BECKHOFF New Automation Technology

Manual | EN

TS6100-0030

TwinCAT 2 | OPC UA Server CE

2023-11-20 | Version: 1.9

Inhaltsverzeichnis

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.
For installation and commissioning of the components, it is absolutely necessary to observe the documentation and the following notes and explanations.
The qualified personnel is obliged to always use the currently valid documentation.

The responsible staff must ensure that the application or use of the products described satisfies all requirements for safety, including all the relevant laws, regulations, guidelines, and standards.

Disclaimer

Trademarks

Patents

Copyright

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.

2 Overview

OPC Unified Architecture (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 webpages of the OPC Foundation.

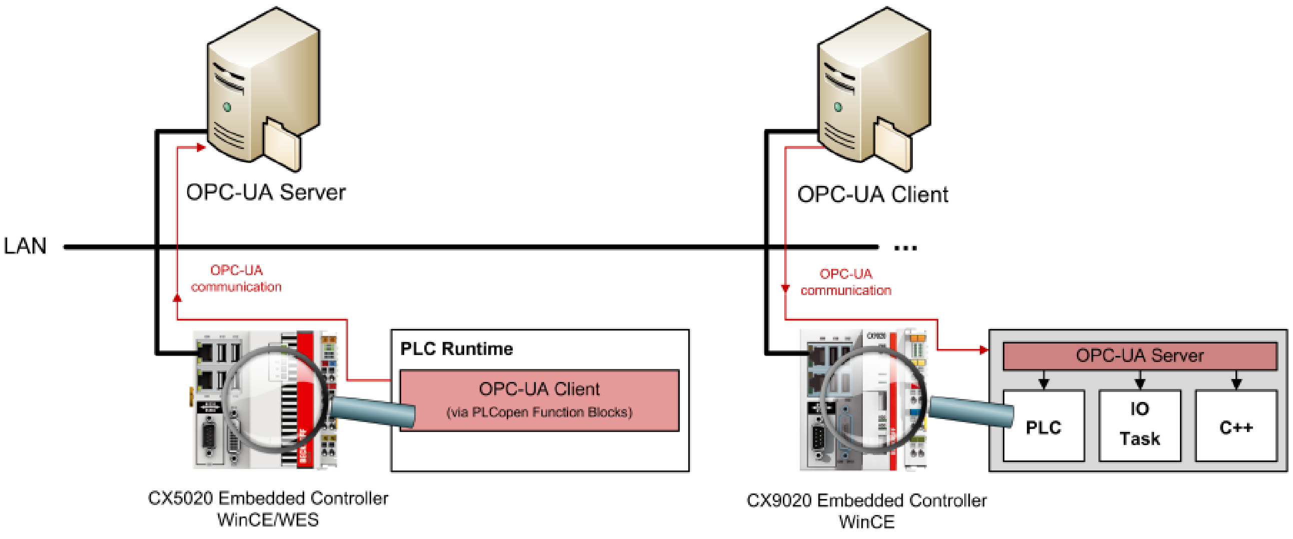
Components

The following software components have been integrated for Win32/64 and Windows CE-based systems:

Software component	Description
OPC UA Server [▸ 35]	Provides an OPC UA Server interface so that UA clients can access the TwinCAT runtime.
OPC UA Client [▸ 177]	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.

The following software components have been integrated for Win32/64-based systems:

Software component (Windows only)	Description
OPC UA Configurator	Graphical user interface for configuring the TwinCAT OPC UA Server
OPC UA Sample Client [▸ 208]	Graphical sample implementation of an OPC UA Client in order to carry out a first connection test with the TwinCAT OPC UA Server.
OPC UA Gateway [▸ 194]	Wrapper technology that provides both an OPC COM DA Server interface and OPC UA Server aggregation capabilities.



2.1 Scenarios

In the following, some scenarios will be illustrated in accordance with which the components can be implemented and used, depending on the application case and infrastructure:

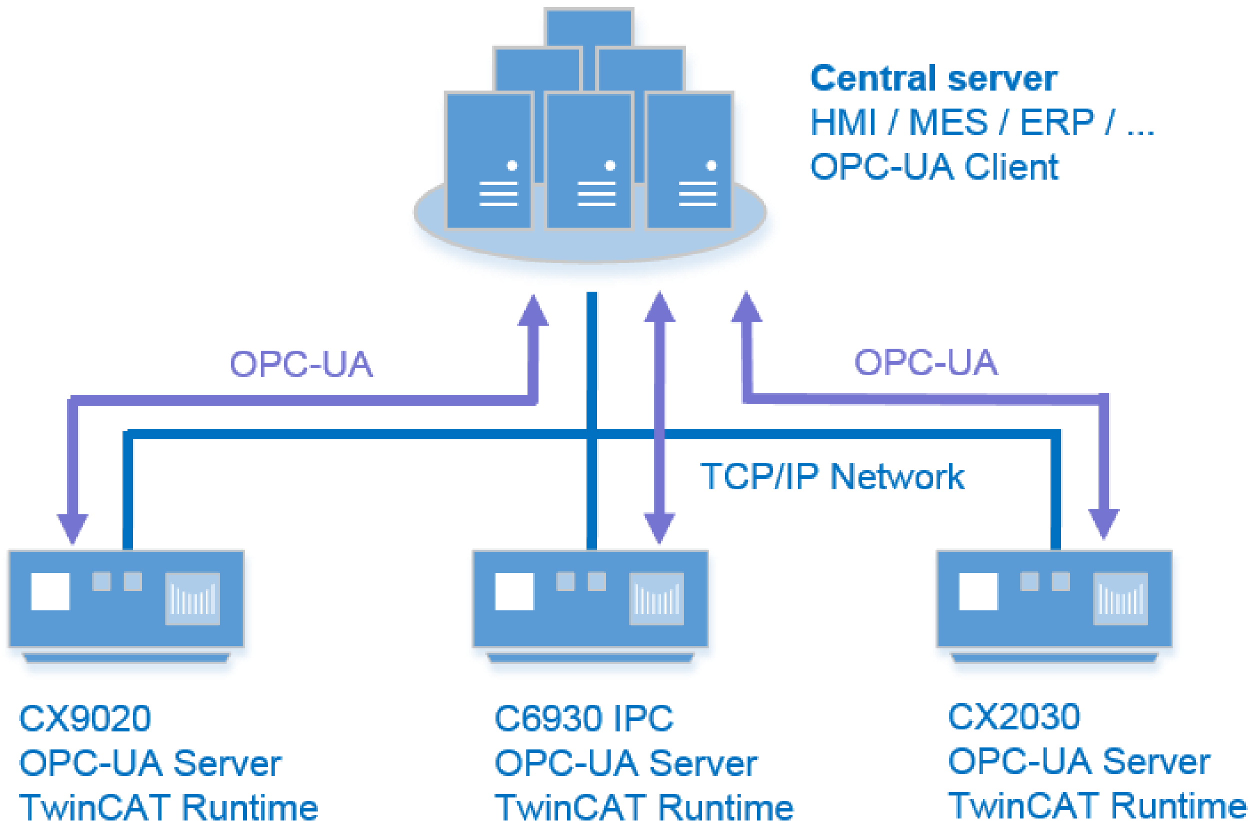
- OPC UA Server: Integrated in Industrial PC or Embedded PC [▸ 9]
- OPC UA Server: Runs on a central computer with connection to remote TwinCAT runtime(s). [▸ 9]
- OPC UA Server: Access to BC Controller [▸ 10]

OPC UA Server: Integrated in Industrial PC or Embedded PC

Recommended scenario

This scenario describes how the TwinCAT OPC UA Server should be used under normal circumstances.

One of the biggest advantages of the TwinCAT OPC UA Server 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 connect to the OPC UA Server and read or write symbol information from the TwinCAT runtime.



The following software components and configurations run on the central server:

- Third party OPC UA Client, which may be an HMI, MES or ERP system, for example.

The following software components and configurations run on the Industrial PC or Embedded PC:

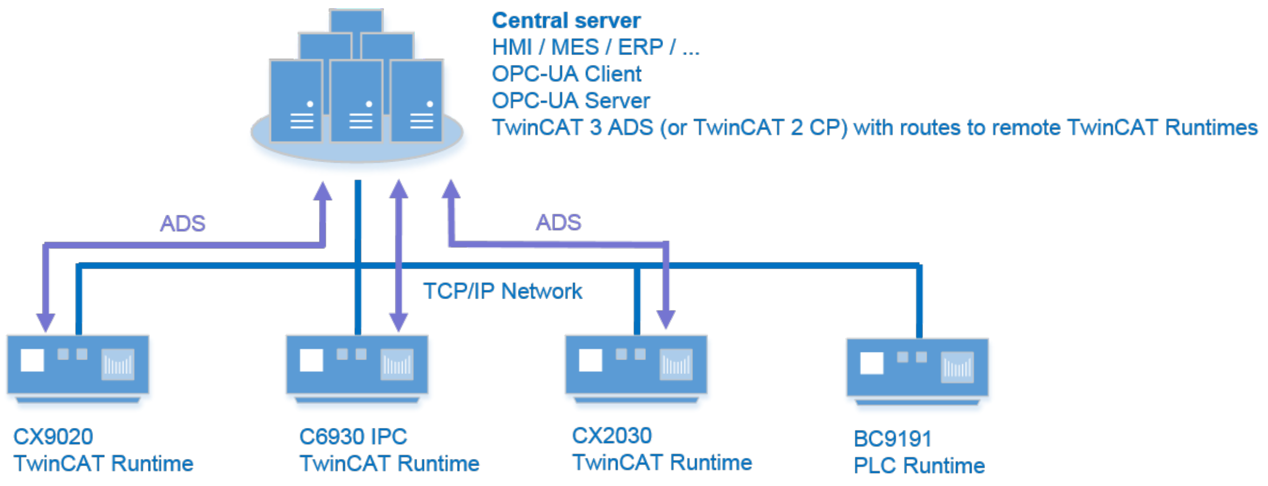
- The OPC UA Server automatically establishes a connection with the first local TwinCAT PLC runtime.
- TwinCAT runtime

This scenario has the following advantages:

- Network usage is optimized because it is based on OPC UA communication technologies, such as OPC UA registrations.
- Memory usage is decentralized. Each device is only responsible for its own memory requirements.
- OPC UA features security mechanisms that are directly integrated in the protocol. This is very useful if one of the Industrial PCs or Embedded PCs is connected via the internet, for example.

OPC UA Server: Runs on a central computer with connection to remote TwinCAT runtime(s)

This scenario describes the conventional implementation of OPC Servers. Servers using the old OPC A technology were often implemented on a central server instead of the Industrial PC on which TwinCAT runtime was executed, in order to avoid DCOM configurations. (Remember: in contrast to OPC UA, OPC DA is based on COM/DCOM technologies)



The following software components and configurations run on the central server:

- TwinCAT 3 ADS (or TwinCAT 2 CP) for the required ADS connectivity
- OPC UA Server with data access devices configured for remote TwinCAT runtimes
- ADS routes to remote TwinCAT runtimes
- Symbol files from any remote TwinCAT runtime
- OPC UA Client, which can be an HMI, MES or ERP system, for example.

The following software components and configurations run on the Industrial PC or Embedded PC:

- TwinCAT runtime
- ADS route to the central server

The more remote TwinCAT runtimes are connected to the central OPC UA Server, the higher the network usage will be.

The OPC UA Server uses advanced ADS recording techniques to query the symbol values of the TwinCAT runtime. The more symbols there are, the more cyclical queries are in progress. As a result, optimized OPC UA communication only takes place locally on the central server (between OPC UA Client and OPC UA Server).

This scenario has the following disadvantages compared to scenario 1:

- Network usage can be very high depending on the number of devices and symbols present.
- The memory requirement on the central server is very high because the OPC UA Server has to import symbol information from every TwinCAT runtime.
- No security between the central server and TwinCAT runtime based on data encryption - ADS was designed for high performance.
- The need to exchange symbol files between TwinCAT runtime and central server after each PLC program modification.

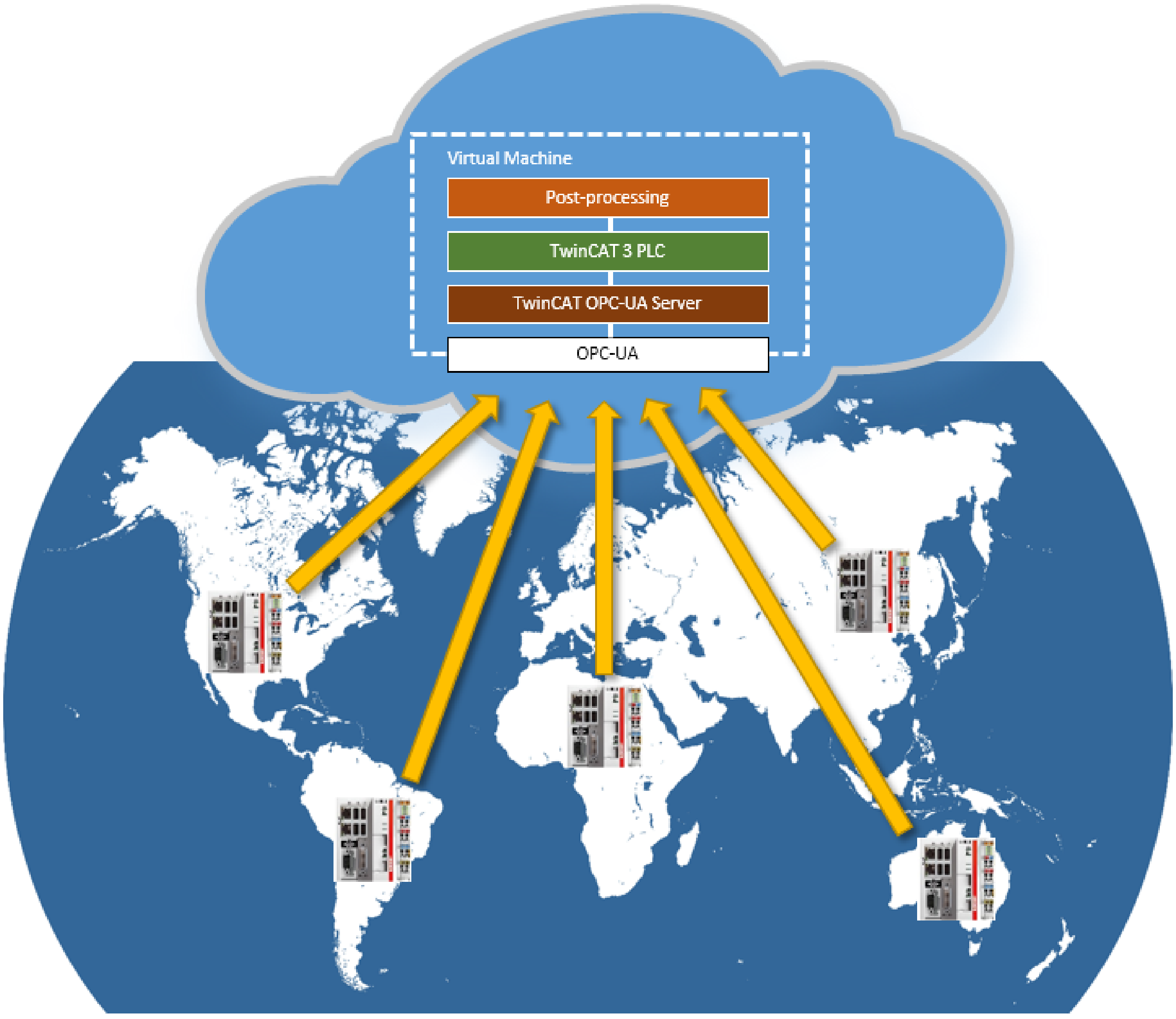
OPC UA Server: Access to BC Controller

Although small BC controllers are not able to run their own OPC UA Server, they may nevertheless have access to the PLC runtime executed on a remote BC Controller, such as a BC9191, and publish their symbol values via OPC UA. In this scenario the OPC UA Server must run together with a TwinCAT 3 ADS (or TwinCAT 2 CP) on another computer, and the scenario must be configured in the same way as scenario 2.

2.2 Application examples

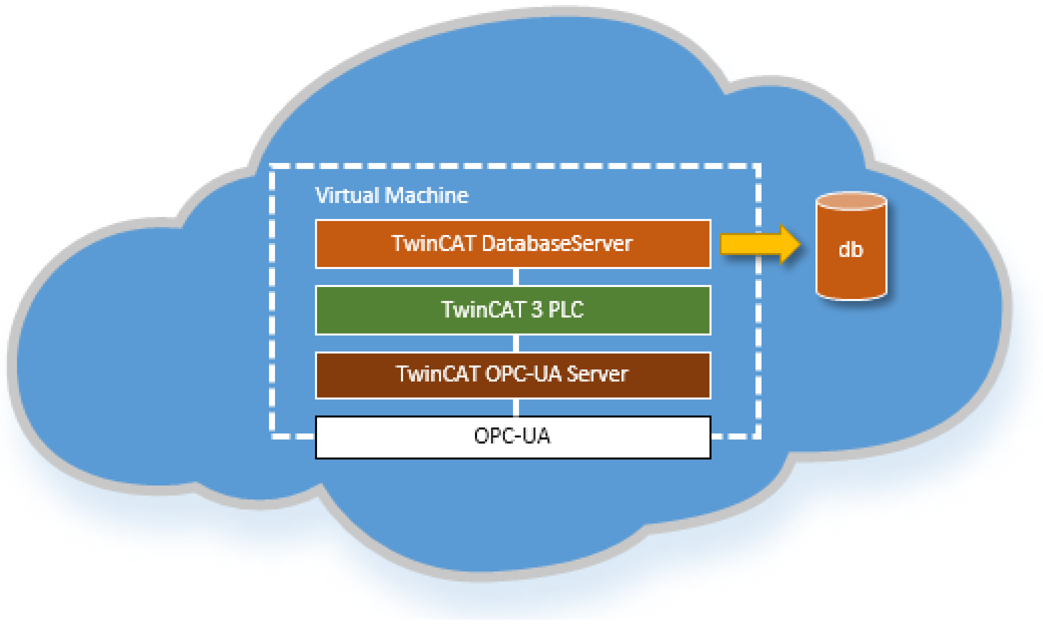
2.2.1 Post-processing in the Cloud

The overall concept of providing a generic, standardized and callable interface in the cloud that accepts incoming data for further analysis is based on an endpoint in the cloud, either a Windows-based virtual machine or a real hardware running a Windows operating system, in addition to the decentralized clients that call this service. In this documentation, this device is generally referred to as OPC UA Server (in the cloud).



Data logger

The documentation shows how computing resources in the Cloud can be accessed and how local TwinCAT PLC devices can be connected in order to use these resources. As an initial example, this use case can be applied to a typical data storage scenario in which decentralized TwinCAT PLC devices send data to the cloud, in which the incoming data is stored in an SQL database via the TwinCAT Database Server. Although the general scenario also involves sending data back and forth (i.e. sending parameters to the cloud, performing calculations in the cloud, and then returning the computed results), this specific scenario does not require the return of results from the cloud.



System requirements: OPC UA Server (the cloud)

The cloud must be able to host one of the following operating system variants. Note that using virtualization technology requires a more costly solution for 64-bit operating systems.

Device is a virtual machine

- 32-bit Windows operating system
- 64-bit Windows operating system with CPU core isolation (two cores required as a minimum)

Device is a dedicated hardware

- 32-bit Windows operating system
- 64-bit Windows operating system

In addition, the following software components must be installed on the device:

- TwinCAT 3 XAR (Runtime only) or XAE (Runtime and Engineering)
- TwinCAT 3 Function TF6100 OPC UA

Note that when installing a complete TwinCAT 3 XAE, additional configuration settings such as handling licenses or the option of debugging the device directly in the cloud may be useful. The following software components must be installed to store data in a central database received from a client:

- TwinCAT 3 Function TF6420 Database

System requirements: Decentralized OPC UA Client

The decentralized OPC UA Client is based on a TwinCAT 3 PLC runtime and can therefore be hosted on any hardware configuration that supports the operation of a TwinCAT 3 PLC. In addition, TwinCAT 3 Function TF6100 OPC UA must be installed on the client device in order to be able to use the TwinCAT OPC UA Client to execute UA methods on the server in the cloud.

Technical description

The implementation of an OPC UA Server in the cloud is very application-specific. Beckhoff provides a general architecture description, which is independent of the individual use case. As an application example, this article frequently refers a data aggregation scenario in which process data is transferred from decentralized clients via OPC UA to a central server, where the data is collected in a central database. The advantage of OPC UA in this case is not only the standardized interface (and therefore to use this concept for any type of OPC UA Client), but also the ability to secure the communication channel by using the integrated security mechanism of OPC UA.

The TwinCAT OPC UA Server is used to disclose the methods defined in IEC61131-3 PLC to the OPC UA namespace. These methods can be called by any TwinCAT OPC UA Client (based on the PLCopen function blocks) or by third party OPC UA Clients.

Decentralized OPC UA Client

The main purpose of the OPC UA Client is to call methods on the remote server. The client is most probably a TwinCAT-3-embedded device that fulfils the decentralized part of the application. For the data aggregation example, this would mean querying process data and sending it to the server by calling a method.

Interface description (to the cloud)

As an interface to a centralized cloud system, there are only two methods provided by the server:

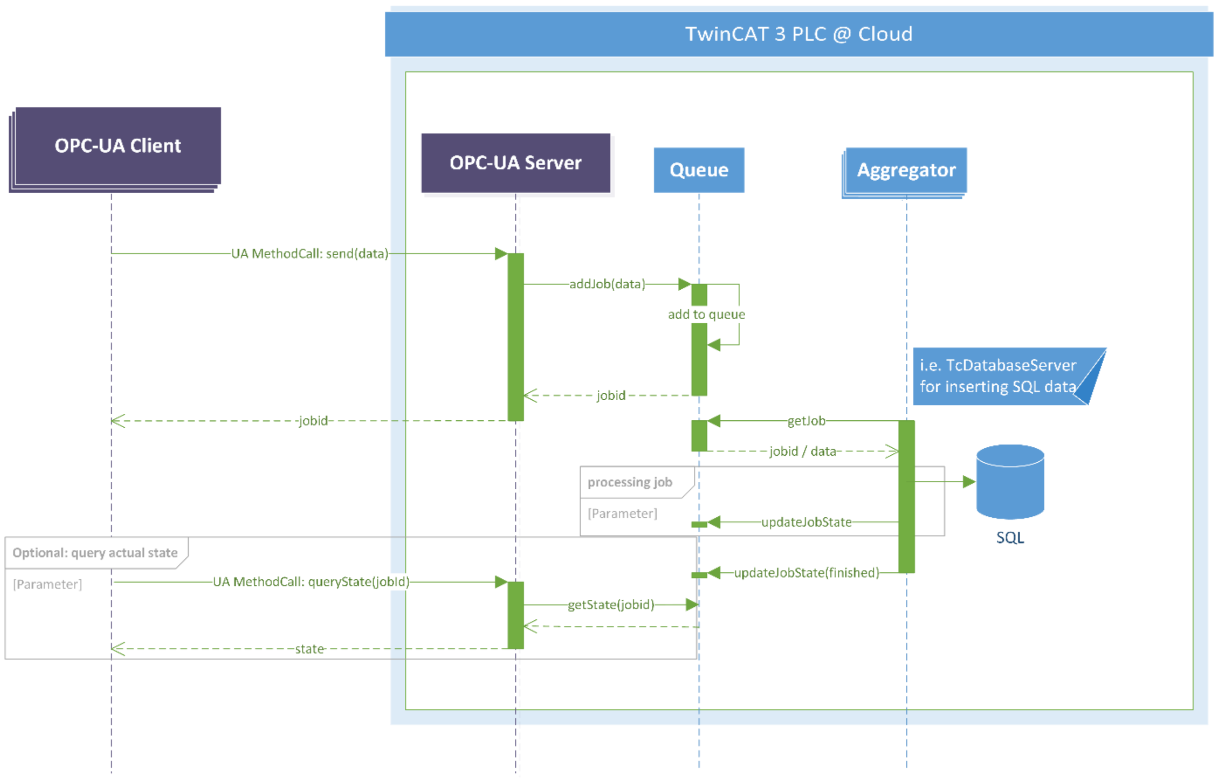
- int Send(data): This method forwards the data using an OPC UA method call and returns a JobID.
- int QueryState(jobID): Using the previously retrieved JobID, the client can cyclically query the current state of the job. This could be omitted if the server in the cloud can handle all situations, or if the client cannot help out in case of problems.

Communication between clients and the cloud can be secured by OPC UA's built-in security concepts, e.g. by using client and server certificates to encrypt data communication.

The cloud

The OPC UA Server in the cloud is based on TwinCAT 3 PLC methods that are disclosed to the OPC UA namespace. The PLC project contains the following three components:

- MethodCall Provider: This component provides the above-mentioned send method, which collects data, creates a JobID and places the data in the queue as different jobs. There may also be a QueryState method to allow OPC UA Clients to query the current job status.
- Queue: The queue stores a list of jobs to be executed. Each job in the queue contains the following information: {ID, State, Data}. New elements (jobs) are added by the MethodCall provider and can be deleted by it. The entire queue can be based on a hash table within the PLC program.
- Aggregator: The application that processes the jobs. It cyclically looks in the queue to see whether there are any jobs to be processed. If this is the case, it takes over the job (sets the status to "processed") and starts processing the job, e.g. connecting to a database via the function blocks of the TwinCAT Database Server. Note that there can be more than one aggregator to enable parallel processing of the queue.



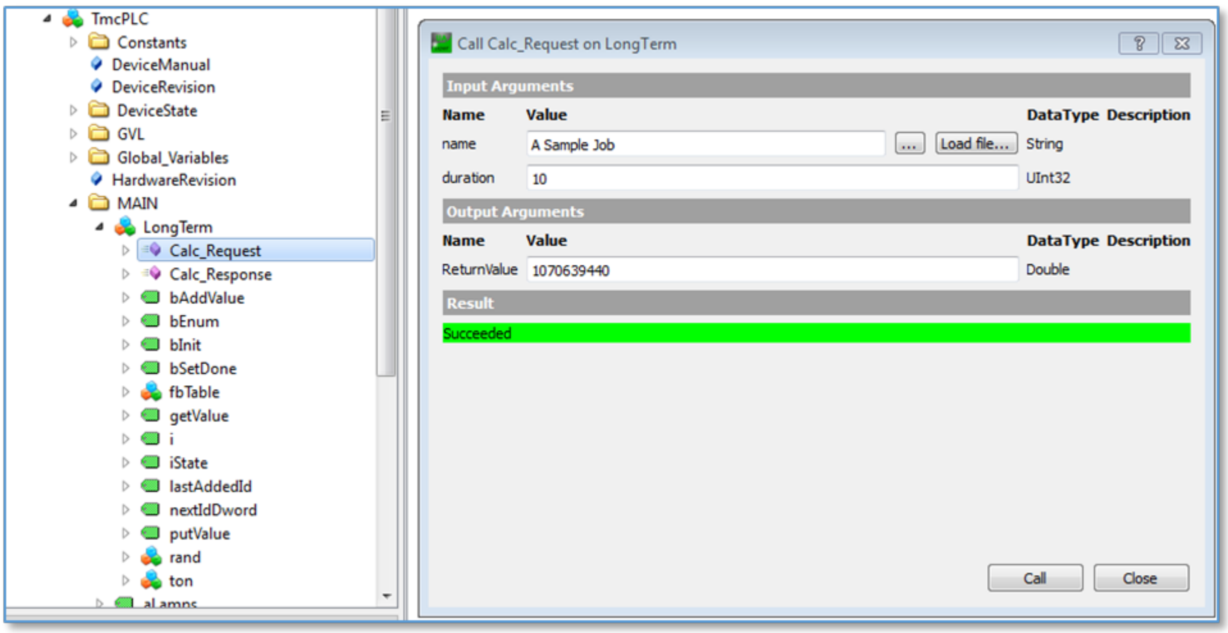
Sample

The following sample illustrates the scenario by executing simple jobs: Jobs can be submitted to the server by an OPC UA Client. In this sample, the job data consists of a definable time interval that is recorded by the aggregator to complete the job. The sample consists of the following PLC components on the OPC UA Server:

- ST_JobEntry: Represents a job in the queue. In terms of the data, the job only consists of a name and a duration. A duration defines the length of time a job takes.
- E_JobState: Represents the state of a job. The sample implementation defines the following values: QUEUED, PROCESSING, READY, FAILED and INVALID.
- LongTerm: Represents the MethodCall server (consisting of the methods Calc_Request and Calc_Response) and the aggregator that processes the job (which is implemented in the function block)
- FB_SpecialHashTableCtrl: Represents the queue in the form of a hash table, as reflected by PLC samples from the Beckhoff Information System. Here, different methods for handling the queue are provided (Add, Count, GetFirst, GetNext, Lookup, Remove, Reset).

Using the sample

UA Expert could be used as a sample client to call the methods provided by the OPC UA Server in the cloud. By calling the Calc_Request method, the client receives a JobID (or 0 for displaying an error):



Input Arguments

Name	Value	DataType	Description
name	A Sample Job	String	
duration	10	UInt32	

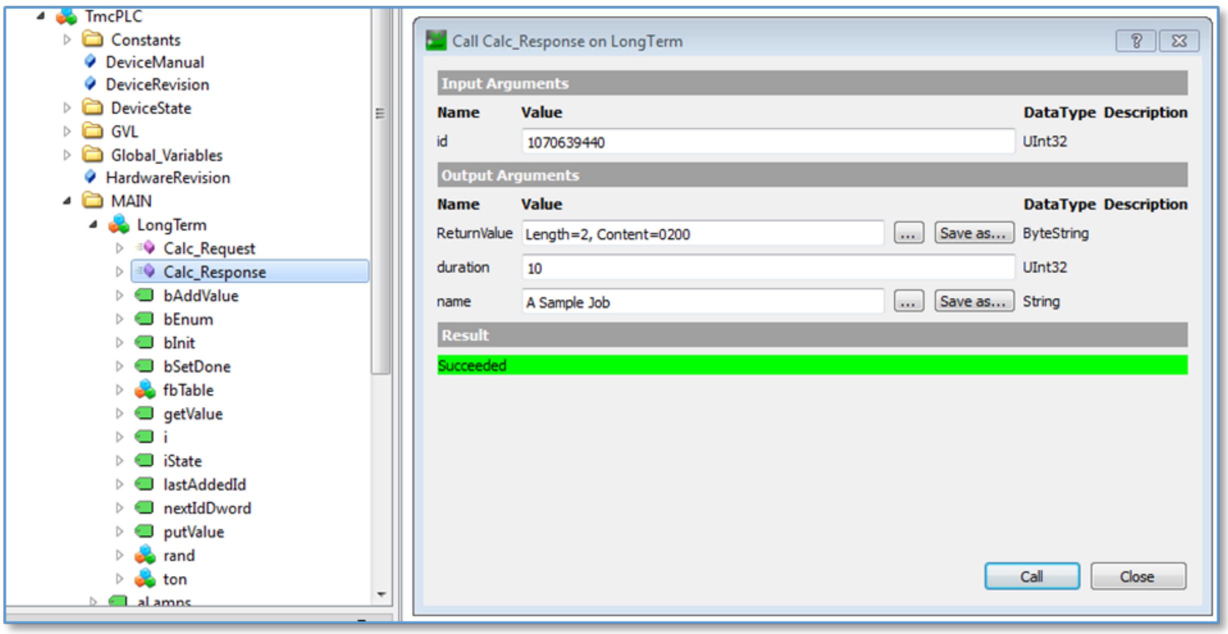
Output Arguments

Name	Value	DataType	Description
ReturnValue	1070639440	Double	

Result

Succeeded

The client can query the JobState by calling the method Calc_Response with the JobID. In addition, the previously set duration and the job name are returned for subsequent reference.


TmcPLC
- Constants
- DeviceManual
- DeviceRevision
- DeviceState
- GVL
- Global_Variables
- HardwareRevision
- MAIN
  - LongTerm
    - Calc_Request
    - Calc_Response
    - bAddValue
    - bEnum
    - bInit
    - bSetDone
    - fbTable
    - getValue
    - i
    - iState
    - lastAddedId
    - nextIdDword
    - putValue
    - rand
    - ton
  - alarms

Call Calc_Response on LongTerm

Input Arguments

Name	Value	DataType	Description
id	1070639440	UInt32	

Output Arguments

Name	Value	DataType	Description
ReturnValue	Length=2, Content=0200	ByteString	
duration	10	UInt32	
name	A Sample Job	String	

Result

Succeeded

Installation

The following chapter describes the installation and configuration of each software component required on the server in the cloud.

Runtime only

If the server does not host the engineering environment of TwinCAT 3, the TwinCAT 3 XAR installation must be used. Note that in this case the entire handling for proper function involves several steps, and future maintenance may be more difficult because the XAR installation lacks the programming and debugging environment. In this scenario, the user must ensure that maintenance of the ADS routes between the actual engineering computer (on which the PLC program for the server is developed) and the device that hosts the server in the cloud must allow not only downloading and debugging of the PLC program, but also activation of licenses for the TwinCAT 3 runtime. Note that you must open the firewall ports to enable ADS traffic. See the ADS documentation for more information. The TC3 License Request Generator tool was developed for easier handling of TwinCAT 3 runtime licenses. It can be downloaded via the Beckhoff FTP server and enables the creation of License Request Files and the import of License Response Files:

https://download.beckhoff.com/download/Software/TwinCAT/Unsupported_Utilities/TC3-LicenseGen/

Runtime and Engineering

This is the recommended setup scenario. The server in the cloud hosts a TwinCAT 3 runtime and the corresponding engineering components to enable debugging and easier handling of the runtime system. In this case it is necessary to install TwinCAT 3 XAE on the system.

Additional software

After successful installation of TwinCAT XAE or XAR, the following software components must be installed and configured on the device:

TF6100 OPC UA

The TF6100 function installs an OPC UA Server (and Client) on the device. The server is required to provide the OPC UA Clients with the PLC methods. When the PLC program is downloaded into the runtime, the OPC UA Server automatically imports the first PLC runtime into its namespace. All variables and methods of the PLC marked as available via OPC UA are imported into the OPC UA namespace and become available to clients. Refer to the TF6100 documentation for more information.

TF6420 Database

If the server is to store the data received from the clients in a database, the TF6420 function must be used. It provides generic function blocks for TwinCAT 3 PLCs for accessing the database, e.g. to insert values into a database table. Install the TF6420 setup and consult the TF6420 documentation for more information on using the corresponding function blocks.

Licensing

All TwinCAT 3 software products require a license to be available on the server. The license can be either a 7-day trial or an unlimited license. Licenses can be activated using the TwinCAT 3 XAE License Manager. If a TwinCAT 3 XAR installation is used on the server, use the TC3 License Generator.

3 Installation

3.1 Setup overview

For better componentization and faster release cycles, the product is shipped in multiple setups. Each product component has its own setup. The following table provides an overview of this.

Setup name	Description
TF6100-OPC-UA-Server	Contains the TwinCAT OPC UA Server and SampleClient.
TF6100-OPC-UA-Client	Contains the TwinCAT OPC UA Client.
TF6100-OPC-UA-Configurator	Includes the TwinCAT OPC UA Configurator as Visual Studio Extension and standalone application.
TF6100-OPC-UA-Gateway	Contains the TwinCAT OPC UA Gateway.

Update installation

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

CE-ARMV4I-LF

For the TwinCAT OPC UA Server a CAB installation file for Windows CE is delivered, which is provided with the suffix CE-ARMV4I-LF. This is a limited server variant for hardware platforms with few main memory. In this variant certain OPC UA features (e.g. Historical Access and Alarms & Conditions) are disabled to keep the memory load as low as possible.

3.2 System requirements

OPC UA Server

Technical data	Description
Operating system	Windows 7, 10 Windows CE 6/7 Windows Embedded 7 Windows Server
Target platforms	PC architecture (x86, x64, ARM)
.NET Framework	4.6.1 (only needed for the SysTray icon)
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)

OPC UA Client

Technical data	Description
Operating system	Windows 7, 10 Windows CE 6/7 Windows Embedded 7 Windows Server
Target platforms	PC architecture (x86, x64, ARM)
.NET Framework	---
TwinCAT version	TwinCAT 2, TwinCAT 3
Minimum TwinCAT installation level	TwinCAT 2 PLC, NC-PTP TwinCAT 3 XAE, XAR
Required TwinCAT license	TS6100 TwinCAT OPC UA (for TwinCAT 2) TF6100 TC3 OPC UA (for TwinCAT 3)

OPC UA I/O Client

Technical data	Description
Operating system	Windows 7, 10 Windows CE 7 Windows Embedded 7 Windows Server
Target platforms	PC architecture (x86, x64, ARM)
.NET Framework	4.6 (only for namespace browser)
TwinCAT version	TwinCAT 3.1 Build 4022.4
Minimum TwinCAT installation level	TwinCAT 3 XAE, XAR
Required TwinCAT license	TF6100 TC3 OPC UA

OPC UA Gateway

Technical data	Description
Operating system	Windows 10 Windows Embedded 7 Windows Server
Target platforms	PC architecture (x86, x64, ARM)
.NET Framework	---
TwinCAT version	---
Minimum TwinCAT installation level	---
Required TwinCAT license	---

OPC UA Configurator Visual Studio

OPC UA Configurator	Description
Operating system	Windows 10 Windows Embedded 7 Windows Server
Target platforms	PC architecture (x86, x64)
Visual Studio dependency	Installing .NET Targeting Package 4 using the Visual Studio Installer
Supported Visual Studio versions	2017, 2019
.NET Framework	4.6.1
Minimum TwinCAT installation level	TwinCAT 3 XAE
Required TwinCAT license	---

OPC UA Configurator Standalone

OPC UA Configurator	Description
Operating system	Windows 10 Windows Embedded 7 Windows Server
Target platforms	PC architecture (x86, x64)
.NET Framework	4.6.1
Minimum TwinCAT installation level	TwinCAT 2 CP, PLC, NC-PTP TwinCAT 3 XAE, XAR, ADS
Required TwinCAT license	---

OPC UA Sample Client

OPC UA Sample Client	Description
Operating system	Windows 10 Windows Embedded 7 Windows Server
Target platforms	PC architecture (x86, x64)
.NET Framework	4.6.1
TwinCAT version	---
Minimum TwinCAT installation level	---
Required TwinCAT license	---

3.3 Installation (TC3)

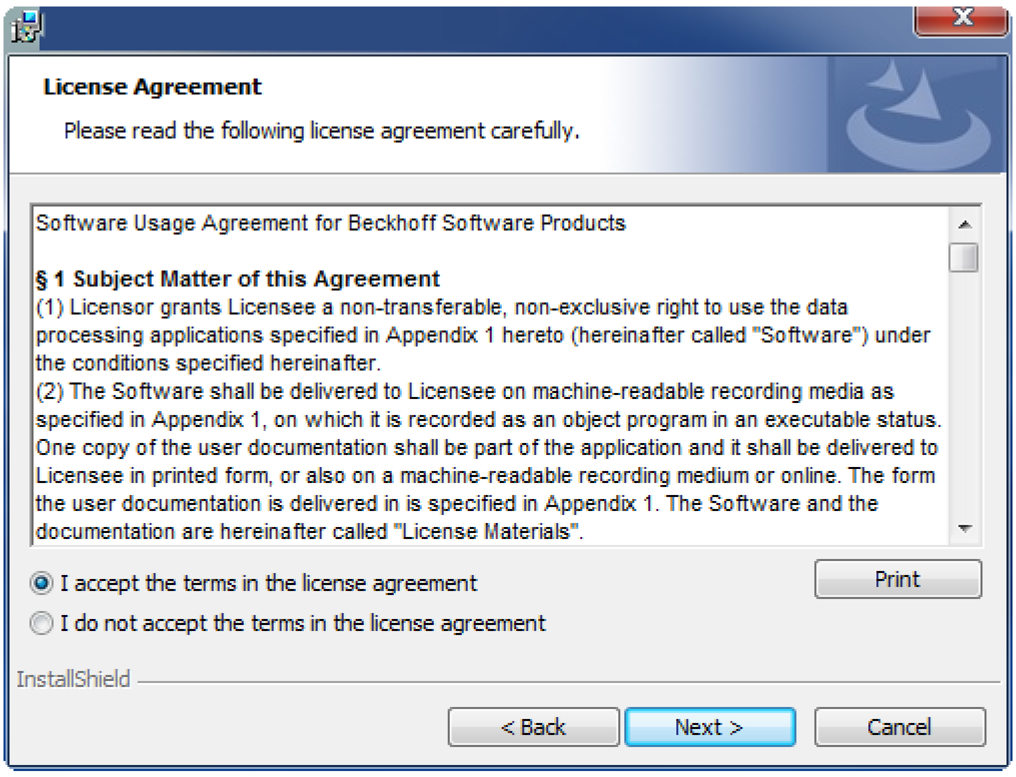
The following section describes how to install the TwinCAT 3 function TF6100 OPC UA for Windows-based operating systems.

✓ The TwinCAT 3 function setup file was downloaded from the Beckhoff website.

1. Run the setup file as administrator. To do this, select the command **Run As Admin** in the context menu of the file.

   ⇨ The installation dialog opens.

2. Accept the end user licensing agreement and click **Next**.



License Agreement

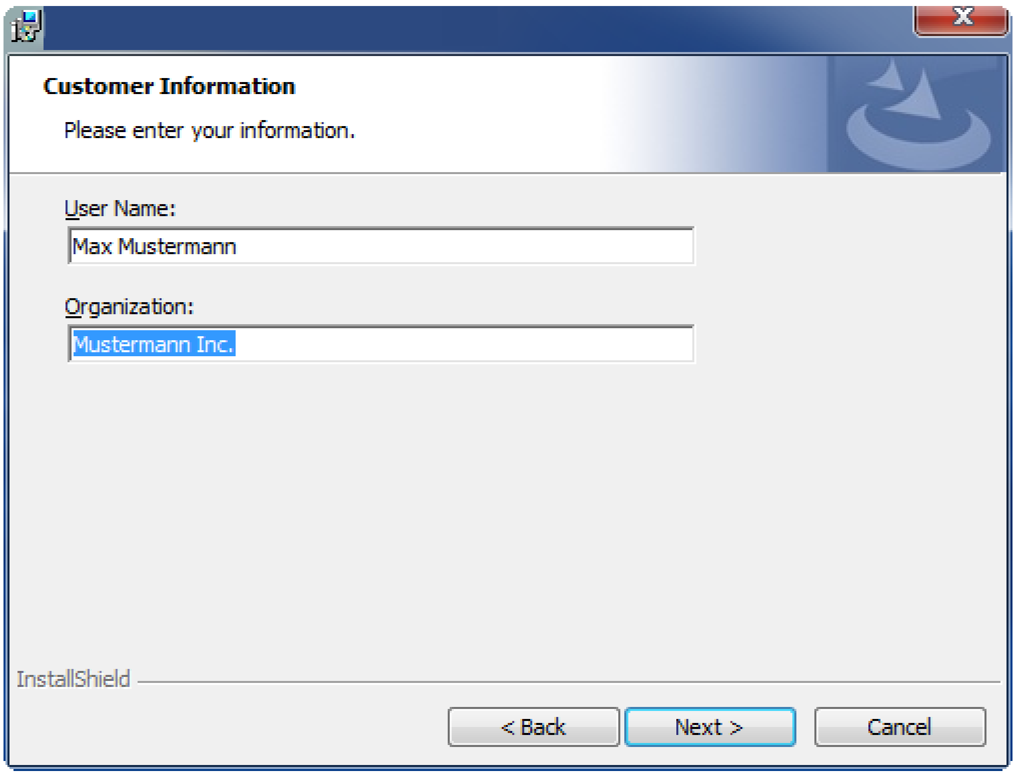
Please read the following license agreement carefully.

