs.
bExecute	BOOL	A rising edge activates processing of the function block.
eOpcUAServerOption	E_OpcUAServerOption [▶ 148]	Specifies the operation to be performed.
tTimeout	TIME	ADS Timeout

Outputs

Name	Type	Description
stOpcUAServerInfo	ST_OpcUAServerInfo [▶ 148]	Contains status information from the server when ServerInfo is selected at the eOpcUAServerOption input.
bBusy	BOOL	TRUE as long as processing of the function block is in progress.
bError	BOOL	Becomes TRUE as soon as an error situation occurs.
nErrorId	UDINT	Contains the error code when an error (bError) occurs.

5.1.2.2 FB_OpcUAServerGetStatus



The function block enables the current status (Running, NotResponding) of a TwinCAT OPC UA Server to be read. It should be noted at this point that this function block deals with the ADS interface of the OPC UA Server. If the OPC UA Server is restarted or shut down, the ADS interface of the server remains accessible. The ADS interface can only be closed by terminating the server process.

Syntax

Definition:

```
FUNCTION_BLOCK FB_OpcUAServerGetStatus
VAR_INPUT
    sNetId             : T_AmsNetId;
    bGetStatus         : BOOL;
    tTimeout           : TIME;
END_VAR
VAR_OUTPUT
    eOPCUAServerStatus : E_OPCUAServerStatus;
    bDone              : BOOL;
    bBusy              : BOOL;
    bError             : BOOL;
    nErrorId           : UDINT;
END_VAR
```

Inputs

Name	Type	Description
sNetId	T_AmsNetId	AmsNetId of the system on which the TwinCAT OPC UA Server runs.
bGetStatus	BOOL	A rising edge activates processing of the function block.
tTimeout	TIME	ADS Timeout

Outputs

Name	Type	Description
eOPCUAServerStatus	E_OpcUAServerStatus [▶ 149]	Contains status information about the server.
bDone	BOOL	TRUE when processing of the function block is complete.
bBusy	BOOL	TRUE as long as processing of the function block is in progress.
bError	BOOL	Becomes TRUE as soon as an error situation occurs.
nErrorId	UDINT	Contains the error code when an error (bError) occurs.

5.2 Tc3_OpcUa

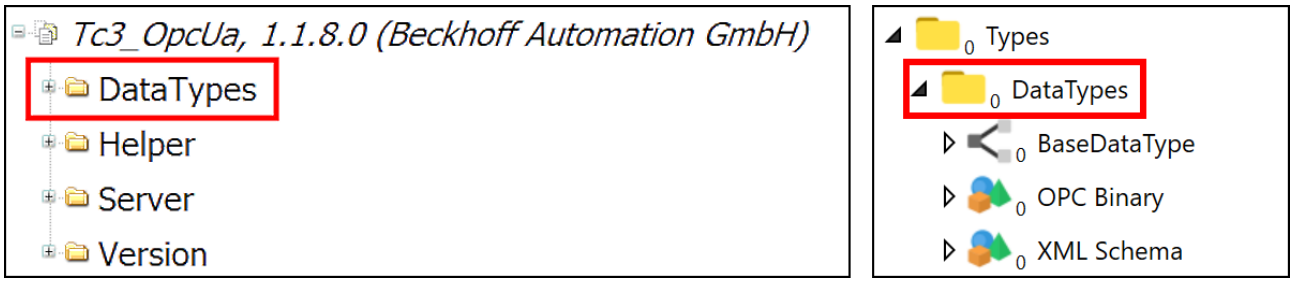
This PLC library contains data types, function blocks and more for use with TwinCAT OPC UA and consists of the following folders at the top level:

Table 1: Folder structure

Order name	Content description
DataTypes	Contains PLC types that map OPC UA data types (UA_DataType).
Helper	Contains PLC types that support the use of OPC UA data types.
Server	Contains PLC types (and more) for use with the TwinCAT OPC UA Server (TF6100).
Version	Contains version information from the PLC library.

5.2.1 Data types

This folder (left image) contains PLC types that map OPC UA data types (UA_DataType) and represents the node "DataTypes" [i=90] from the namespace http://opcfoundation.org/UA/ (right image).



5.2.1.1 BaseDataType

This folder represents the "BaseDataType" node [i=24] and contains PLC types that map its subtypes. BaseDataType itself does not have an equivalent PLC type.

5.2.1.1.1 UA_Boolean

This PLC data type maps the node "Boolean" [i=1]: reference.opcfoundation.org
This built-in data type defines a value that is either TRUE or FALSE.

Syntax

```
TYPE UA_Boolean : BOOL;
END_TYPE
```

5.2.1.1.2 UA_Byte

This PLC data type maps the node "Byte" [i=3]: reference.opcfoundation.org
This built-in data type defines a value that is either TRUE or FALSE.

Syntax

```
TYPE UA_Byte : USINT;
END_TYPE
```

5.2.1.1.3 UA_ByteString

This PLC data type maps the node "ByteString" [i=1]: reference.opcfoundation.org
This built-in data type defines a value that represents a sequence of byte values.

Syntax

```
TYPE UA_ByteString :
STRUCT
    nLength : UDINT;
    aData : ARRAY [0..255] OF BYTE;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
nLength	UDINT	Number of bytes used in aData
aData	ARRAY [0..255] OF BYTE	User data of the byte string

5.2.1.1.4 UA_DateTime

This PLC data type maps the node "DateTime" [i=13]: reference.opcfoundation.org
This built-in data type defines a date and time in the Gregorian calendar.

Syntax

```
TYPE UA_DateTime : DT;
END_TYPE
```

5.2.1.1.5 UA_Double

This PLC data type maps the node "Double" [i=11]: reference.opcfoundation.org
This built-in data type defines a floating point value according to the ISO/IEC/IEEE 60559:2020 definition with double accuracy (64 bits).

Syntax

```
TYPE UA_Double : LREAL;
END_TYPE
```

5.2.1.1.6 UA_Float

This PLC data type maps the node "Float" [i=10]: reference.opcfoundation.org
This built-in data type defines a floating point value according to the ISO/IEC/IEEE 60559:2020 definition with single accuracy (32 bits).

Syntax

```
TYPE UA_Float : REAL;
END_TYPE
```

5.2.1.1.7 UA_Int16

This PLC data type maps the node "Int16" [i=4]: reference.opcfoundation.org
This built-in data type defines a signed integer between -32,768 and 32,767.

Syntax

```
TYPE UA_Int16 : INT;
END_TYPE
```

5.2.1.1.8 UA_Int32

This PLC data type maps the node "Int32" [i=6]: reference.opcfoundation.org
This built-in data type defines a signed integer between -2,147,483,648 and 2,147,483,647.

Syntax

```
TYPE UA_Int32 : DINT;
END_TYPE
```

5.2.1.1.9 UA_Int64

This PLC data type maps the node "Int64" [i=8]: reference.opcfoundation.org
This built-in data type defines a signed integer between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807.

Syntax

```
TYPE UA_Int64 : LINT;
END_TYPE
```

5.2.1.1.10 UA_LocaleId

This PLC data type maps the node "LocaleId" [i=295]: reference.opcfoundation.org
This simple data type is a character string consisting of a language code and an optional country code in accordance with RFC 5646 (e.g. "de-DE").

Syntax