Software Usage Agreement for Beckhoff Software Products

§ 1 Subject Matter of this Agreement
(1) Licensor grants Licensee a non-transferable, non-exclusive right to use the data processing applications specified in Appendix 1 hereto (hereinafter called "Software") under the conditions specified hereinafter.
(2) The Software shall be delivered to Licensee on machine-readable recording media as specified in Appendix 1, on which it is recorded as an object program in an executable status. One copy of the user documentation shall be part of the application and it shall be delivered to Licensee in printed form, or also on a machine-readable recording medium or online. The form the user documentation is delivered in is specified in Appendix 1. The Software and the documentation are hereinafter called "License Materials".

[✓ I accept the terms in the license agreement]
I do not accept the terms in the license agreement

3. Enter your user data.



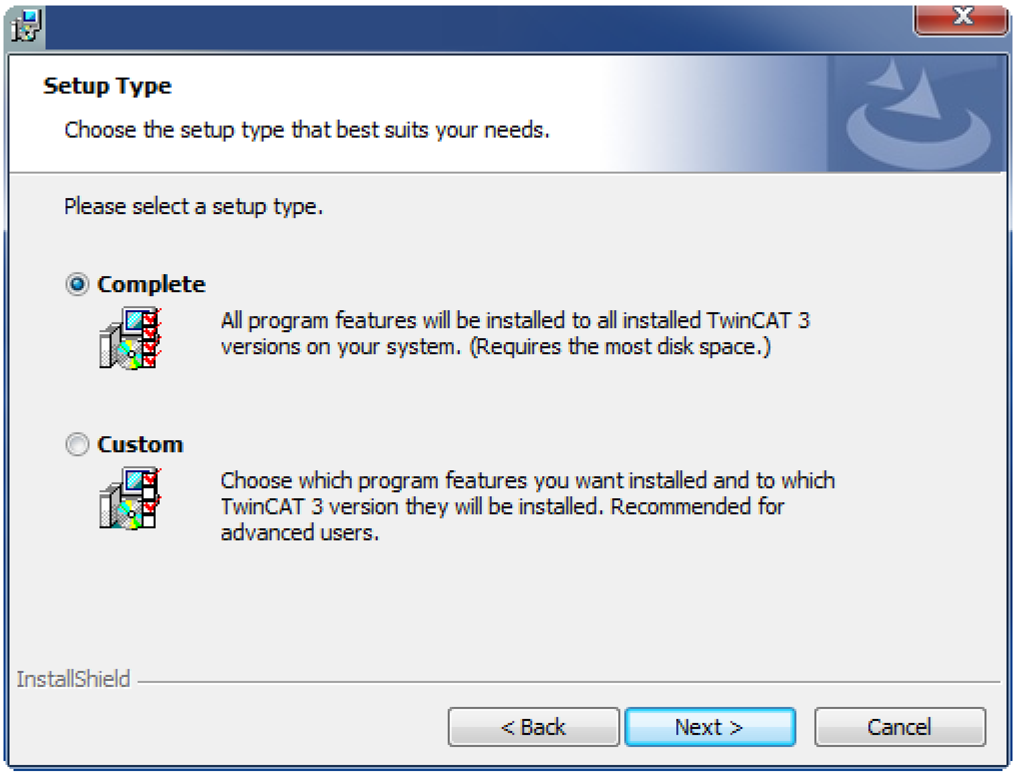
Customer Information

Please enter your information.

User Name: Max Mustermann

Organization: Mustermann Inc.

4. If you want to install the full version of the TwinCAT 3 function, select **Complete** as installation type. If you want to install the TwinCAT 3 function components separately, select **Custom**.



Setup Type

Choose the setup type that best suits your needs.

Please select a setup type.

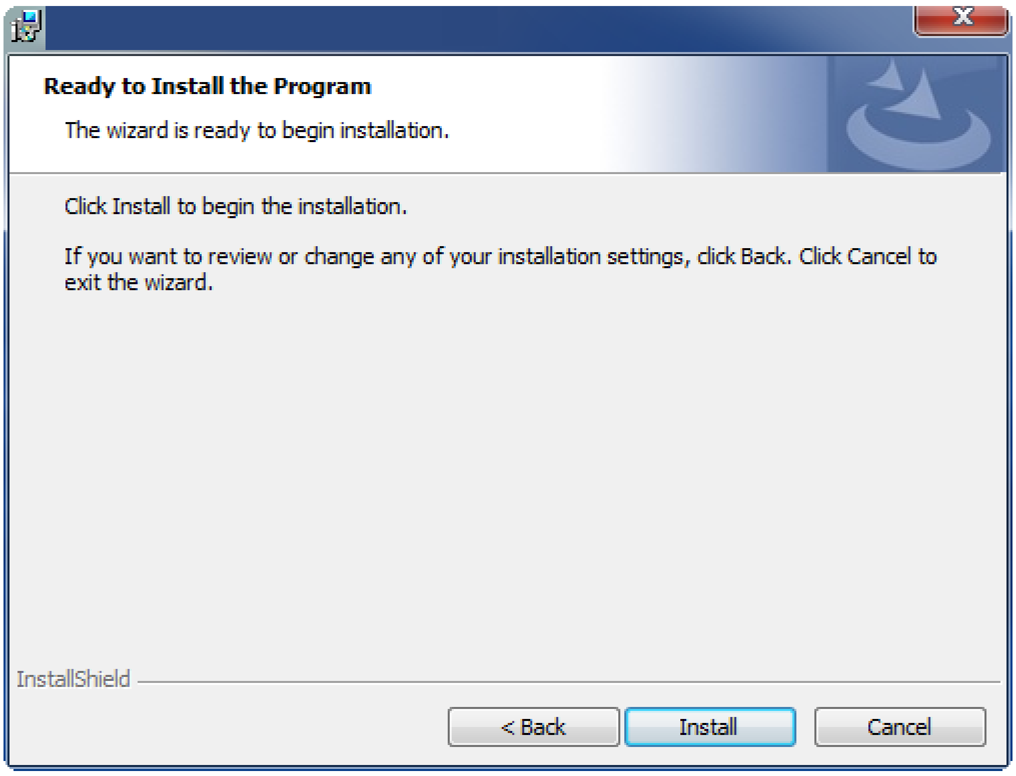
[✓ Complete]

All program features will be installed to all installed TwinCAT 3 versions on your system. (Requires the most disk space.)

Custom

Choose which program features you want installed and to which TwinCAT 3 version they will be installed. Recommended for advanced users.

5. Select **Next**, then **Install** to start the installation.



Ready to Install the Program

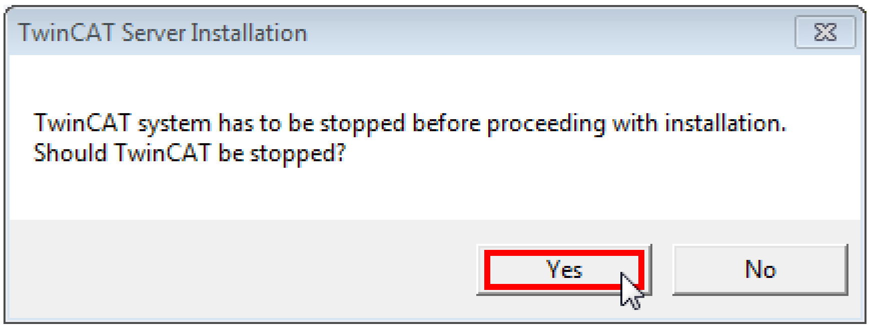
The wizard is ready to begin installation.

Click Install to begin the installation.

If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.

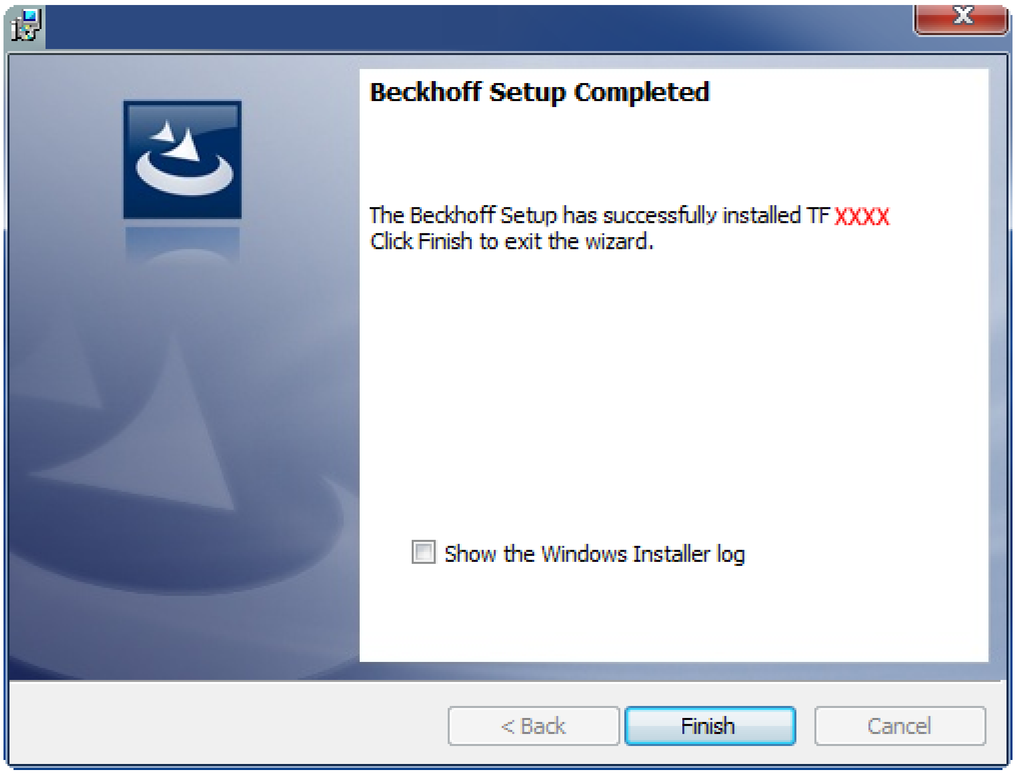
⇨ A dialog box informs you that the TwinCAT system must be stopped to proceed with the installation.

6. Confirm the dialog with **Yes**.


TwinCAT Server Installation

TwinCAT system has to be stopped before proceeding with installation.
Should TwinCAT be stopped?

7. Select **Finish** to exit the setup.


Beckhoff Setup Completed

The Beckhoff Setup has successfully installed TF XXXX
Click Finish to exit the wizard.

Show the Windows Installer log

⇨ The TwinCAT 3 function has been successfully installed and can be licensed (see Licensing (TC3) [▶ 32]

3.4 Installation (TC2)

The following section describes how to install the TwinCAT 2 Supplement TwinCAT OPC UA for Windows-based operating systems.

- Download and install the setup file [▶ 22]
- After the installation [▶ 25]

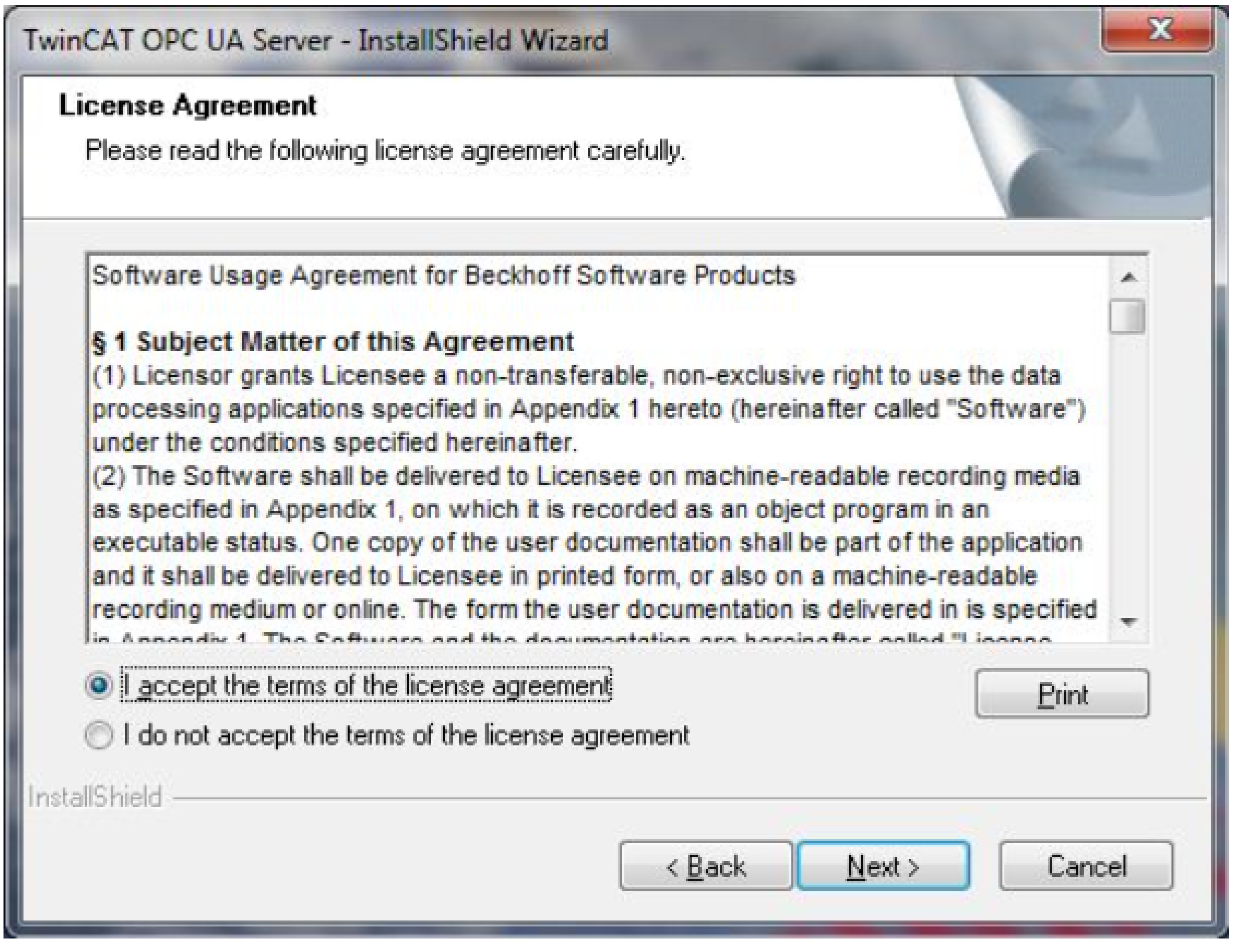
Download and install the setup file

Like many other TwinCAT add-on products, OPC UA is available as a 30-day demo version or full version for download from the Beckhoff website www.beckhoff.com.

✓ The setup file for the TwinCAT 2 Supplement has been downloaded from the Beckhoff homepage.

1. Run the downloaded setup file *TcOpcUaSvr.exe* as an administrator. To do this, select the command **Run As Admin** in the context menu of the file.
   ⇨ The installation dialog opens.
2. Select an installation language.

3. Click **Next** and accept the license agreement.



TwinCAT OPC UA Server - InstallShield Wizard

License Agreement

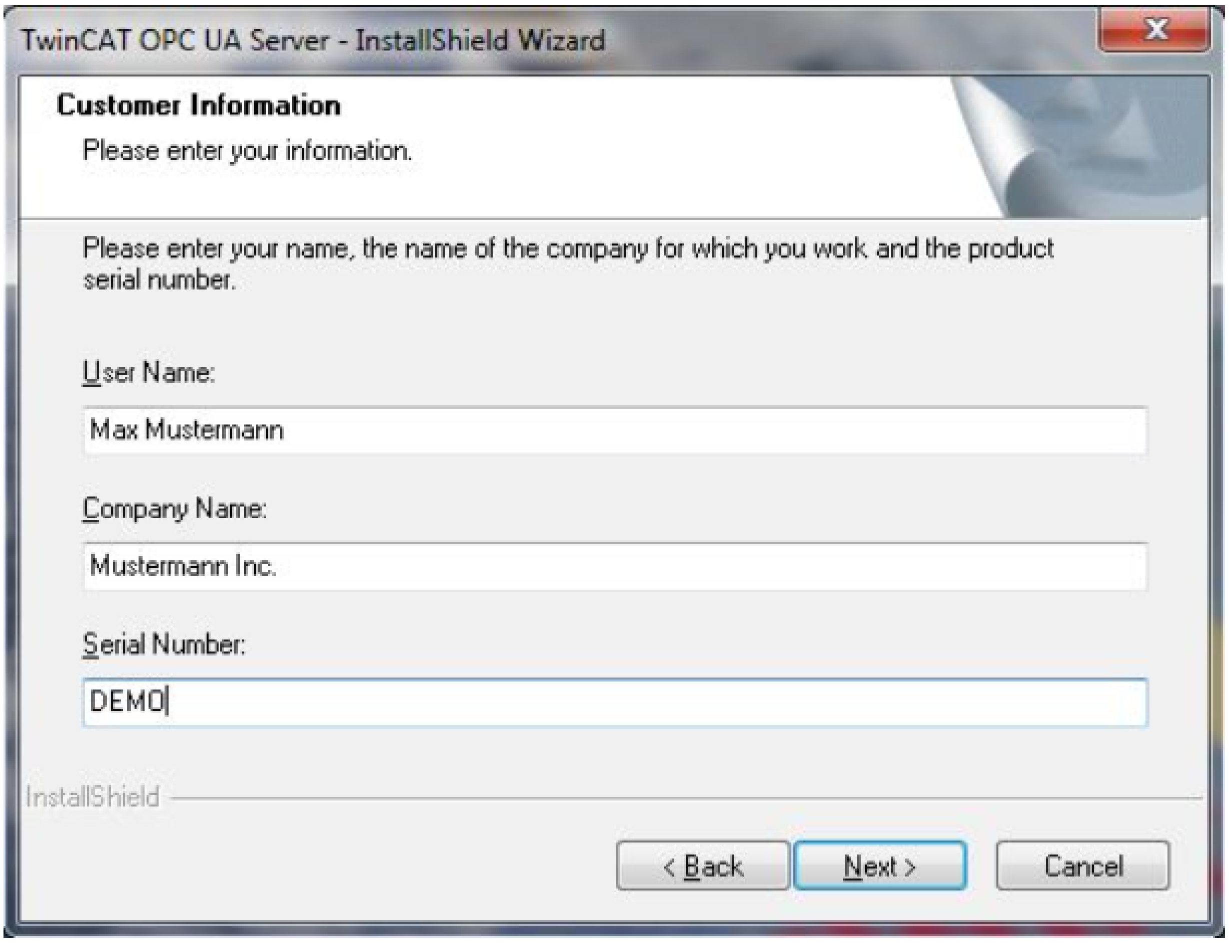
Please read the following license agreement carefully.

Software Usage Agreement for Beckhoff Software Products

§ 1 Subject Matter of this Agreement
(1) Licensor grants Licensee a non-transferable, non-exclusive right to use the data processing applications specified in Appendix 1 hereto (hereinafter called "Software") under the conditions specified hereinafter.
(2) The Software shall be delivered to Licensee on machine-readable recording media as specified in Appendix 1, on which it is recorded as an object program in an executable status. One copy of the user documentation shall be part of the application and it shall be delivered to Licensee in printed form, or also on a machine-readable recording medium or online. The form the user documentation is delivered in is specified

(●) I accept the terms of the license agreement
( ) I do not accept the terms of the license agreement

4. Enter your information. All fields are mandatory. If you want to install a 30-day demo, please enter "DEMO" as license key.



TwinCAT OPC UA Server - InstallShield Wizard

Customer Information

Please enter your information.

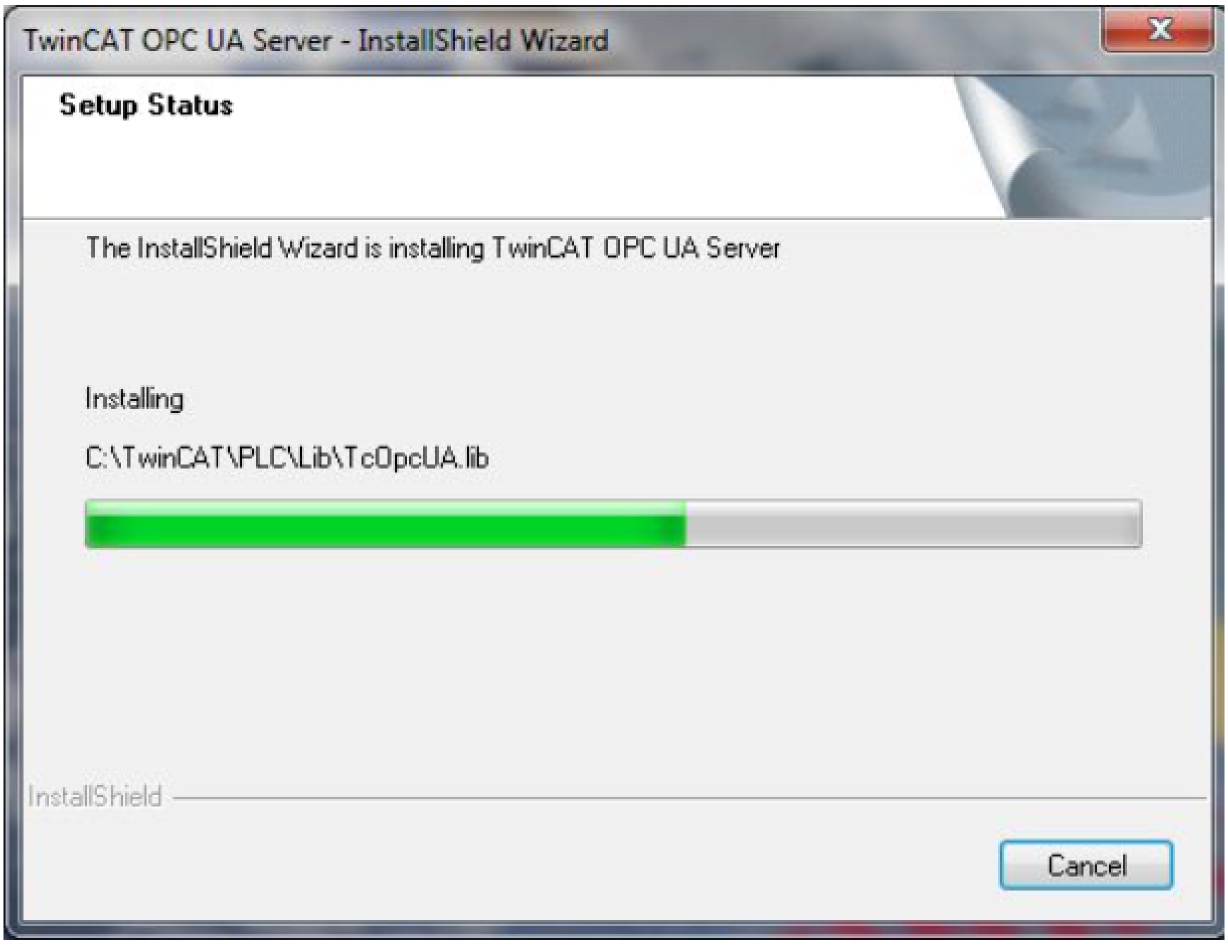
Please enter your name, the name of the company for which you work and the product serial number.

User Name: Max Mustermann

Company Name: Mustermann Inc.

Serial Number: DEMO

5. Click **Install** to start the installation.



TwinCAT OPC UA Server - InstallShield Wizard

Setup Status

The InstallShield Wizard is installing TwinCAT OPC UA Server

Installing

C:\TwinCAT\PLC\Lib\TcOpcUA.lib

6. Restart the computer to complete the installation.

After the installation

The TwinCAT add-on OPC UA is automatically configured during installation, and no further settings are required for using this product. Further steps may include:

- Establish an initial connection to the installed UA server and test its configuration with the OPC UA Sample Client. (See Sample Client [▸ 208])
- Personalize the UA server setup using the OPC UA Configurator. (See Configurator [▸ 115])
- Also, make sure your firewall opens TCP port 4840 because the OPC UC Server requires this port.

3.5 Installation Windows CE (TC3)

The following section describes how to install the TwinCAT 3 function TF6100 OPC UA on a Beckhoff Embedded PC with Windows CE.

- Download and install the setup file [▸ 26]
- Transfer the CAB file to the Windows CE device [▸ 26]
- Run the CAB file on the Windows CE device [▸ 26]

If an older version of TF6100 is already installed on the Windows CE device, it can be updated:

- Software upgrade [▸ 27]

Download and install the setup file

The executable file for Windows CE is part of the TF6100 OPC UA setup. This is made available on the Beckhoff website www.beckhoff.com and automatically contains all versions for Windows XP, Windows 7 and Windows CE (x86 and ARM).

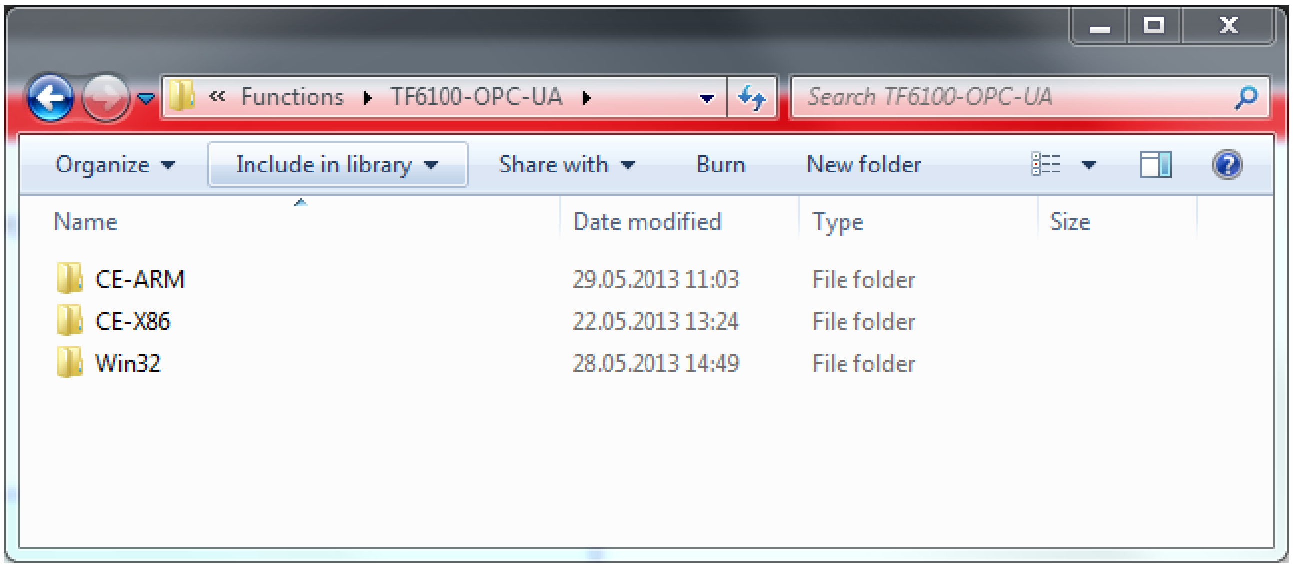
Download the setup and install the function as described in the section Installation.

After the installation, the installation folder contains three directories (one directory per hardware platform)

- **CE-ARM:** ARM-based embedded PCs running Windows CE, e.g. CX8090, CX9020
- **CE-X86:** X86-based embedded PCs running Windows CE, e.g. CX50xx, CX20x0
- **Win32:** embedded PCs running Windows XP, Windows 7 or Windows Embedded Standard

The CE-ARM and CE-X86 directories contain the CAB files of the TwinCAT 3 function for Windows CE in relation to the respective hardware platform of the Windows CE device.

Example: installation folder "TF6310"



Name	Date modified	Type	Size
CE-ARM	29.05.2013 11:03	File folder	
CE-X86	22.05.2013 13:24	File folder	
Win32	28.05.2013 14:49	File folder	

Transfer the CAB file to the Windows CE device

Transfer the corresponding executable file to the Windows CE device.

There are various options for transferring the executable file:

- via network shares
- via the integrated FTP server
- via ActiveSync
- via CF/SD cards

Further information can be found in the Beckhoff Information System in the "Operating Systems" documentation (Embedded PC > Operating Systems > CE).

Run the CAB file on the Windows CE device

After transferring the CAB file to the Windows CE device, double-click the file there. Confirm the installation dialog with **OK**. Then restart the Windows CE device.

After restarting the device, the files (Client and Server) are automatically loaded in the background and are available.

The software is installed in the following directory on the Windows CE device:
\Hard Disk\TwinCAT\Functions\TF6310-TCP-IP

Software upgrade

If an older TF6100 version is already installed on the Windows CE device, carry out the following steps on the Windows CE device to upgrade to a new version:

1. Open the CE Explorer by clicking **Start > Run** and entering "Explorer".
2. Navigate to *\Hard Disk\TwinCAT\Functions\Tf6100-OPC-UA\Server* or (in a second step) *\Hard Disk\TwinCAT\Functions\Tf6100-OPC-UA\Client*.
3. Rename the file*TcOpcUaServer.exe* or *TcOpcUaClient.exe*.
4. Restart the Windows CE device.
5. Transfer the new CAB file to the Windows CE device.
6. Run the CAB file on the Windows CE device and install the new version.
7. Delete the old (re-named) files.
8. Restart the Windows CE device.

⇨ The new version is active after the restart.

3.6 Installation Windows CE (TC2)

The following section describes how to install the TwinCAT 2 Supplement on a Beckhoff Embedded PC with Windows CE, e.g. CX1000, CX1020, CX9000, CX9001, CX9010, CX8090, CP62xx or CP69xx.

- Download and install the setup file [▶ 27]
- Transfer the setup file to a Windows CE device [▶ 30]
- Run the setup file on the Windows CE device [▶ 30]

Installation on small embedded platforms (CX9001, CX9010)

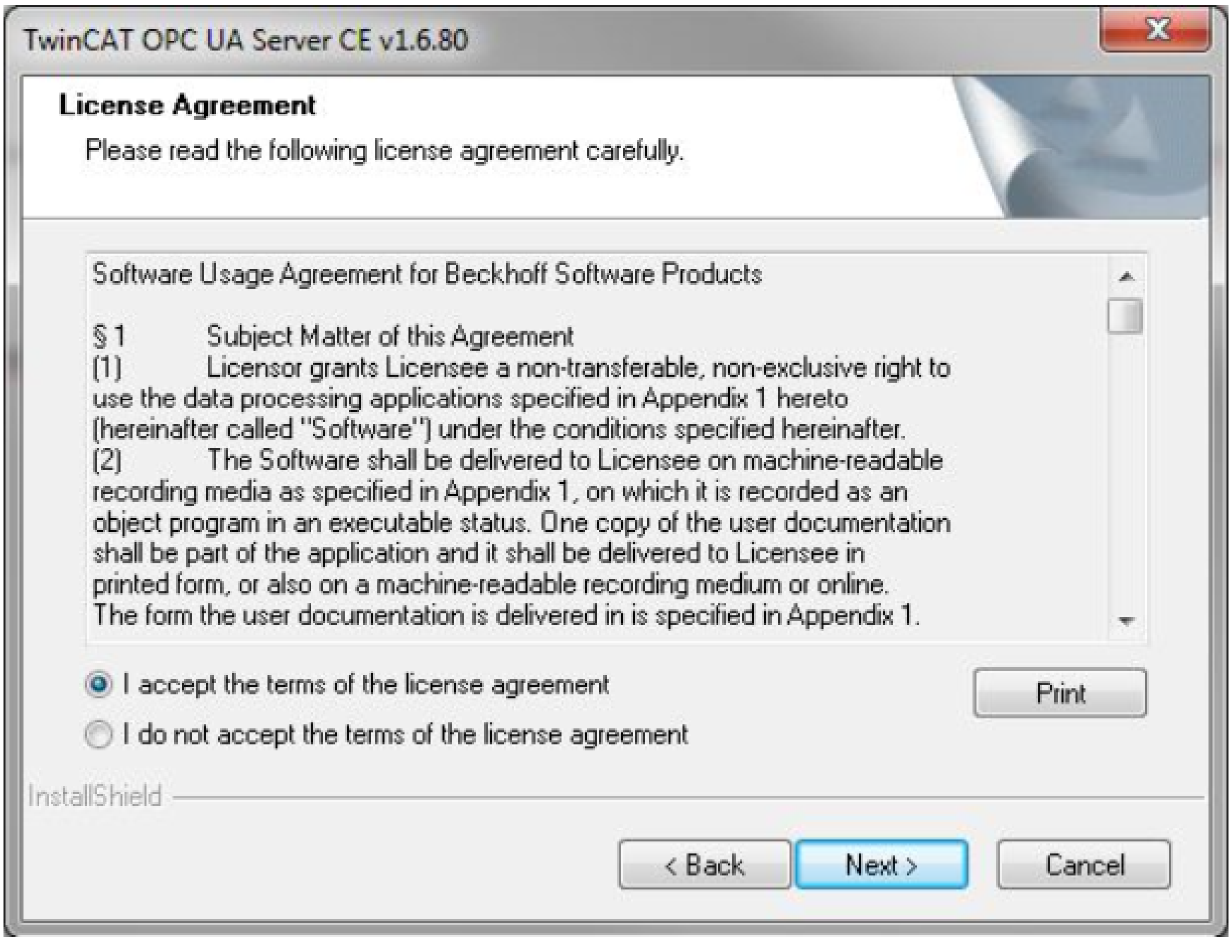
Very small embedded devices may require some additional manual steps to install the CAB file. These steps may include, for example, the deleting of unnecessary files from the memories of the devices so that there is sufficient space to install all the files of the application.

Download and install the setup file

Just like many other TwinCAT supplementary products, OPC UA for CE is available as a download on the Beckhoff homepage. To obtain the installation files for Windows CE, you must first install the downloaded setup file on a host computer. This can be any Windows CE-based system.

1. Double-click the downloaded file *TcOpcUaSvrCE.exe*.
2. Select an installation language.

3. Click **Next** and accept the license agreement.



License Agreement

Please read the following license agreement carefully.

Software Usage Agreement for Beckhoff Software Products

§ 1 Subject Matter of this Agreement

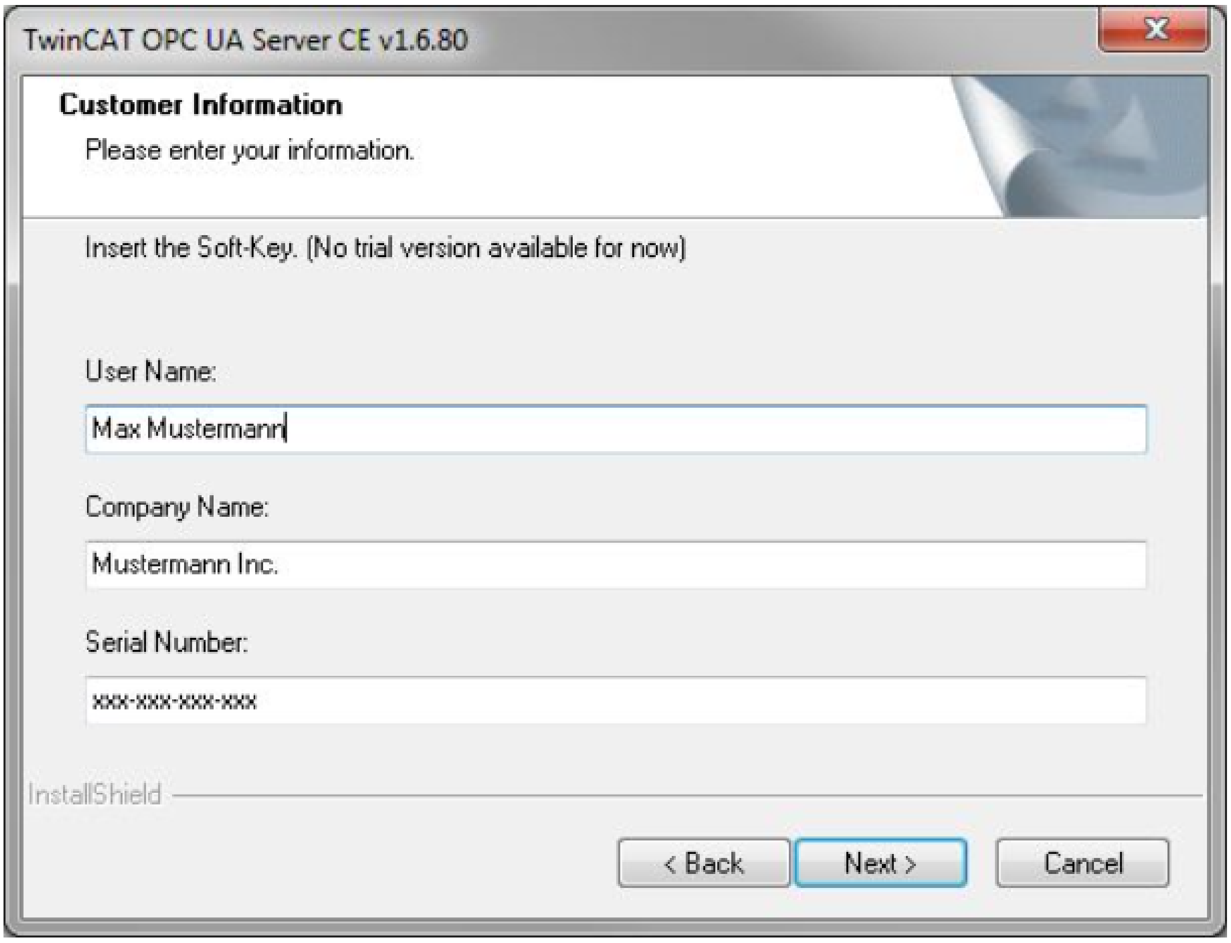
(1) Licensor grants Licensee a non-transferable, non-exclusive right to use the data processing applications specified in Appendix 1 hereto (hereinafter called "Software") under the conditions specified hereinafter.

(2) The Software shall be delivered to Licensee on machine-readable recording media as specified in Appendix 1, on which it is recorded as an object program in an executable status. One copy of the user documentation shall be part of the application and it shall be delivered to Licensee in printed form, or also on a machine-readable recording medium or online. The form the user documentation is delivered in is specified in Appendix 1.

(● I accept the terms of the license agreement)

○ I do not accept the terms of the license agreement

4. Enter your information. All fields are mandatory. Note that OPC UA for CE is not currently available as a demo version. Therefore, you will need a valid license key to continue with the installation.



Customer Information

Please enter your information.

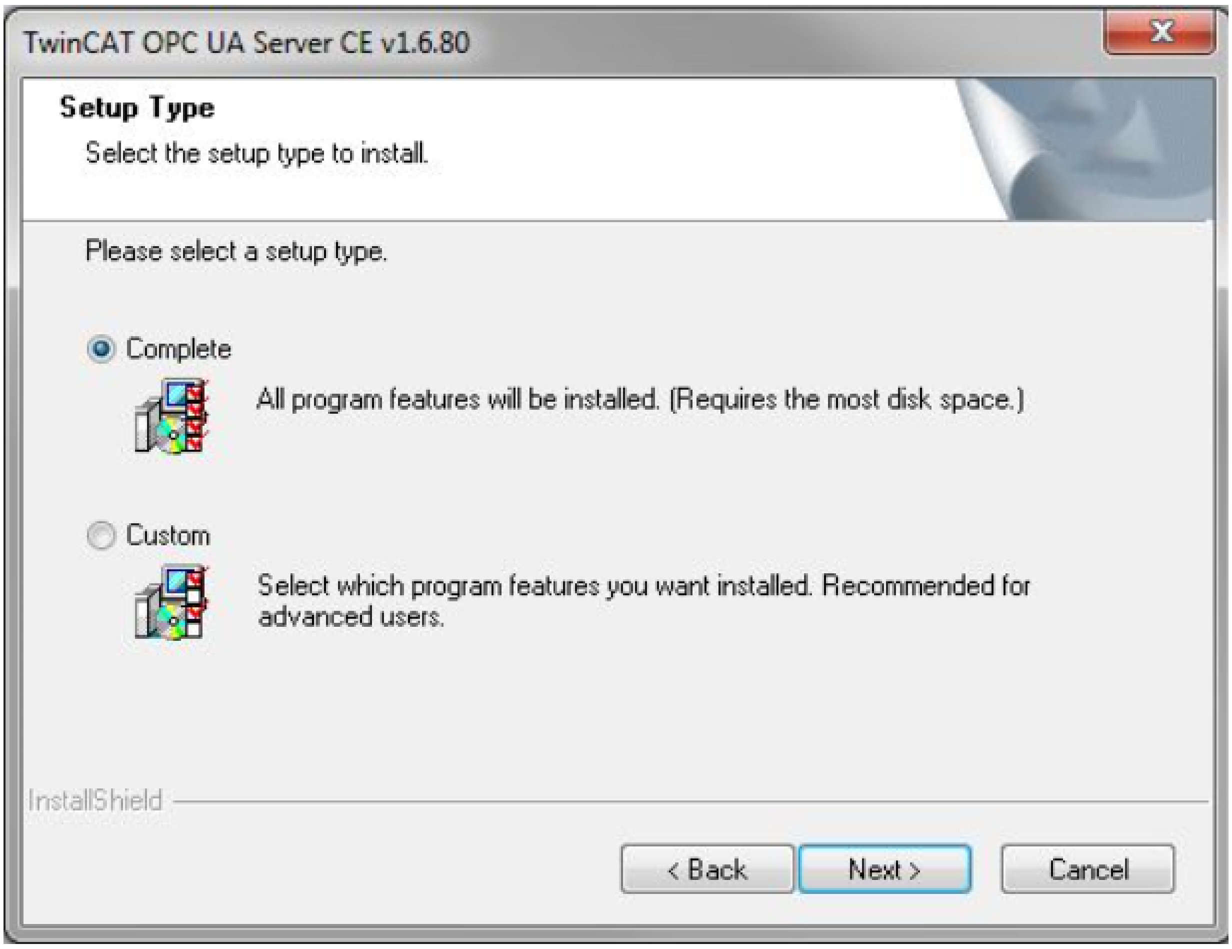
Insert the Soft-Key. (No trial version available for now)

User Name: Max Mustermann

Company Name: Mustermann Inc.

Serial Number: xxx-xxx-xxx-xxx

5. Select **Full** as the installation type and click on **Continue**



TwinCAT OPC UA Server CE v1.6.80

Setup Type

Select the setup type to install.

Please select a setup type.

(●) Complete

All program features will be installed. (Requires the most disk space.)

( ) Custom

Select which program features you want installed. Recommended for advanced users.

6. Click **Install** to start the installation.

⇨ After installation you will find the setup files for Windows CE in the directory ...*TwinCAT\CE*. This directory contains setup files for the following CE platforms:

- TwinCAT OPC UA Client CE\I586: OPC UA Client (PLC library) for x86-based CPUs (such as CX10xx, CP62xx, C69xx,...)
- TwinCAT OPC UA Client CE\ARMV4I: OPC UA Client (PLC library) for ARM-based CPUs (such as CX9001, CX9010, CP6608, ...)
- TwinCAT OPC UA Server CE\I586: OPC UA Server for x86-based CPUs (such as CX10xx, CP62xx, C69xx, ...)
- TwinCAT OPC UA Server CE\ARMV4I: OPC UA Server for ARM-based CPUs (such as CX9001, CX9010, CP6608, ...)

Transfer the setup file to the Windows CE device

Transfer the corresponding executable file to the Windows CE device. There are various options for transferring the setup file:

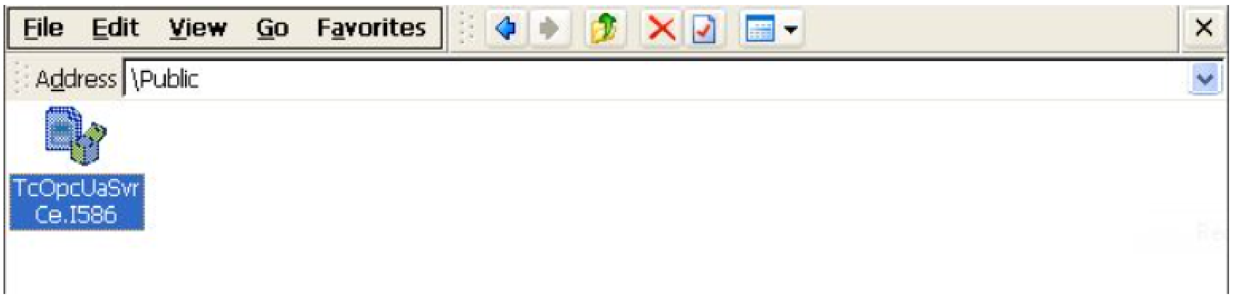
- from a shared folder
- via the integrated FTP server
- via ActiveSync
- via a CF card

Further information can be found in the Beckhoff Information System in the "Operating Systems" documentation (Embedded PC > Operating Systems > CE).

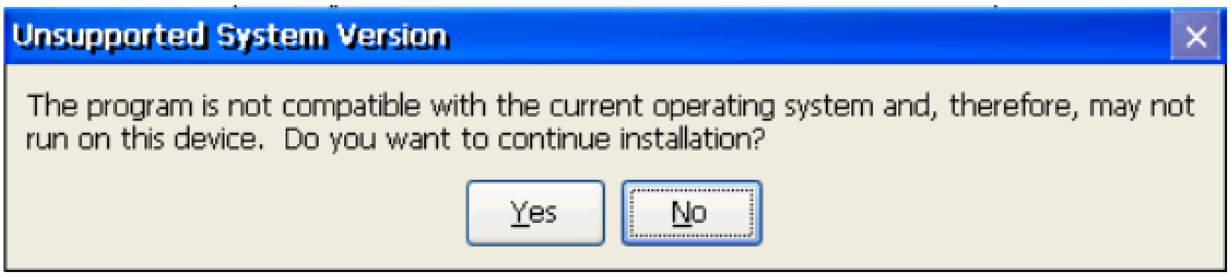
Run the setup file on the Windows CE device

Execute the transferred setup file *TcOpcUaSvrCe.xxxx.CAB* on the CE device:

1. Navigate to the directory to which you have transferred the setup file.

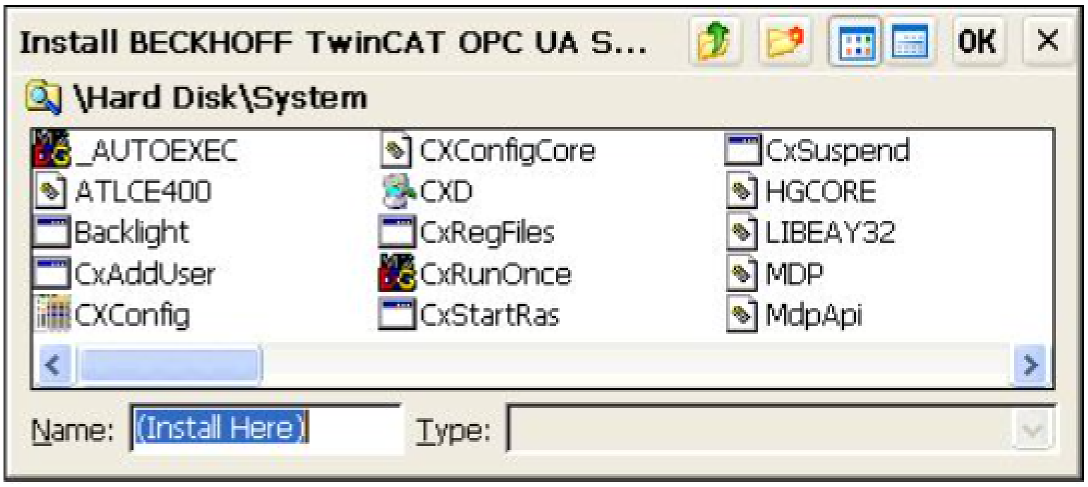


2. Double-click the CAB file. If a message dialog box appears that says this program is not compatible with the current operating system, make sure you are using the correct CAB file (ARM, I586) for your IPC/ Embedded PC.
3. If you are sure that the CAB file matches the Embedded PC/IPC, confirm this message dialog with **Yes**.


Unsupported System Version

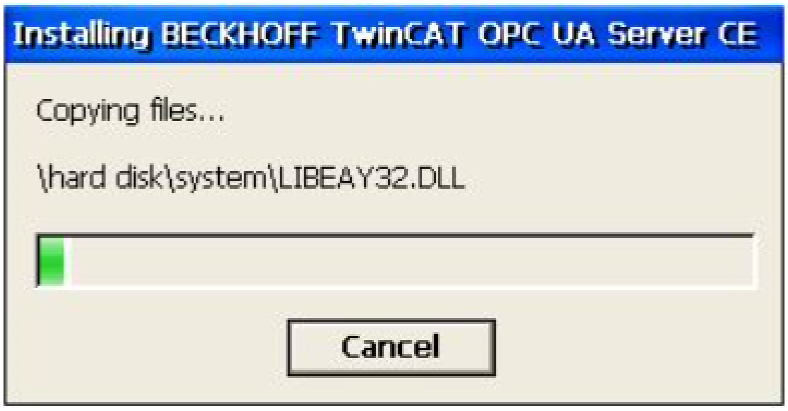
The program is not compatible with the current operating system and, therefore, may not run on this device. Do you want to continue installation?

4. Select \Hard Disk\\System\\ as destination directory


Install BECKHOFF TwinCAT OPC UA S...

\Hard Disk\System

5. Click **OK** to start the installation.


Installing BECKHOFF TwinCAT OPC UA Server CE

Copying files...

\hard disk\system\LIBEAY32.DLL

⇨ After installation, the setup file is automatically deleted.

Once you have installed the OPC UA Server, this component will be available after restarting your Windows CE device.

3.7 Licensing (TC3)

The TwinCAT 3 function can be activated as a full version or as a 7-day test version. Both license types can be activated via the TwinCAT 3 development environment (XAE).

The licensing of a TwinCAT 3 function is described below. The description is divided into the following sections:

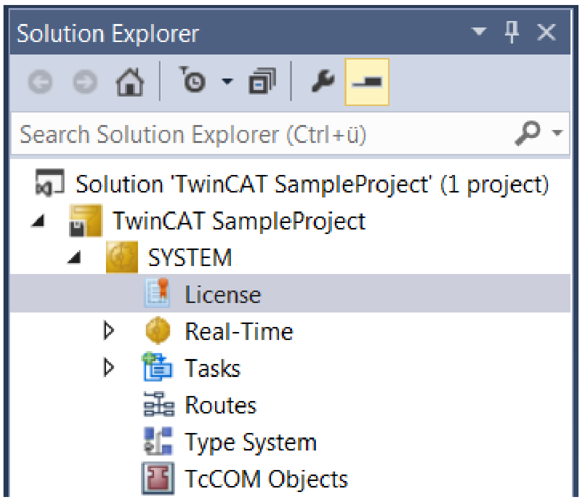
- Licensing a 7-day trial version [▸ 32]
- Licensing a full version [▸ 34]

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

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

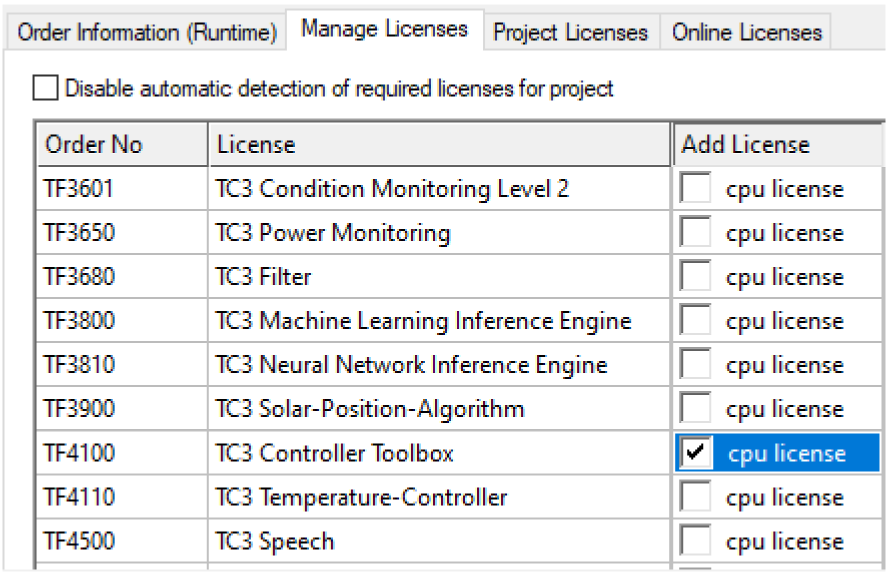
A 7-day test version cannot be enabled for a TwinCAT 3 license dongle.

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").

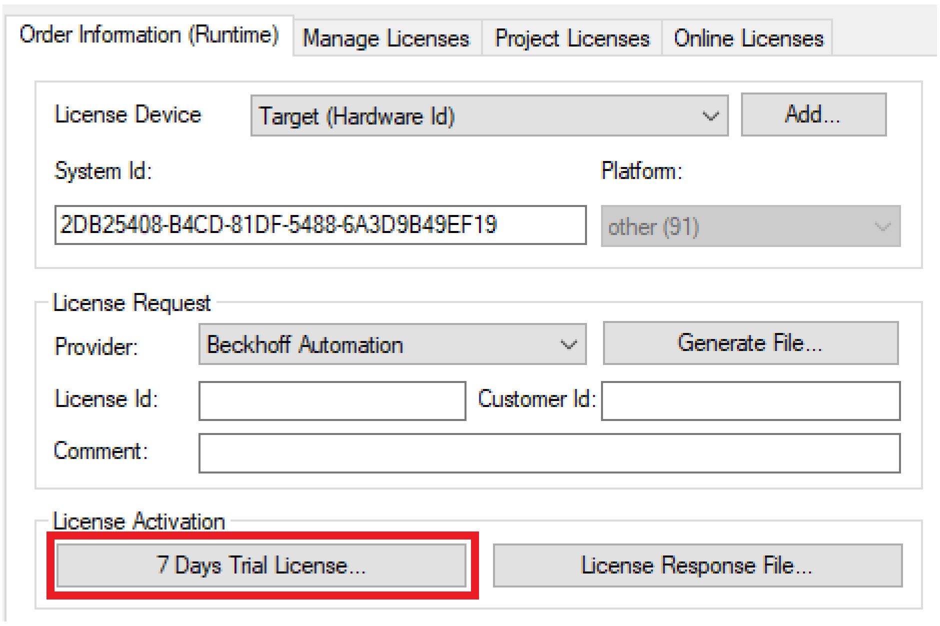


Order Information (Runtime) | Manage Licenses | Project Licenses | Online Licenses

☐ Disable automatic detection of required licenses for project

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.



Order Information (Runtime) | Manage Licenses | Project Licenses | Online Licenses

License Device: Target (Hardware Id)

System Id: 2DB25408-B4CD-81DF-5488-6A3D9B49EF19

Platform: other (91)

License Request

Provider: Beckhoff Automation

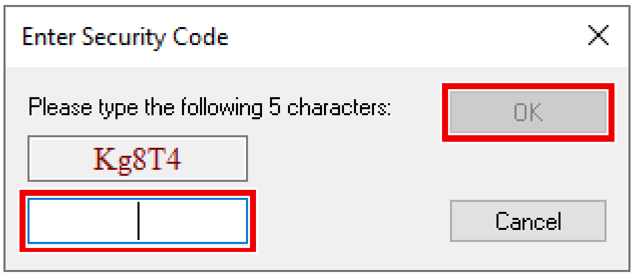
License Id:

Customer Id:

Comment:

License Activation

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



Enter Security Code

Please type the following 5 characters:

Kg8T4

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".

3.8 Licensing (TC2)

The licensing of the TwinCAT OPC UA for TwinCAT 2 takes place during the installation [▶ 22] by entering a license key.

4 Technical introduction

4.1 Server

4.1.1 Overview

The TwinCAT OPC UA Server provides a standardized communication interface for accessing symbol values from the TwinCAT runtime. This facilitates integration of third-party software for reading or writing variable values. This part of the documentation describes the various configuration options available for the TwinCAT OPC UA Server.

We recommend following our QuickStart [▸ 35] guide. Afterwards, please also refer to our article Recommended steps [▸ 42] and Optimizations [▸ 44].

4.1.2 Quick start

After successful installation, execute the following steps to make PLC variables available via the TwinCAT OPC UA Server.

- Step 1: initialize OPC UA Server
- Step 2: configure PLC variables for the OPC UA access
- Step 3: configure the download of the symbol file
- Step 4: activate the license for the server
- Step 5: activate the project
- Step 6: establish a connection to the OPC UA Server

Delivery state

This quick-start procedure assumes that the TwinCAT OPC UA Server is in the delivery state. The server is automatically configured to access the first (local) PLC runtime.

Step 1: initialize OPC UA Server

Initialize the TwinCAT OPC UA Server as described in the corresponding documentation article for initialization [▸ 37].

Step 2: configure PLC variables for the OPC UA access

Open an existing PLC project and insert the following comment before the selected variables. Alternatively you can create a new PLC project.

TwinCAT 3 (TMC import):

```
{attribute 'OPC.UA.DA' := '1'}
bVariable : BOOL;
```

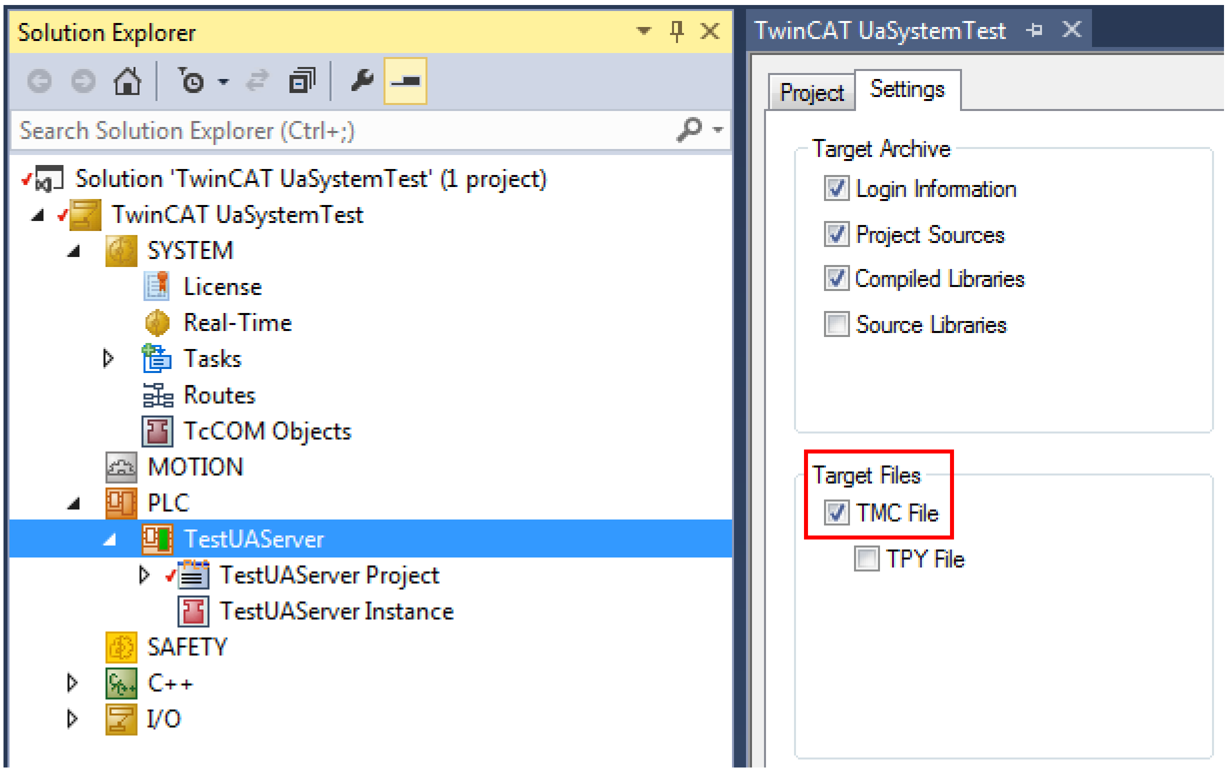
TwinCAT 2 (TPY import):

```
bVariable : BOOL; (*~ (OPC:1:some description) *)
```

Step 3: configure the download of the symbol file

By default, the TwinCAT OPC UA Server establishes a connection with the first PLC runtime on the local system and uses the corresponding symbol file to build the namespace.

To make the symbol file available, activate the download of the symbol file in the settings of the PLC project.



Solution 'TwinCAT UaSystemTest' (1 project)
TwinCAT UaSystemTest
SYSTEM
License
Real-Time
Tasks
Routes
TcCOM Objects
MOTION
PLC
TestUAServer
TestUAServer Project
TestUAServer Instance
SAFETY
C++
I/O

Target Archive
- [✓] Login Information
- [✓] Project Sources
- [✓] Compiled Libraries
- [ ] Source Libraries

Target Files
- [✓] TMC File
- [ ] TPY File

Step 4: activate the license for the server

Check whether a license exists. In TwinCAT 3 you can enter a TF6100 license in the TwinCAT license management (see Licensing (TC3) [▸ 32]). A 7-day trial license is sufficient for a basic introduction.

Step 5: activate the project

Activate the TwinCAT project and restart TwinCAT. This will also restart the TwinCAT OPC UA Server (please refer to the note below). Then log into the PLC runtime and start the PLC program.

NOTICE

Note on the Windows CE platform

Please note that the TwinCAT OPC UA Server under Windows CE is not restarted with a TwinCAT restart. Here you have to either restart the CE device or call the Restart() method from the configuration namespace [▸ 110] to restart the server so that it reads the TMC file and provides the PLC variables.

Step 6: establish a connection to the OPC UA Server

To establish a connection from an OPC UA Client, the client must establish a connection with the URL of the OPC UA Server, e.g. opc.tcp://CX-12345:4840 or opc.tcp://192.168.1.1:4840.

Default port and end points

Please also note our information on the default port and end points [▸ 105] used by the server.

To connect to the TwinCAT OPC UA Server you can use the OPC UA Sample Client provided, which can be called up via the Windows start menu.

For more advanced testing we recommend using the free UA Expert software from Unified Automation.

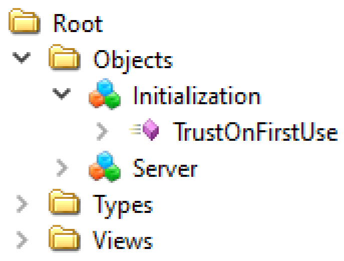
After successful connection to the server you will find the released PLC variables under the object PLC1.

4.1.3 Initialization

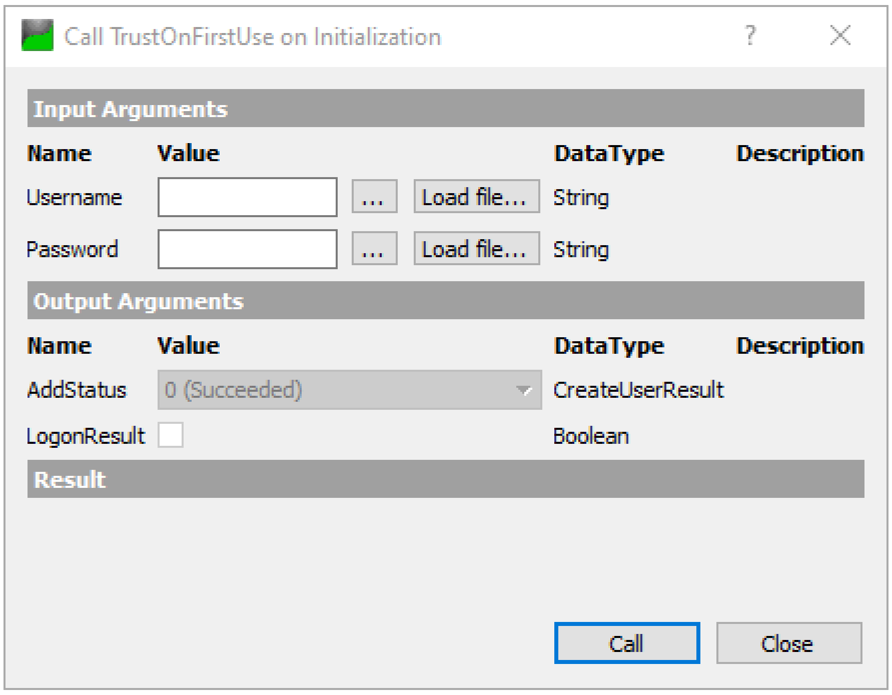
Starting with setup version 4.4.0, the TwinCAT OPC UA Server requires an initialization phase, which is based on the TOFU principle (Trust On First Use). This means that the server must be actively initialized by the user so that it can be used for its various functions (Data Access, Historical Access, 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:


Call TrustOnFirstUse on Initialization

Input Arguments

Name	Value	DataType	Description
Username		String	
Password		String	

Output Arguments

Name	Value	DataType	Description
AddStatus	0 (Succeeded)	CreateUserResult	
LogonResult	☐	Boolean	

Result

Parameter	Description
[in] Username	User name 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 user name/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.

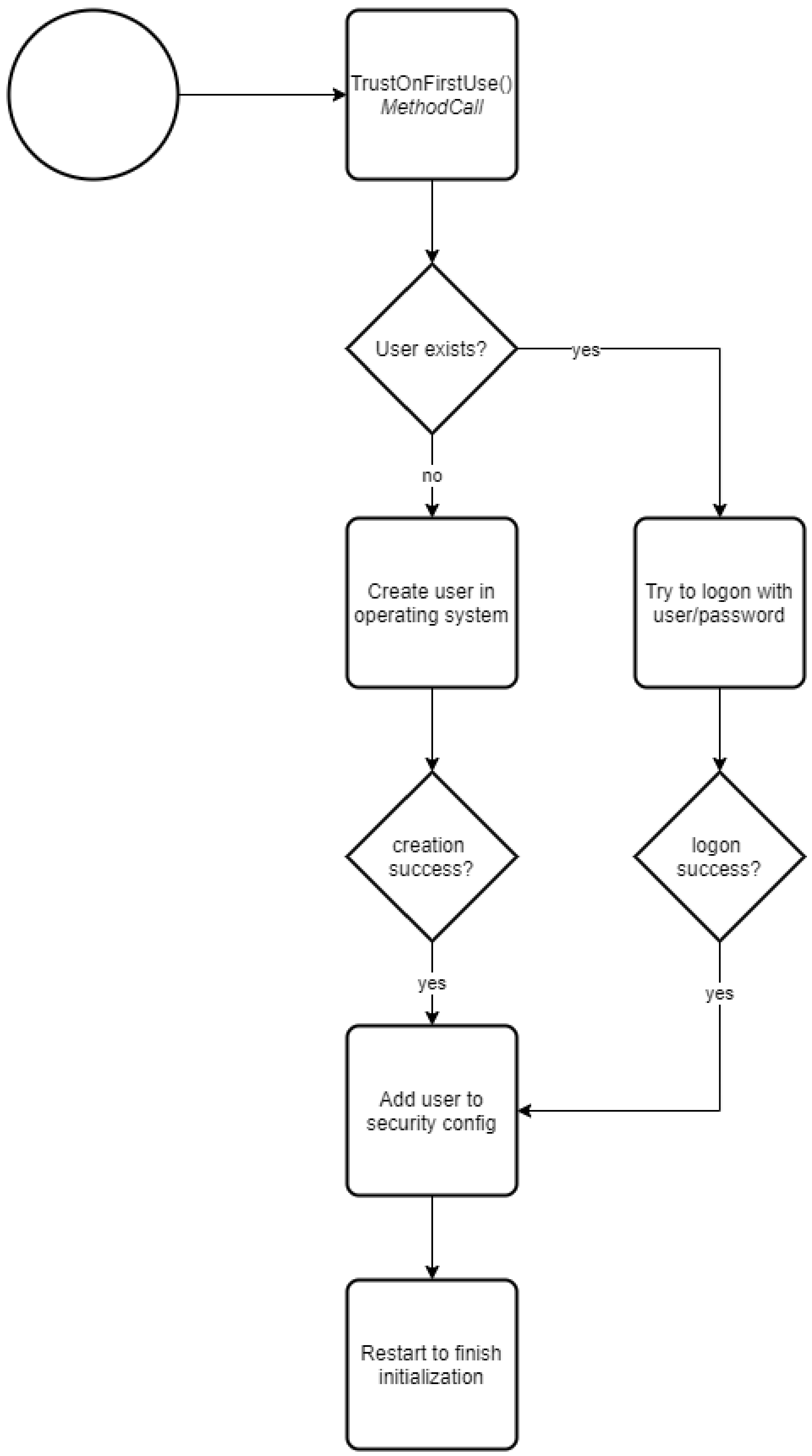
The server is initialized by calling this method. The method tries to create a user specified by the user in the lower-level operating system of the server. If this is successful, the user is automatically added to the security configuration (TcUaSecurityConfig.xml) of the server and defined as server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

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

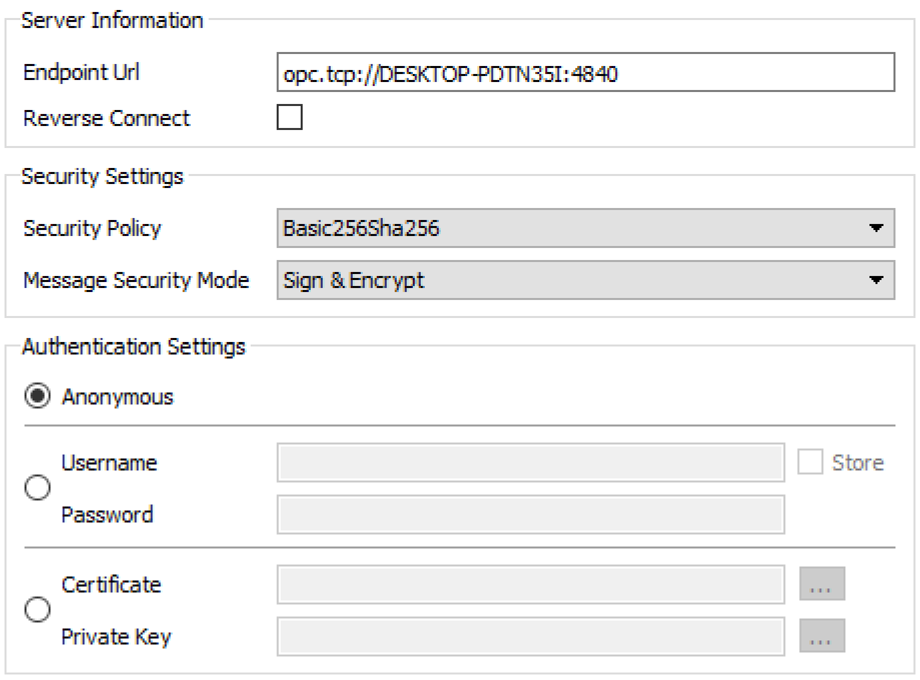

TrustOnFirstUse()
MethodCall
User exists?
yes
no
Create user in operating system
Try to logon with user/password
creation success?
logon success?
yes
yes
Add user to security config
Restart to finish initialization

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.



Server Information

Endpoint Url	opc.tcp://DESKTOP-PDTN35I:4840
Reverse Connect	☐

Security Settings

Security Policy	Basic256Sha256
Message Security Mode	Sign & Encrypt

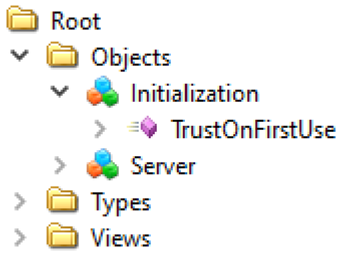
Authentication Settings

● Anonymous

○ Username / Password ☐ Store

○ Certificate / Private Key

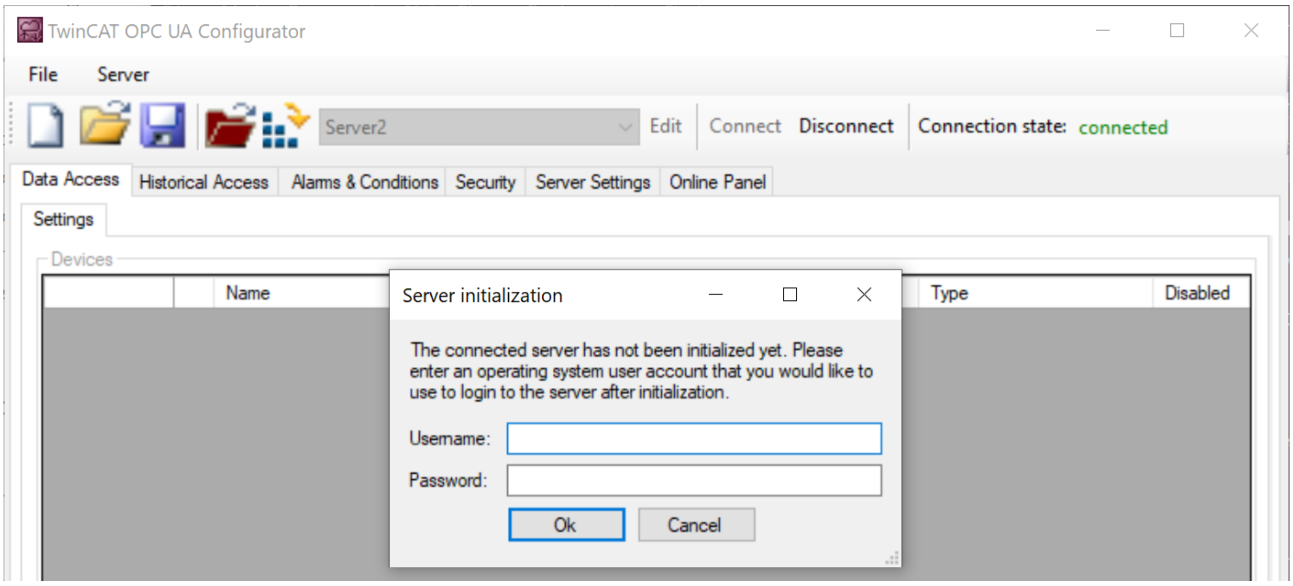
After the connection has been established, the initialization object together with the TrustOnFirstUse method can be found in the server's address space.



```
Root
  Objects
    Initialization
      TrustOnFirstUse
    Server
  Types
  Views
```

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:



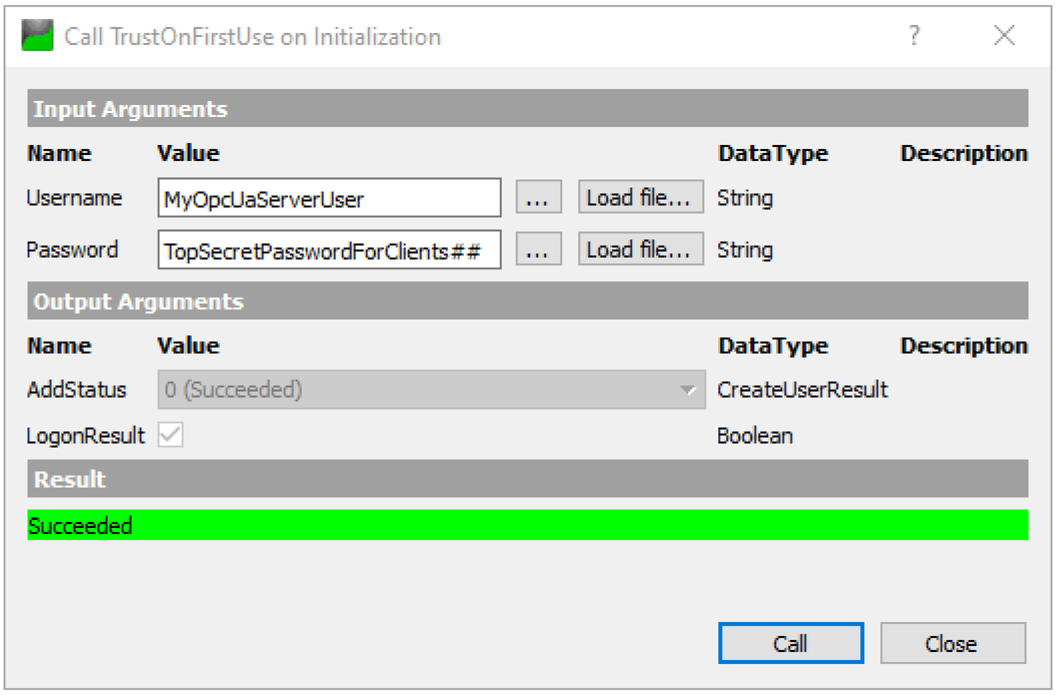
The connected server has not been initialized yet. Please enter an operating system user account that you would like to use to login to the server after initialization.

Username:

Password:

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

In UA Expert, you now call the TrustOnFirstUse method to create a user and configure the server for that user. "MyOpcUaServerUser" was used as the user name 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.



Input Arguments

Name	Value	DataType	Description
Username	MyOpcUaServerUser	String	
Password	TopSecretPasswordForClients##	String	

Output Arguments

Name	Value	DataType	Description
AddStatus	0 (Succeeded)	CreateUserResult	
LogonResult	✓	Boolean	

Result

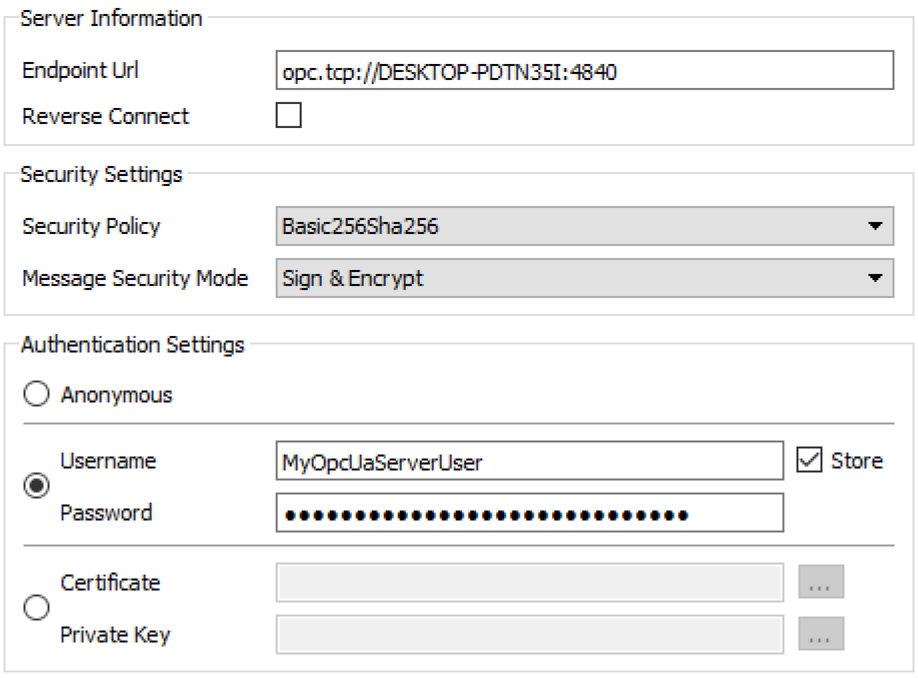
Succeeded

The AddStatus parameter indicates that the user was successfully created in the operating system's user database. The LogonResult parameter 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

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 login must be done using the specified user name.


Server Information

Endpoint Url: opc.tcp://DESKTOP-PDTN35I:4840

Reverse Connect: ☐

Security Settings

Security Policy: Basic256Sha256

Message Security Mode: Sign & Encrypt

Authentication Settings

○ Anonymous

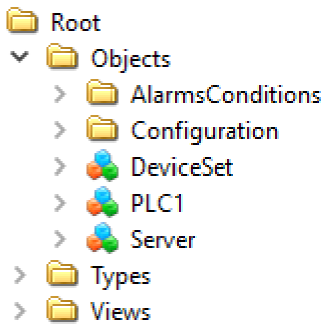
● Username: MyOpcUaServerUser ☑ Store

Password: ••••••••••••••••••••••••••••

○ Certificate:

Private Key:

After the connection has been established, the regular namespaces and objects can now be found in the server's address space and the project planning for the application can begin.



Authorizations of the TOFU user

The user configured by the TOFU mechanism has full access to the server, which may not be desirable. We therefore recommend creating an explicit user for pure data access in the next step. This is described in more detail in the article Recommended steps [▶ 42].

4.1.4 Recommended steps

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

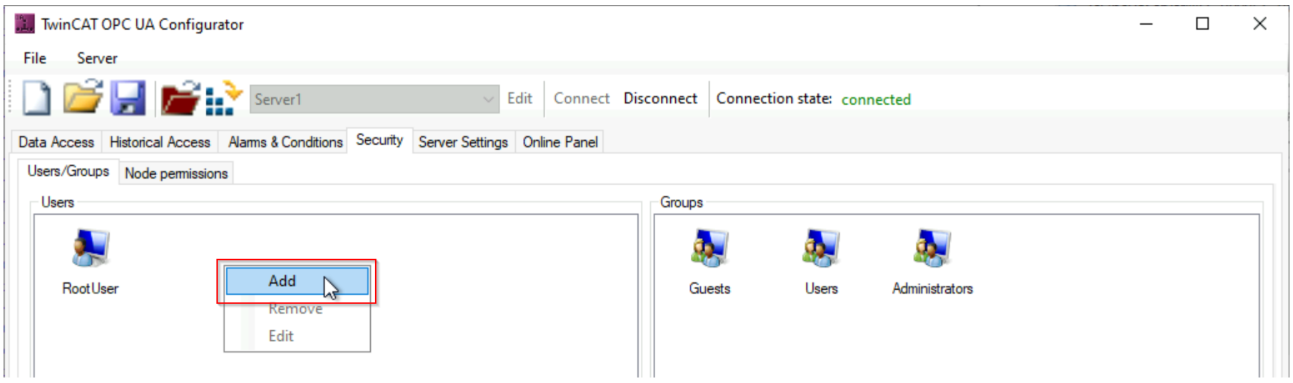
Use secure IdentityToken

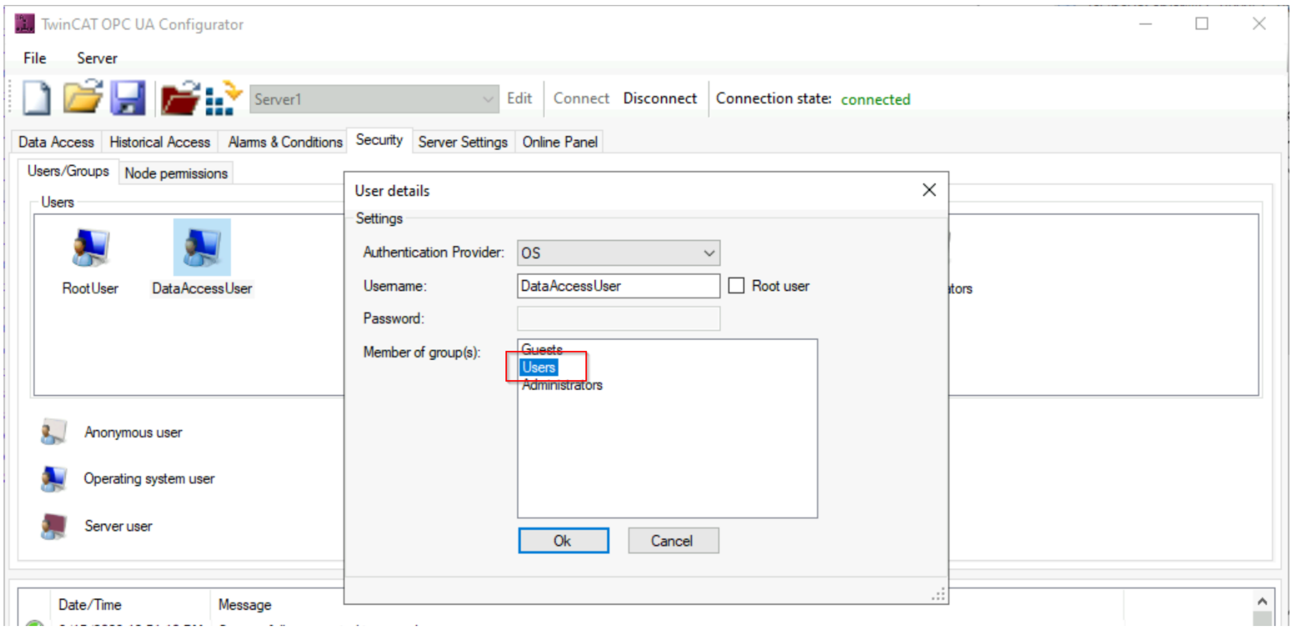
The one-time initialization [▶ 37] of the server disables the "Anonymous" IdentityToken. For security reasons, we recommend leaving this disabled and only allowing access to the server for authenticated client applications, e.g. via the user name/password authentication configured by default during initialization.

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. However, 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.

We therefore recommend configuring an additional, dedicated user who is given the necessary authorizations to access variables on a Data Access device, but who is not allowed to access the configuration namespace [▶ 110]. This setting can be made via the configurator [▶ 115] by adding a new user who is assigned to the "Users" group.





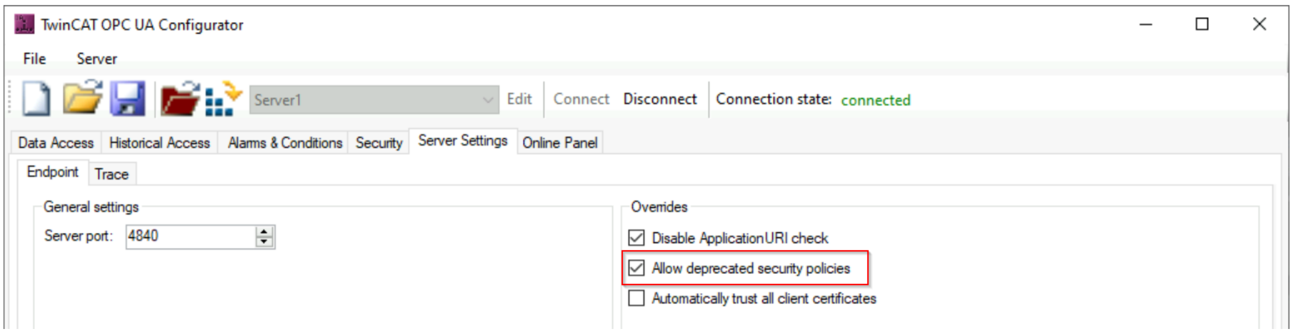
Authentication Provider: OS
Username: DataAccessUser
☐ Root user
Password:
Member of group(s): Users

The newly configured user then has all the necessary authorizations 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.

Leave insecure endpoints disabled

Endpoints [▶ 105] classified as unsafe are not offered by the TwinCAT OPC UA Server by default. A configuration switch in TcUaServerConfig.xml can be used to make unsafe endpoints available in the server again, but we strongly do not recommend this.