```
TYPE UA_LocaleId : STRING(6);
END_TYPE
```

5.2.1.1.11 UA_LocalizedText

This PLC data type maps the node "LocalizedText" [i=21]: reference.opcfoundation.org
This built-in data type defines a structure consisting of a string/text and a locale identifier for localized translations.

Syntax

```
{attribute 'pack_mode' := '1'}
TYPE UA_LocalizedText :
STRUCT
    sLocaleId : UA_LocaleId;
    {attribute 'TcEncoding' := 'UTF-8'}
    sText : STRING(256);
END_STRUCT
END_TYPE
```

Parameters

Name	Type	Description
sLocaleId	Tc3_OpcUa.UA_LocaleId	Locale identifier
sText	STRING(256)	Localized string/text

5.2.1.1.12 UA_NodeId

This PLC data type maps the node "NodeId" [i=17]: reference.opcfoundation.org
This built-in data type consists of three elements (namespace index, ID type, identifier) that uniquely identify a node within a server.

Syntax

```
{attribute 'pack_mode' := '1'}
TYPE UA_NodeId :
STRUCT
    xml : UA_String;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
xml	Tc3_OpcUa.UA_String	Complete NodeId encoded as a string e.g. ns=5;i=1234

5.2.1.1.13 UA_SByte

This PLC data type maps the node "SByte" [i=2]: reference.opcfoundation.org
This built-in data type defines a signed integer between -128 and 127.

Syntax

```
TYPE UA_SByte : SINT;
END_TYPE
```

5.2.1.1.14 UA_String

This PLC data type maps the node "String" [i=12]: reference.opcfoundation.org
This built-in data type defines a Unicode string that should not contain any control characters other than spaces.

Syntax

```
TYPE UA_String : STRING;
END_TYPE
```

5.2.1.1.15 UA_UInt16

This PLC data type maps the node "UInt16" [i=5]: reference.opcfoundation.org
This built-in data type defines an unsigned integer between 0 and 65,535.

Syntax

```
TYPE UA_UInt16: UINT;
END_TYPE
```

5.2.1.1.16 UA_UInt32

This PLC data type maps the node "UInt32" [i=7]: reference.opcfoundation.org
This built-in data type defines an unsigned integer between 0 and 4,294,967,295.

Syntax

```
TYPE UA_UInt32 : UDINT
END_TYPE
```

5.2.1.1.17 UA_UInt64

This PLC data type maps the node "UInt64" [i=9]: reference.opcfoundation.org
This built-in data type defines an unsigned integer between 0 and 18,446,744,073,709,551,615.

Syntax

```
TYPE UA_UInt64 : ULINT
END_TYPE
```

5.2.1.1.18 UA_UtcTime

This PLC data type maps the node "UtcTime" [i=294]: reference.opcfoundation.org
This simple data type is a DateTime value for representing coordinated universal time (UTC), which is used exclusively in OPC UA communication.

Syntax

```
TYPE UA_UtcTime : UA_DateTime
END_TYPE
```

5.2.1.1.19 Structure

This folder represents the "Structure" node [i=22] and contains PLC types that map its subtypes. "Structure" itself does not have an equivalent PLC type. Concrete OPC UA structure types derived from "Structure" can be mapped as PLC data types:

Mapping of an OPC UA structure type as a PLC data type:

Requirements

1. The data structure in the PLC matches the OPC UA data structure. In particular:
   - Sequence of fields
   - Field labels (see Restrictions)
   - Data types
2. The PLC type definition contains the OPC UA type information as attributes.
   The TwinCAT OPC UA Server requires these attributes in order to assign the PLC data type to the correct OPC UA data type. Simple PLC data types do not require these attributes.

The required attributes can be written manually above the instance or type declaration or retrieved for copying via the TwinCAT OPC UA Nodeset Editor.

Step-by-step with the Nodeset Editor:

1 - Finding and opening the structure type.

5 - Creating the DUT in TwinCAT.

2 – Selection and copying of all names and DataType BrowseNames of the fields

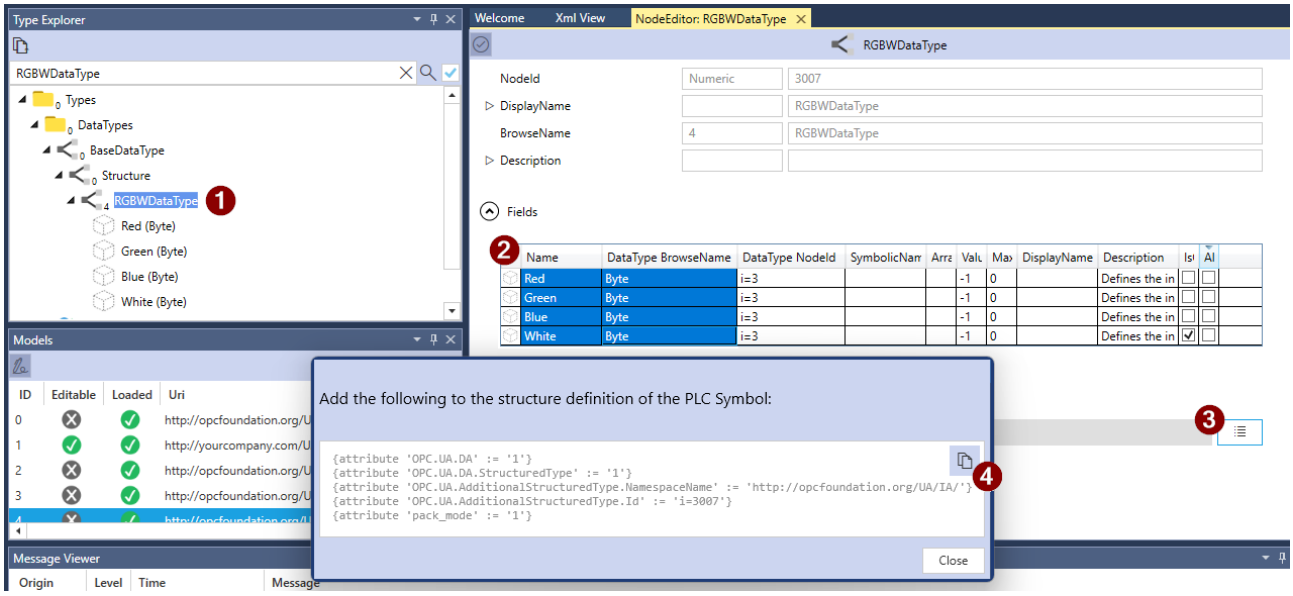
5 - Insert in TwinCAT between STRUCT and END_STRUCT