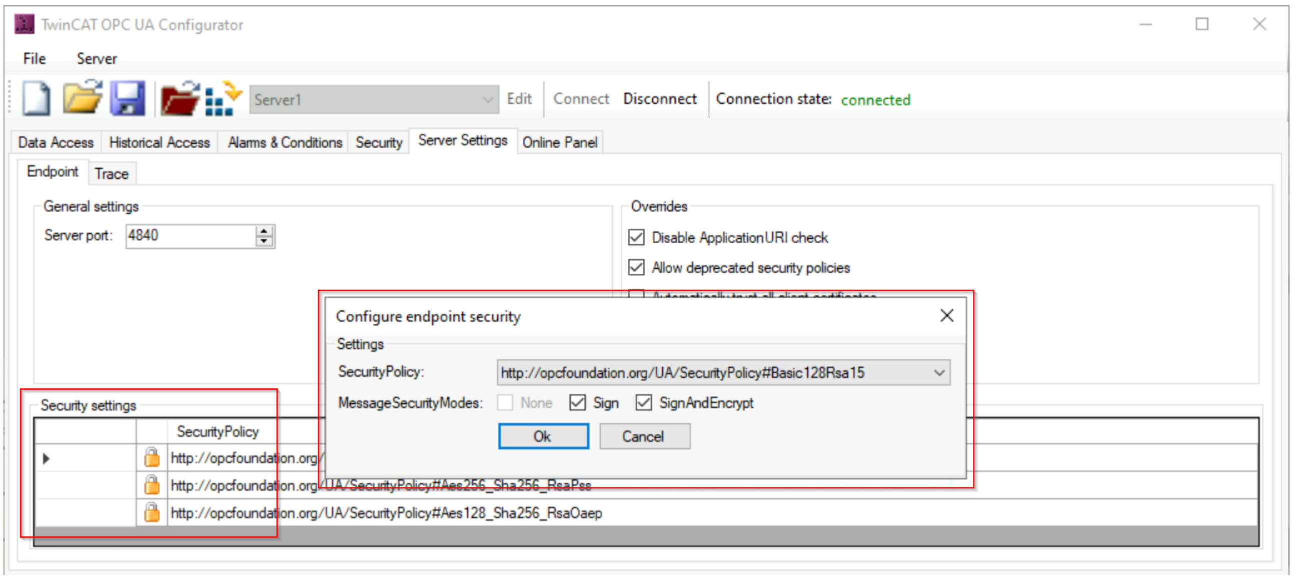


Server port: 4840
☑ Disable ApplicationURI check
☑ Allow deprecated security policies
☐ Automatically trust all client certificates

Alternatively, you can also manually adjust the corresponding entry in the configuration file TcUaServerConfig.xml:

```
<AllowDepcrecatedSecurityPolicies>true</AllowDeprecatedSecurityPolicies>
```

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:



Server port: 4840

☑ Disable ApplicationURI check
☑ Allow deprecated security policies

Configure endpoint security

Settings

SecurityPolicy: http://opcfoundation.org/UA/SecurityPolicy#Basic128Rsa15

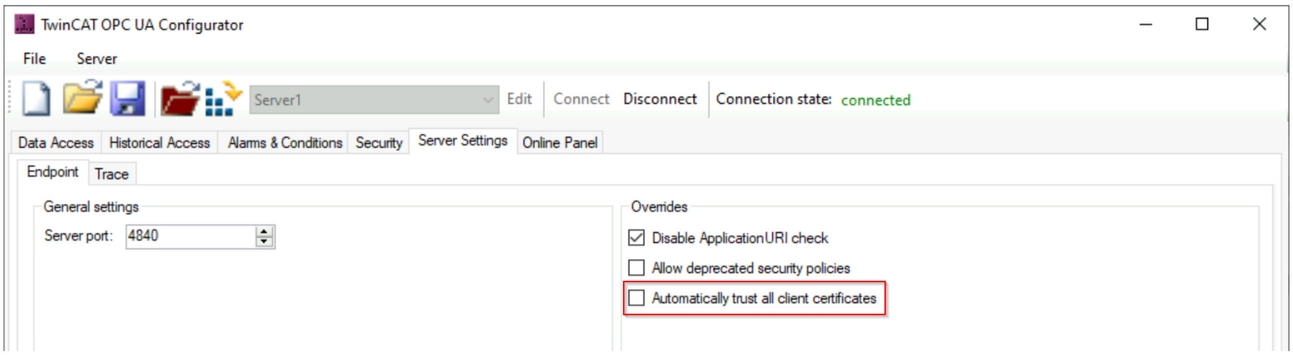
MessageSecurityModes: ☐ None ☑ Sign ☑ SignAndEncrypt

http://opcfoundation.org/
http://opcfoundation.org/UA/SecurityPolicy#Aes256_Sha256_RsaPss
http://opcfoundation.org/UA/SecurityPolicy#Aes128_Sha256_RsaOaep

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

Disable 'AutomaticallyTrustAllClientCertificates'

By default, the server is configured to automatically trust all client certificates. For security reasons, we recommend disabling this setting. This setting can be made via the configurator [▶ 115]:



Server port: 4840

☑ Disable ApplicationURI check
☐ Allow deprecated security policies
☐ Automatically trust all client certificates

Alternatively, you can also manually adjust the corresponding entry in the configuration file TcUaServerConfig.xml:

```
<AutomaticallyTrustAllClientCertificates>false</AutomaticallyTrustAllClientCertificates>
```

After disabling this setting, a trust relationship [▶ 108] must be established between the client and server by having the applications trust each other's certificates.

4.1.5 Optimizations

There are many different ways to optimize the communication connection between OPC UA Client and server or PLC.

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.

The following article provides further information on the following topics:

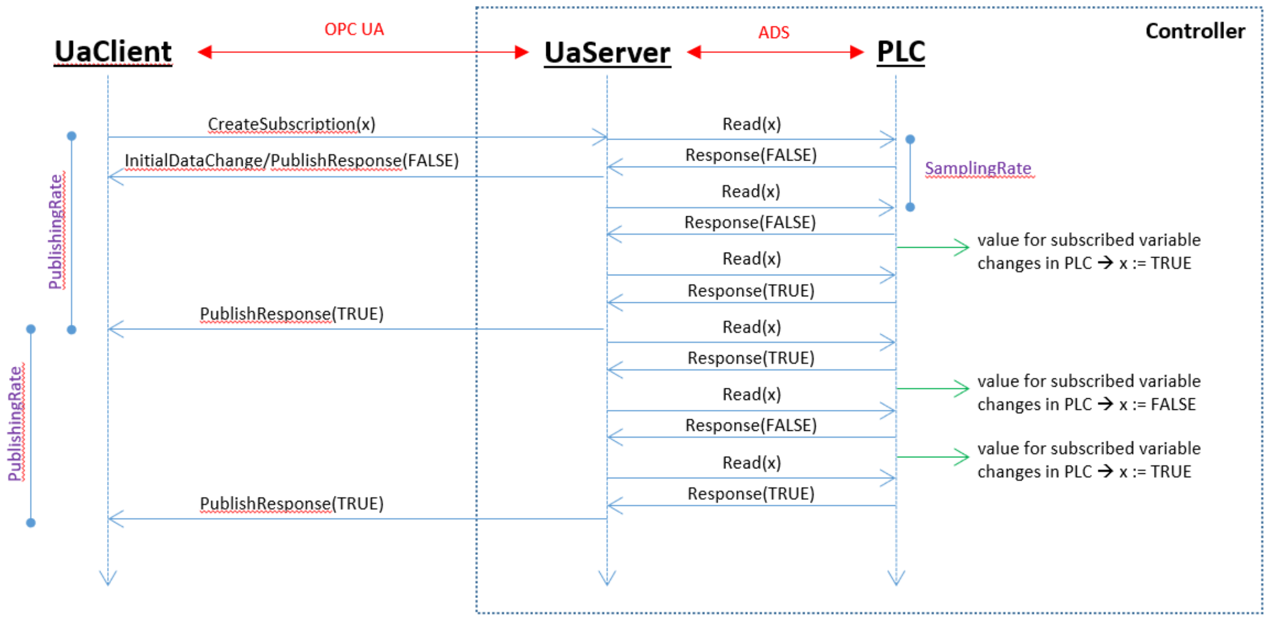
- SamplingInterval vs. PublishingInterval
- StructuredTypes
- StructuredTypes and their member variables

SamplingInterval vs. PublishingInterval

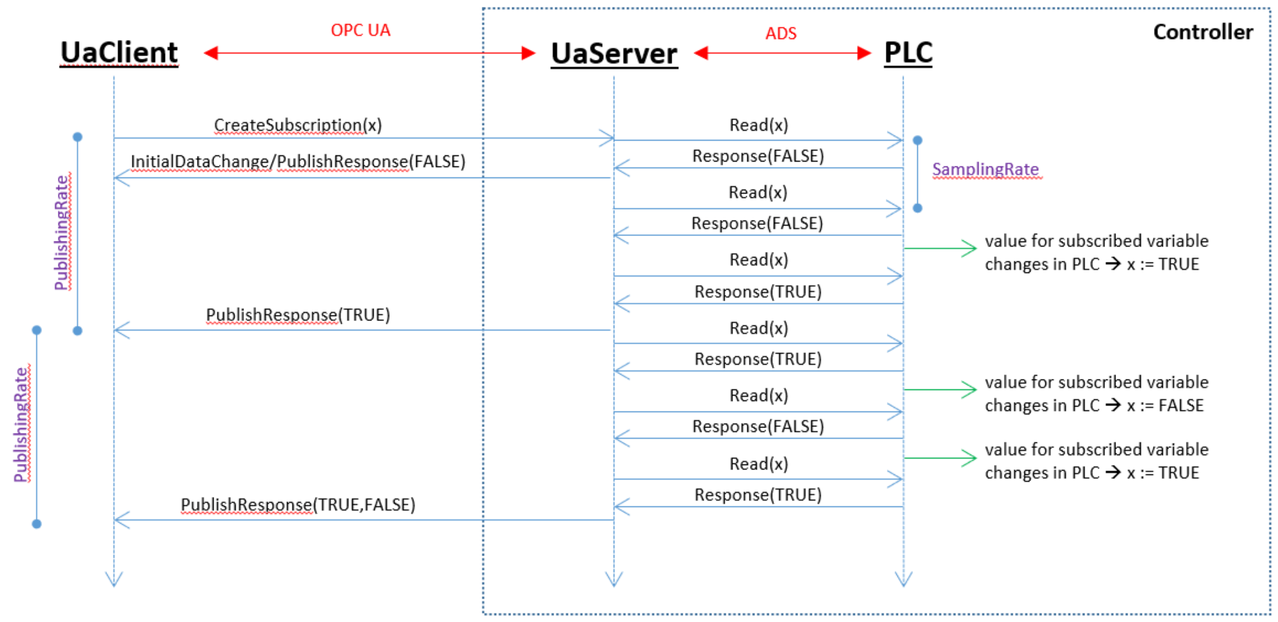
When creating a subscription, an OPC UA Client uses various parameters for the subscription and the so-called MonitoredItems contained in it to receive notifications about variable changes. The following table explains two of these parameters, which will then be described in more detail.

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 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.

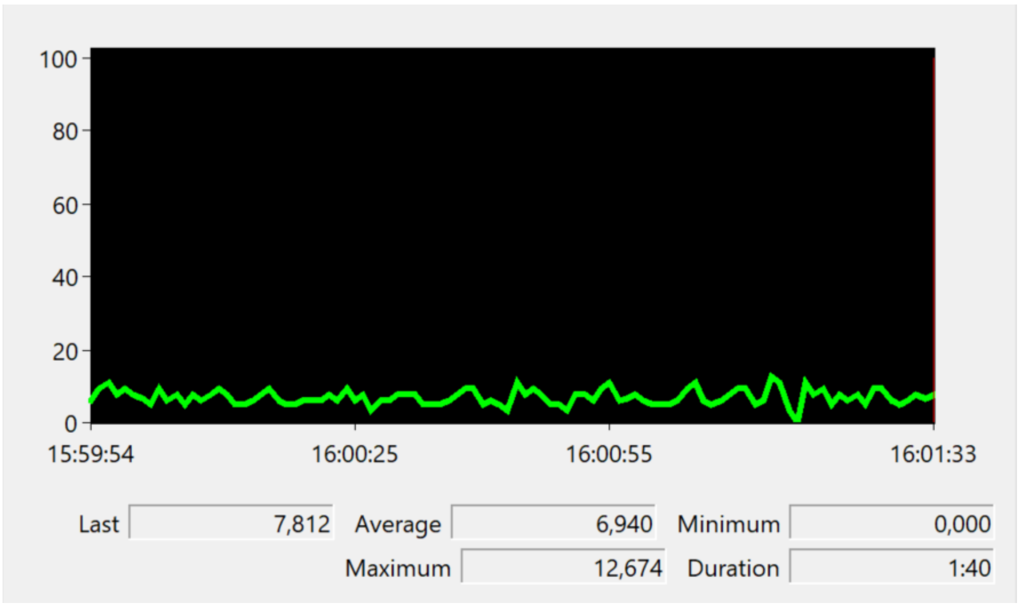


With the first PublishResponse it can be seen that only one value change is transmitted to the client, because also only one value change has taken place in the PLC. With the second PublishResponse it can be seen that two value changes have occurred in the PLC and both are also transmitted to the client.

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.

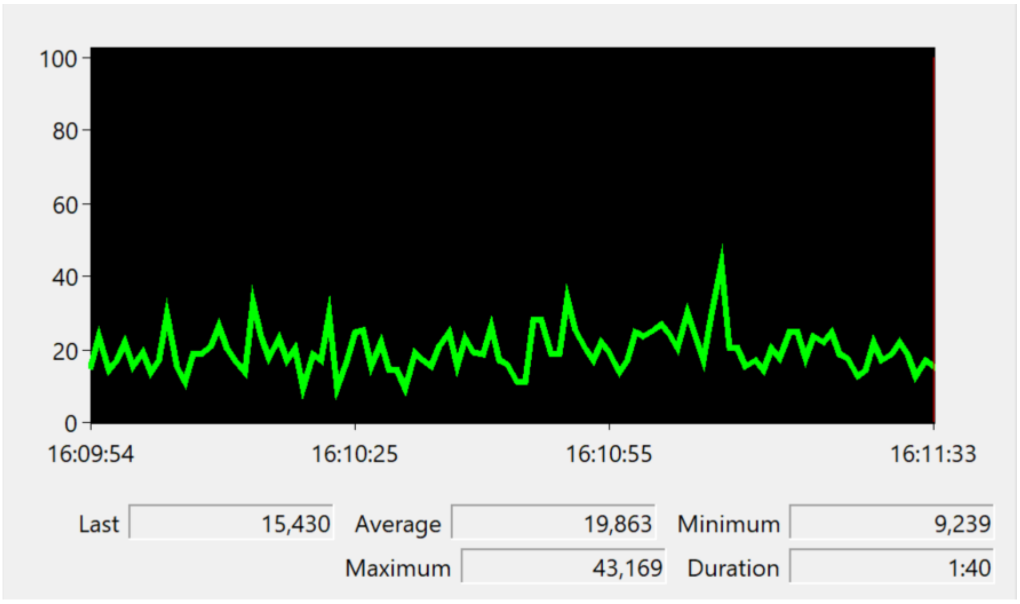
Depending on the application scenario used, the two parameters should be set appropriately. For example, if the OPC UA Client is a visualization, then fast PublishingIntervals and SamplingRates make only limited sense, since the human eye cannot process information faster than ~200 ms anyway. The use 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.



Last	7,812	Average	6,940	Minimum	0,000
		Maximum	12,674	Duration	1:40

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%.



Last	15,430	Average	19,863	Minimum	9,239
		Maximum	43,169	Duration	1:40

StructuredTypes

With the help of OPC UA StructuredTypes the TwinCAT OPC UA Server can make IEC61131 data structures [▶ 57] available in its address space for clients. However, a data structure can also be provided as a non-StructuredType, i.e. the root element is a FolderType (without value) and access can then be made to the individual member variables. In ADS communication with the PLC, both variants behave fundamentally differently.

When an OPC UA Client accesses individual member variables of a non-StructuredType, it may well happen that the ADS communication is divided among several ADS read/write requests, depending on the number of variables. This in turn can result in the ADS Read/Write Requests being processed by the PLC in different PLC cycles. The data consistency of the respective data structure can therefore not be guaranteed.

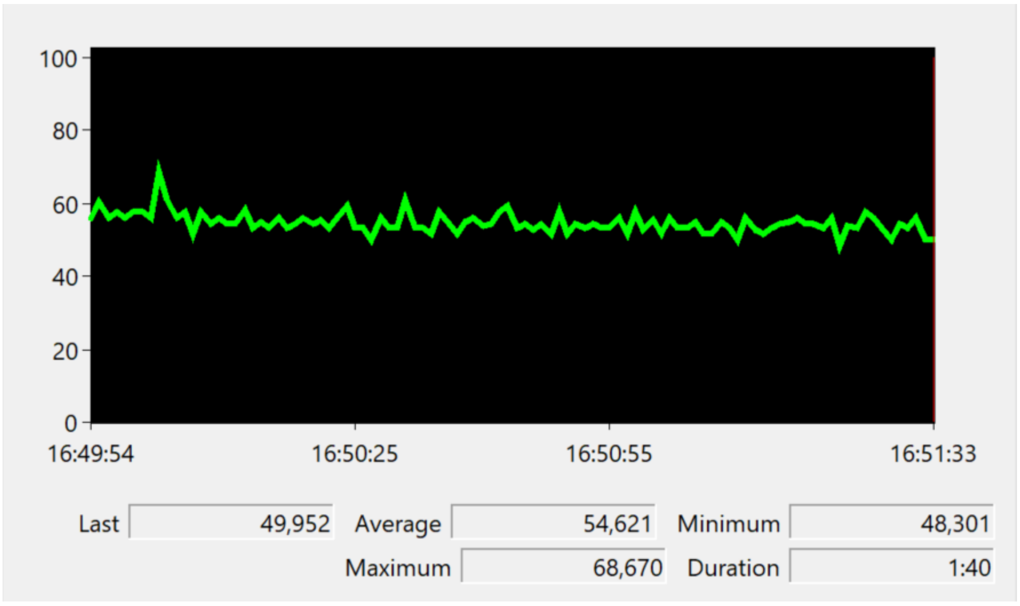
When an OPC UA Client accesses a StructuredType, on the other hand, it is ensured that the StructuredType is processed by the PLC in a single ADS Read/Write Request, thus ensuring data consistency.

When using StructuredTypes for large PLC data structures, it should be noted that the resulting ADS Read/Write Request or Response can be correspondingly large and require a correspondingly large amount of memory in the TwinCAT ADS router. In addition, StructuredTypes have to be encoded or decoded accordingly by the OPC UA Server, which requires additional CPU time. Especially in connection with the parameters for SamplingInterval and PublishingInterval that can be set for a subscription, further optimizations can be made here.

In the following sample an OPC UA Client has created a subscription with a StructuredType on the TwinCAT OPC UA Server. The lower-level PLC data structure is structured as follows:

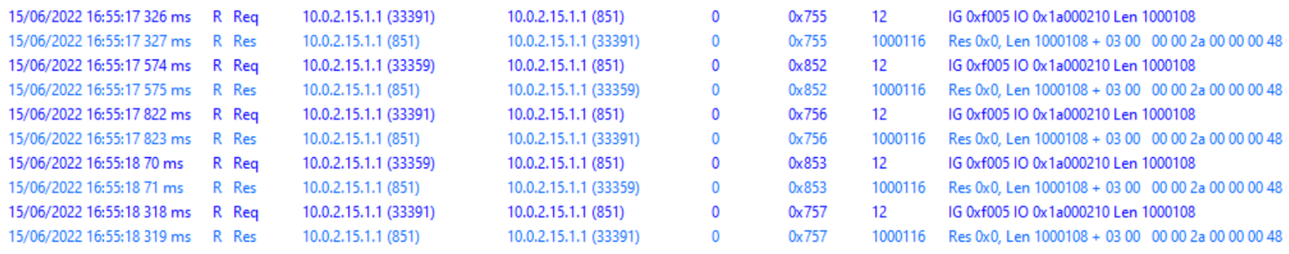
```
TYPE ST_TEST :
STRUCT
  stComplex : ST_Complex_1;
  strString5 : STRING[5];
  eEnum : E_Enum_1;
  strString3 : STRING[3];
  arrComplex : ARRAY[0..9999] OF ST_Complex_1;
  arrDint : ARRAY[0..9999] OF DINT;
END_STRUCT
END_TYPE
```

The structure ST_Complex_1 used as member variable is about 91 bytes. In total, therefore, 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.



Last	49,952	Average	54,621	Minimum	48,301
		Maximum	68,670	Duration	1:40

According to the selected SamplingInterval, an ADS Read Request 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).



```
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

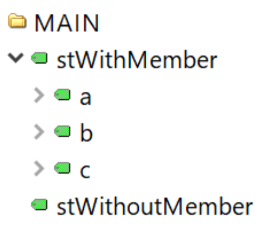
By default, the member variables of a StructuredType are represented and made available as separate nodes in the server's address space. This requires additional main memory, because the TwinCAT OPC UA Server allocates main memory for each node. However, an OPC UA Client that works exclusively with the StructuredType itself, i.e. "the root element", does not need these additional nodes. These can therefore be explicitly hidden via the OPC.UA.DA:=2 attribute. Sample:

```
TYPE ST_Test :
STRUCT
  a : DINT;
  b : STRING;
  c : DINT;
END_STRUCT
END_TYPE

{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.StructuredType' := '1'}
{attribute 'OPC.UA.DA.Description' := 'Complex (structured) type '}
stWithMember : ST_Test;

{attribute 'OPC.UA.DA' := '2'}
{attribute 'OPC.UA.DA.StructuredType' := '1'}
{attribute 'OPC.UA.DA.Description' := 'Complex (structured) type '}
stWithoutMember : ST_Test;
```

This declaration results in the address space shown below:



4.1.6 Data Access

4.1.6.1 PLC

This section describes how to configure the namespace of the OPC UA Server such that it contains variables from a TwinCAT PLC runtime. The TwinCAT OPC UA Server can represent several namespaces, i.e., several PLC runtimes. In order for a PLC variable to be accessible via the respective namespace, it must be explicitly enabled for this purpose in the PLC program.

Quick start

Note that the OPC UA Server always connects by default to the first local PLC runtime system, therefore a configuration is not necessary in most cases (see also Quick start [▶ 35]). A separate configuration is only necessary in a few cases, e.g., if additional runtimes are to be displayed in the server.

This documentation article contains the following topics:

- General Information [▶ 49]
- Step 1: Selecting PLC variables to be publicly accessible via OPC UA [▶ 50]
- Step 2: Downloading the symbol description and activating it in the PLC project [▶ 53]
- [Optional] Step 3: Configuring the data access device in the OPC UA Server [▶ 54]
- [Optional] Step 4: Explicit hiding of variables [▶ 54]

General Information

Several parameters are available for configuring the server and accessing PLC variables, which can be set via the TwinCAT OPC UA Configurator.

Parameter	Description	Possible values
ADS Port	Defines the ADS port under which the PLC runtime can be reached. The ADS port can be read in the properties of the PLC project.	800 (BC Controller) 801 (TwinCAT 2) 811 (TwinCAT 2) … 851 (TwinCAT 3 - standard) 852 (TwinCAT 3) …
AutoCfg	Defines the runtime type, e.g., PLC, C++, I/O. Some AutoCfg options are available as filtered or unfiltered variants. Filtered means the user can determine which symbols are to be published via OPC UA (see below). When using an unfiltered option, each symbol is made available via OPC UA.	7 TwinCAT 2 (TPY) 8 TwinCAT 2 (TPY) filtered 4040 TwinCAT 3 (TMC) 4041 TwinCAT 3 (TMC) filtered
AutoCfgSymFile	Symbol file of the respective PLC runtime. By default, the automatically generated symbol file of the first PLC runtime of the local system is imported.	Path to the symbol file. Points by default to the symbol file (TMC) of the first local PLC runtime.
IoMode	Defines the method for accessing symbols. This is particularly important for accessing BC devices.	1 (access via handle - default) 3 (Access to BC Controller)
Array expansion	By default, subelements of an array are not mapped as separate nodes in the UA namespace. Instead, only the array is mapped as a single element. UA Clients can nevertheless access subelements via their IndexRange. (Some older OPC UA Clients do not yet support this option). The flag was introduced so that access is nevertheless possible for these clients. It ensures that every array position is displayed as a separate node in the UA namespace. This leads to higher memory requirement of the OPC UA Server.	0 (disabled - default) 1 (enabled)
Disabled	Disables the PLC runtime in the UA namespace, so that the corresponding node is not displayed. It is advisable to enable this parameter if certain PLC runtimes are not yet available at the time of project planning, for example because the corresponding devices are not yet connected to the network.	0 (disabled - default) 1 (enabled)

The following section uses a sample to illustrate how the variables can be imported from a PLC program into the server. It is assumed that the PLC runtime and the TwinCAT OPC UA Server in its standard configuration (delivery state after installation) are on the same computer.

Step 1: Configuring PLC variables

The TwinCAT OPC UA Server automatically establishes a connection to the first PLC runtime on the local system. The PLC symbols marked in the PLC program for OPC UA are taken into account when the server is started. A comment at the appropriate position (instance, structure, variable) in the PLC program code is used for identification (see the following samples).

Sample 1

In this sample, the PLC variables bMemFlag1, bMemFlag2, bMemAlarm2 and iReadOnly are enabled via OPC UA. The PLC variable bMemAlarm1 should not be accessible via the OPC UA Server.

TwinCAT 3 (TMC import):

```
{attribute 'OPC.UA.DA' := '1'}
bMemFlag1 : BOOL;

{attribute 'OPC.UA.DA' := '1'}
bMemFlag2 : BOOL;

bMemAlarm1 : BOOL;

{attribute 'OPC.UA.DA' := '1'}
bMemAlarm2 : BOOL;

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

TwinCAT 2 (TPY import):

```
bMemFlag1 : BOOL; (*~ (OPC:1:some description) *)

bMemFlag2 : BOOL; (*~ (OPC:1:some description) *)

bMemAlarm1 : BOOL;

bMemAlarm2 : BOOL; (*~ (OPC:1:some description) *)

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

Due of the additional comment `OPC.UA.DA.Access` the access level for the variable iReadOnly is set to "ReadOnly". The various options can be found in the complete list of PLC comments (see List of attributes and comments).

TPY import for TwinCAT 3

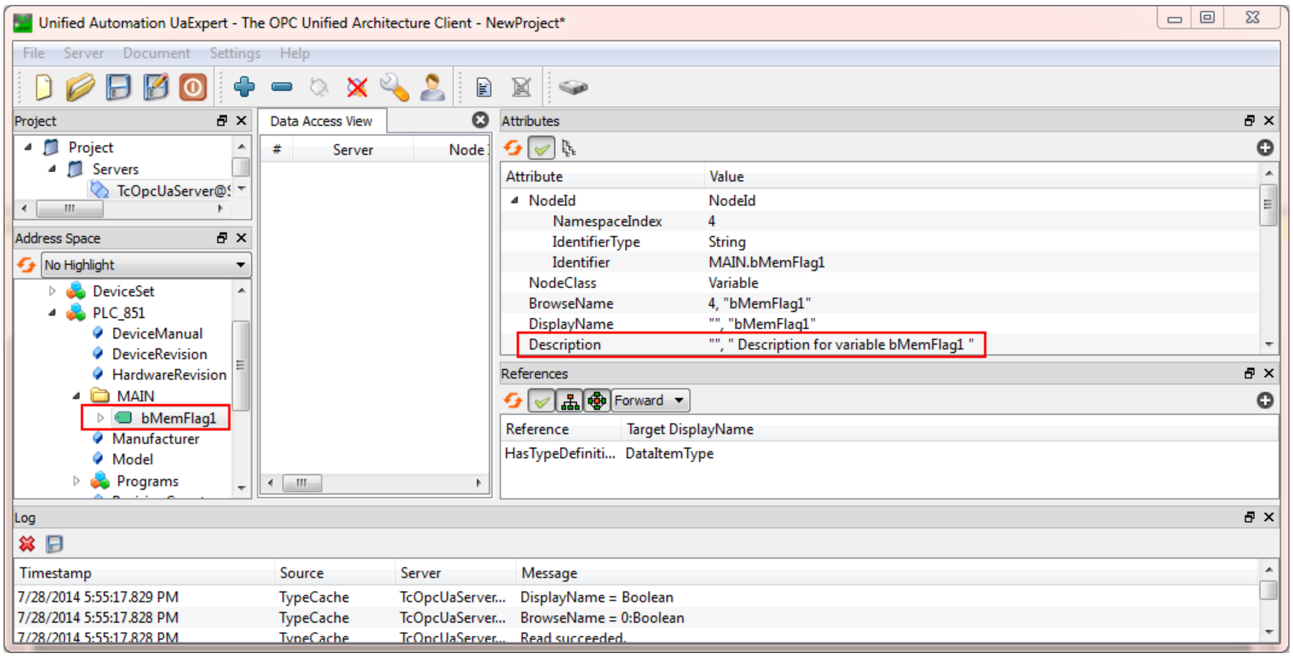
The TPY import can also be used in TwinCAT 3 PLC projects for migration purposes, e.g. if you convert a TwinCAT 2 project into a TwinCAT 3 project and do not want to make any changes to the PLC comments. Please note, however, that some advanced functions are only available for the newer import mechanism (TMC).

You may assign a description to a variable in the server, which is stored in the corresponding OPC UA attribute ("Description"). This function is only available for TwinCAT 3.

TwinCAT 3 (TMC import):

```
{attribute 'OPC.UA.DA' := '1'}
bMemFlag1 : BOOL; (* Description for variable bMemFlag1 *)
```

The comment automatically becomes the description attribute of the node in the UA namespace.



Attribute	Value
NodeId	NodeId
NamespaceIndex	4
IdentifierType	String
Identifier	MAIN.bMemFlag1
NodeClass	Variable
BrowseName	4, "bMemFlag1"
DisplayName	"", "bMemFlag1"
Description	"", " Description for variable bMemFlag1 "

Reference	Target DisplayName
HasTypeDefiniti...	DataItemType

```
Timestamp                  Source      Server            Message
7/28/2014 5:55:17.829 PM   TypeCache   TcOpcUaServer...  DisplayName = Boolean
7/28/2014 5:55:17.828 PM   TypeCache   TcOpcUaServer...  BrowseName = 0:Boolean
7/28/2014 5:55:17.828 PM   TypeCache   TcOpcUaServer...  Read succeeded.
```

Sample 2:

In this sample, the two instances fbTest1 and fbTest2 of the function block FB_BLOCK1 should be available via OPC UA. When an entire instance is released, all its symbols are also available via OPC UA. The PLC program looks like the following:

TwinCAT 3 (with TMC import):

```
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
```

TwinCAT 2 (with TPY import):

```
PROGRAM MAIN
VAR
    fbTest1 : FB_BLOCK1; (*~ (OPC:1:some description) *)
    fbTest2 : FB_BLOCK1;
END_VAR
```

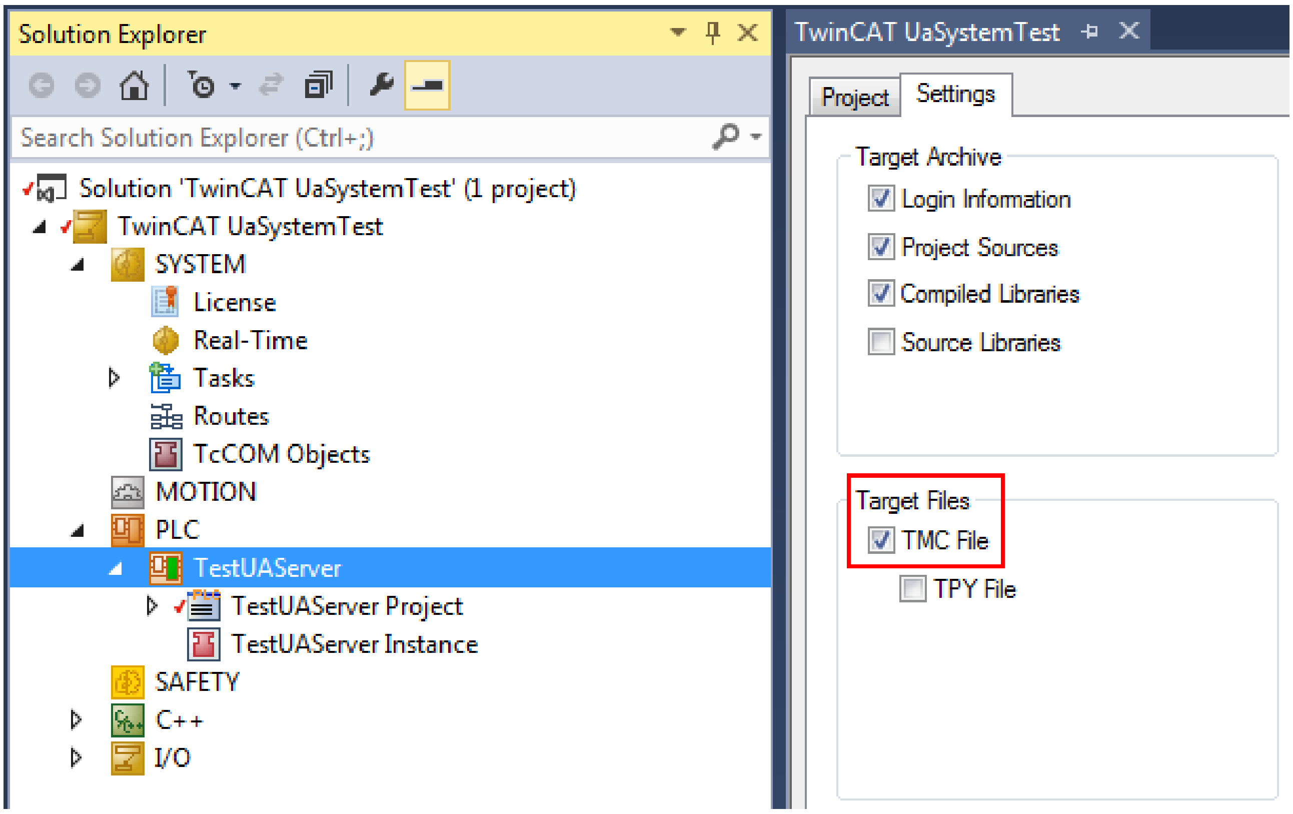
```
FUNCTION_BLOCK FB_BLOCK1
VAR_INPUT
    ni1 : INT; (*~ (OPC:1:some description) *)
    ni2 : INT;
END_VAR
VAR_OUTPUT
    no1 : INT; (*~ (OPC:1:some description) *)
    no2 : INT;
END_VAR
VAR
    nx1 : INT; (*~ (OPC:1:some description) *)
    nx2 : INT;
END_VAR
```

The instance fbTest1 is enabled for OPC UA, whereby all contained symbols are automatically enabled for OPC UA, i.e. fbTest.ni1, fbTest.ni2, etc. The instance fbTest2 is not marked for OPC UA, but the three variables contained in it - ni1, no1 and nx1 - were marked in the function block. They are therefore available in all instances via OPC UA.

Step 2: Activating the download of the symbol file

The symbol file contains information about all variables available in a PLC project. The TwinCAT OPC UA Server needs this information to configure its namespace. By default, the current symbol information is automatically generated and stored in a symbol file. This is located in the project folder of the corresponding TwinCAT project. To ensure that the symbol file is transferred to the target runtime, enable the download of the symbol file in the settings of the PLC project.

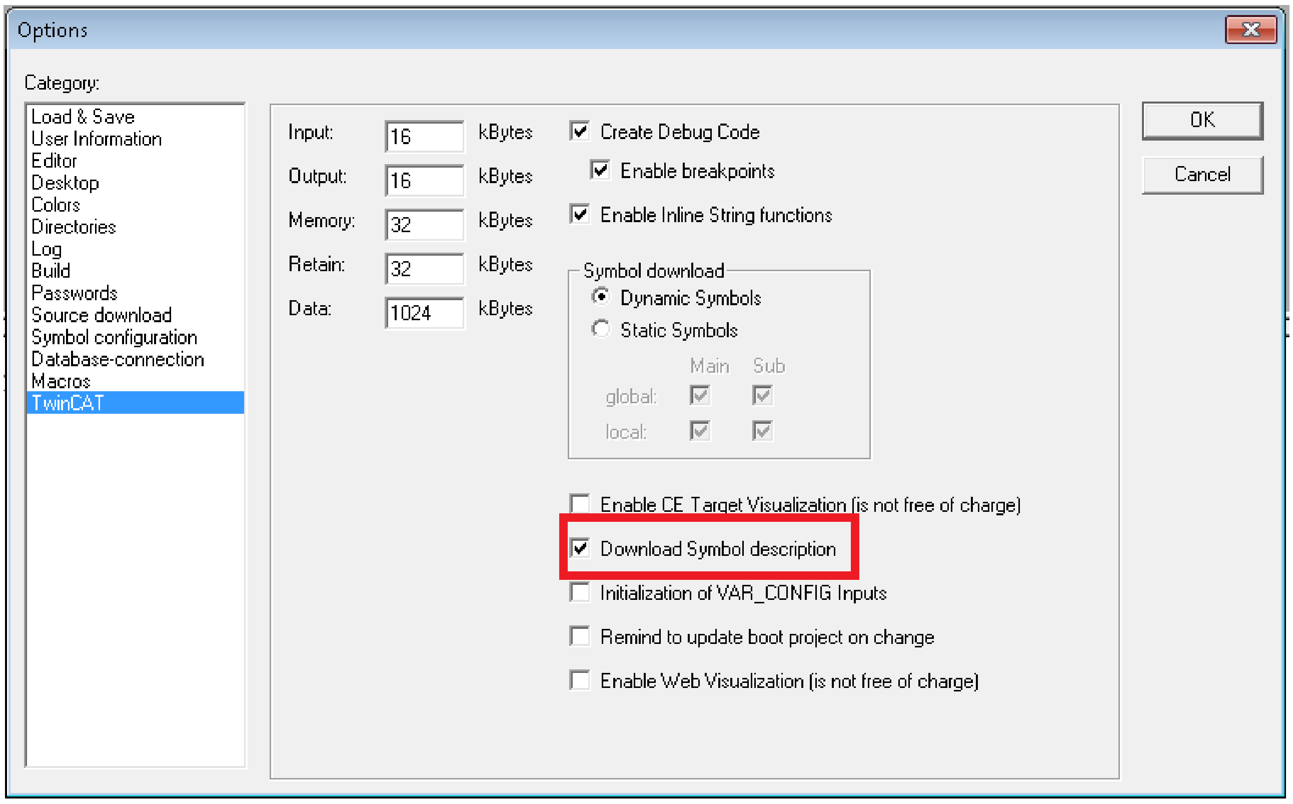
TwinCAT 3:



Target Archive
- [✓] Login Information
- [✓] Project Sources
- [✓] Compiled Libraries
- [ ] Source Libraries

Target Files
- [✓] TMC File
- [ ] TPY File

TwinCAT 2:


Options

Category:

- Load & Save
- User Information
- Editor
- Desktop
- Colors
- Directories
- Log
- Build
- Passwords
- Source download
- Symbol configuration
- Database-connection
- Macros
- TwinCAT

Input:	16	kBytes
Output:	16	kBytes
Memory:	32	kBytes
Retain:	32	kBytes
Data:	1024	kBytes

- [✓] Create Debug Code
- [✓] Enable breakpoints
- [✓] Enable Inline String functions

Symbol download
- (●) Dynamic Symbols
- ( ) Static Symbols

	Main	Sub
global:	✓	✓
local:	✓	✓

- [ ] Enable CE Target Visualization (is not free of charge)
- [✓] Download Symbol description
- [ ] Initialization of VAR_CONFIG Inputs
- [ ] Remind to update boot project on change
- [ ] Enable Web Visualization (is not free of charge)

[Optional] Step 3: Connecting further runtimes

By default, the TwinCAT OPC UA Server connects to the first PLC runtime on the local system. To make more than one PLC runtime available on the server, or if the PLC runtime is located on another system, corresponding settings are required in the TwinCAT OPC UA Configurator.

Configure all further necessary parameters as described in the section General information [▶ 49].

Connecting remote computers

To configure a runtime that is located on a remote Industrial PC or Embedded PC, enter the correct parameters for AmsNetId and AdsPort and provide the corresponding symbol file of the PLC program on this PC. Also, make sure that an ADS route to the remote system has been established.

[Optional] Step 4: Explicit hiding of symbols

In addition to the regular PLC attributes for activating a PLC symbol, you can explicitly exclude PLC symbols from publication in the OPC UA namespace. There may be several reasons for this:

- You want to publish a data structure, but not all its subelements (Sample 1).
- You have enabled a data structure in the definition and want to hide an individual instance of this data structure (Sample 2).

Sample 1:

```
{attribute 'OPC.UA.DA' := '1'}
TYPE ST_TEST :
STRUCT
    a : INT;
    {attribute 'OPC.UA.DA' := '0'}
    b : DINT;
END_STRUCT
END_TYPE

PROGRAM MAIN
VAR
    instance1 : ST_TEST;
    instance2 : ST_TEST;
END_VAR
```

In this case, the PLC attribute has been added to the structure definition, so that each instance of this structure should be available on the OPC UA Server. Subelement b (and all its subelements, if b is another data structure) will be excluded from publication on the OPC UA Server.

Sample 2:

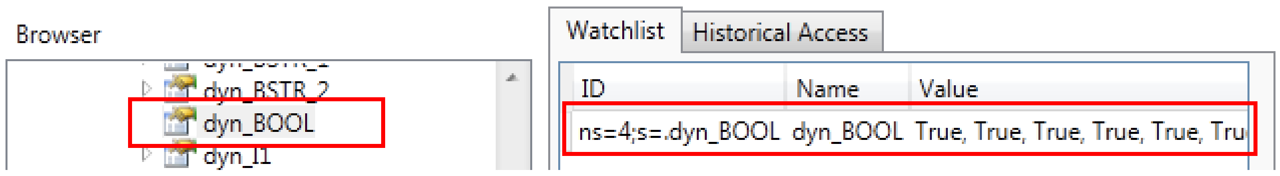
```
{attribute 'OPC.UA.DA' := '1'}
TYPE ST_TEST :
STRUCT
  a : INT;
  b : DINT;
END_STRUCT
END_TYPE

PROGRAM MAIN
VAR
  instance1 : ST_TEST;
  {attribute 'OPC.UA.DA' := '0'}
  instance2 : ST_TEST;
END_VAR
```

Although the PLC attribute is added to the structure definition, which makes each instance of this structure available on the OPC UA Server, instance2 receives the PLC attribute to disable this inheritance. This means that only instance1 is available in the UA namespace, not instance2.

4.1.6.1.1 Arrays

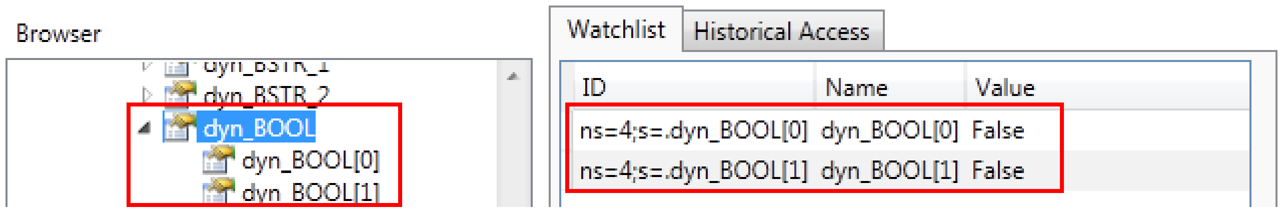
By default arrays are regarded as individual nodes in the UA namespace. This means that if you define, for example, an array dyn_BOOL[10] in the PLC (and have also enabled it for OPC UA), it will subsequently appear in the UA namespace as follows:



ID	Name	Value
ns=4;s=.dyn_BOOL	dyn_BOOL	True, True, True, True, True, Tru

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:



ID	Name	Value
ns=4;s=.dyn_BOOL[0]	dyn_BOOL[0]	False
ns=4;s=.dyn_BOOL[1]	dyn_BOOL[1]	False

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.

4.1.6.1.2 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 BadOutOfRange status code exception 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.

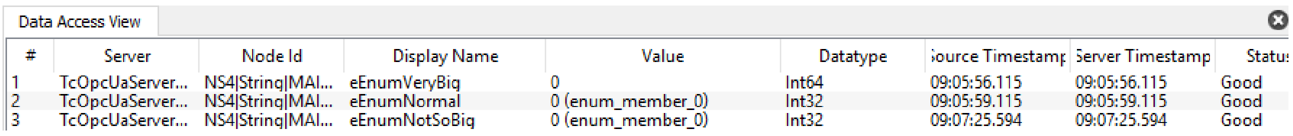
Let's assume we have the following enumeration definitions in our PLC code:

```
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):



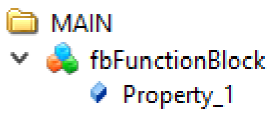
#	Server	Node Id	Display Name	Value	Datatype	ource Timestamp	Server Timestamp	Statu
1	TcOpcUaServer...	NS4|String|MAI...	eEnumVeryBig	0	Int64	09:05:56.115	09:05:56.115	Good
2	TcOpcUaServer...	NS4|String|MAI...	eEnumNormal	0 (enum_member_0)	Int32	09:05:59.115	09:05:59.115	Good
3	TcOpcUaServer...	NS4|String|MAI...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

4.1.6.1.3 Properties

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 [▶ 49].