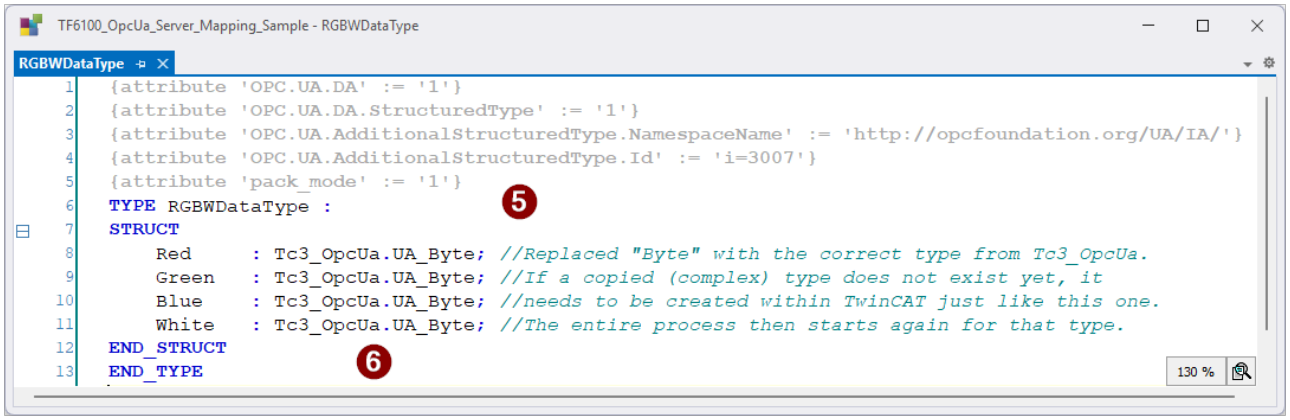
3 - Opening of the pragmas for the type

4 - Copying the pragmas

5 - Inserting in TwinCAT above TYPE

6 - Replacing the DataType BrowseNames with the appropriate PLC types
(e.g. Boolean => "Tc3_OpcUa.UA_Boolean" or "BOOL").
Missing (structure) types may also need to be created.





Meaning of the attributes:

- **OPC.UA.DA**
  Activates the provision of the data type and any sub-elements via OPC UA.
- **OPC.UA.DA.StructuredType**
  Identifies the type as a structured OPC UA data type.
- **OPC.UA.AdditionalStructuredType.NamespaceName**
  Specifies the NamespaceUri of the data type from the nodeset.
- **OPC.UA.AdditionalStructuredType.Id**
  Contains the NodeId of the OPC UA data type, in this case i=3007.
- **pack_mode := '1'**
  Ensures that the memory layout of the structure is defined and compatible.

5.2.1.1.19.1 UA_EUInformation

This PLC data type maps the node "EUInformation" [i=887]: reference.opcfoundation.org
This structured data type contains information about units of measurement (e.g. namespace URI, unit ID, display name and description).

Syntax

```
{attribute 'pack_mode' := '1'}
TYPE UA_EUInformation :
STRUCT
    sNamespaceURI:  UA_String;
    nUnitId:        UA_Int32;
    sDisplayName:   UA_LocalizedText;
    sDescription:   UA_LocalizedText;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
sNamespaceURI	Tc3_OpcUa.UA_String	URI of the organization that defines this unit of measurement
nUnitId	Tc3_OpcUa.UA_Int32	Numerical identifier for programmatic evaluation
sDisplayName	Tc3_OpcUa.UA_LocalizedTExt	Usually abbreviation of the unit of measurement
sDescription	Tc3_OpcUa.UA_LocalizedTExt	Full (written in full) name of the unit of measurement

5.2.1.1.19.2 UA_OptionSet

This PLC data type maps the "OptionSet" node [i=12755]: reference.opcfoundation.org
This abstract data type is the base type for all data types that represent a bit mask with more than 64 bits or where the validity of individual bits must be marked.

Syntax

```
TYPE UA_OptionSet :
STRUCT
END_STRUCT
END_TYPE
```

Using UA_OptionSet

To define a specific OptionSet and use it with TF6100 (Server) and TE6100 (Nodeset Editor), it must be derived from Tc3_OpcUa.UA_OptionSet.

Sample implementation of a specific option set:

```
TYPE UA_OptionSet_Sample EXTENDS UA_OptionSet :
STRUCT
    BitOne : UA_OptionSet_Option;
    BitTwo : UA_OptionSet_Option;
END_STRUCT
END_TYPE
```

5.2.1.1.19.2.1 UA_OptionSet_Option

This PLC data type is an auxiliary data type for defining your own OptionSet data types.
This data type maps an option (a bit) in an OptionSet (a bit mask) including a validity specification.

Syntax

```
TYPE UA_OptionSet_Option :
STRUCT
    value : BOOL;
    valid : BOOL;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
value	BOOL	Value of this option
Valid	BOOL	Validity specification of this option

5.2.1.1.19.3 UA_Range

This PLC data type maps the "Range" node [i=884]: reference.opcfoundation.org
This structured data type defines a value range with a lower (low) and an upper (high) double value.

Syntax

```
{attribute 'pack_mode' := '1'}
TYPE UA_Range :
STRUCT
    fLow : UA_Double;
    fHigh: UA_Double;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
fLow	Tc3_OpcUa.UA_Double	Lower limit of the value range
fHigh	Tc3_OpcUa.UA_Double	Upper limit of the value range

5.2.1.1.19.4 UA_TimeZoneDataType

This PLC data type maps the node "TimeZoneDataType" [i=8912]: reference.opcfoundation.org
This structured data type defines the local time, which may or may not take daylight saving time into account.

Syntax

```
{attribute 'pack_mode' := '1'}
TYPE UA_TimeZoneDataType :
STRUCT
    nOffset                 : UA_Int16;
    bDaylightSavingInOffset : UA_Boolean;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
nOffset	Tc3_OpcUa.UA_Int16	Time offset in minutes relative to UTC time
nDaylightSavingInOffset	Tc3_OpcUa.UA_Boolean	Indication of whether the offset includes the summer time correction

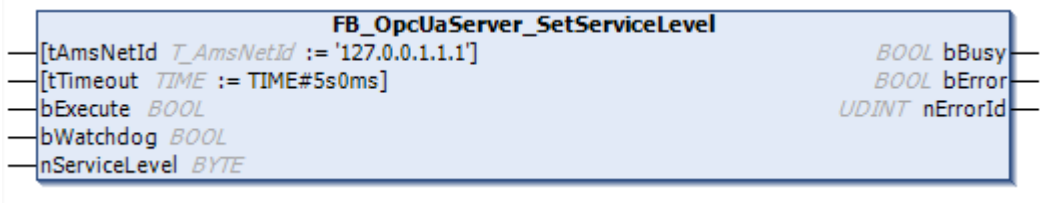
5.2.2 Function blocks

This folder contains PLC function blocks for use with TwinCAT OPC UA.

5.2.2.1 Server

This folder contains PLC function blocks for use with the TwinCAT OPC UA Server (TF6100).

5.2.2.1.1 FB_OpcUaServer_SetServiceLevel



The TwinCAT OPC UA Server currently sets the ServiceLevel statically to the value 255. This function block enables the ServiceLevel to be set via an ADS interface in the server. A distinction is made between use with and without a watchdog.

Further information on the service level can be found in the ServiceLevel Documentation of the TwinCAT OPC UA Server.

Syntax

```
FUNCTION_BLOCK FB_OpcUaServer_SetServiceLevel
VAR_INPUT
    tAmsNetId              : T_AmsNetId := '127.0.0.1.1.1';
    tTimeout               : TIME := T#5s;;
    bExecute               : BOOL;
    bWatchdog              : BOOL;
    nServiceLevel          : BYTE;
END_VAR
VAR_OUTPUT
    bBusy                  : BOOL;
    bError                 : BOOL;
    nErrorId               : UDINT;
END_VAR
```

Inputs

Name	Type	Description
tAmsNetId	T_AmsNetId	AmsNetId of the system on which the TwinCAT OPC UA Server runs. If no information is provided, the local host is used.
tTimeout	TIME	Maximum time for executing the ADS command. The setting is 5 seconds by default.
bExecute	BOOL	Optional parameter to specify a URI for the WebSocket opening handshake.
bWatchdog	BOOL	Determines whether the ServiceLevel is set with or without a watchdog.
nServiceLevel	BYTE	ServiceLevel that is set in the TwinCAT OPC UA Server.

Outputs

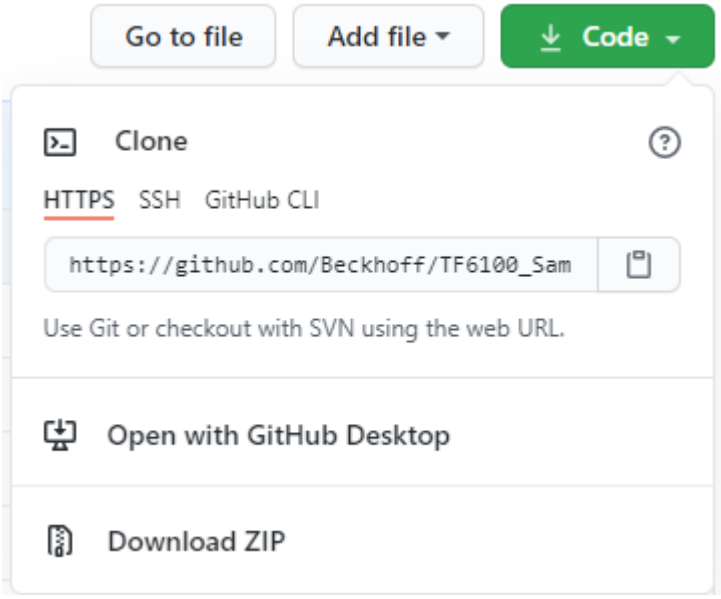
Name	Type	Description
bBusy	BOOL	Is TRUE as long as the function block is busy with processing.
bError	BOOL	Becomes TRUE when an error situation occurs.
nErrorId	UDINT	ADS error code.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4026.x	IPC or CX (x86, x64, Arm®)	Tc3_OpcUa (1.1.8 or higher)

6 Samples

Sample code and configurations for this product can be obtained from the corresponding repository on GitHub: https://github.com/Beckhoff/TF6100_Samples. There you have the possibility to clone the repository or download a ZIP file with the sample.

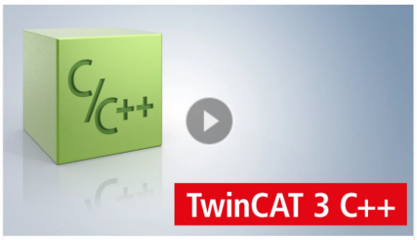


The following samples exist:

Name	TwinCAT Version	Description
TF6100_OpcUa_Client_Sample	TwinCAT 3	This sample contains sample code for various functions of the TwinCAT OPC UA Client (PLCOpen function blocks). These include Browse, Connect, HistoryUpdate, MethodCall, Read and Write. The server sample for access is also included.
TF6100_OpcUa_Server_Sample	TwinCAT 3	This sample contains a PLC with extensive provision of PLC data for the TwinCAT OPC UA Server (OPC UA Data Access).
TS6100_OpcUa_Client_Sample	TwinCAT 2	This sample contains sample code for various functions of the TwinCAT OPC UA Client (PLCOpen function blocks). These include MethodCall, Read and Write.
TS6100_OpcUa_Server_Sample	TwinCAT 2	This sample demonstrates the use of PLC comments to release variables via the TwinCAT OPC UA Server.

7 Tutorials

Video tutorials for this product can be found on our website at https://www.beckhoff.com/tutorials. The video tutorials offer a quick start to the product and the individual product facets.



14.03.2024 | Multimedia
Tutorial: TwinCAT 3 C++ overview
Get an overview of the C++ integration into TwinCAT.
Learn more →



12.02.2024 | Multimedia
Tutorial: Chart synchronization in TwinCAT Scope
Learn how to dock and synchronize charts in TwinCAT Scope.
Learn more →



12.02.2024 | Multimedia
Tutorial: Getting started with TF6100 TwinCAT 3 OPC UA Server
Learn how to configure the TwinCAT OPC UA Server.
Learn more →

8 Appendix

8.1 Attributes and comments

The following table provides an overview of all pragmas and comments that can be configured in the server. These can be defined in the various real-time environments of TwinCAT to enable different functionalities. A detailed description of the use of attributes and comments can be found in chapter Enabling symbols [▶ 59].

Using the tags in the PLC

When using the pragmas in the PLC, please ensure that you place the key and value in quotation marks. You can find an example in the chapter Enabling symbols\PLC [▶ 60].

TwinCAT 3

Key	Value	Meaning
OPC.UA.DA	0	Blocks a symbol explicitly for OPC UA.
OPC.UA.DA	1	Enables a symbol for OPC UA.
OPC.UA.DA	2	Enables a symbol for OPC UA. In the case of a structure and the StructuredDataType, member variables are not loaded into the server's address space as separate nodes.
OPC.UA.DA.Access	1	Sets a node read-only [▶ 91].
OPC.UA.DA.Access	2	Sets a node write-only.
OPC.UA.DA.Access	3	Enables read/write access to a node (default if not set).
OPC.UA.DA.Alias	<string>	Defines a different name (alias) for a node.
OPC.UA.DA.Description	<string>	Defines the content of the OPC UA attribute "Description [▶ 90]".
OPC.UA.DA.StructuredType	0	Disables the StructuredDataType for a structure [▶ 82].
OPC.UA.DA.StructuredType	1	Enables the StructuredDataType for a structure [▶ 82].
OPC.UA.DA.Status	quality	Manually define the StatusCode [▶ 86] of a symbol in the OPC UA namespace.

TwinCAT 2

PLC comment	Meaning
(*~ (OPC:0:not available) *)	Locks a variable for OPC UA, whereupon it is no longer visible in the UA namespace.
(*~ (OPC:1:available) *)	Enables a variable for OPC UA, whereupon it becomes visible in the UA namespace. This tag must always be set if you want to use a variable for UA.
(*~ (OPC_PROP[0005]:1:read-only) *)	Sets the write protection for a variable. Must be used together with (*~ (OPC: 1: available) *).
(*~ (OPC_UA_PROP[5100] : x: Alias name) *)	Specifies x as node name in the UA namespace, so-called alias mapping.
(*~ (OPC_UA_PROP[5000]:x:Storage media) *)	Enables a variable for "Historical Access". Must be used together with (*~ (OPC: 1: available) *). x defines the storage medium for storing the data values: 1 = RAM

PLC comment	Meaning
	2 = File 3 = SQL Compact Database 4 = SQL Server Database
(*~ (OPC_UA_PROP[5000][1]:x:SamplingRate) *)	Determines the sampling rate at which the variable values are to be stored, in [ms] depending on the parameter "x"
(*~ (OPC_UA_PROP[5000][2]:x:Buffer) *)	Defines the maximum number of values that remain in the data memory, depending on the parameter "x".