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.

4.1.6.1.4 StructuredTypes

StructuredTypes allow you to read or write structures without interpreting each byte, because the UA Server returns the information type of each element of the structure. Based on complex functions in modern OPC UA SDKs, OPC UA Clients can search and interpret this structural information.

From version 2.2.x of the OPC UA Server, structures of the TwinCAT 3 runtime (TMC and TMI import only) are generated as a StructuredType in the UA namespace.

Sample:

STRUCT ST_Communication:

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

Program MAIN:

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

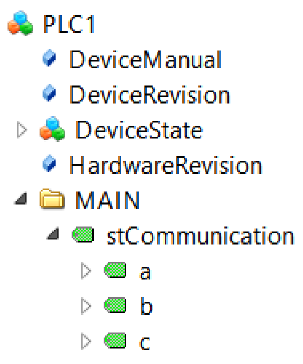
Filtered mode

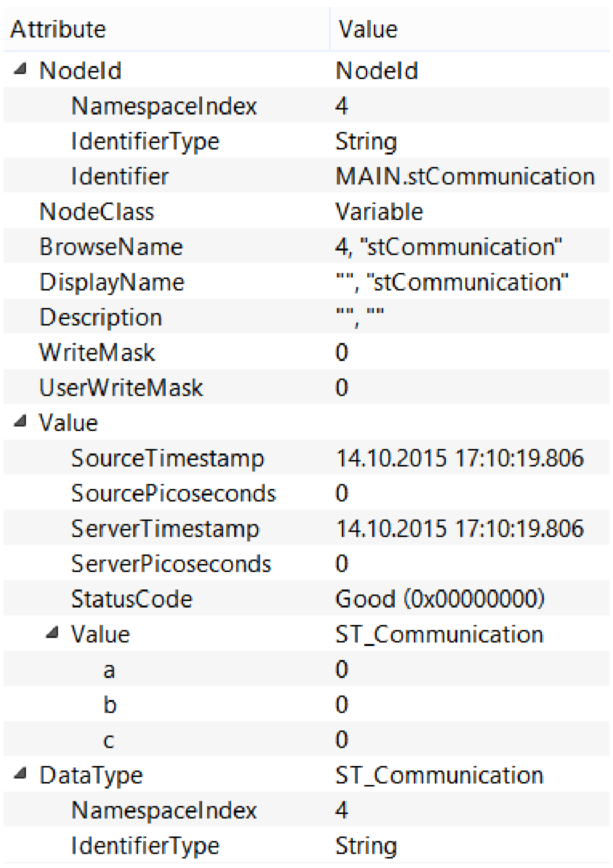
If filters are used to make symbols available via OPC UA, a STRUCT or function block must be fully available in the UA namespace in order to be displayed as a StructuredType.

Pointers and references

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

The instance stCommunication is then displayed in the UA namespace as a StructuredType, including all member variables:





Attribute	Value
◢ NodeId	NodeId
NamespaceIndex	4
IdentifierType	String
Identifier	MAIN.stCommunication
NodeClass	Variable
BrowseName	4, "stCommunication"
DisplayName	"", "stCommunication"
Description	"", ""
WriteMask	0
UserWriteMask	0
◢ Value	
SourceTimestamp	14.10.2015 17:10:19.806
SourcePicoseconds	0
ServerTimestamp	14.10.2015 17:10:19.806
ServerPicoseconds	0
StatusCode	Good (0x00000000)
◢ Value	ST_Communication
a	0
b	0
c	0
◢ DataType	ST_Communication
NamespaceIndex	4
IdentifierType	String

Alternatively, the STRUCT definition can also be assigned the PLC attribute to make all instances of STRUCT available as StructuredType. If the member variables are not to be displayed explicitly, they can be hidden with the OPC.UA.DA:=2 attribute.

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

In order to deactivate StructuredType of a certain instance, use the following attribute:

```
PROGRAM MAIN
VAR
  {attribute 'OPC.UA.DA' := '1'}
  {attribute 'OPC.UA.DA.StructuredType' := '0'}
  stCommunication : ST_Communication;
END_VAR
```

Function block StructuredType

In addition, each function block of the TwinCAT 3 PLC also contains a child node, FunctionBlock, which contains the entire function block as a StructuredType.

Sample:

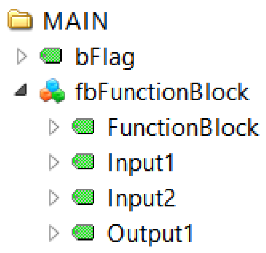
Function block:

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

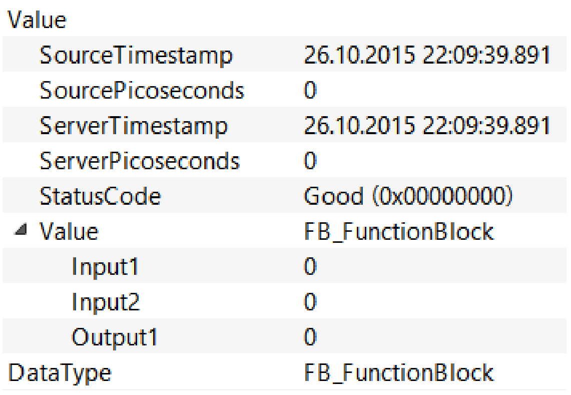
Instance of the function block:

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

Instance of the function block in the OPC UA namespace:



FunctionBlock node with StructuredType:



Value	
SourceTimestamp	26.10.2015 22:09:39.891
SourcePicoseconds	0
ServerTimestamp	26.10.2015 22:09:39.891
ServerPicoseconds	0
StatusCode	Good (0x00000000)
Value	FB_FunctionBlock
Input1	0
Input2	0
Output1	0
DataType	FB_FunctionBlock

Alternatively, the function block can also receive a PLC attribute to make all instances of the function block available as StructuredType.

```
{attribute 'OPC.UA.DA.StructuredType' := '1'}
FUNCTION_BLOCK FB_FunctionBlock
VAR_INPUT
  Input1 : INT;
  Input2 : LREAL;
END_VAR
VAR_OUTPUT
  Output1 : LREAL;
END_VAR
```

In order to deactivate StructuredType of a certain instance, use the following attribute:

```
PROGRAM MAIN
VAR
  {attribute 'OPC.UA.DA' := '1'}
  {attribute 'OPC.UA.DA.StructuredType' := '0'}
  fbFunctionBlock : FB_FunctionBlock;
END_VAR
```

Maximum size of the structure

The maximum size of a structure is 16 kB by default. 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. You can change this default in the file *TcUaDaConfig.xml*. To do this, the key <MaxStructureSize> must be added in the file, and a new value for the maximum size of a structure must be set in bytes. If a structure exceeds <MaxStructureSize>, it is imported as FolderType, where each structure element is available as a single node.

4.1.6.1.5 AnalogItemTypes

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: Normal 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 hereby 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.

4.1.6.1.6 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 [▶ 57], 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 [▶ 57] in the server, but only as FolderType. However, access to the individual reference variables within the structure works.

4.1.6.1.7 Type system

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.

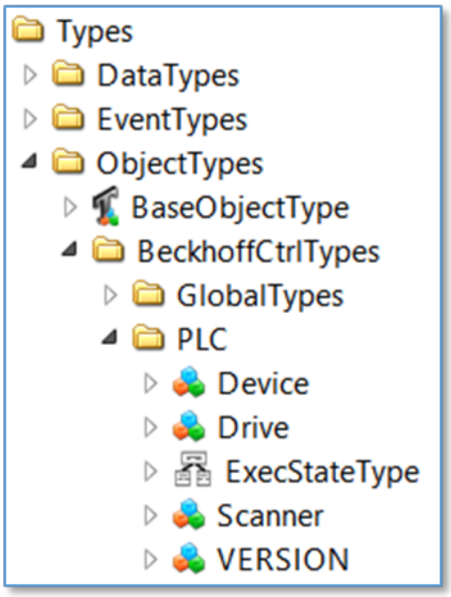
The OPC UA Server publishes type information from the IEC61131 world in its namespace. This includes not only base types such as BOOL, INT, DINT, or REAL, but also extended type information such as the current class (function block) or structure that represents an object.

Type information

Type information is part of the UA namespace. The OPC UA Server extends the basic 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:

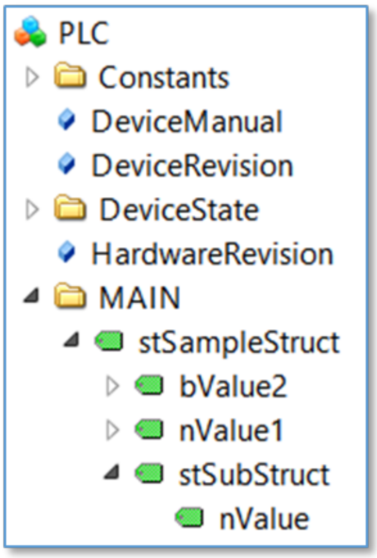


Every non-standard data type is entered in the BeckhoffCtrlTypes area.

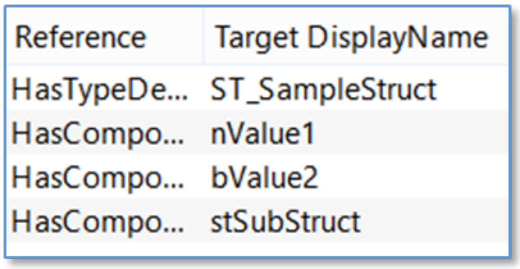
Basic principles

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).

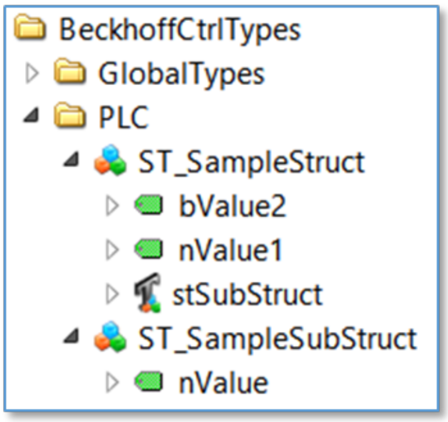

PLC
Constants
DeviceManual
DeviceRevision
DeviceState
HardwareRevision
MAIN
stSampleStruct
bValue2
nValue1
stSubStruct
nValue

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:

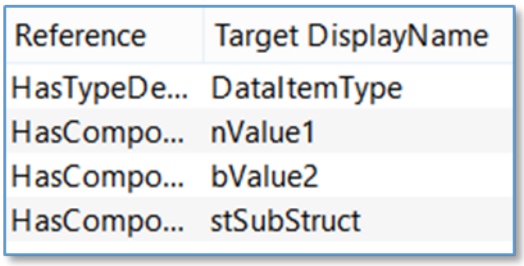


Reference	Target DisplayName
HasTypeDe...	ST_SampleStruct
HasCompo...	nValue1
HasCompo...	bValue2
HasCompo...	stSubStruct

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


BeckhoffCtrlTypes
GlobalTypes
PLC
ST_SampleStruct
bValue2
nValue1
stSubStruct
ST_SampleSubStruct
nValue

And in the references of ST_SampleStruct:

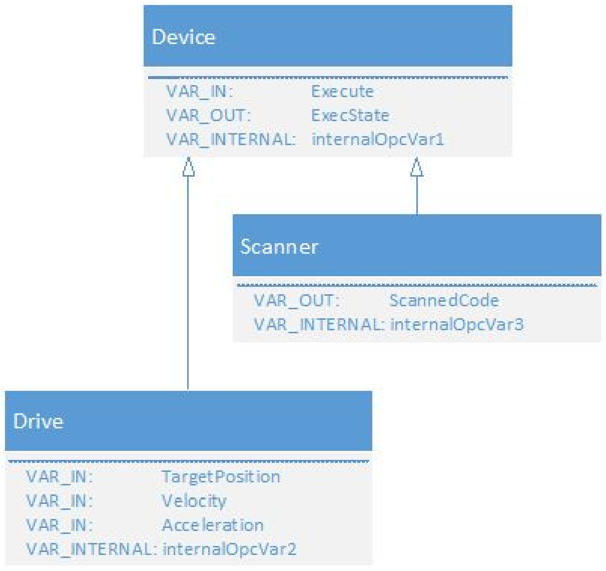


Reference	Target DisplayName
HasTypeDe...	DataItemType
HasCompo...	nValue1
HasCompo...	bValue2
HasCompo...	stSubStruct

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

Object-orientated extensions

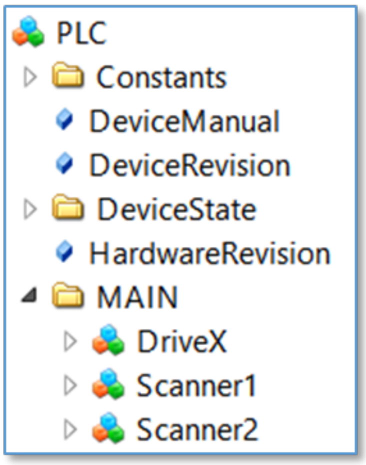
Assuming the TwinCAT 3 PLC runtime contains a PLC program whose structure can be visualized as follows:



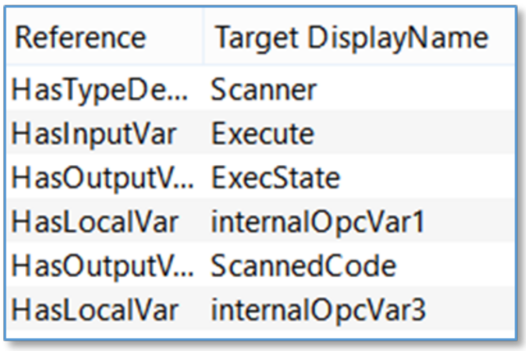
The two function blocks Scanner and Drive are derived from the base class Device by using object-oriented extensions of IEC61131-3. The MAIN program now contains the following declarations:

```
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 basic 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.

4.1.6.1.8 StatusCode

The OPC UA Server enables a PLC application to change the OPC UA StatusCode.

Perform the following steps to configure and affect the StatusCode of a variable:

- Creating a PLC structure [▸ 64]
- Overwriting StatusCode [▸ 65]

Creating a PLC structure

So that the data consistency is guaranteed, the concept of changing the OPC UA StatusCode is based on StructuredTypes (see StructuredTypes [▸ 57]). Each variable whose UA StatusCode is to be affected must be included in a STRUCT.

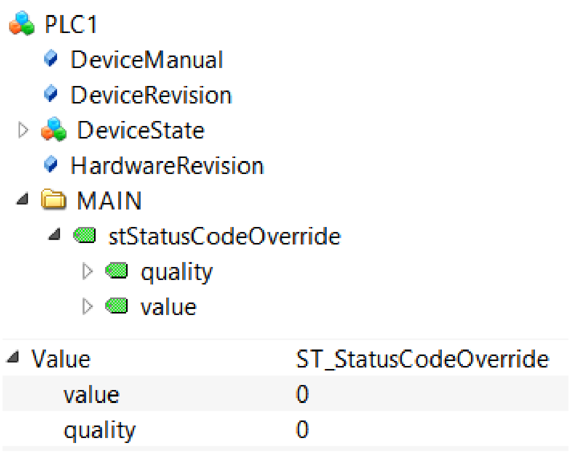
Create a new STRUCT and add a PLC attribute before the STRUCT definition. The STRUCT contains the variable itself and a variable of the data type DINT, which represents the StatusCode and to which reference is made in the attribute in front of the STRUCT definition.

```
{attribute 'OPC.UA.DA.STATUS' := 'quality'}
TYPE ST_StatusCodeOverride :
STRUCT
  value   : REAL;
  quality: DINT;
END_STRUCT
END_TYPE
```

Now create an instance of this STRUCT, for example in the MAIN program, and add the regular PLC attribute so that this instance becomes available via OPC UA.

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

This instance is now available inside the OPC UA namespace as a StructuredType.

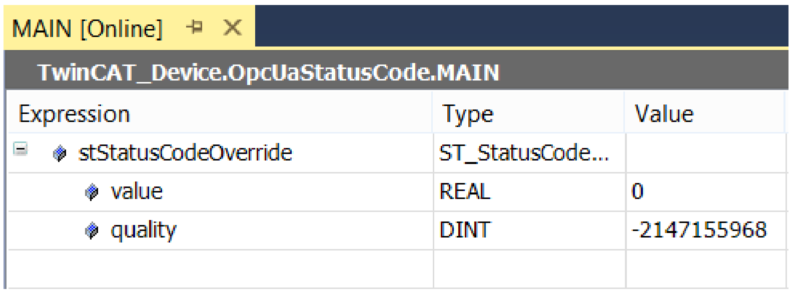


```
PLC1
    DeviceManual
    DeviceRevision
  ▷ DeviceState
    HardwareRevision
  ◢ MAIN
     ◢ stStatusCodeOverride
        ▷ quality
        ▷ value
```

◢ Value	ST_StatusCodeOverride
value	0
quality	0

Overwriting StatusCode

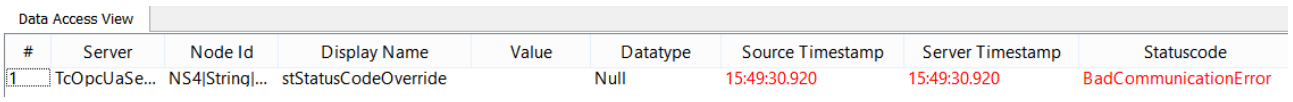
To overwrite the UA StatusCode for the STRUCT, simply edit the value of the variable "quality". If you set this, for example, to "-2147155968", the StatusCode of the STRUCT changes to "BadCommunicationError".



MAIN [Online]

TwinCAT_Device.OpcUaStatusCode.MAIN

Expression	Type	Value
stStatusCodeOverride	ST_StatusCode...	
value	REAL	0
quality	DINT	-2147155968



Data Access View

#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	TcOpcUaSe...	NS4|String|...	stStatusCodeOverride		Null	15:49:30.920	15:49:30.920	BadCommunicationError

The value must be determined according to the definition in the OPC UA specification.

The following table lists only some of the available StatusCodes and their corresponding decimal representation. Consult the official OPC UA specification for a more comprehensive list of the StatusCodes.

StatusCode	Hex	Decimal
BadUnexpectedError	0x80010000	-2147418112
BadInternalError	0x80020000	-2147352576
BadCommunicationError	0x80050000	-2147155968
BadTimeout	0x800A0000	-2146828288
BadServiceNotSupported	0x800B0000	-2146762752

UA StatusCodes

When calculating the decimal representation of other UA StatusCodes on the basis of their hexadecimal representation, make sure that your computer is set to DWORD, e.g. the Windows computer ("Programmer" view).

4.1.6.1.9 List of attributes and comments

The runtime variables for various OPC UA functions (e.g. data access or historical access) are configured directly in the PLC program or in the TMC code editor (if using TwinCAT 3 C++). The advantage is that PLC developers can decide directly in the program with which they are familiar whether and how a variable is to be enabled for OPC UA. You activate a variable by inserting a comment and the corresponding OPC UA tag in front of the variable, e.g.:

TwinCAT 3 PLC (TMC):

```
{attribute 'OPC.UA.DA' := '1'}
bVariable : BOOL;
```

TwinCAT 2 PLC (TPY):

```
bVariable : BOOL; (*~ (OPC:1:available)*)
```

You can find a detailed description of the use of attributes and comments in the sections concerning the corresponding runtime components:

- PLC
- C++
- I/O
- Matlab/Simulink

The following table shows an overview of all definable tags and their meaning. The subsections of the corresponding function contain a detailed description of the functional principle.

TwinCAT 3 (TMC):

OPC UA function	PLC tag	C++ TMC code editor (Optional features)	Meaning
Data Access (DA)	{attribute 'OPC.UA.DA' := '0'}	Name: OPC.UA.DA Value: 0	Locks a variable for OPC UA, whereupon it is no longer visible in the UA namespace.
Data Access (DA)	{attribute 'OPC.UA.DA' := '1'}	Name: OPC.UA.DA Value: 1	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.
Data Access (DA)	{attribute 'OPC.UA.DA.Access' := 'x'}	Name: OPC.UA.DA.Access Value: see right column	Sets read/write access for a variable, depending on parameter "x". 0 = none 1 = read-only 2 = write access only 3 = read and write access (default if no tag is used)
Data Access (DA)	{attribute 'OPC.UA.DA.Alias' := '1'}	Name: OPC.UA.DA.Alias Value: see right column	Specifies x as node name in the UA namespace, so-called alias mapping.
Data Access (DA)	{attribute 'OPC.UA.DA.Description' := 'x'}	Name: OPC.UA.DA.Description Value: see right column	Sets a text for the OPC UA attribute "Description".
Data Access (DA)	{attribute 'OPC.UA.DA.StructuredType' := '0'}	Name: OPC.UA.DA.StructuredType Value: 0	Disables StructuredType for a STRUCT or a function block
Data Access (DA)	{attribute 'OPC.UA.DA.StructuredType' := '1'}	Name: OPC.UA.DA.StructuredType Value: 1	Enables StructuredType for a STRUCT or a function block
Data Access	{attribute 'OPC.UA.DA.Status' := 'quality'}	Name: OPC.UA.DA.Status Value: quality	Manually define the StatusCode of a symbol in the UA namespace. Only for data structures. The value "quality" specifies which DINT subelement of the data structure determines the StatusCode . See corresponding article in the documentation for more information.
Historical Access (HA) [▸ 76]	{attribute 'OPC.UA.HA' := '1'}	Name: OPC.UA.HA Value: 1	Enables a variable for Historical Access. Must be used with {attribute 'OPC.UA.DA' := '1'}.

OPC UA function	PLC tag	C++ TMC code editor (Optional features)	Meaning
Historical Access (HA) [▶ 76]	{attribute 'OPC.UA.HA.Storage' := 'x'}	Name: OPC.UA.HA.Storage Value: see right column	Defines the storage location for Historical Access, depending on parameter "x" 1 = RAM 2 = File 3 = SQL Compact Database 4 = SQL Server Database
Historical Access (HA) [▶ 76]	{attribute 'OPC.UA.HA.Sampling' := 'x'}	Name: OPC.UA.HA.Sampling Value: see right column	Determines the sampling rate at which the variable values are to be stored, in [ms] depending on the parameter "x"
Historical Access (HA) [▶ 76]	{attribute 'OPC.UA.HA.Buffer' := 'x'}	Name: OPC.UA.HA.Buffer Value: see right column	Defines the maximum number of values that remain in the data memory, depending on the parameter "x".

TwinCAT 2 (TPY):

OPC UA function	PLC tag	Meaning
Data Access (DA)	(*~ (OPC:0:not available) *)	Locks a variable for OPC UA, whereupon it is no longer visible in the UA namespace.
Data Access (DA)	(*~ (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.
Data Access (DA)	(*~ (OPC_PROP[0005]:1:read-only) *)	Sets the write protection for a variable. Must be used together with (*~ (OPC: 1: available) *).
Data Access (DA)	(*~ (OPC_UA_PROP[5100] : x: Alias name) *)	Specifies x as node name in the UA namespace, so-called alias mapping.
Historical Access (HA) [▶ 76]	(*~ (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 2 = File 3 = SQL Compact Database 4 = SQL Server Database
Historical Access (HA) [▶ 76]	(*~ (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"
Historical Access (HA) [▶ 76]	(*~ (OPC_UA_PROP[5000][2]:x:Buffer) *)	Defines the maximum number of values that remain in the data memory, depending on the parameter "x".

4.1.6.2 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. For this you can either use the OPC UA Configurator or carry out the configuration directly in the file *ServerConfig.xml* of the OPC UA Server.

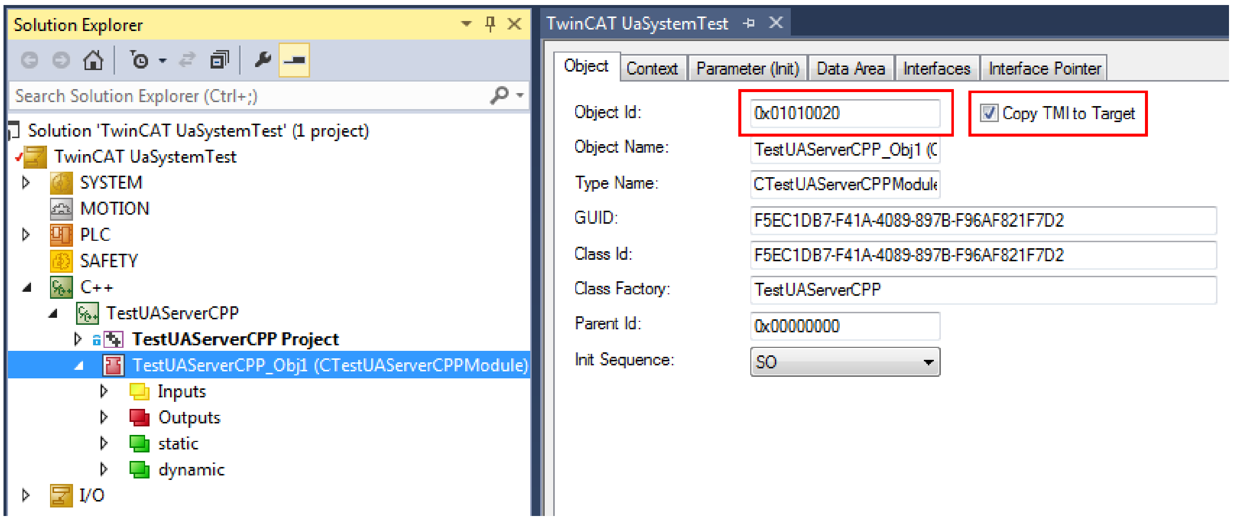
This section contains the following topics:

- Step 1: Project-related settings [▶ 69]
- Step 2: Configuring the UA Server [▶ 70]

Step 1: Project-related settings

To configure certain symbols contained in an instance of a C++ module in such a way that they are accessible via OPC UA, the following settings are necessary in the corresponding C++ module instance in TwinCAT XAE.

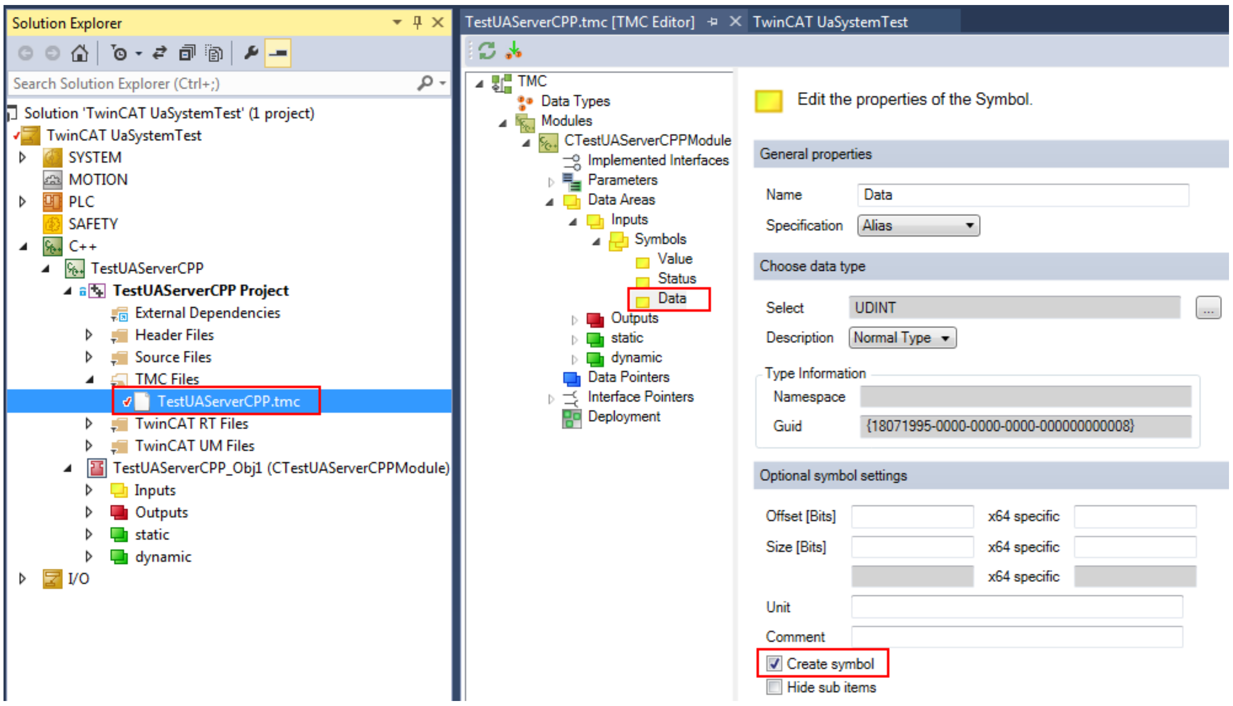
1. The OPC UA Server requires the TMI symbol file, which is not passed to the target runtime by default. Enable the transfer by setting the following options:



Object Id: 0x01010020
☑ Copy TMI to Target
Object Name: TestUAServerCPP_Obj1 (C
Type Name: CTestUAServerCPPModule
GUID: F5EC1DB7-F41A-4089-897B-F96AF821F7D2
Class Id: F5EC1DB7-F41A-4089-897B-F96AF821F7D2
Class Factory: TestUAServerCPP
Parent Id: 0x00000000
Init Sequence: SO

⇨ The generated TMI file is named after the Object ID, e.g. "Obj_01010020.tmi" and stored in the TwinCAT boot directory, e.g. *C:\TwinCAT\3.1\Boot\Tmi*.

2. Select which symbols are to be made accessible to the corresponding variables by checking the **Create symbol** checkbox in the TMC code generator. Then execute the TMC CodeGenerator.



Name: Data
Specification: Alias
Select: UDINT
Description: Normal Type
Namespace:
Guid: {18071995-0000-0000-0000-000000000008}
Offset [Bits]:
Size [Bits]:
Unit:
Comment:
☑ Create symbol
☐ Hide sub items

Step 2: Configuring the UA Server

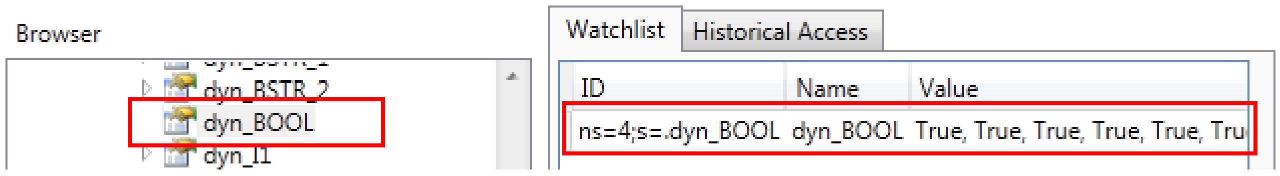
You can configure and parameterize the access to TwinCAT 3 C++ modules simply with the help of the OPC UA Configurator. To do this, add a new device of the type "CPP TwinCAT 3 (TMI) filtered" in the **Data Access** area. The **SymbolFile** field must point to the TMI file created for the C++ module instance. This TMI file is located in the TwinCAT boot directory, e.g. *C:\TwinCAT\3.1\Boot\Tmi*, and is named after the ObjectID of the TwinCAT 3 C++ module instance.

The configurable parameters describe the following functions:

Parameter	Description	Possible values
ADS Port	Defines the ADS port under which the C++ module instance is accessible. The ADS port can be read in the properties of the C++ task.	351 352 …
AutoCfg	First defines the type of the target runtime used for communication, e.g. PLC, C++, I/O. A further distinction can then be made within these categories. Each AutoCfg option is available as an unfiltered or filtered option. Filtered means the user can determine which C++ symbols are made publicly accessible via OPC UA. When using an unfiltered option, each C++ symbol is published to OPC UA.	4020 CPP TwinCAT 3 (TMI) 4021 CPP TwinCAT 3 (TMI) filtered
AutoCfgSymFile	Symbol file (TMI) of the corresponding C++ module instance.	Path to the symbol file. (TMI)
IoMode	Defines the method for accessing symbols.	1 (access via handle - default)
ArraySubItemLegacy Support	By default, subelements of an array are not mapped as separate nodes in the UA namespace. Instead, only the array is mapped as a single element. Nonetheless, UA Clients can access subelements via their "IndexRange". (Some older OPC UA Clients do not yet support this option). The flag was introduced so that access is nevertheless possible for these clients. It ensures that every array position is displayed as a separate node in the UA namespace. This leads to higher memory requirement of the OPC UA Server.	0 (disabled - default) 1 (enabled)
Disabled	Disables the C++ module instance in the UA namespace, so that the corresponding node is not displayed. It is advisable to enable this parameter if certain C++ runtimes are not yet available at the time of project planning, for example because the corresponding devices are not yet connected to the network.	0 (disabled - default) 1 (enabled)

4.1.6.2.1 Arrays

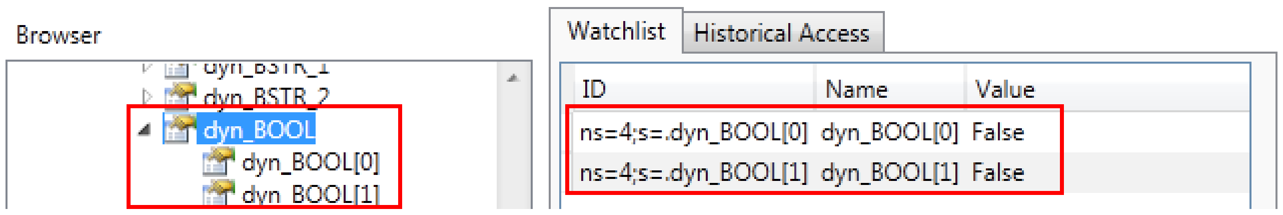
By default arrays are regarded as individual nodes in the UA namespace. This means that if you define, for example, an array dyn_BOOL[10] in the PLC (and have also enabled it for OPC UA), it will subsequently appear in the UA namespace as follows:



ID	Name	Value
ns=4;s=.dyn_BOOL	dyn_BOOL	True, True, True, True, True, Tru

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:



ID	Name	Value
ns=4;s=.dyn_BOOL[0]	dyn_BOOL[0]	False
ns=4;s=.dyn_BOOL[1]	dyn_BOOL[1]	False

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.

4.1.6.3 Matlab/Simulink

This section describes how to use TwinCAT 3 TMI files for TcCOM modules to construct the namespace of the OPC UA Server.

This section contains the following topics:

- General Information [▸ 72]
- Generating and importing a TMI file for TcCom modules [▸ 72]
- Filtered or unfiltered mode [▸ 74]

General Information

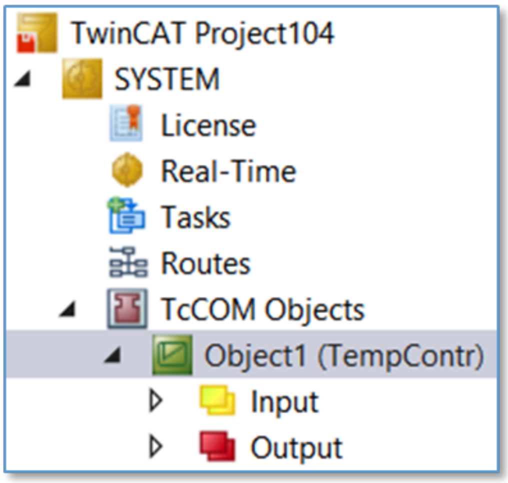
If you use TwinCAT 3 C++ or the Matlab/Simulink integration, you can generate a symbol file (TMI file) for the arising TcCom modules from the TwinCAT 3 XAE.

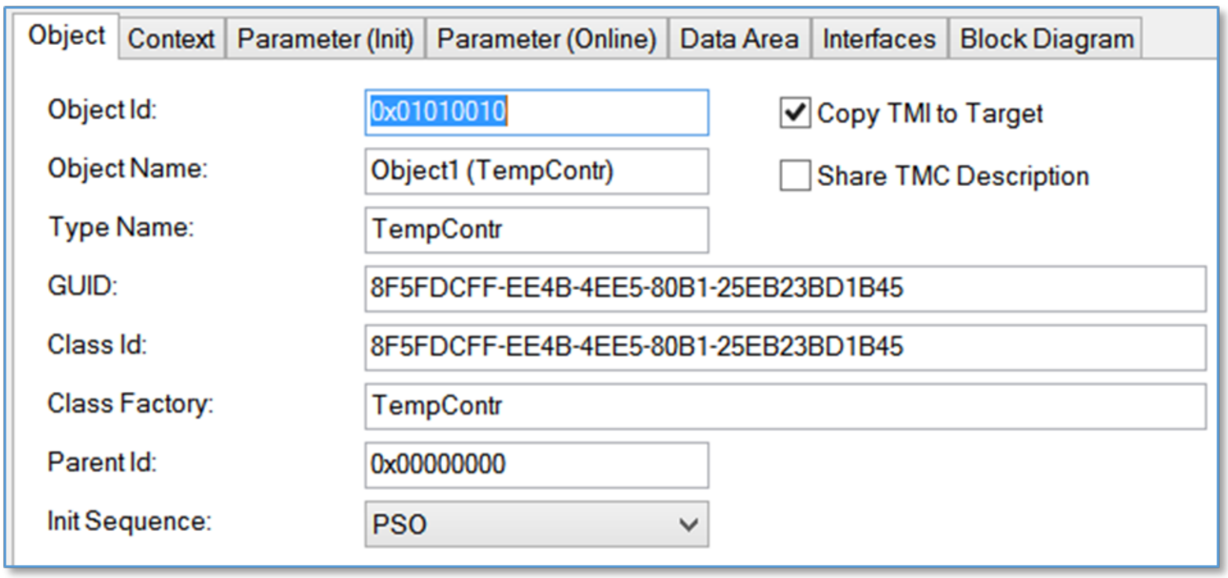
The OPC UA Server imports this TMI file and generates its namespace using the symbol information contained in the file. The namespace can then consist of variables (= symbols) that are contained in the respective TcCom module.

The import of TMI files is configured in the OPC UA Configurator.

Generating and importing a TMI file for TcCom modules

In order for a TMI file to be generated and for TwinCAT to automatically copy the TMI file to the target system, enable the option **Copy TMI to Target** in the settings of the TcCOM module.



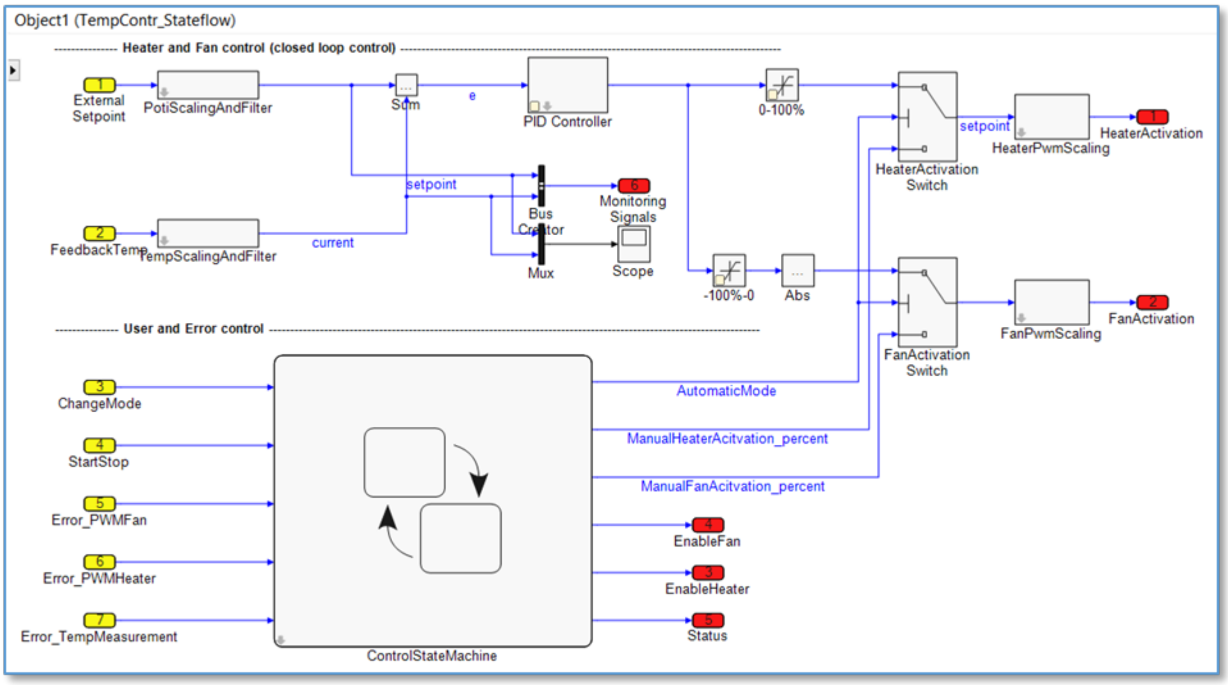


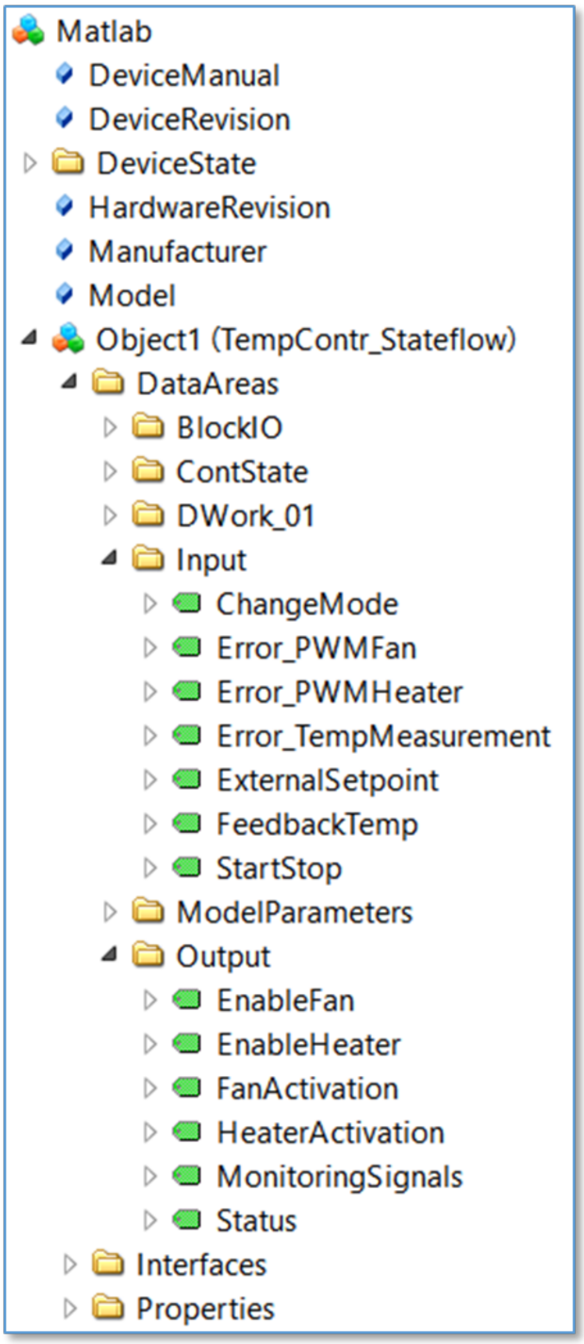
Object Id:	0x01010010
Object Name:	Object1 (TempContr)
Type Name:	TempContr
GUID:	8F5FDCFF-EE4B-4EE5-80B1-25EB23BD1B45
Class Id:	8F5FDCFF-EE4B-4EE5-80B1-25EB23BD1B45
Class Factory:	TempContr
Parent Id:	0x00000000
Init Sequence:	PSO

☑ Copy TMI to Target
☐ Share TMC Description

After activating the project, the boot directory of the target system contains the TMI file for the TcCOM module. This file can then be imported into the OPC UA namespace with the help of the OPC UA Configurator.

The configured OPC UA namespace then contains a representation of the Matlab/Simulink block diagram:





Matlab
DeviceManual
DeviceRevision
DeviceState
HardwareRevision
Manufacturer
Model
Object1 (TempContr_Stateflow)
DataAreas
BlockIO
ContState
DWork_01
Input
ChangeMode
Error_PWMFan
Error_PWMHeater
Error_TempMeasurement
ExternalSetpoint
FeedbackTemp
StartStop
ModelParameters
Output
EnableFan
EnableHeater
FanActivation
HeaterActivation
MonitoringSignals
Status
Interfaces
Properties

Filtered or unfiltered mode

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

4.1.6.4 I/O task

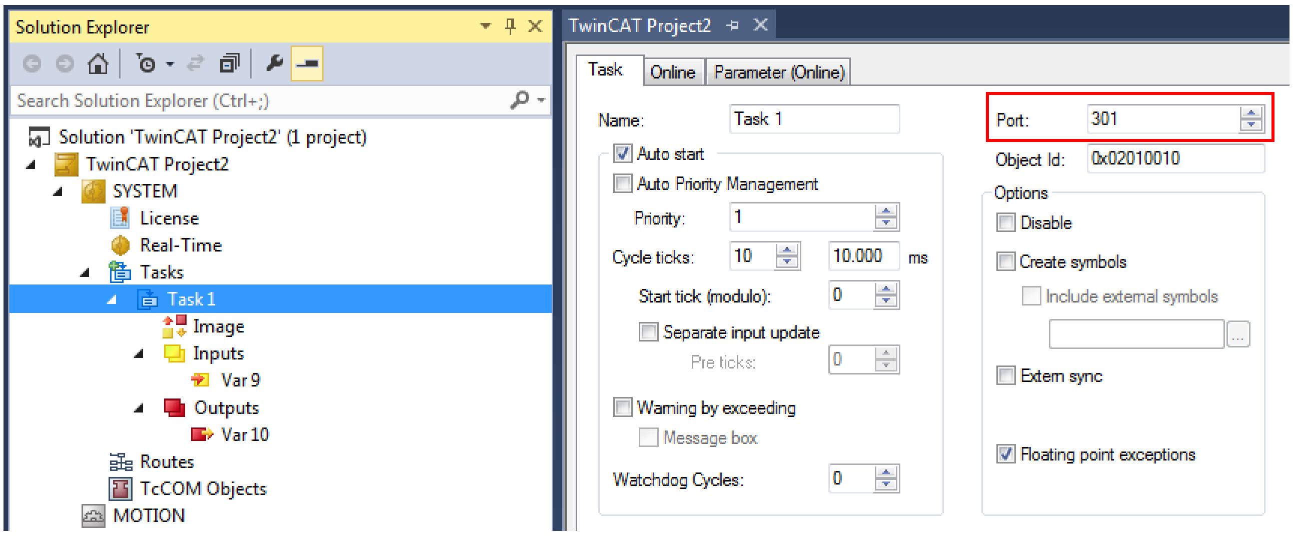
This section describes how you can make IO task variables accessible via the data access functions of the TwinCAT OPC UA Server.

This section contains the following topics:

- General Information [▶ 75]
- Step 1: Configure a TwinCAT I/O task to enable symbol handling. [▶ 75]
- Step 2: Add the I/O task to the OPC UA namespace [▶ 76]

General Information

You can publish the variables of a task with process image via OPC UA.



Name: Task 1
Auto start [✓]
Auto Priority Management [ ]
Priority: 1
Cycle ticks: 10 10.000 ms
Start tick (modulo): 0
Separate input update [ ]
Pre ticks: 0
Warning by exceeding [ ]
Message box [ ]
Watchdog Cycles: 0
Port: 301
Object Id: 0x02010010
Options
Disable [ ]
Create symbols [ ]
Include external symbols [ ]
Extern sync [ ]
Floating point exceptions [✓]

The way in which the variables are displayed and how they are communicated with is determined by various parameters. You can define these parameters with the help of the OPC UA Configurator or directly in the configuration file *ServerConfig.xml*.

The following table provides an overview of all parameters that are important when accessing the I/O task.

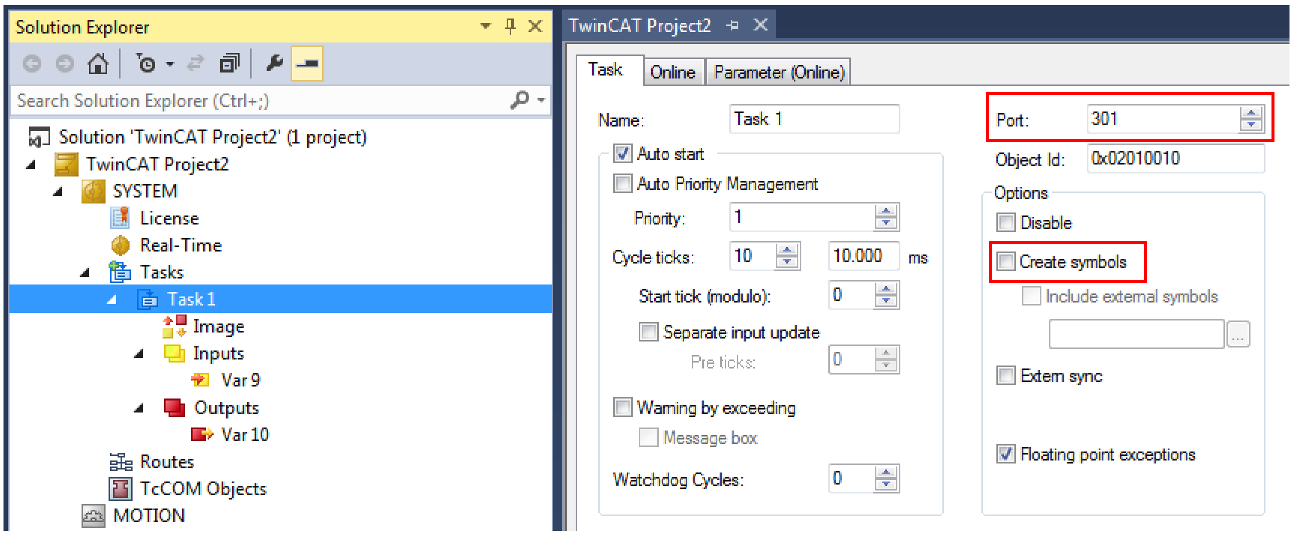
Parameter	Description	Possible values
ADS Port	Defines the ADS port under which the I/O task is accessible. The ADS port can be read out in the properties of the I/O task.	301 302 …
AutoCfg	Initially defines the type of target runtime used for the communication, e.g. PLC, C++, I/O. A finer distinction can subsequently be made within these categories. Each AutoCfg option is available as an unfiltered or filtered option. Filtered means that users can determine which I/O variables are made publicly available to the OPC UA via comments. When using an unfiltered option, each task variable is published to OPC UA.	107 I/O TwinCAT 2/3 108 I/O TwinCAT 2/3 filtered
AutoCfgSymFile	Specifies the path of the CurrentConfig.xml file, which is normally located in the folder *C:\TwinCAT\3.1\Boot\*.	Path of CurrentConfig.xml
Disabled	Disables the I/O task in the UA namespace, so that the corresponding node is not displayed. It is advisable to enable this parameter if certain runtimes are not yet available at the time of project planning, for example, because the corresponding devices are not yet connected to the network.	0 (disabled - default) 1 (enabled)

The following steps describe how to import variables from an I/O task into the UA namespace. This requires the OPC UA Server and the runtime to be on the same computer.

Step 1: Configure a TwinCAT I/O task to enable symbol handling.

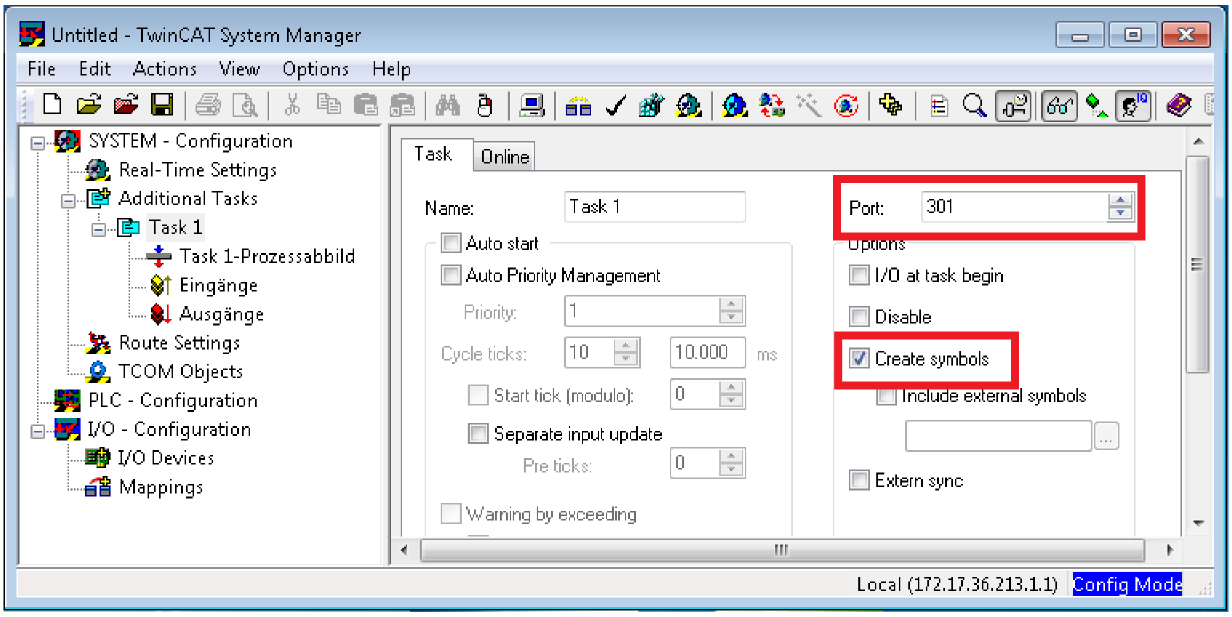
Open the I/O task settings and enable the **Create symbols** option. At this point, note the ADS port of the I/O task (in the example, this is 301).

TwinCAT 3:



Name: Task 1
Port: 301
Object Id: 0x02010010
[✓] Auto start
[ ] Auto Priority Management
Priority: 1
Cycle ticks: 10 10.000 ms
Start tick (modulo): 0
[ ] Separate input update
Pre ticks: 0
[ ] Warning by exceeding
[ ] Message box
Watchdog Cycles: 0
Options
[ ] Disable
[ ] Create symbols
[ ] Include external symbols
[ ] Extern sync
[✓] Floating point exceptions

TwinCAT 2:



Name: Task 1
Port: 301
[ ] Auto start
[ ] Auto Priority Management
Priority: 1
Cycle ticks: 10 10.000 ms
[ ] Start tick (modulo): 0
[ ] Separate input update
Pre ticks: 0
[ ] Warning by exceeding
Options
[ ] I/O at task begin
[ ] Disable
[✓] Create symbols
[ ] Include external symbols
[ ] Extern sync

Step 2: Add the I/O task to the OPC UA namespace

Configure the OPC UA Server so that it offers access to the I/O task. Using the OPC UA Configurator, add a new device of type "IO TwinCAT 2/3" and configure its settings for AdsNetId, AdsPort and SymbolFile (path to the *CurrentConfig.xml* file).

After setting the corresponding properties, restart the OPC UA Server to enable these settings.

4.1.7 Historical Access

This section describes the steps necessary to configure the variables in the namespace of the OPC UA Server for Historical Access (HA).

Historical Access is an OPC UA function, in which the variable values are either stored permanently on a data storage device (file or database) or in the device RAM, so that they can be retrieved later by the client. The way in which the OPC UA Server reads and stores the variable values can be configured.

The Historical Access configuration is available for variables from any runtime system (PLC, C++, ...). As a prerequisite, the respective variable must first be enabled for OPC UA. For details on how to do this please refer to the respective document under "Data Access".

Requirements and recommendations

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

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 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	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).

The individual memory types are configured as HistoryAdapters in the server. This is done via the TwinCAT OPC UA Server configurator. Each HistoryAdapter (except for "Volatile" -> RAM) can be used repeatedly, e.g., if the historical values for individual variables are to be stored in different data memories. In the HistoryNodes area, the individual nodes are configured, assigned to a data memory and assigned a SamplingRate and a maximum size for the ring buffer to be used in the data memory.

The attribute `HistoryWriteable="true"` ensures that the data memory for this node can be filled with values via a HistoryUpdate() call. Such a call is provided, for example, by the TwinCAT OPC UA Client via the function block UA_HistoryUpdate. If you configure a node with this attribute, then the server doesn't normally "sample" the values itself, but receives them from other clients. In such a configuration the SamplingRate is therefore usually 0.

Requirements

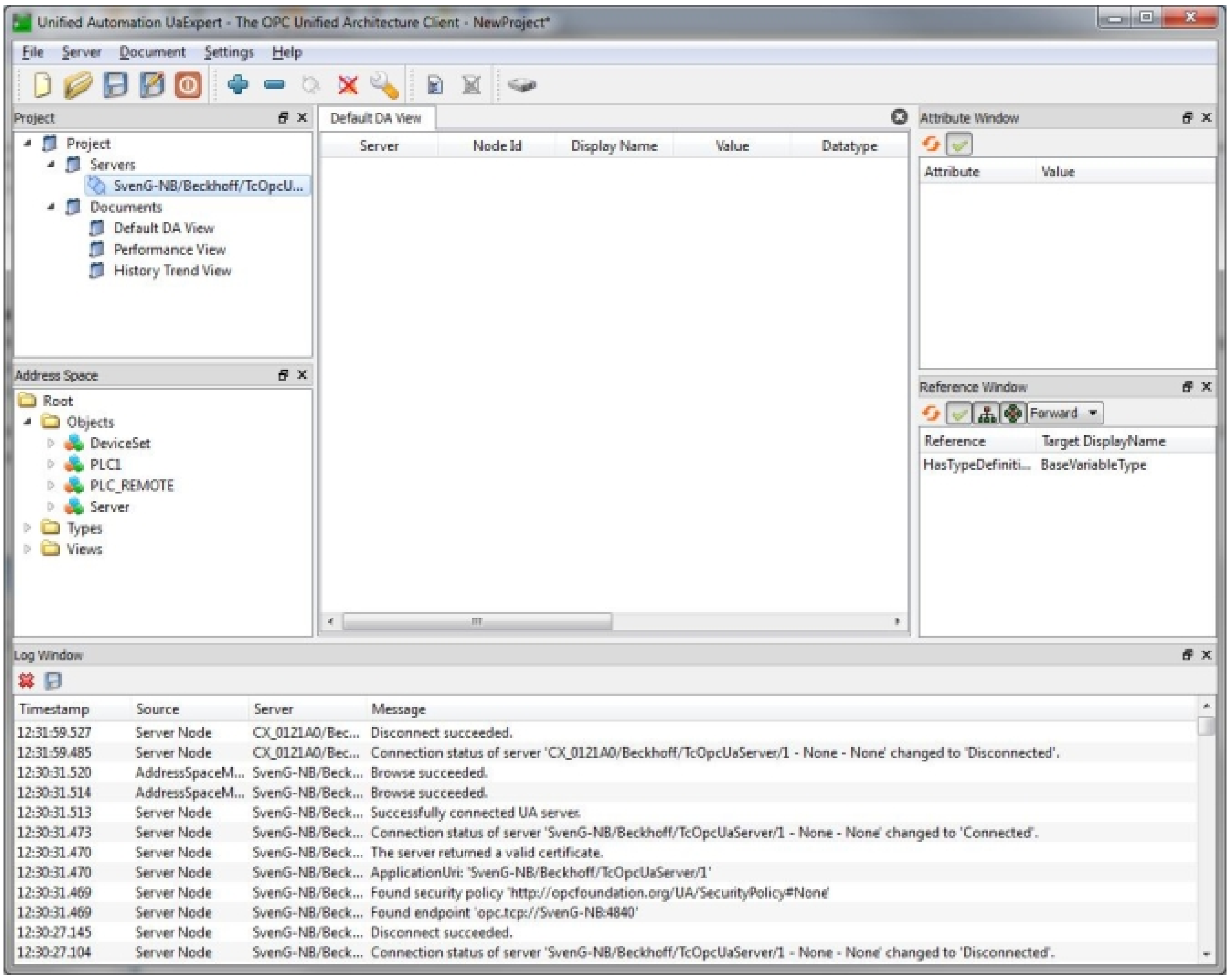
Products	Setup versions	Target platform
TF6100	4.x.x	IPC or CX (x86, x64, ARM)

4.1.7.1 Display historical data

Displaying Historical Access values in an OPC UA Client

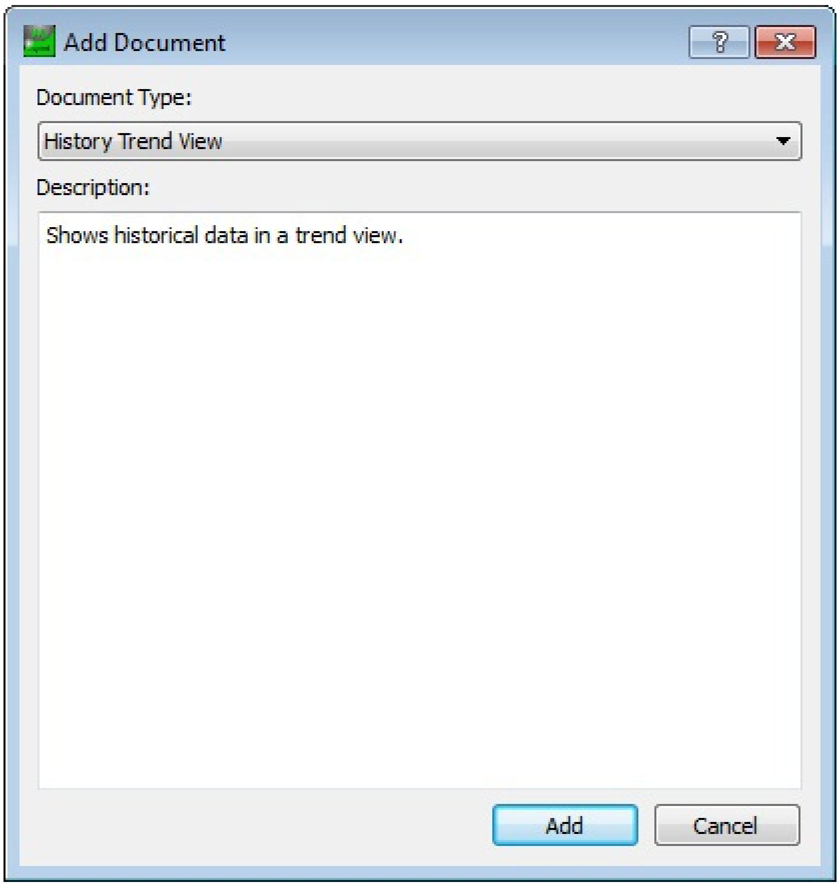
The following step-by-step instructions describe how to configure the UA Expert software in order to access historical data.

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


Unified Automation UaExpert - The OPC Unified Architecture Client - NewProject*
File Server Document Settings Help
Project
Project
Servers
SvenG-NB/Beckhoff/TcOpcU...
Documents
Default DA View
Performance View
History Trend View
Address Space
Root
Objects
DeviceSet
PLC1
PLC_REMOTE
Server
Types
Views
Default DA View
Server Node Id Display Name Value Datatype
Attribute Window
Attribute Value
Reference Window
Forward
Reference Target DisplayName
HasTypeDefiniti... BaseVariableType
Log Window