8.2 32-bit and 64-bit process

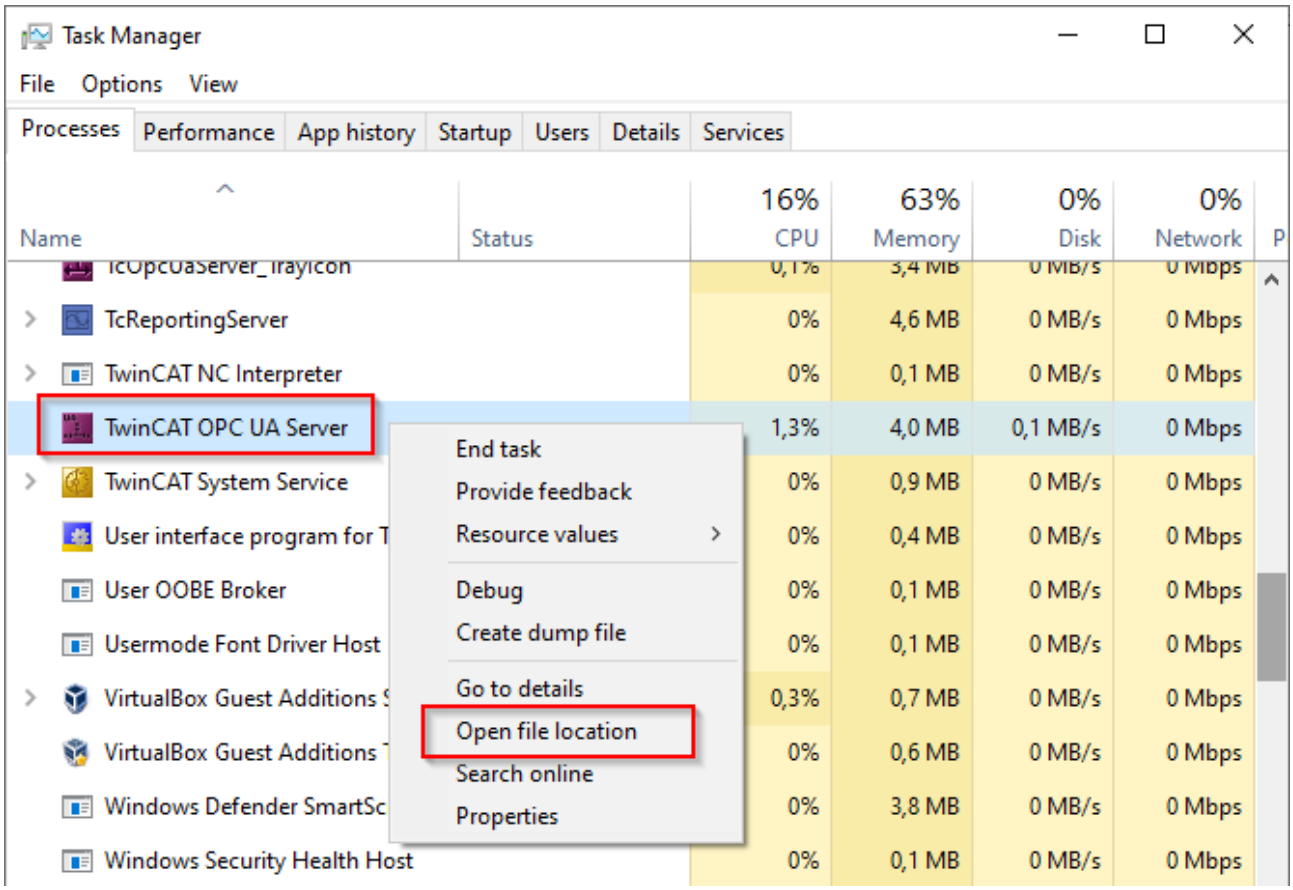
The TwinCAT OPC UA Server is available as a 32-bit and 64-bit process. The corresponding files are installed automatically via the setup or package.

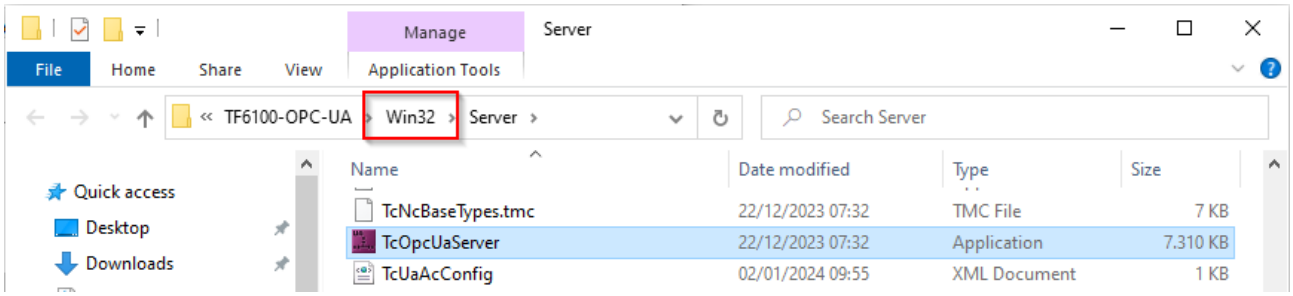
Which variant is active on your system?

✓ A little trick in the Windows Task Manager allows you to easily recognize which variant is currently active on your system.

1. Open the Task Manager.
2. Navigate to the "TwinCAT OPC UA Server" process.
3. Open the file location via the context menu.

⇨ In the address bar of the File Explorer you can now see which variant of TcOpcUaServer.exe is currently registered on your system and is therefore active:





Register/unregister

To register or unregister another variant of the TwinCAT OPC UA Server, please follow the steps below.



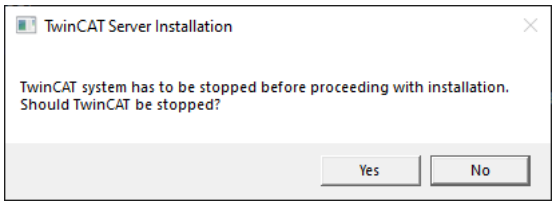
Before changing the variant, please note that you must first end the running process.

Switch from 32-bit to 64-bit process:

- End the running process of the TcOpcUaServer.exe file in the Task Manager.
- Open the Windows command prompt as an administrator and change to the installation directory [▶ 54] of the 32-bit version.
- Enter the following command to unregister the process:

```
TcOpcUaServer.exe /UnRegTcServer2
```

A dialog box informs you that the TwinCAT system service must be restarted for unregistration. Confirm this dialog box with **Yes**.



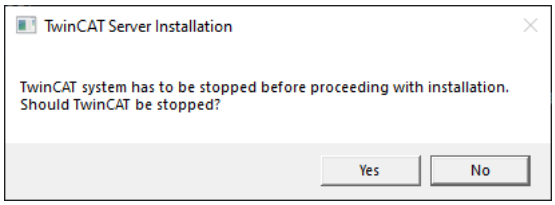
As a result of this command, the TwinCAT OPC UA Server is no longer started automatically with TwinCAT.

In the next step, you will learn how to register the 64-bit variant of the server with the TwinCAT system service.

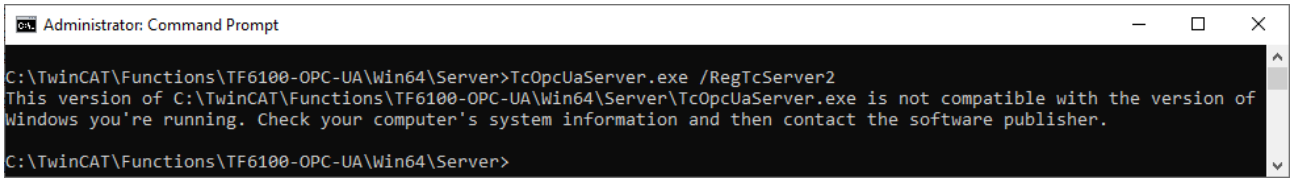
- Change to the server installation directory [▶ 54] of the 64-bit variant and enter the following command to register the process:

```
TcOpcUaServer.exe /RegTcServer2
```

A dialog box informs you once again that the TwinCAT system service must be restarted for unregistration. Confirm this dialog box with "Yes".



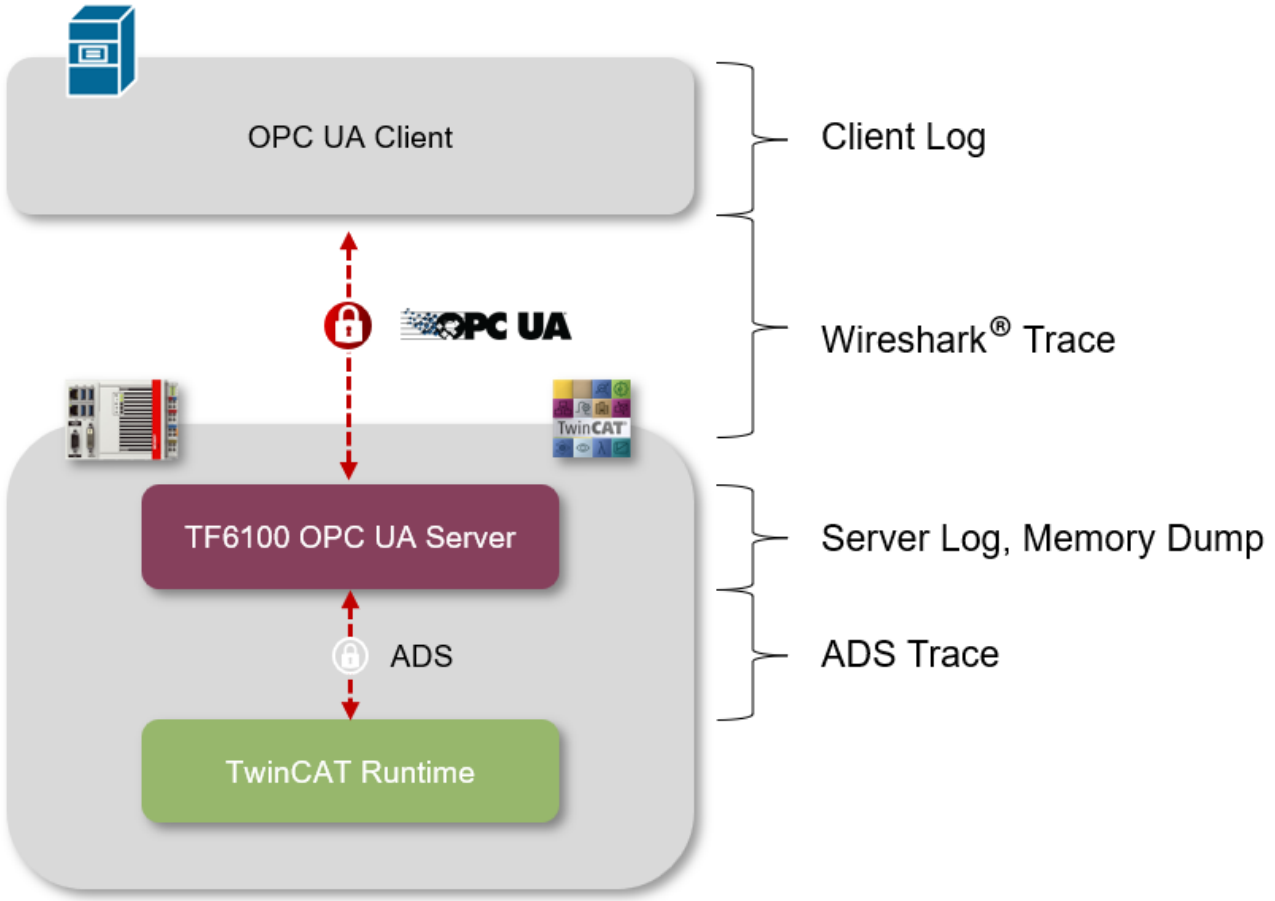
As a result of this command, the TwinCAT OPC UA Server is registered with the TwinCAT system service and starts automatically together with TwinCAT. If you get the error message when entering this command, make sure that you are not inadvertently trying to register the 64-bit process on a 32-bit system.



Conversely, the 32-bit process can of course be registered on a 64-bit system.

8.3 Error diagnosis

In the event of undesirable operating behavior, an extended diagnosis may be necessary. Depending on the situation, this may include the following measures: ADS Trace, Wireshark® Trace, log file, Memory Dump. A log function may also be available and useful on the client side. The appropriateness of the action depends heavily on the appearance of the behavior. Take another look at the software architecture of the TwinCAT OPC UA Server and compare it with the operating behavior. The following figure illustrates where the individual measures (described in more detail in the table) could be helpful.



Measure	Use case	Link
Client Log	Useful for recording extended protocol functions on the client side.	Please contact the manufacturer of the client for more information about its logging functions.
Wireshark® Trace	Useful for examining communication between client and server.	https://www.wireshark.org/
Server Log	Useful for logging advanced server-side logging functions.	See chapter Logging [▶ 145] in this documentation.
Memory Dump	It can be useful to create a Memory Dump in case the TwinCAT OPC UA Server is	Microsoft Debug Diagnostics

Measure	Use case	Link
	unexpectedly terminated. The tools required for this depend on the operating system.	
ADS Trace	Useful for examining the communication between server and PLC.	Beckhoff Download Finder (TF6010 TC3 ADS Monitor)

Support

The Beckhoff support department [▶ 171] will be happy to assist you in implementing the appropriate measures.