```
Timestamp     Source           Server              Message
12:31:59.527  Server Node      CX_0121A0/Bec...    Disconnect succeeded.
12:31:59.485  Server Node      CX_0121A0/Bec...    Connection status of server 'CX_0121A0/Beckhoff/TcOpcUaServer/1 - None - None' changed to 'Disconnected'.
12:30:31.520  AddressSpaceM... SvenG-NB/Beck...    Browse succeeded.
12:30:31.514  AddressSpaceM... SvenG-NB/Beck...    Browse succeeded.
12:30:31.513  Server Node      SvenG-NB/Beck...    Successfully connected UA server.
12:30:31.473  Server Node      SvenG-NB/Beck...    Connection status of server 'SvenG-NB/Beckhoff/TcOpcUaServer/1 - None - None' changed to 'Connected'.
12:30:31.470  Server Node      SvenG-NB/Beck...    The server returned a valid certificate.
12:30:31.470  Server Node      SvenG-NB/Beck...    ApplicationUri: 'SvenG-NB/Beckhoff/TcOpcUaServer/1'
12:30:31.469  Server Node      SvenG-NB/Beck...    Found security policy 'http://opcfoundation.org/UA/SecurityPolicy#None'
12:30:31.469  Server Node      SvenG-NB/Beck...    Found endpoint 'opc.tcp://SvenG-NB:4840'
12:30:27.145  Server Node      SvenG-NB/Beck...    Disconnect succeeded.
12:30:27.104  Server Node      SvenG-NB/Beck...    Connection status of server 'SvenG-NB/Beckhoff/TcOpcUaServer/1 - None - None' changed to 'Disconnected'.
```

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

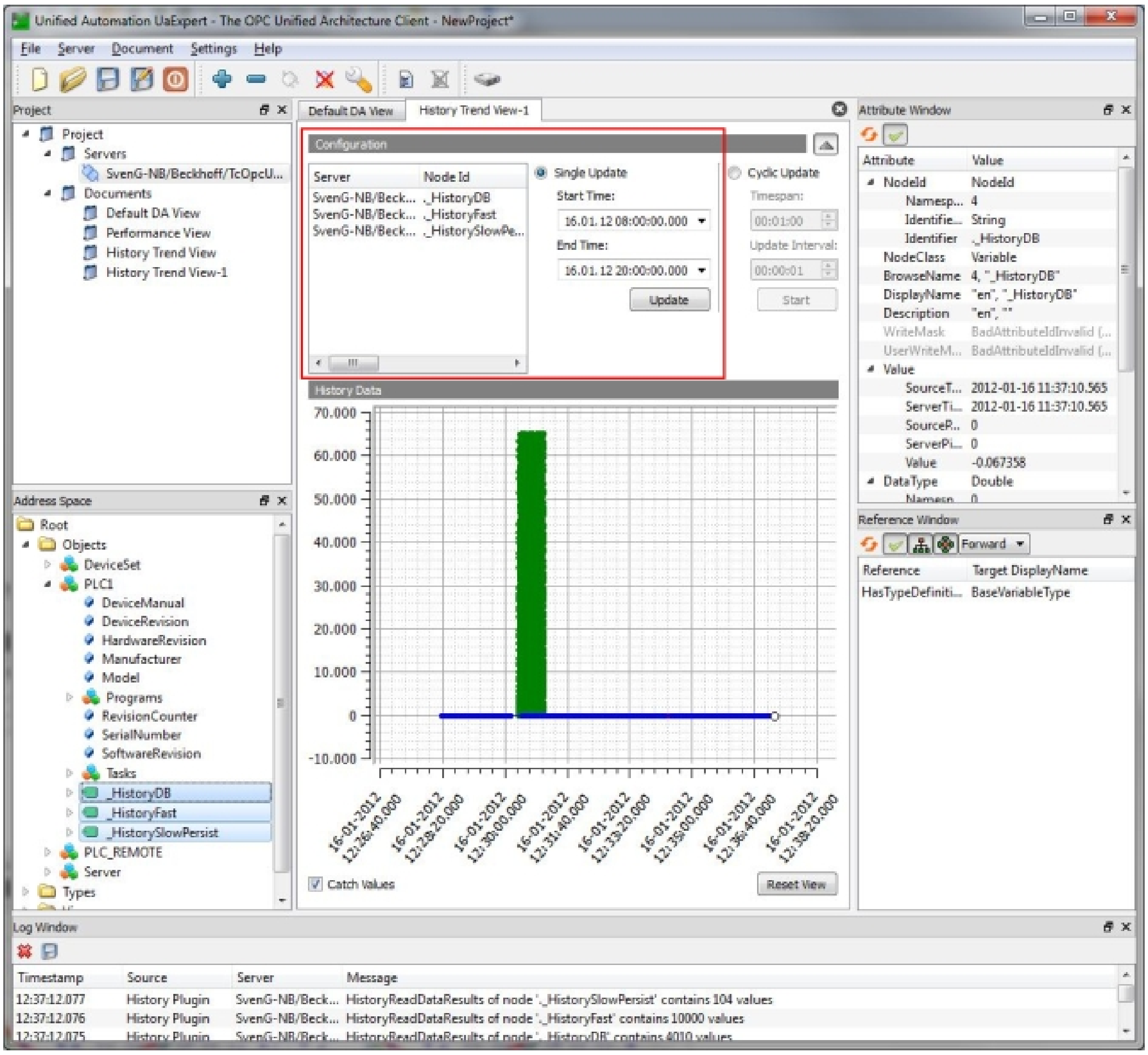


Document Type: History Trend View

Description: Shows historical data in a 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.



Server	Node Id
SvenG-NB/Beck...	_HistoryDB
SvenG-NB/Beck...	_HistoryFast
SvenG-NB/Beck...	_HistorySlowPe...

Start Time: 16.01.12 08:00:00.000
End Time: 16.01.12 20:00:00.000
Timespan: 00:01:00
Update Interval: 00:00:01

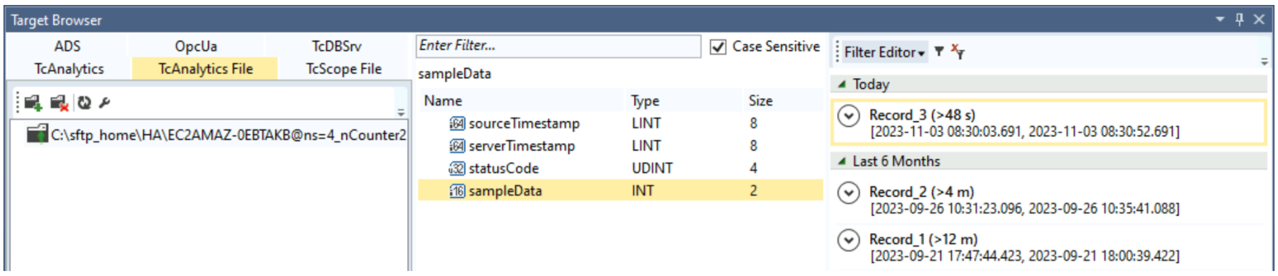
Attribute	Value
NodeId	NodeId
Namesp...	4
Identifie...	String
Identifier	_HistoryDB
NodeClass	Variable
BrowseName	4, "_HistoryDB"
DisplayName	"en", "_HistoryDB"
Description	"en", ""
WriteMask	BadAttributeIdInvalid (...
UserWriteM...	BadAttributeIdInvalid (...
Value	
SourceT...	2012-01-16 11:37:10.565
ServerTi...	2012-01-16 11:37:10.565
SourceP...	0
ServerPi...	0
Value	-0.067358
DataType	Double

Reference	Target DisplayName
HasTypeDefiniti...	BaseVariableType

```
Timestamp     Source          Server              Message
12:37:12.077  History Plugin  SvenG-NB/Beck...    HistoryReadDataResults of node '_HistorySlowPersist' contains 104 values
12:37:12.076  History Plugin  SvenG-NB/Beck...    HistoryReadDataResults of node '_HistoryFast' contains 10000 values
12:37:12.075  History Plugin  SvenG-NB/Beck...    HistoryReadDataResults of node '_HistoryDB' contains 4010 values
```

Display Historical Access values in the 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.



C:\sftp_home\HA\EC2AMAZ-0EBTAKB@ns=4_nCounter2

sampleData

Name	Type	Size
sourceTimestamp	LINT	8
serverTimestamp	LINT	8
statusCode	UDINT	4
sampleData	INT	2

Record_3 (>48 s)
[2023-11-03 08:30:03.691, 2023-11-03 08:30:52.691]

Record_2 (>4 m)
[2023-09-26 10:31:23.096, 2023-09-26 10:35:41.088]

Record_1 (>12 m)
[2023-09-21 17:47:44.423, 2023-09-21 18:00:39.422]

4.1.8 Alarms and Conditions

The steps required to configure OPC UA Alarms and Conditions (A&C) on the OPC UA Server are described in this section. The basic concept is independent of TwinCAT runtime, which means the configuration steps for PLC, C++, TcCOM runtime or only I/O task are the same.

OPC UA Alarms and Conditions (Part 9 of the OPC UA specification) describes a model for monitoring process values and outputting alarms and events when a runtime symbol changes its state.

Requirements

The following requirements apply to the use of OPC UA Alarms and Conditions:

- The runtime symbol to be monitored must be available in the namespace.
- The OPC UA Client must support Alarms and Conditions. In this section UA Expert (from Unified Automation) is used as the reference UA Client.

General Information

Execute the following steps once to release a symbol for Alarms and Conditions:

- Step 1: Activating runtime symbol for data access [▶ 82] (so that the symbol is generally accessible via OPC UA.)
- Step 2: Activating A&C for a symbol [▶ 82]
- Step 3: Transferring your own user data with an event [▶ 83]
- Step 4: Triggering an event using the FireEvent method [▶ 85]
- Step 5: Configuring multilingual alarm texts [▶ 86]
- Step 6: Registering A&C with a reference OPC UA Client [▶ 86]

These steps are explained in more detail below. At the end of this section you will find information about receiving configured alarms via A&C with the UA Expert Reference Client.

Supported alarm types

The implementation of OPC UA Alarms and Conditions currently supports the following alarm types:

- LimitAlarmType: Define different limits for a symbol. If a limit is reached, the UA Server issues an alarm.
- OffNormalAlarmType: Define a value that is "normal". If the current value deviates from the "normal" value, the UA Server issues an alarm.

Step 1: Activating runtime symbol for data access

A variable must be available in the OPC UA Server in order for it to be configured for A&C. To do this in the case of a PLC variable, provide it with an attribute (see PLC [▶ 49]).

Step 2: Activating A&C for a symbol

You can configure a runtime symbol for A&C with the OPC UA Server Configurator. The Configurator has a simple graphical user interface for editing the XML file on which it is based. The configurator is available in two versions, depending on the setup version: Standalone or integrated in Visual Studio [▶ 127].

The following program extract shows a sample of this XML file to better understand the general behavior and structure of the A&C implementation.

```
<TcUaAcConfig>
  <ConditionController Name="ConditionController1" >
    <Condition Name="Counter" Severity="200">
    <LimitAlarmType LowLowLimit="-10" LowLimit="0" HighLimit="10" HighHighLimit="20"
MessageNormal="100" MessageLowLow="10" MessageLow="11" MessageHigh="12" MessageHighHigh="13"/>
      <ItemToMonitor SamplingRate="100" NS="urn:[NodeName]:BeckhoffAutomation:Ua:PLC1"
NodeId="s=MAIN.nCounter1" />
    </Condition>
    <Condition Name="Switch" Severity="500">
```

```
      <OffNormalAlarmType Normal="0" MessageNormal="100" MessageOffNormal="20" />
      <ItemToMonitor SamplingRate="100" NS="urn:[NodeName]:BeckhoffAutomation:Ua:PLC1"
NodeId="s=MAIN.bSwitch" />
    </Condition>
    <Condition Name="Struct" Severity="300">
      <LimitAlarmType LowLowLimit="-10" LowLimit="0" HighLimit="10" HighHighLimit="20"
MessageNormal="100" MessageLowLow="10" MessageLow="11" MessageHigh="12" MessageHighHigh="13"/>
      <ItemToMonitor SamplingRate="100" NS="urn:[NodeName]:BeckhoffAutomation:Ua:PLC1"
NodeId="s=MAIN.stStruct" />
    </Condition>
  </ConditionController>
  <ConditionController Name="ConditionController2" >
    <Condition Name="Counter2" Severity="200">
      <LimitAlarmType LowLowLimit="-10" LowLimit="0" HighLimit="10" HighHighLimit="20"
MessageNormal="100" MessageLowLow="10" MessageLow="11" MessageHigh="12" MessageHighHigh="13"/>
      <ItemToMonitor SamplingRate="100" NS="urn:[NodeName]:BeckhoffAutomation:Ua:PLC1"
NodeId="s=MAIN.nCounter2" />
    </Condition>
  </ConditionController>
</TcUaAcConfig>
```

A "condition" defines the runtime symbol to be monitored and the alarm limits and texts. Each condition is organized in a so-called "ConditionController", the object that the OPC UA Clients subsequently subscribe to.

When creating a condition, you must specify the NamespaceName (NS) and NodeID for referring to the UA node to be monitored. The standalone configurator provides a simple browsing mechanism for selecting a node. The configurator integrated in Visual Studio uses the target browser (Extension - OPC UA). In XML, the placeholder [NodeName] in NamespaceName can be used to switch the XML file between different hardware systems. NamespaceName always contains the host name of the IPC or Embedded PC on which the OPC UA Server is running. If [NodeName] is selected, this tag will be replaced by the host name of the current IPC or Embedded PC on which the UA Server is running.

SamplingRate determines how often the UA Server should request a value from the node to determine whether one of the alarm limits has been reached.

The alarm texts are identified by an ID. The ID uniquely identifies an alarm text from the resource file (see Step 5: Configuring multilingual alarm texts [▶ 86]).

Step 3: Transferring own user data with an alarm

Alarms can contain fields with their own user data, which complement the data output with the alarm. These user data fields can be created and filled out in the runtime application. To do this, create a STRUCT and name its first element "value". In case of an alarm, all the following elements are then sent in an additional user data field.

Sample PLC application:

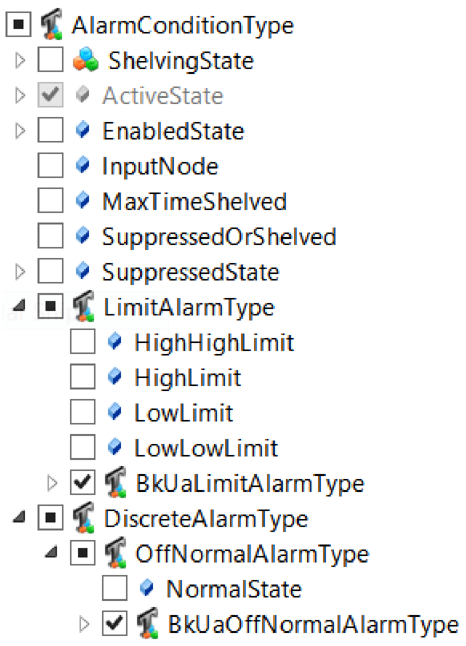
```
TYPE ST_CustomStruct :
STRUCT
  value : INT;
  data  : ST_SomeStruct;
END_STRUCT
END_TYPE

TYPE ST_SomeStruct :
STRUCT
  Data1 : INT;
  Data2 : REAL;
  Data3 : LREAL;
END_STRUCT
END_TYPE
```

The instance of ST_CustomStruct is then enabled for data access via the regular mechanism, e.g. in TwinCAT 3: To do this the STRUCT must be activated as a StructuredType [▶ 57].

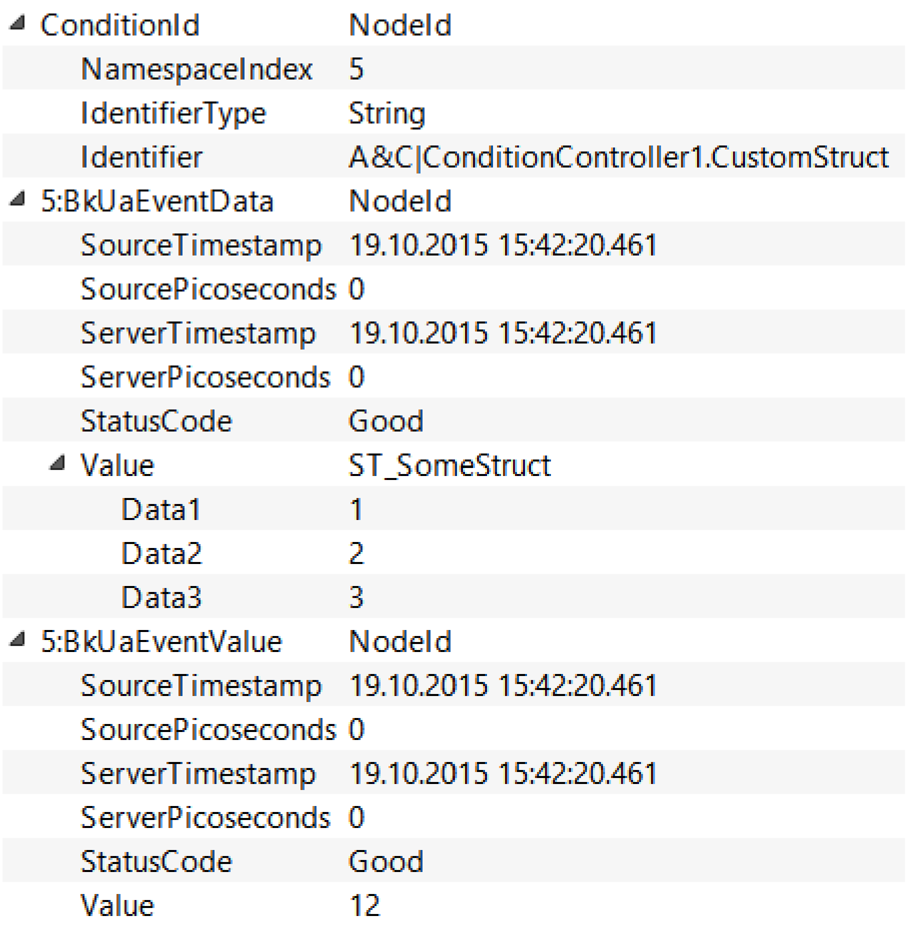
```
PROGRAM MAIN
VAR
  {attribute 'OPC.UA.DA' := '1'}
  {attribute 'OPC.UA.DA.StructuredType' := '1'}
  stCustomStruct : ST_CustomStruct;
END_VAR
```

When logging on to a ConditionController, the OPC UA Clients must subscribe to special AlarmConditionTypes, i.e. "BkUaLimitAlarmType" and "BkUaOffNormalAlarmType", so that they can receive the special user data fields when an alarm is received.