The following table provides an overview of possible unexpected operating behavior and measures to resolve it.

Behavior	Action(s)
An OPC UA client does not see the PLC namespace.	TF6100 Setup Version 3.x and older: this status indicates a missing license. Check whether you have activated a valid TF6100 license.
An OPC UA client is assigned the StatusCode 0x810e0000 when reading nodes.	TF6100 Setup Version 4.x: this status indicates a missing license. Check whether you have activated a valid TF6100 license.
The variables enabled via comments/attributes are not displayed in the OPC UA server.	Check whether the symbol file has been correctly transferred to the controller (e.g., check boxes in the PLC project), verify that it exists in the boot directory and that the path to the symbol file in the configuration file of the server refers to the correct symbol file. You can also use the DeviceState Node in the respective namespace to check any error messages that may have occurred. An entry is made here if the symbol file was not found. Also check that the comments/attributes are spelled correctly.
The server does not comply with the sampling rate/publishing interval required by the OPC UA Client.	OPC UA client/server is not a real-time protocol, i.e., there is no guarantee that the server will always meet 100% of the sampling rate or publishing interval required by the client. The available sampling rates and publishing intervals can be viewed in the server configuration file and modified if required (<AvailableSamplingRates> and <MinPublishingInterval>).
An OPC UA Client cannot connect to the server although the server is displayed in the Windows Task Manager. The error message “Host unreachable” (or similar) is displayed.	Check whether firewall settings prevent communication with the server. The server port must be open for incoming TCP communication so that a client can connect.
An OPC UA client sees the server’s endpoints, but a connection with them fails with the error message “Host unreachable”.	Check that the name resolution in your network is working properly and that the server is accessible under its host name. Even if the OPC UA Client apparently connects to the IP address of the server (e.g., opc.tcp://192.168.0.1:4840) to access the server’s endpoints, the server always returns its own host name in its endpoints. If the client connects directly to one of the endpoints, it will use the host name of the server again. If the name resolution does not work, the connection fails.
An OPC UA Client sees the endpoints of the server, but a connection to a secure endpoint fails. The error message “BadSecurityChecksFailed” is displayed.	Check whether the server trusts the client certificate. The required configuration steps can be found in section Certificate exchange. In this case, it must also be ensured that a signature hash algorithm of the SHA2 group (SHA256, SHA364, SHA512) is used to sign the client

Behavior	Action(s)
	certificate. If deprecated algorithms such as SHA1 are used, the TwinCAT OPC UA Server does not allow a connection.
When using an SQL server to store Historical Access information, the values are not added to the SQL database.	Check the access data to the SQL Server and verify that the SQL Server is also accessible in the network. Also make sure that you are using a “Big Windows” operating system on the TwinCAT OPC UA Server, since SQL Server cannot be used for Historical Access under Windows CE (although SQL Compact is OK).
When reading nodes, an OPC UA client receives the error message “BadDeviceFailure”.	This is an indication that the associated ADS device cannot be reached, for example if no PLC program has been started. Check the connectivity with the ADS device and make sure that the appropriate runtime is active.
When reading nodes, an OPC UA client receives the error message “BadLicenseExpired”.	This is an indication that no TF6100 license is active on the system or that it has expired. Please make sure that you have activated a TF6100 license on the device on which the server was installed.
Arrays are not displayed in the namespace with full resolution.	By default, arrays of simple data types are not displayed in expanded form in the namespace. However, individual array indices can still be addressed using the IndexRange function of OPC UA. An OPC UA Client should therefore support this function. If this is not the case, a radio button in the configuration file of the server can be used to display an array in expanded form, so that each individual array element can be addressed as a separate node. This radio button is described in the chapter Arrays [▶ 66].
The following message appears repeatedly in the error log: “CBkUaPlcSamplingEngine_Poll : !!CRITICAL!! Sampling exceeded n times in a row, leaving no time left to rest.”	This message is an indication that the server cannot process the variables added by the client to a subscription at the requested sampling rate. The reason for this is typically an excessive amount of data, e.g. when sampling large data structures (StructuredTypes [▶ 82]). Please note that this message does not necessarily represent a problem. In many OPC UA Client applications, the sampling rate settings are generalized and made for all variables, even if they could be sampled more slowly. In any case, a first step towards optimization should always include checking and, if necessary, reducing the sampling rate. It should also be checked whether the amount of data can be reduced if large structured types are used. Upgrading to a faster CPU is also helpful. However, experience has shown that reducing the sampling rate and the amount of data is sufficient to achieve initial success.
The call of a job method [▶ 112] fails with the status code “BadResourceUnavailable”	If a job method returns with this status code, this is an indication that the signature of the function block representing the job method has not been set correctly. First check the presence of the Start() method. Further note: in this case, you would see in an ADS trace that the associated call (AdsReadWriteRequest) of the method returns the result 0x710 (“Symbol not found”).
During TOFU initialization on an UMRT system, an "Access denied (not root)" error is displayed.	The TwinCAT OPC UA Server is started on an UMRT system with restricted authorizations, which can prevent the TOFU user being entered in the user database. In this case, please add the TOFU user to the user database yourself, as described in our article on initialization [▶ 40].

8.4 ADS Return Codes

Grouping of error codes:
Global error codes: 0x0000 [▶ 168]... (0x9811_0000 ...)
Router error codes: 0x500 [▶ 168]... (0x9811_0500 ...)
General ADS errors: 0x700 [▶ 169]... (0x9811_0700 ...)
RTime error codes: 0x1000 [▶ 170]... (0x9811_1000 ...)

Global error codes

Hex	Dec	HRESULT	Name	Description
0x0	0	0x98110000	ERR_NOERROR	No error.
0x1	1	0x98110001	ERR_INTERNAL	Internal error.
0x2	2	0x98110002	ERR_NORTIME	No real time.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Allocation locked – memory error.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Mailbox full – the ADS message could not be sent. Reducing the number of ADS messages per cycle will help.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Wrong HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Target port not found – ADS server is not started, not reachable or not installed.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Target computer not found – AMS route was not found.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unknown command ID.
0x9	9	0x98110009	ERR_BADTASKID	Invalid task ID.
0xA	10	0x9811000A	ERR_NOIO	No IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unknown AMS command.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 error.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port not connected.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Invalid AMS length.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Invalid AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installation level is too low –TwinCAT 2 license error.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	No debugging available.
0x12	18	0x98110012	ERR_PORTDISABLED	Port disabled – TwinCAT system service not started.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port already connected.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 error.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync error.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	No index map for AMS Sync available.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Invalid AMS port.
0x19	25	0x98110019	ERR_NOMEMORY	No memory.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP send error.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host unreachable.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Invalid AMS fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS send error – secure ADS connection failed.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Access denied – secure ADS access denied.

Router error codes

Hex	Dec	HRESULT	Name	Description
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Locked memory cannot be allocated.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	The router memory size could not be changed.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	The mailbox has reached the maximum number of possible messages.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	The Debug mailbox has reached the maximum number of possible messages.
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	The port type is unknown.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	The router is not initialized.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	The port number is already assigned.