```
■ AlarmConditionType
  ▷ ☐ ShelvingState
  ▷ ☑ ActiveState
  ▷ ☐ EnabledState
    ☐ InputNode
    ☐ MaxTimeShelved
    ☐ SuppressedOrShelved
  ▷ ☐ SuppressedState
  ◢ ■ LimitAlarmType
      ☐ HighHighLimit
      ☐ HighLimit
      ☐ LowLimit
      ☐ LowLowLimit
    ▷ ☑ BkUaLimitAlarmType
  ◢ ■ DiscreteAlarmType
    ◢ ■ OffNormalAlarmType
        ☐ NormalState
      ▷ ☑ BkUaOffNormalAlarmType
```

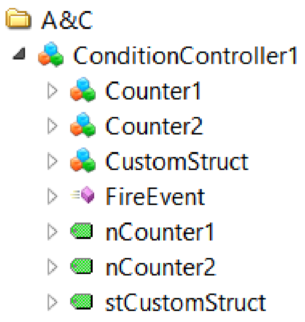
The OPC UA Client then receives the user data in the fields BkUaEventData and BkUaEventValue of the incoming alarm. In the above sample, these are the value of the PLC variables and the user data represented by the PLC structure ST_SomeStruct.

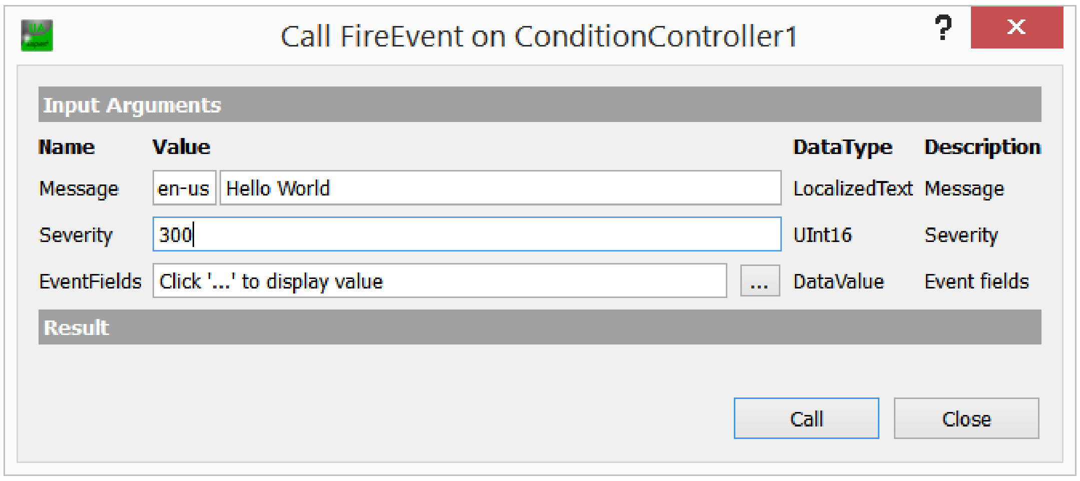


◢ ConditionId	NodeId
NamespaceIndex	5
IdentifierType	String
Identifier	A&C|ConditionController1.CustomStruct
◢ 5:BkUaEventData	NodeId
SourceTimestamp	19.10.2015 15:42:20.461
SourcePicoseconds	0
ServerTimestamp	19.10.2015 15:42:20.461
ServerPicoseconds	0
StatusCode	Good
◢ Value	ST_SomeStruct
Data1	1
Data2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

Step 4: Triggering an event using the FireEvent method

Each ConditionController includes a FireEvent method with which the OPC UA Clients can trigger a general event with user-defined EventFields.



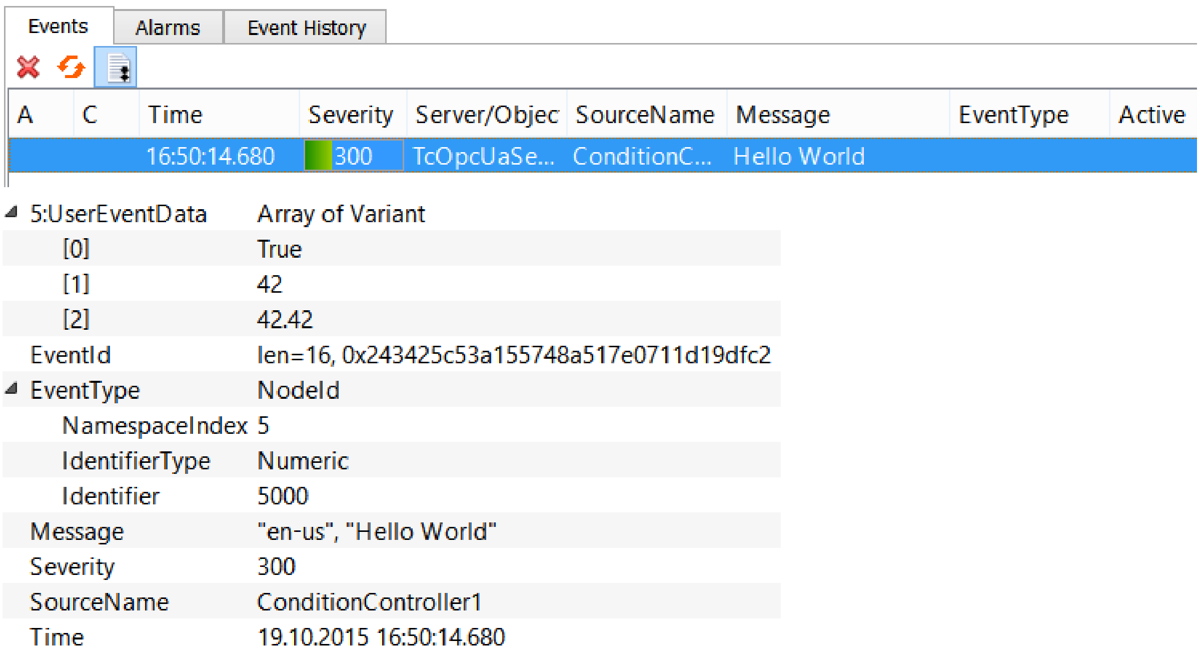

Call FireEvent on ConditionController1

Input Arguments

Name	Value	DataType	Description
Message	en-us Hello World	LocalizedText	Message
Severity	300	UInt16	Severity
EventFields	Click '...' to display value	DataValue	Event fields

Result

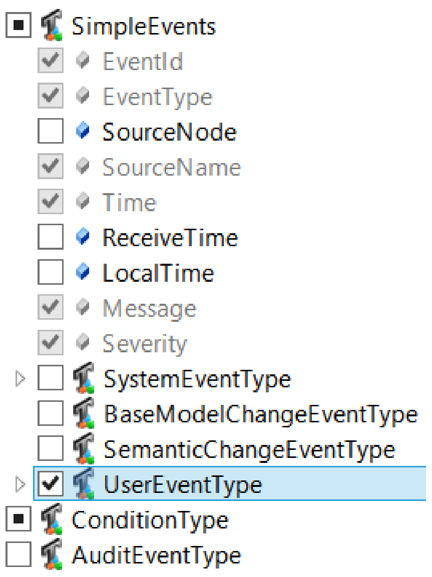
If the method is executed, an event is output on the OPC UA Server. Other OPC UA Clients can receive these events when they subscribe to the corresponding ConditionController.



A	C	Time	Severity	Server/Objec	SourceName	Message	EventType	Active
		16:50:14.680	300	TcOpcUaSe...	ConditionC...	Hello World		

5:UserEventData	Array of Variant
[0]	True
[1]	42
[2]	42.42
EventId	len=16, 0x243425c53a155748a517e0711d19dfc2
EventType	NodeId
NamespaceIndex	5
IdentifierType	Numeric
Identifier	5000
Message	"en-us", "Hello World"
Severity	300
SourceName	ConditionController1
Time	19.10.2015 16:50:14.680

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



Step 5: Configuring multilingual alarm texts

The A&C implementation in the OPC UA Server supports the use of multilingual alarm texts. The alarm text that is used depends on the language with which the UA Client connects to the Server.

Alarm texts are configured in XML files. There is a separate file for each language. These files are located in the res (Resource) folder on the OPC UA Server. With the configurator you can simply add or remove alarm texts without having to directly edit the XML files. Each alarm text has its own ID, which occurs only once in the file.

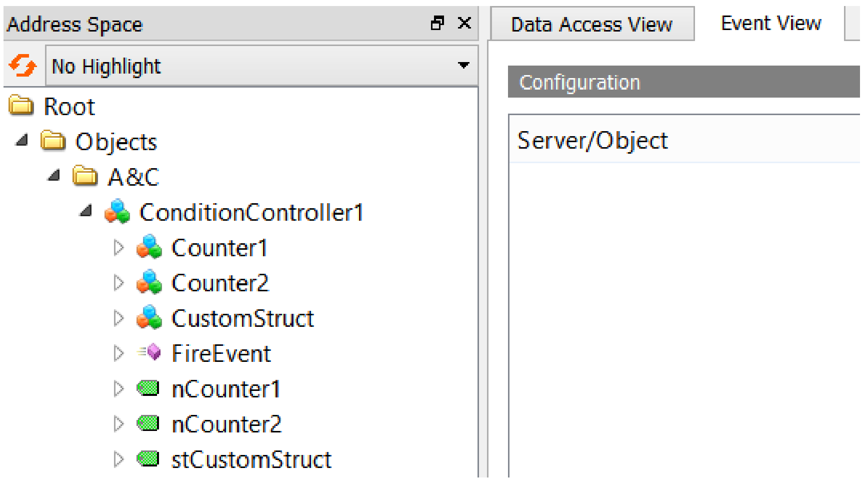
Sample:

```
<TcOpcUaSvrRes Lang="en">
  <Text ID="0">Text not available</Text>
  <Text ID="1">Some alarm text</Text>
  <Text ID="2">Value is High range</Text>
  <Text ID="3">Value is HighHigh range</Text>
  <Text ID="4">Value is OffNormal</Text>
  ...
</TcOpcUaSvrRes>
```

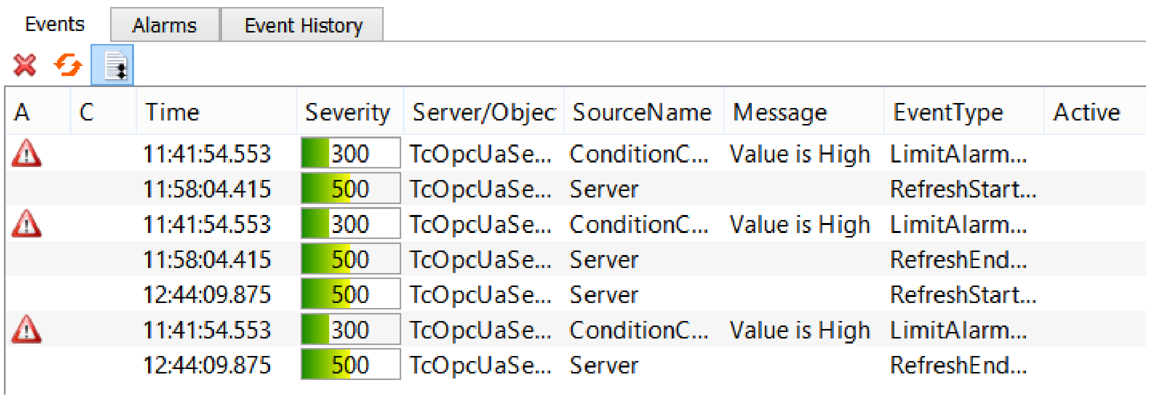
Step 6: Registering A&C with a reference OPC UA Client

An OPC UA Client must log on to a ConditionController so that it can receive events for conditions that are configured for this particular ConditionController. The UA Expert provides functions for subscribing to and receiving UA events.

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



You can subscribe to a ConditionController by dragging the corresponding object in Event View. The events for this ConditionController are displayed in the Event Window.



A	C	Time	Severity	Server/Objec	SourceName	Message	EventType	Active
⚠		11:41:54.553	300	TcOpcUaSe...	ConditionC...	Value is High	LimitAlarm...	
		11:58:04.415	500	TcOpcUaSe...	Server		RefreshStart...	
⚠		11:41:54.553	300	TcOpcUaSe...	ConditionC...	Value is High	LimitAlarm...	
		11:58:04.415	500	TcOpcUaSe...	Server		RefreshEnd...	
		12:44:09.875	500	TcOpcUaSe...	Server		RefreshStart...	
⚠		11:41:54.553	300	TcOpcUaSe...	ConditionC...	Value is High	LimitAlarm...	
		12:44:09.875	500	TcOpcUaSe...	Server		RefreshEnd...	

It may be necessary to subscribe to special alarm and/or event types in order to receive all fields of an incoming event or alarm.

4.1.9 Method Call

RPC methods can be called via the TwinCAT OPC UA Server in the PLC [▶ 90] or in C++ [▶ 91]. The handling of an RPC method call via OPC UA is performed in the same way on OPC UA level for both options:

- 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 was not called, the server returns an error message in the "Bad_XYZ" 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.

4.1.9.1 Job methods

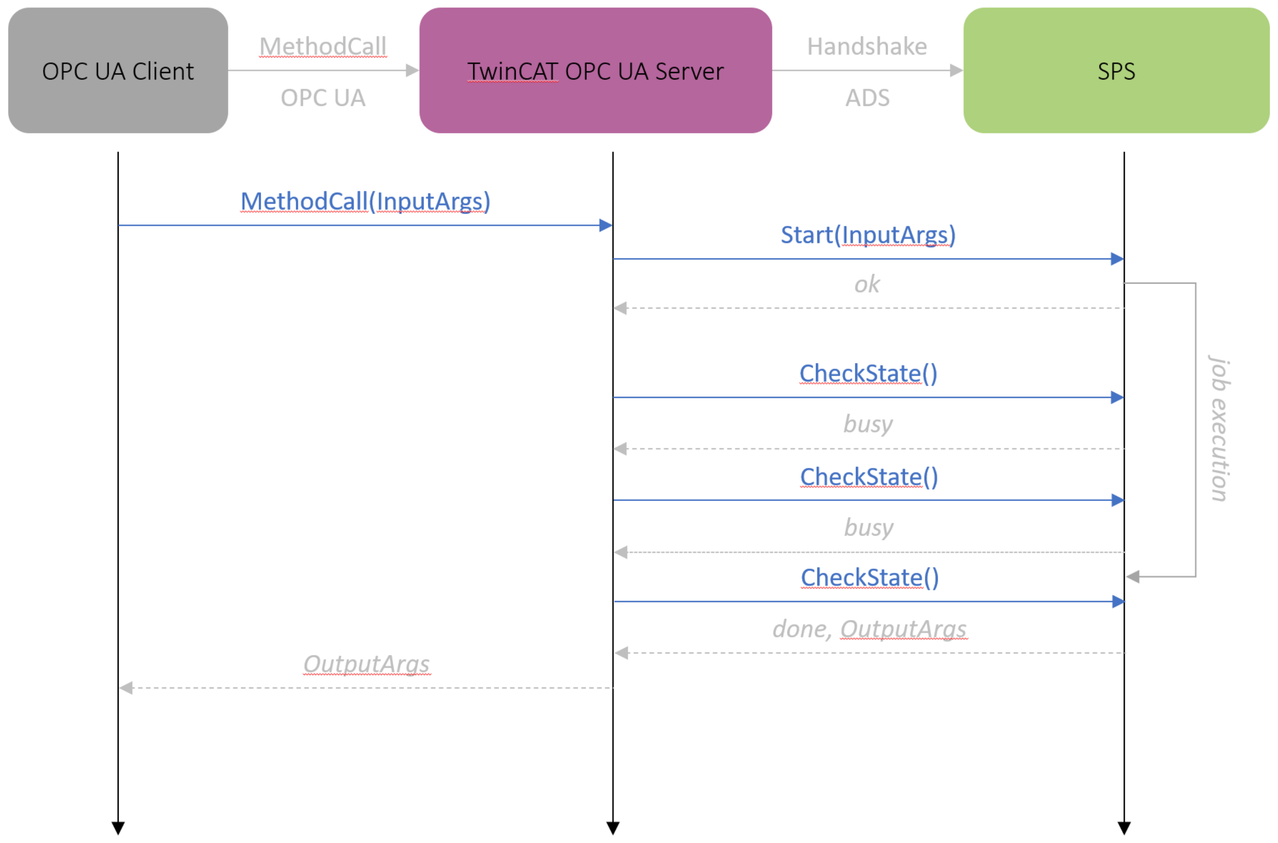
The concept of job methods has a fundamental difference compared to regular method calls: the OPC UA methods are no longer mapped 1:1 to a PLC method, but instead to a function block with a specific signature. This also allows method calls to be realized that take longer than one cycle from the perspective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

The PLC-side structure of such a job method is defined as follows. There is a function block that 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. 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

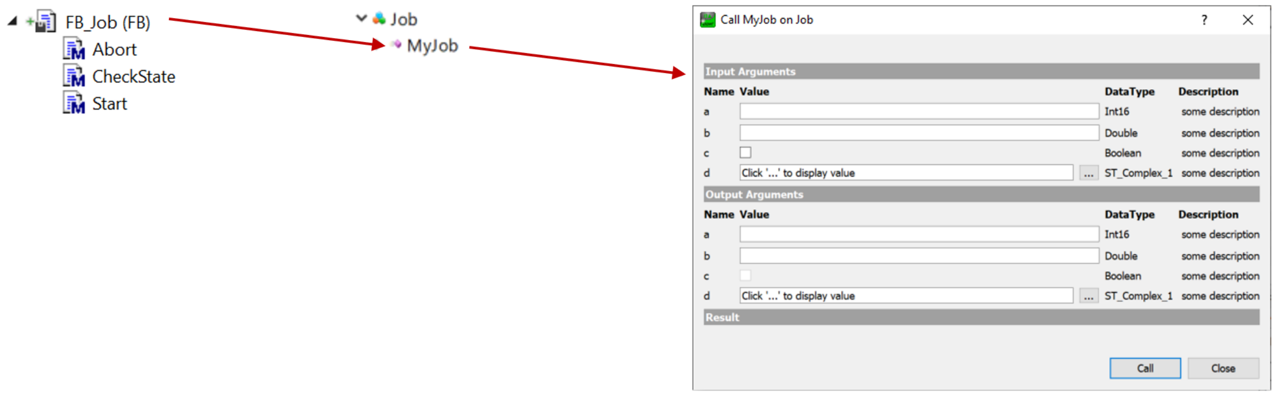
The handshake mechanism between server and PLC can be represented in simplified form as follows.



Sample

The following sample is also available in executable form at Samples [▶ 254]. This part of the documentation is intended to explain the basic concepts of how the system works. The sample "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 sample, a function block called FB_Job was created for the OPC UA method with the name "MyJob", which has the required signature mentioned above.



Call MyJob on Job

Input Arguments

Name	Value	DataType	Description
a		Int16	some description
b		Double	some description
c	☐	Boolean	some description
d	Click '...' to display value ...	ST_Complex_1	some description

Output Arguments

Name	Value	DataType	Description
a		Int16	some description
b		Double	some description
c		Boolean	some description
d	Click '...' to display value ...	ST_Complex_1	some description

Result

The function block declaration contains the OPC.UA.DA.JobMethod attribute 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). Sample:

```
{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. Sample:

```
{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
```

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. Sample:

```
{attribute 'TcRpcEnable' := '1'}
METHOD CheckState : HRESULT
VAR_OUTPUT
  a : INT; // some description
  b : LREAL; // some description
  c : BOOL; // some description
  d : ST_Complex_1; // some description
END_VAR
```

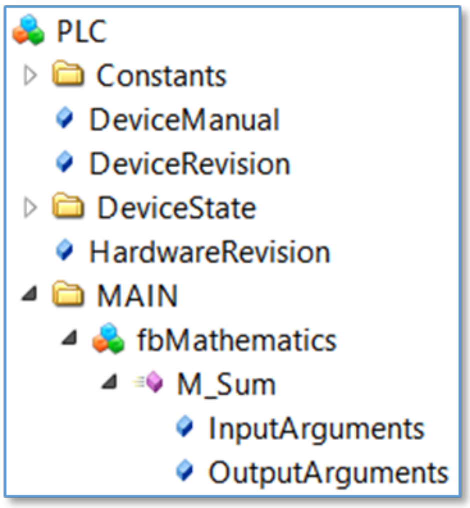
(In this sample, the values of the input parameters are applied 1:1 to the output parameters for illustrative purposes. The input and output parameters are therefore identical.)

4.1.9.2 PLC

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. The OPC UA Server imports PLC methods such as OPC UA methods via its TMC import.

IEC61131 method

Although the PLC method appears to be a normal method in the UA namespace, it is still an IEC61131 method that runs within the real-time context and therefore falls under the context of a real-time task. The PLC developer must therefore take precautions so that the execution time of the method matches the task cycle time.



Methods in IEC61131-3

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
```

Filtered or unfiltered mode

Depending on the import mode used, you also have to activate 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.

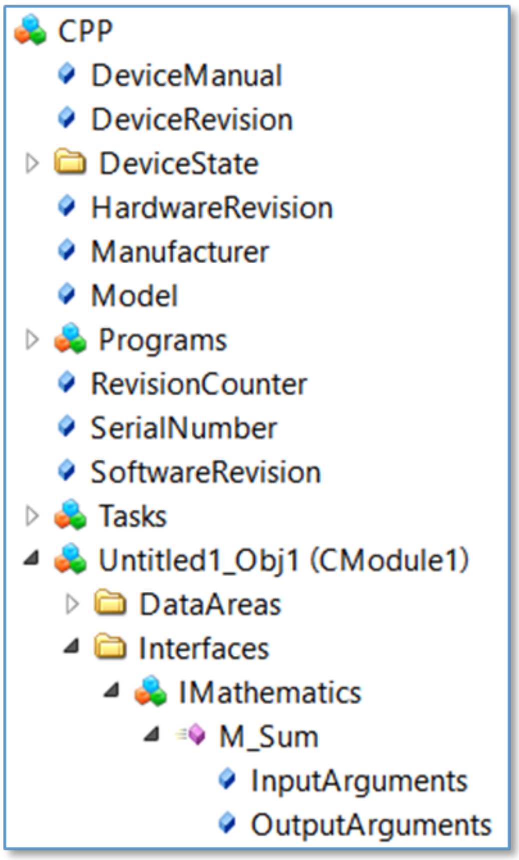
08:47:37.677Z|1|11A0* Error when importing method 'METH_PArray': VAR_IN_OUT pointer variables are not allowed!

4.1.9.3 C++

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 efficient execution of RPC calls in the C++ real-time context and thus reduces the classic handshake patterns for communication between devices. The OPC UA Server imports C++ methods such as OPC UA methods via its TMI import.

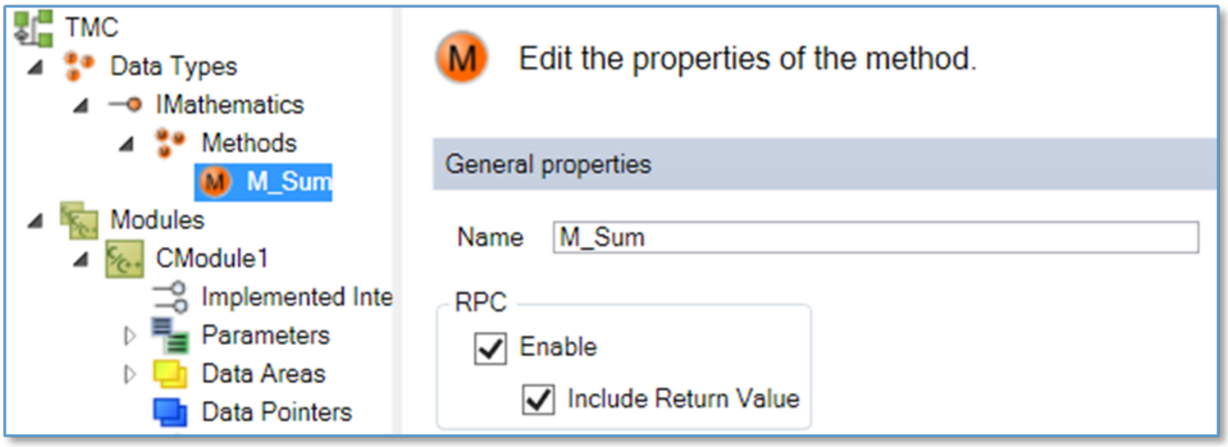
Real-time method

Although the C++ method appears to be a normal method in the UA namespace, it is still a real-time method that runs within the real-time context and therefore falls under the context of a real-time task. The TwinCAT C++ developer must therefore take precautions so that the execution time of the method matches the task cycle time.



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.



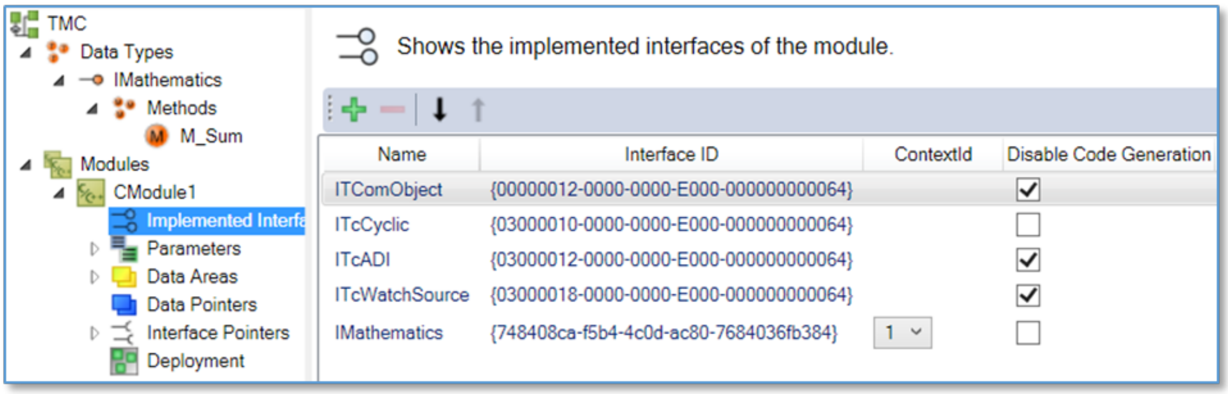
Edit the properties of the method.

General properties

Name: M_Sum

RPC
- [✓] Enable
- [✓] Include Return Value

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.

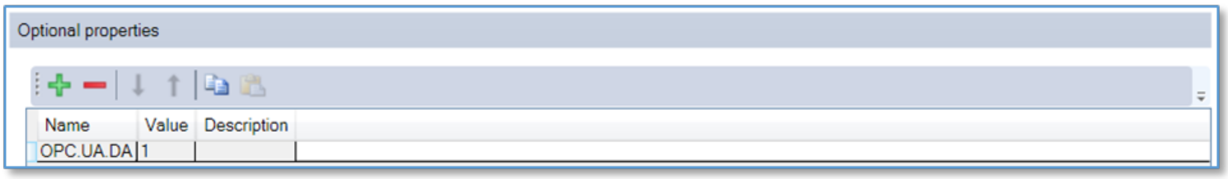


Shows the implemented interfaces of the module.

Name	Interface ID	ContextId	Disable Code Generation
ITComObject	{00000012-0000-0000-E000-000000000064}		✓
ITcCyclic	{03000010-0000-0000-E000-000000000064}		
ITcADI	{03000012-0000-0000-E000-000000000064}		✓
ITcWatchSource	{03000018-0000-0000-E000-000000000064}		✓
IMathematics	{748408ca-f5b4-4c0d-ac80-7684036fb384}	1	

Filtered or unfiltered mode

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.



Optional properties

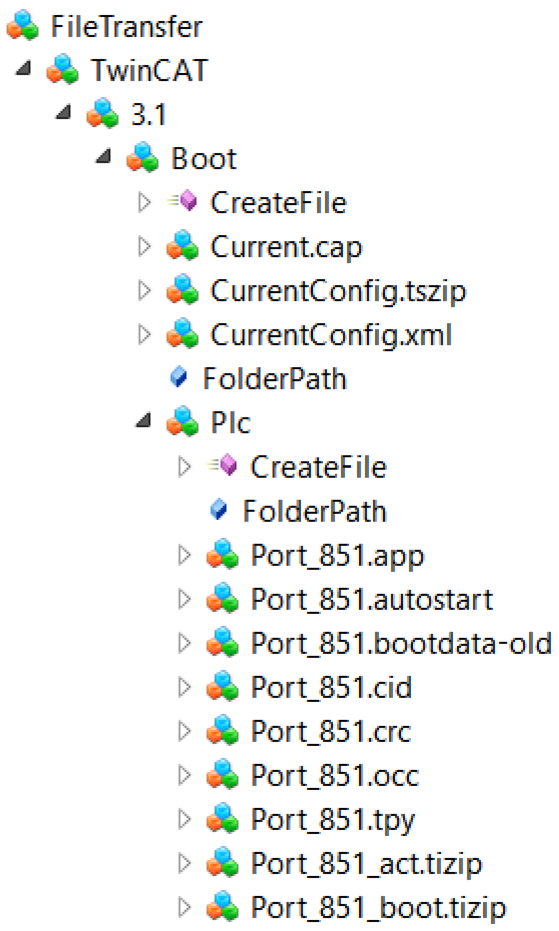
Name	Value	Description
OPC.UA.DA	1	

4.1.10 File transfer

4.1.10.1 Access to files and folders via OPC UA

From OPC UA specification version 1.02, OPC UA contains a specialized ObjectType for file transfer, which is described in Appendix C of the specification. 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.

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>
```

Sample: Reading a file with UA Expert

General file handling is described in Appendix C of the OPC UA specification. Reading a file via UA can 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 [▸ 95]).

- 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

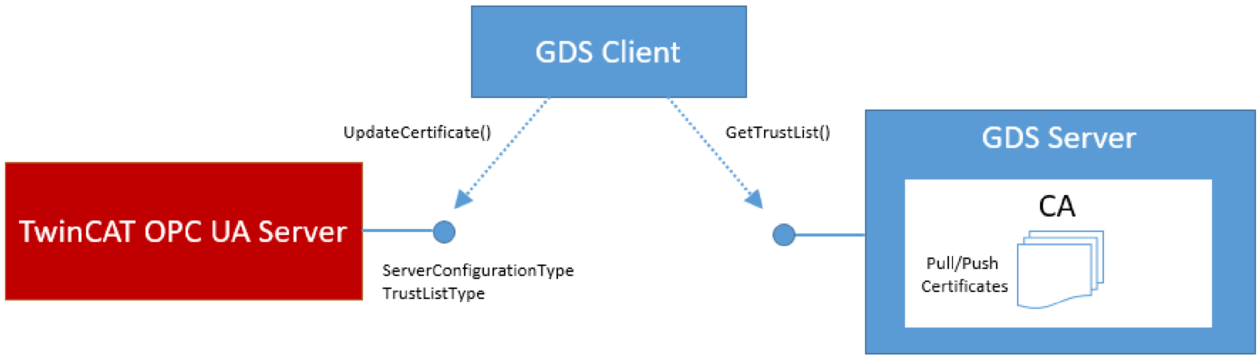
The number of files opened in parallel is in principle unlimited and is subject only to any restrictions of the underlying operating system. However, files are subject to a 60 seconds timeout. After this timeout, open files are not automatically closed immediately. Instead, they are marked as "to close". 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 released. If an OPC UA Client disconnects from the server and still has files open, all FileHandles belonging to this session will be closed automatically.

4.1.11 Global Discovery Service

The TwinCAT OPC UA Server allows integration of the server application into a Global Discovery Service (GDS), so that it can issue certificates for the server and provide certificate revocation lists (CRL). Two models are supported, push and pull, which are explained in more detail below:

Push

In this model, the server includes a standardized interface that a GDS client can use to connect to a Global Discovery Service at the request of the server, register the server application there and request a server certificate, including the current CRL. The certificate is then activated on the server.



As a prerequisite for using this functionality, the GDS client must authenticate itself on the server with a user account that has administrator rights [▶ 134] (IsRoot = true).

GDS Client

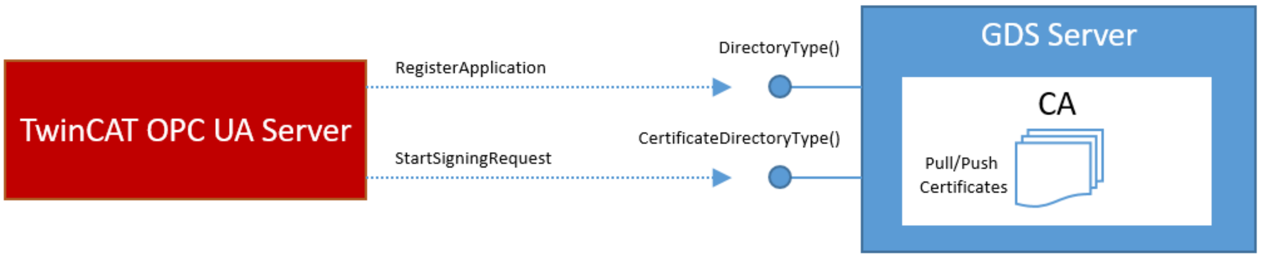
Any client that supports the push model can be used as a GDS Client. Various OPC UA toolkit manufacturers offer corresponding software packages. Alternatively, the OPC Foundation also provides a GDS Sample Client on Github.

GDS Server

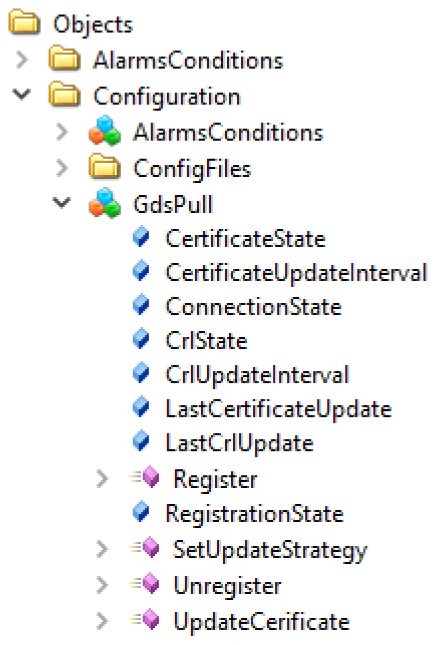
In principle, any GDS can be used as a Global Discovery Service. Various OPC UA toolkit manufacturers offer corresponding software packages. Alternatively, the OPC Foundation also provides a GDS Sample Server on Github.

Pull

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



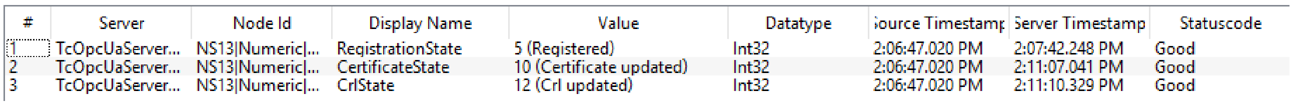
The TwinCAT OPC UA Server offers an option to activate and configure the GDS pull functions via its configuration namespace.



```
Objects
  > AlarmsConditions
  v Configuration
      > AlarmsConditions
      > ConfigFiles
      v GdsPull
            CertificateState
            CertificateUpdateInterval
            ConnectionState
            CrlState
            CrlUpdateInterval
            LastCertificateUpdate
            LastCrlUpdate
          > Register
            RegistrationState
          > SetUpdateStrategy
          > Unregister
          > UpdateCerificate
```

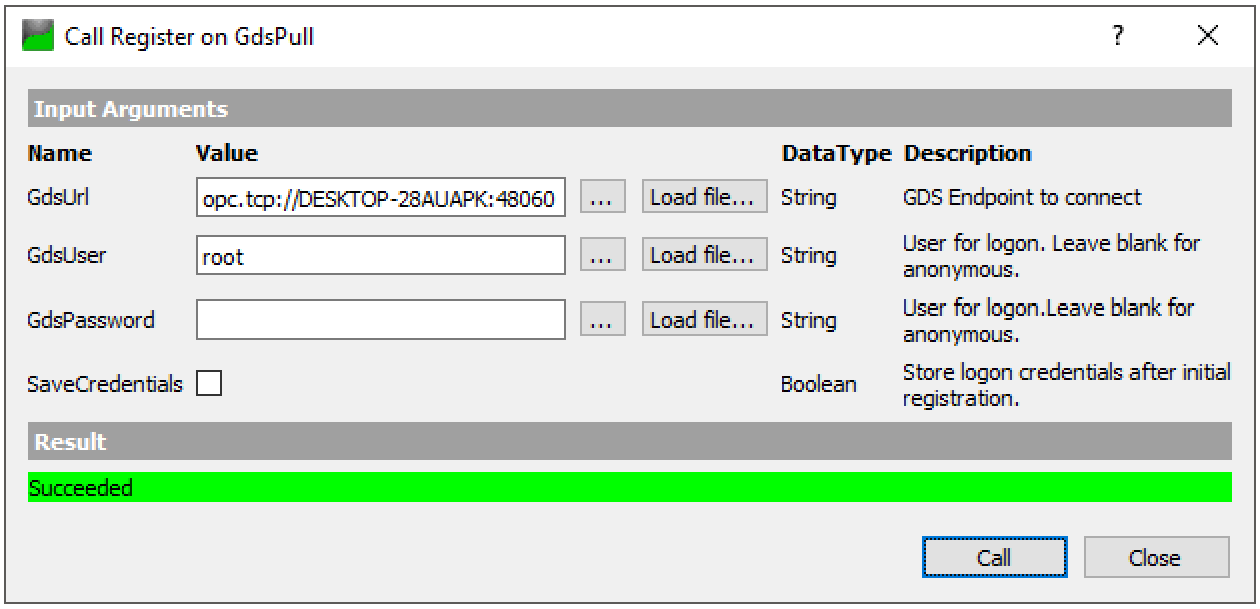
Registration at the Global Discovery Service

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	erver 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 Arguments

Name	Value	DataType	Description
GdsUrl	opc.tcp://DESKTOP-28AUAPK:48060	String	GDS Endpoint to connect
GdsUser	root	String	User for logon. Leave blank for anonymous.
GdsPassword		String	User for logon.Leave blank for anonymous.
SaveCredentials	☐	Boolean	Store logon credentials after initial registration.

Result

Succeeded

Input parameter	Meaning
GdsUrl	The URL of the Global Discovery Service in the format opc.tcp://hostnameOrIpAddress:port
GdsUser	User name 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, since it was designed exclusively for Global Discovery Service, which require user name/password authentication. However, this 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

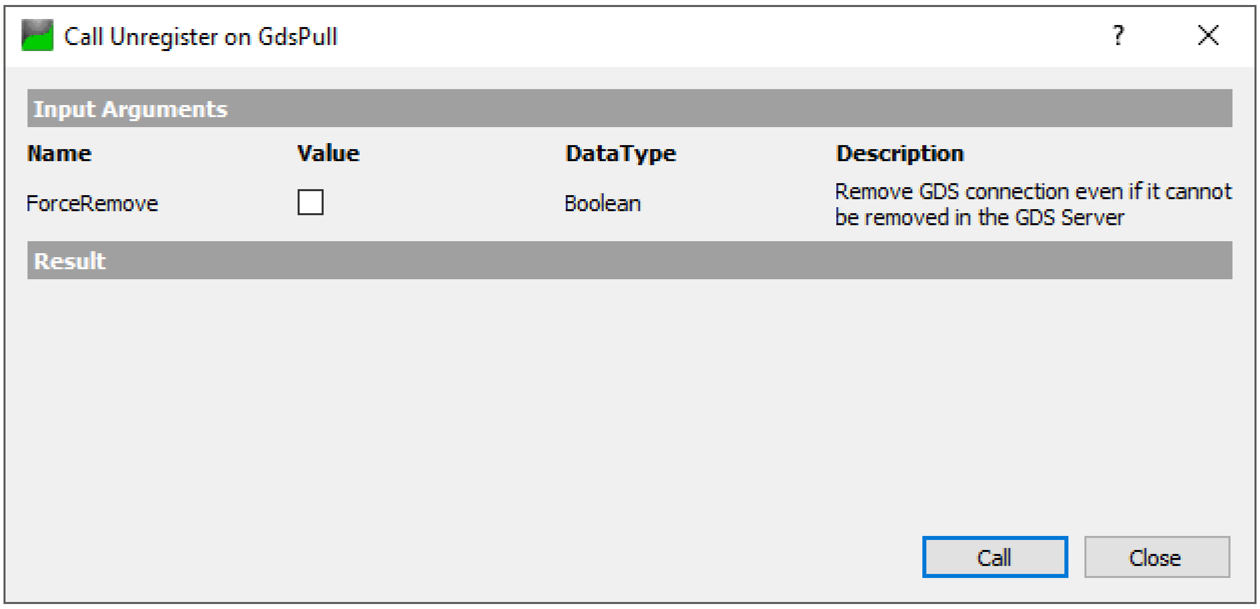
After registering the server application with the Global Discovery Service and obtaining a server certificate, the server reinitializes its endpoints once, causing connected clients to momentarily lose connectivity.

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.

De-registration at the Global Discovery Service

The Unregister() method can be used to de-register the TwinCAT OPC UA Server application at the GDS. Successful execution of the method causes the server application to be de-registered at the GDS and the contents of the file TcUaGdsClientConfig.xml to be deleted.

The method expects the following input parameters:


Call Unregister on GdsPull

Input Arguments

Name	Value	DataType	Description
ForceRemove	☐	Boolean	Remove GDS connection even if it cannot be removed in the GDS Server

Result

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.

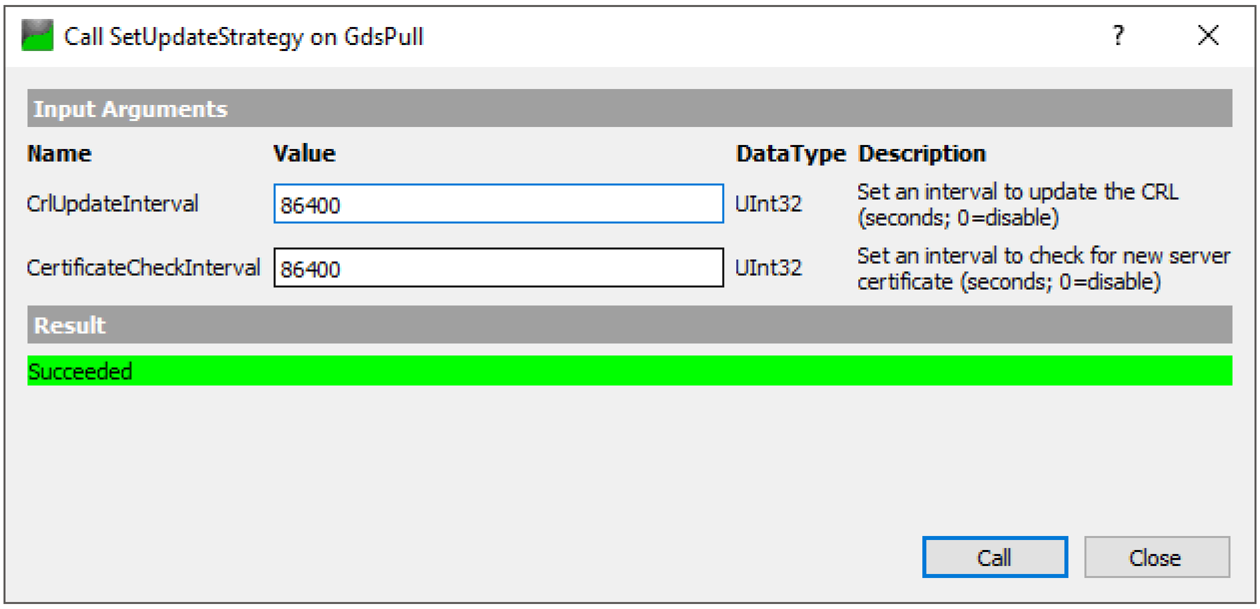
Updating the server certificate

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:


Call SetUpdateStrategy on GdsPull

Input Arguments

Name	Value	DataType	Description
CrlUpdateInterval	86400	UInt32	Set an interval to update the CRL (seconds; 0=disable)
CertificateCheckInterval	86400	UInt32	Set an interval to check for new server certificate (seconds; 0=disable)

Result

Succeeded

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

4.1.12 TwinCAT EventLogger

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.

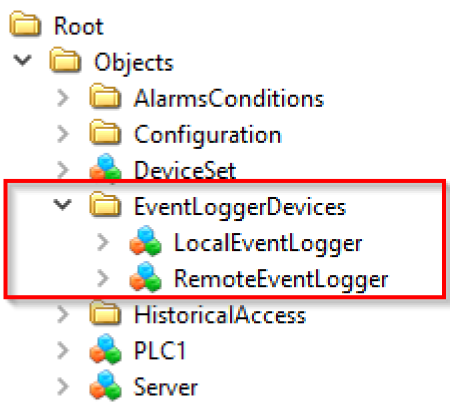
The integration of the TwinCAT EventLogger is not available for Windows CE. This means that TwinCAT OPC UA Servers running on Windows CE devices do not show the EventLoggerDevices folder. In this case it is only possible to integrate the TwinCAT EventLogger as a remote eventlogger in a TwinCAT OPC UA Server on another operating system.

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>
```

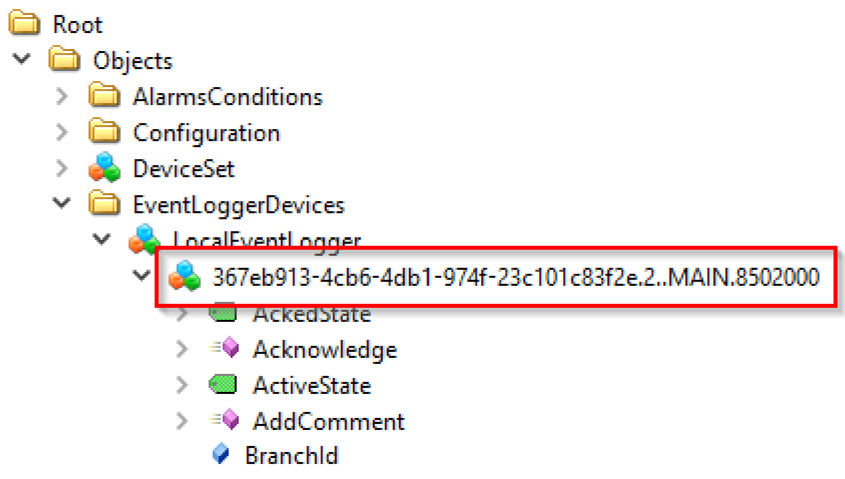
The individual device entries are then displayed in the "EventLoggerDevices" folder in the server's namespace.