Hex	Dec	HRESULT	Name	Description
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	The port is not registered.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	The maximum number of ports has been reached.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	The port is invalid.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	The router is not active.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	The mailbox has reached the maximum number for fragmented messages.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	A fragment timeout has occurred.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	The port is removed.

General ADS error codes

Hex	Dec	HRESULT	Name	Description
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	General device error.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service is not supported by the server.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Invalid index group.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Invalid index offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Reading or writing not permitted. Several causes are possible. For example, an incorrect password was entered when creating routes.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parameter size not correct.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Invalid data values.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Device is not ready to operate.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Device is busy.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Invalid operating system context. This can result from use of ADS blocks in different tasks. It may be possible to resolve this through multitasking synchronization in the PLC.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Insufficient memory.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Invalid parameter values.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Not found (files, ...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax error in file or command.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objects do not match.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Object already exists.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol not found.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Invalid symbol version. This can occur due to an online change. Create a new handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Device (server) is in invalid state.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode not supported.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification handle is invalid.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification client not registered.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	No further handle available.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Notification size too large.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Device not initialized.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Device has a timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface query failed.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Wrong interface requested.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class ID is invalid.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object ID is invalid.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Request pending.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Request is aborted.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal warning.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Invalid array index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol not active.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Access denied. Several causes are possible. For example, a unidirectional ADS route is used in the opposite direction.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Missing license.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	License expired.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	License exceeded.

Hex	Dec	HRESULT	Name	Description
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Invalid license.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	License problem: System ID is invalid.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	License not limited in time.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Licensing problem: time in the future.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	License period too long.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception at system startup.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	License file read twice.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Invalid signature.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Invalid certificate.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public key not known from OEM.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	License not valid for this system ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo license prohibited.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Invalid function ID.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Outside the valid range.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Invalid alignment.
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Invalid platform level.
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Context – forward to passive level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Context – forward to dispatch level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Context – forward to real-time.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Client error.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Service contains an invalid parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling list is empty.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var connection already in use.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	The called ID is already in use.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout has occurred – the remote terminal is not responding in the specified ADS timeout. The route setting of the remote terminal may be configured incorrectly.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Error in Win32 subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Invalid client timeout value.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port not open.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	No AMS address.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Internal error in Ads sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Hash table overflow.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Key not found in the table.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	No symbols in the cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Invalid response received.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port is locked.
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	The request was canceled.

RTime error codes

Hex	Dec	HRESULT	Name	Description
0x1000	4096	0x98111000	RTERR_INTERNAL	Internal error in the real-time system.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer value is not valid.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task pointer has the invalid value 0 (zero).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack pointer has the invalid value 0 (zero).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	The request task priority is already assigned.
0x1005	4101	0x98111005	RTERR_NOMORETCB	No free TCB (Task Control Block) available. The maximum number of TCBs is 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	No free semaphores available. The maximum number of semaphores is 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	No free space available in the queue. The maximum number of positions in the queue is 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	An external synchronization interrupt is already applied.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	No external sync interrupt applied.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Application of the external synchronization interrupt has failed.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Call of a service function in the wrong context
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel® VT-x extension is not supported.

Hex	Dec	HRESULT	Name	Description
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel® VT-x extension is not enabled in the BIOS.
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Missing function in Intel® VT-x extension.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Activation of Intel® VT-x fails.

Specific positive HRESULT Return Codes:

HRESULT	Name	Description
0x0000_0000	S_OK	No error.
0x0000_0001	S_FALSE	No error. Example: successful processing, but with a negative or incomplete result.
0x0000_0203	S_PENDING	No error. Example: successful processing, but no result is available yet.
0x0000_0256	S_WATCHDOG_TIMEOUT	No error. Example: successful processing, but a timeout occurred.

TCP Winsock error codes

Hex	Dec	Name	Description
0x274C	10060	WSAETIMEDOUT	A connection timeout has occurred - error while establishing the connection, because the remote terminal did not respond properly after a certain period of time, or the established connection could not be maintained because the connected host did not respond.
0x274D	10061	WSAECONNREFUSED	Connection refused - no connection could be established because the target computer has explicitly rejected it. This error usually results from an attempt to connect to a service that is inactive on the external host, that is, a service for which no server application is running.
0x2751	10065	WSAEHOSTUNREACH	No route to host - a socket operation referred to an unavailable host.
More Winsock error codes: Win32 error codes			

8.5 Support and Service

Beckhoff and their partners around the world offer comprehensive support and service, making available fast and competent assistance with all questions related to Beckhoff products and system solutions.

Download finder

Our download finder contains all the files that we offer you for downloading. You will find application reports, technical documentation, technical drawings, configuration files and much more.

The downloads are available in various formats.

Beckhoff's branch offices and representatives

Please contact your Beckhoff branch office or representative for local support and service on Beckhoff products!

The addresses of Beckhoff's branch offices and representatives round the world can be found on our internet page: www.beckhoff.com

You will also find further documentation for Beckhoff components there.

Beckhoff Support

Support offers you comprehensive technical assistance, helping you not only with the application of individual Beckhoff products, but also with other, wide-ranging services:

- support
- design, programming and commissioning of complex automation systems
- and extensive training program for Beckhoff system components

Hotline: +49 5246 963-157

e-mail: support@beckhoff.com

Beckhoff Service

The Beckhoff Service Center supports you in all matters of after-sales service:

- on-site service
- repair service
- spare parts service
- hotline service

Hotline: +49 5246 963-460
e-mail: service@beckhoff.com

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

Phone: +49 5246 963-0
e-mail: info@beckhoff.com
web: www.beckhoff.com

Trademark statements

Beckhoff®, ATRO®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, MX-System®, Safety over EtherCAT®, TC/BSD®, TwinCAT®, TwinCAT/BSD®, TwinSAFE®, XFC®, XPlanar® and XTS® are registered and licensed trademarks of Beckhoff Automation GmbH.

Third-party trademark statements

Arm, Arm9 and Cortex are trademarks or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere.

DALI, DALI-2, D4i, DALI+, and DiiA are trademarks in various countries in the exclusive use of the Digital Illumination Interface Alliance.

Docker is a trademark or registered trademark of Docker, Inc. in the United States and/or other countries. Docker, Inc. and other parties may also have trademark rights in other terms used herein.
DSP System Toolbox, Embedded Coder, MATLAB, MATLAB Coder, MATLAB Compiler, MathWorks, Predictive Maintenance Toolbox, Simscape, Simscape™ Multibody™, Simulink, Simulink Coder, Stateflow and ThingSpeak are registered trademarks of The MathWorks, Inc.

Excel, IntelliSense, Microsoft, Microsoft Azure, Microsoft Edge, PowerShell, Visual Studio, Windows and Xbox are trademarks of the Microsoft group of companies.

MAX®, Stratix®, Cyclone®, Altera®, Agilex™, Arria®, Intel, the Intel logo, Intel Core, Xeon, Intel Atom, Celeron and Pentium are trademarks of Intel Corporation or its subsidiaries.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Wireshark is a registered trademark of Sysdig, Inc.

OPC UA is a registered trademark of the OPC Foundation.

More Information:
www.beckhoff.com/tf6100

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com