```
Root
  Objects
    AlarmsConditions
    Configuration
    DeviceSet
    EventLoggerDevices
      LocalEventLogger
      RemoteEventLogger
    HistoricalAccess
    PLC1
    Server
```

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.



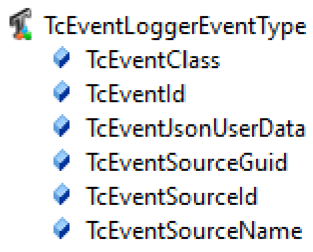
```
Root
  Objects
    AlarmsConditions
    Configuration
    DeviceSet
    EventLoggerDevices
      LocalEventLogger
        367eb913-4cb6-4db1-974f-23c101c83f2e.2..MAIN.8502000
          AckedState
          Acknowledge
          ActiveState
          AddComment
          BranchId
```

OPC UA Alarm/Event Types

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

TcEventLoggerEventType

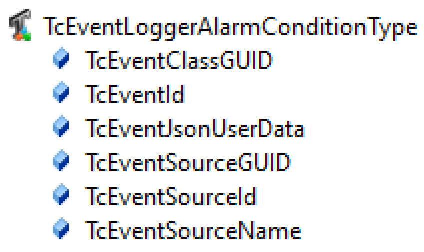
The TcEventLoggerEventType is derived from the BaseEventType and extends it with TwinCAT EventLogger-specific properties:


TcEventLoggerEventType
- TcEventClass
- TcEventId
- TcEventJsonUserData
- TcEventSourceGuid
- TcEventSourceId
- TcEventSourceName

```
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:


TcEventLoggerAlarmConditionType
- TcEventClassGUID
- TcEventId
- TcEventJsonUserData
- TcEventSourceGUID
- TcEventSourceId
- TcEventSourceName

```
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:

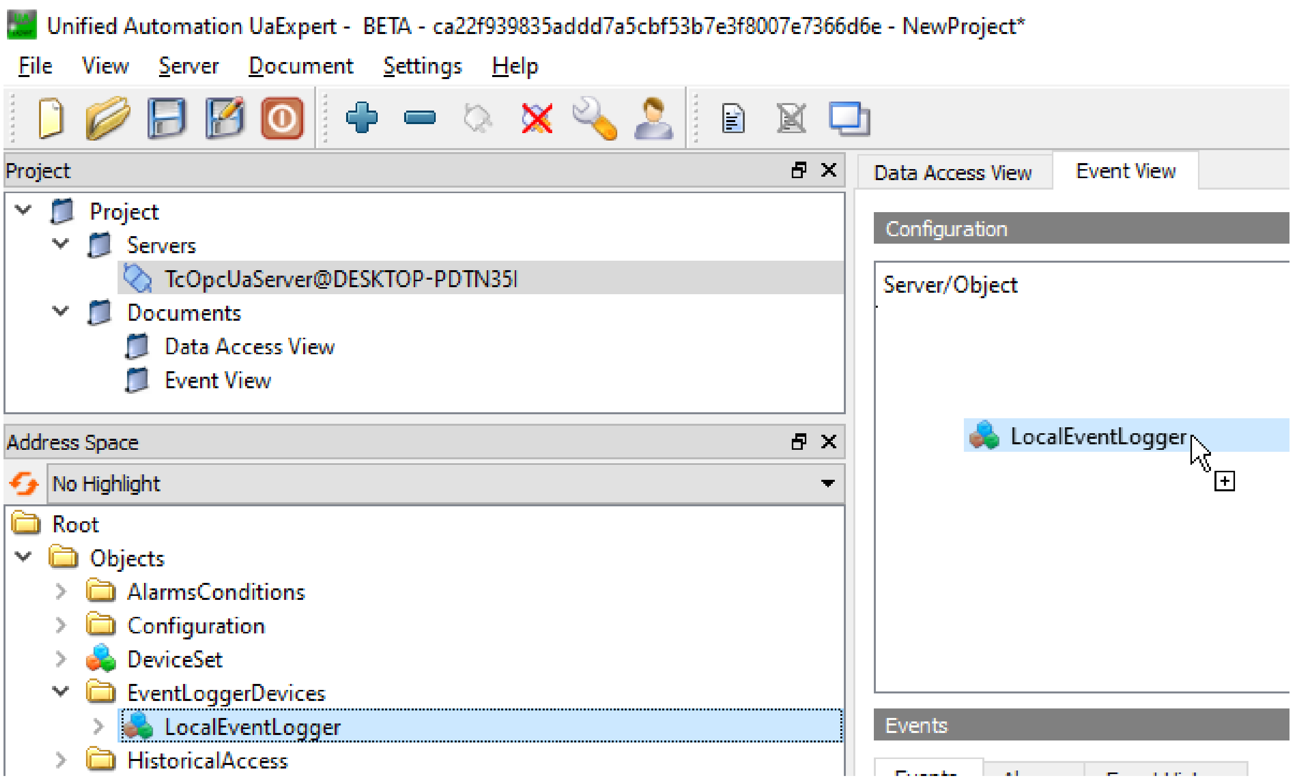
```
<TcUaEventLogConfig>
  <EventLoggerDevice Name="LocalEventLogger" AmsAddr="127.0.0.1.1.1"/>
</TcUaEventLogConfig>
```

Step 2: Activating the TwinCAT EventLogger sample

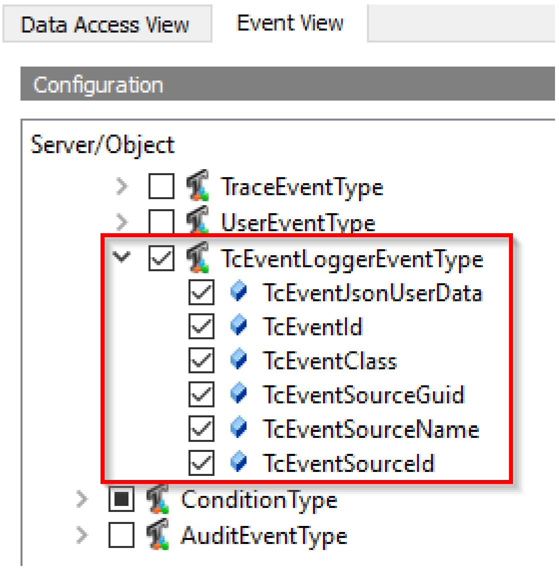
Before activating the TwinCAT EventLogger sample, we first deactivated the automatic firing of an event or alarm. In the present sample version this is done by initializing bSend and bAlmRaise with the value FALSE. The project is then activated and executed in the local PLC runtime.

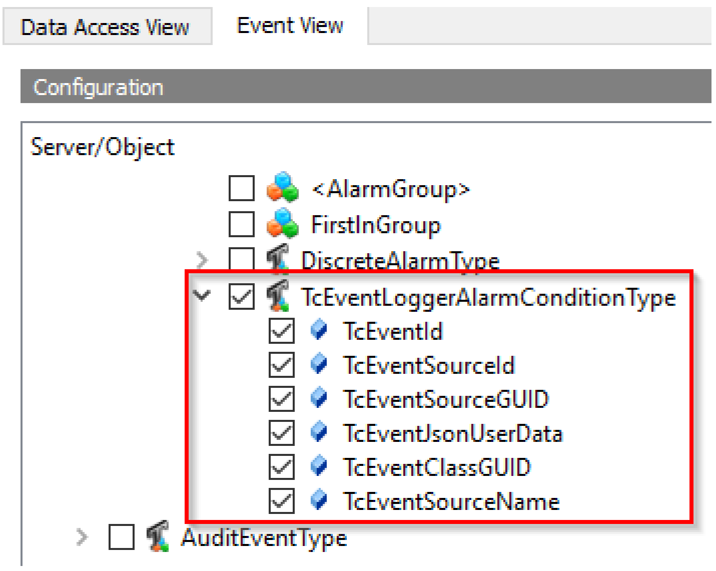
Step 3: Connecting an OPC UA Client to the Server

The UA Expert has now been 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:

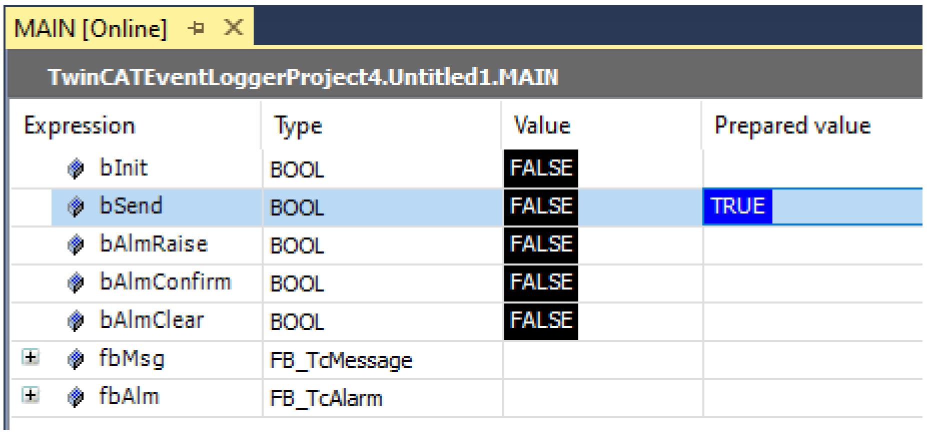




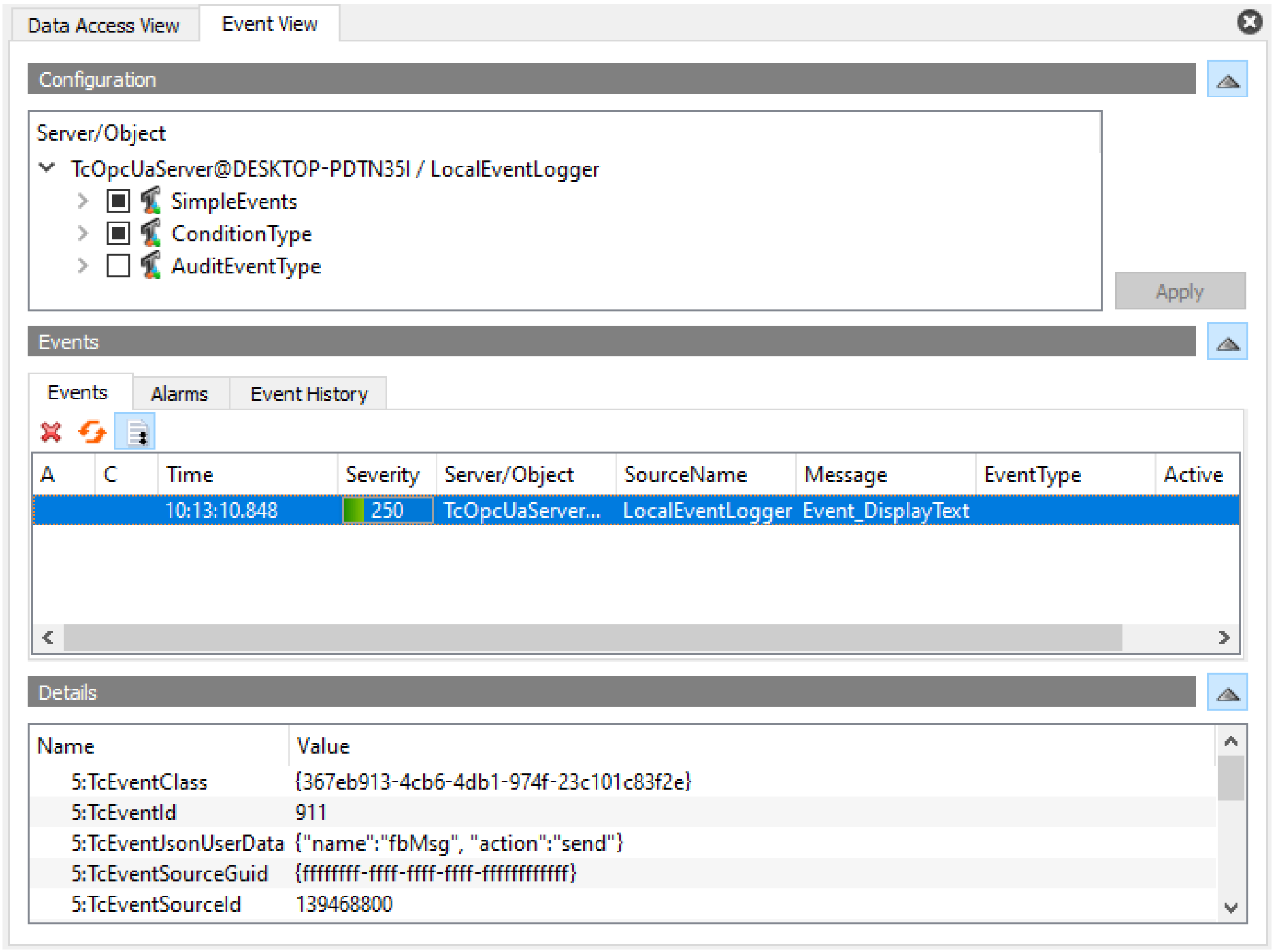
By clicking on the **Apply** button, these filters are applied.

Step 4: Firing an event

In the TwinCAT EventLogger Sample we now 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.



Expression	Type	Value	Prepared value
bInit	BOOL	FALSE	
bSend	BOOL	FALSE	TRUE
bAlmRaise	BOOL	FALSE	
bAlmConfirm	BOOL	FALSE	
bAlmClear	BOOL	FALSE	
fbMsg	FB_TcMessage		
fbAlm	FB_TcAlarm		



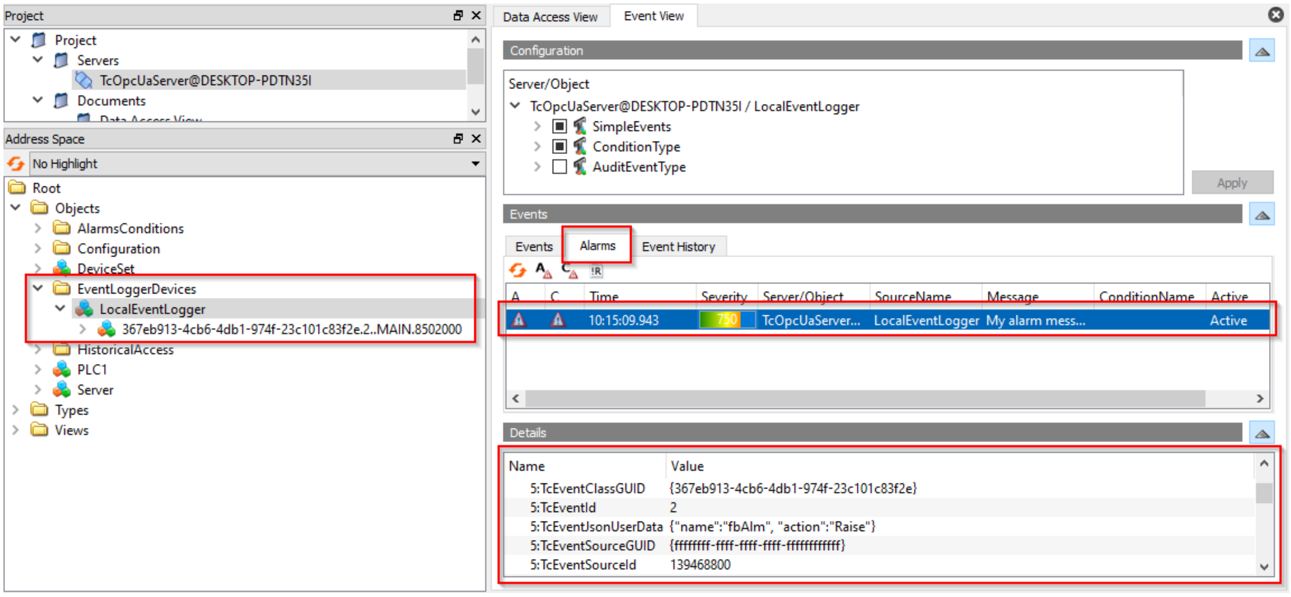
Server/Object
TcOpcUaServer@DESKTOP-PDTN35I / LocalEventLogger
SimpleEvents
ConditionType
AuditEventType

A	C	Time	Severity	Server/Object	SourceName	Message	EventType	Active
		10:13:10.848	250	TcOpcUaServer...	LocalEventLogger	Event_DisplayText		

Name	Value
5:TcEventClass	{367eb913-4cb6-4db1-974f-23c101c83f2e}
5:TcEventId	911
5:TcEventJsonUserData	{"name":"fbMsg", "action":"send"}
5:TcEventSourceGuid	{ffffffff-ffff-ffff-ffff-ffffffffffff}
5:TcEventSourceId	139468800

Step 5: Firing an alarm

In the TwinCAT EventLogger sample we now 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.



Project
Servers
TcOpcUaServer@DESKTOP-PDTN35I
Documents

Root
Objects
AlarmsConditions
Configuration
DeviceSet
EventLoggerDevices
LocalEventLogger
367eb913-4cb6-4db1-974f-23c101c83f2e.2..MAIN.8502000
HistoricalAccess
PLC1
Server
Types
Views

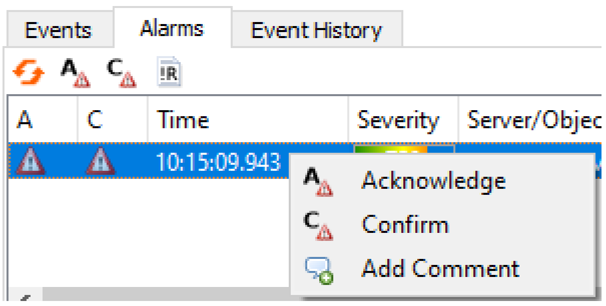
Server/Object
TcOpcUaServer@DESKTOP-PDTN35I / LocalEventLogger
SimpleEvents
ConditionType
AuditEventType

A	C	Time	Severity	Server/Object	SourceName	Message	ConditionName	Active
⚠	⚠	10:15:09.943	750	TcOpcUaServer...	LocalEventLogger	My alarm mess...		Active

Name	Value
5:TcEventClassGUID	{367eb913-4cb6-4db1-974f-23c101c83f2e}
5:TcEventId	2
5:TcEventJsonUserData	{"name":"fbAlm", "action":"Raise"}
5:TcEventSourceGUID	{ffffffff-ffff-ffff-ffff-ffffffffffff}
5:TcEventSourceId	139468800

Step 6: Acknowledging an alarm

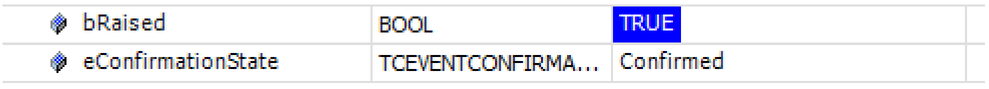
In addition to receiving an alarm, it can also be acknowledged. The acknowledgement is also reported by the server to the TwinCAT EventLogger. In UA Expert, an alarm can be acknowledged via the context menu in the Event View.



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

4.1.13 Security

4.1.13.1 Overview

One of the reasons for the success of OPC UA as communication technology is the various integrated security mechanisms. OPC UA-based data communication can be secured on two levels:

1. Transport layer
2. Application level

Endpoints

A server offers the client a list of different endpoints [▸ 105] 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 user name/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 [▸ 108].

Application level

In addition to the transport layer, a communication connection can also be secured at the application layer. For this purpose, various authentication mechanisms [▸ 106] are available, which are offered by the server endpoint.

Also see about this

Access rights [▸ 109]

4.1.13.2 Endpoints

The TwinCAT OPC UA Server makes various end points available for OPC UA Clients via the default port 4840/tcp. The end points 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 Local Discovery Server (LDS) from the OPC Foundation, which is used by some vendors with OPC UA software packages.

Safe end points

Please note that in order to use the secure end points, a trust relationship must be established between server and client, which is usually done via their certificates. The configuration of such a trust relationship on the server side is explained here [▶ 108].

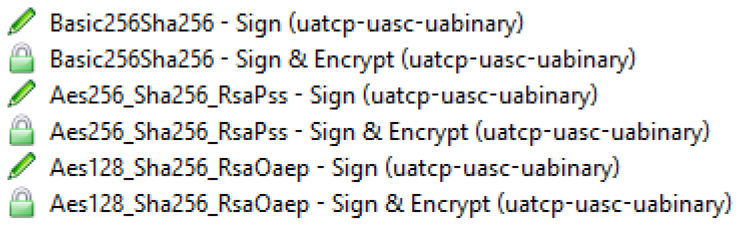
Deprecated endpoints

A configuration switch (<AllowDeprecatedSecurityPolicies>) in TcUaServerConfig.xml can be used to reactivate security policies that are obsolete and classified as unsafe. However, we recommend leaving this configuration switch disabled for security reasons.

List of endpoints

The list provides an overview of the endpoints of the OPC UA Server. This includes endpoints that have already been discontinued. The following screenshot shows the endpoints currently contained in the OPC UA Server (version 3.2.0.62). The options are signing only or signing in combination with encryption.

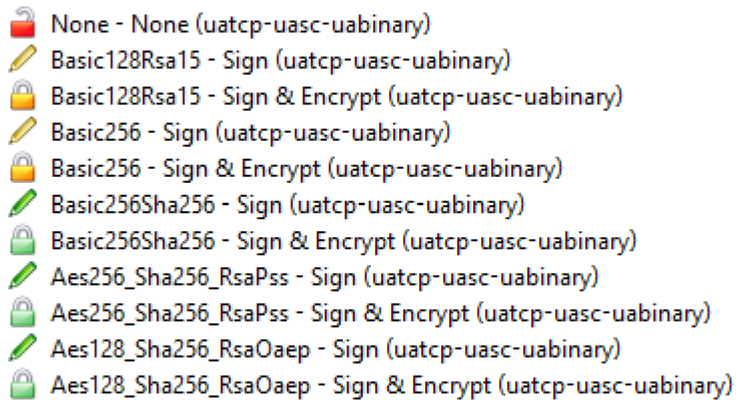
From setup version 4.3.28 (server version 3.2.0.62) or higher, the unencrypted endpoint is disabled by default for security and certification reasons.


Basic256Sha256 - Sign (uatcp-uasc-uabinary)
Basic256Sha256 - Sign & Encrypt (uatcp-uasc-uabinary)
Aes256_Sha256_RsaPss - Sign (uatcp-uasc-uabinary)
Aes256_Sha256_RsaPss - Sign & Encrypt (uatcp-uasc-uabinary)
Aes128_Sha256_RsaOaep - Sign (uatcp-uasc-uabinary)
Aes128_Sha256_RsaOaep - Sign & Encrypt (uatcp-uasc-uabinary)

The available endpoints are based on the safety of the security mechanisms. If security profiles are classified as potentially unsafe over time, they are no longer used in the TwinCAT OPC UA Server by default and are replaced by newer and more secure encryption algorithms.

Security profile	Security mode	Short description
None	None	This is the unencrypted endpoint, which can be used to communicate without security. Disabled by default since version 4.3.28. Can be reactivated if required by configuring the server accordingly.
Basic128Rsa15 (obsolete)	Sign / Sign & Encrypt	This endpoint is considered obsolete from a security perspective and is disabled by default. Can be re-enabled if required by configuring the server accordingly.
Basic256 (obsolete)	Sign / Sign & Encrypt	This endpoint is considered obsolete from a security perspective and is disabled by default. Can be re-enabled if required by configuring the server accordingly.
Basic256Sha256	Sign / Sign & Encrypt	Endpoint currently present in the server for secure signing and encryption.
Aes256_Sha256_RsaPss	Sign / Sign & Encrypt	Endpoint currently present in the server for secure signing and encryption.
Aes256_Sha256_RsaOaep	Sign / Sign & Encrypt	Endpoint currently present in the server for secure signing and encryption.

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


None - None (uatcp-uasc-uabinary)
Basic128Rsa15 - Sign (uatcp-uasc-uabinary)
Basic128Rsa15 - Sign & Encrypt (uatcp-uasc-uabinary)
Basic256 - Sign (uatcp-uasc-uabinary)
Basic256 - Sign & Encrypt (uatcp-uasc-uabinary)
Basic256Sha256 - Sign (uatcp-uasc-uabinary)
Basic256Sha256 - Sign & Encrypt (uatcp-uasc-uabinary)
Aes256_Sha256_RsaPss - Sign (uatcp-uasc-uabinary)
Aes256_Sha256_RsaPss - Sign & Encrypt (uatcp-uasc-uabinary)
Aes128_Sha256_RsaOaep - Sign (uatcp-uasc-uabinary)
Aes128_Sha256_RsaOaep - Sign & Encrypt (uatcp-uasc-uabinary)

4.1.13.3 Authentication

An OPC UA client application can authenticate itself to the server via various IdentityTokens:

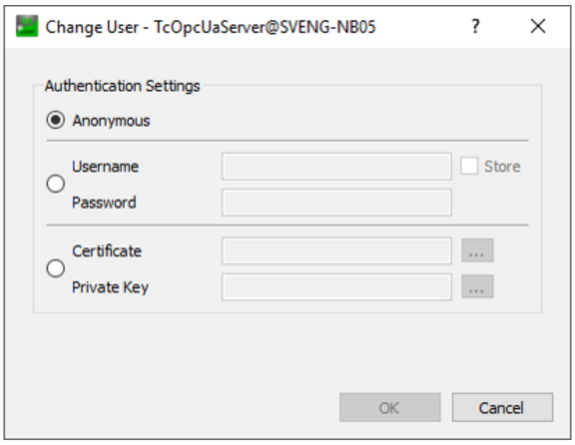
- Anonymous
- User name/Password
- User certificate

Delivery state

The server is delivered with the Anonymous IdentityToken enabled, but its use requires a one-time initialization [▶ 37]. The Anonymous IdentityToken is then disabled and client applications must authenticate to the server with a valid User IdentityToken.

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. We recommend deactivating this authentication type after commissioning the server. This can be done via the TwinCAT OPC UA Configurator. In the following you will find an example screenshot from the OPC UA Client application "UA Expert":


Change User - TcOpcUaServer@SVENG-NB05

Authentication Settings
- [✓ Anonymous]
- Username / Password — Store
- Certificate / Private Key

User name/Password

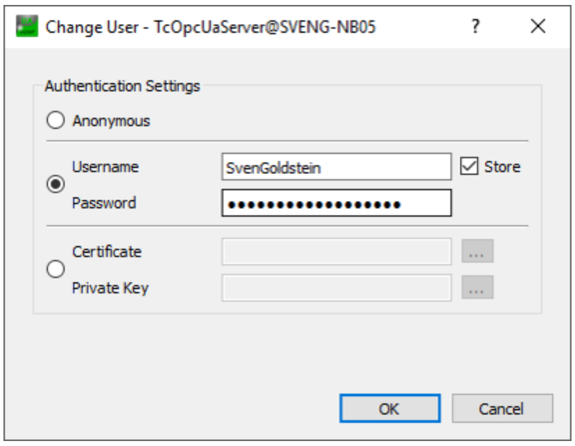
This type of authentication uses a user name/password combination to authenticate the client to the server application. On the server, access rights can then be defined for the respective user identity. The user identity can be defined on different levels:

- 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 User IdentityTokens

If User IdentityTokens are to be used to authenticate client applications, we recommend using operating system users.

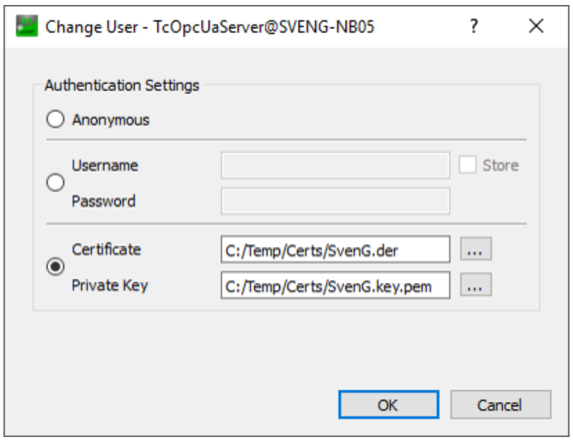
In the following you will find an example screenshot from the OPC UA Client application "UA Expert":


Change User - TcOpcUaServer@SVENG-NB05

Authentication Settings
- Anonymous
- [✓ Username: SvenGoldstein] [✓ Store]
  Password: ••••••••••••••••••
- Certificate / Private Key

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. In the following you will find an example screenshot from the OPC UA Client application "UA Expert":


Change User - TcOpcUaServer@SVENG-NB05

Authentication Settings

○ Anonymous

○ Username / Password — Store

● Certificate: C:/Temp/Certs/SvenG.der

Private Key: C:/Temp/Certs/SvenG.key.pem

OK Cancel

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 [▸ 108]).

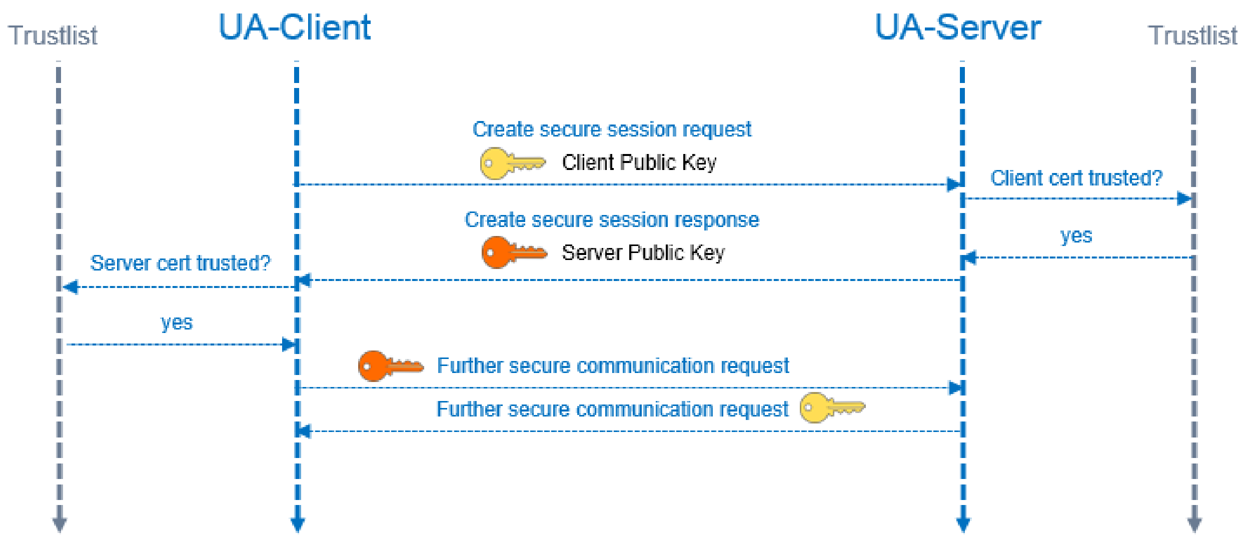
Also see about this

🗎 Access rights [▸ 109]

4.1.13.4 Certificate exchange

To secure the communication connection at transport layer via a secure endpoint [▸ 105], it is necessary to establish a mutual trust between client and server.

By default, both the TwinCAT OPC UA Server and the TwinCAT OPC UA Client generate a machine-specific, self-signed certificate for authentication of the respective application at the first start.


Trustlist
UA-Client
UA-Server
Trustlist
Create secure session request
Client Public Key
Client cert trusted?
Create secure session response
Server Public Key
yes
Server cert trusted?
yes
Further secure communication request
Further secure communication request

Set up a trust relationship on the server

To establish a trust relationship between any OPC UA Client and the TwinCAT OPC UA Server, you need the public key of the client certificate. The server must trust this. This can be done via the file system, for example. The server manages the trust settings for client certificates in the PKI subdirectory.

- Trusted certificates: %InstallDir%\Server\PKI\CA\trusted\certs
- Untrusted certificates: %InstallDir%\Server\PKI\CA\rejected\certs

By moving client certificates between these directories, the trust settings can be adjusted accordingly. The public key of a client certificate is automatically stored in the above untrusted certificate directory the first time the client attempts to connect to a secure endpoint. By subsequently moving the public key to the trusted certificate directory, the client is trusted the next time it attempts to connect.

AutomaticallyTrustAllClientCertificates

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

Set up a trust relationship on the client

Depending on the OPC UA Client employed, different steps may need to be taken so that the OPC UA Client trusts the OPC UA Server. Typically, for client applications with a graphical user interface, a warning message is displayed the first time you connect to the server, whereby the server certificate can then be classified as trustworthy.

The following instruction is therefore only valid for the TwinCAT OPC UA Client.

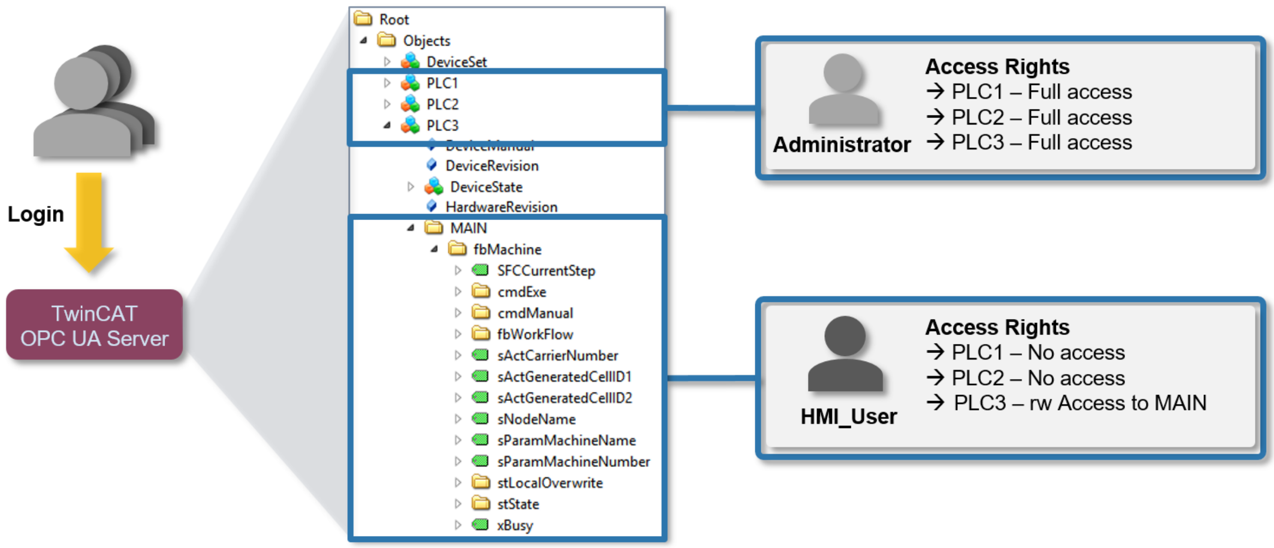
The public key of the OPC UA Server is located as a DER file in the following directory: *%InstallDir%\Server\PKI\CA\own\certs*

In the case of the TwinCAT OPC UA Client, copy the file into the corresponding "Trusted" directory: *%InstallDir%\Client\PKI\CA\trusted\certs*

4.1.13.5 Access rights

The TwinCAT OPC UA Server enables the configuration of access rights for specific authenticated user identities [▶ 106]. These access rights can be configured for entire namespaces as well as for individual nodes.

This allows you to fine-granulate the access to ADS devices (for example, to different PLC runtimes) as well as variables. These security settings are available for all ADS devices that can be displayed in the server namespace.



See also:

Configuration > Setup Version 4.x.x. > Configuring the security settings in the configurator [▶ 134].

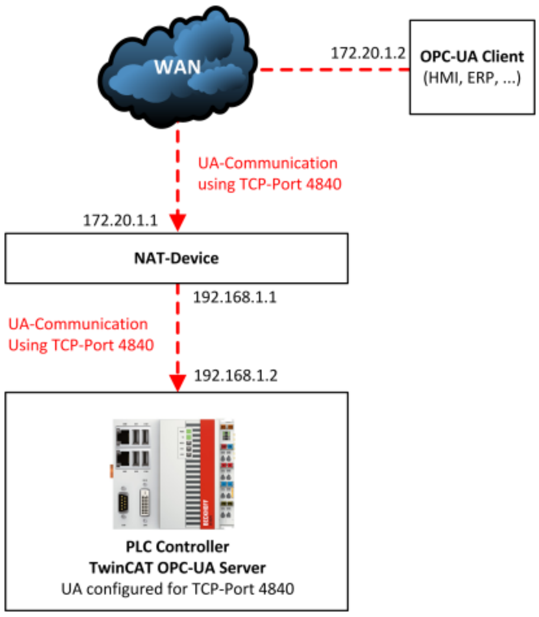
Requirements

Products	Setup versions	Target platform
TF6100	4.x.x	IPC or CX (x86, x64, ARM)

4.1.14 Miscellaneous

4.1.14.1 Configuring firewalls

To make OPC UA communication also possible via a NAT device, e.g. an Internet router, this device must be able to forward the UA port used to the TwinCAT OPC UA Server (so-called "port forwarding"). By default the TwinCAT OPC UA Server is configured for UA communication via the TCP port 4840; however, you can adapt this configuration yourself if necessary, either via the server configuration file or using the OPC UA Configurator. The following figure illustrates the relationship between port forwarding and the UA Server.

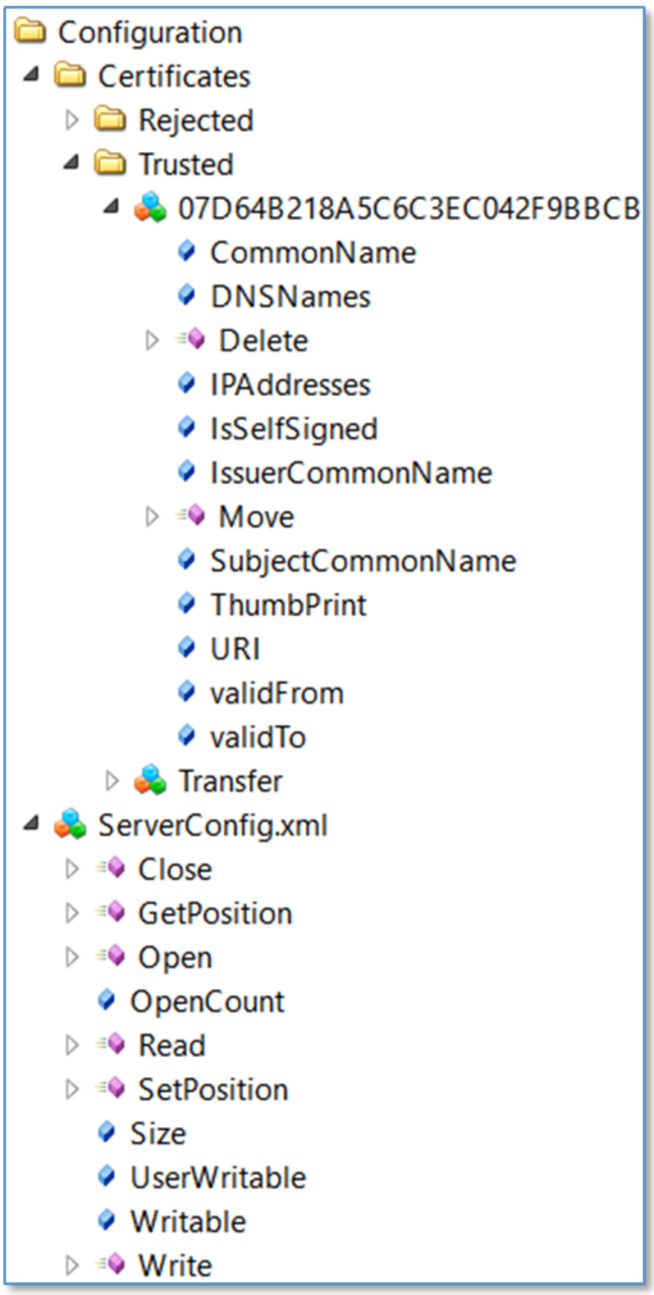


In this example, the OPC UA Client establishes a UA connection to the NAT device via TCP port 4840, which then forwards this communication connection to the TwinCAT OPC UA Server via port forwarding. The NAT device thus only needs to forward the UA port configured in the TwinCAT OPC UA Server to the target device. The port used by the UA Server can either be viewed in the server configuration file or conveniently through the UA Configurator (in delivery state this is always TCP port 4840). For the appropriate configuration of your NAT device for port forwarding please refer to its documentation.

4.1.14.2 Namespace for configuration of the server

From server version 2.1.x the TwinCAT OPC UA Server provides a configuration namespace that contains the following functionalities:

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



Management of the server configuration files

The server configuration files are published in the namespace as a regular OPC UA FileType and therefore provide the methods and properties required to access the file.

Management of certificates

Each client certificate known to the server is published in the namespace as an OPC UA CertificateType. 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 trust lists by calling the Move() method.

In addition, various properties provide additional information about the certificates themselves for easier identification.

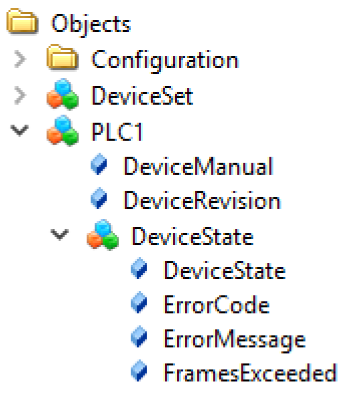
Using the configuration namespace

The configuration namespace is enabled by default for ease of use and is available to users. This enables the TwinCAT OPC UA Configurator to connect to a server and apply the corresponding server configuration.

We recommend that you only make the namespace accessible to authenticated users once the server configuration is complete. This means that an OPC UA Client must authenticate itself vis-à-vis the OPC UA Server by providing a valid user name/password combination to access the namespace. Click here [▸ 134] to find out how to do this.

4.1.14.3 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 readable error message is then displayed in the ErrorMessage Property.

4.1.14.4 ReverseConnect

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.

Client compatibility

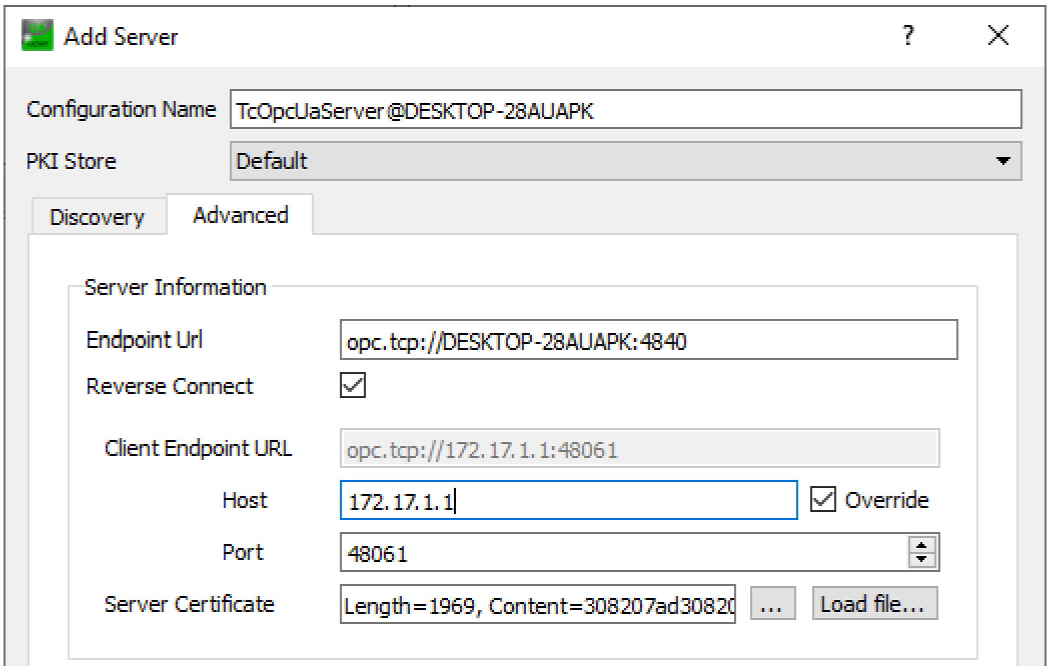
Please note that the OPC UA Client must also support this function and must be accessible via its ReverseConnect URL.

Configuration in the server

A list of OPC UA Clients can be configured in the TcUaServerConfig.xml within the <UaEndpoint>.

```
<!-- List with Clients -->
<ReverseConnect>
  <Url>opc.tcp://172.17.1.1:48061</Url>
</ReverseConnect>
```

Example configuration in UA Client (UA Expert)


Add Server

Configuration Name: TcOpcUaServer@DESKTOP-28AUAPK

PKI Store: Default

Discovery | Advanced

Server Information

Endpoint Url: opc.tcp://DESKTOP-28AUAPK:4840

Reverse Connect: ☑

Client Endpoint URL: opc.tcp://172.17.1.1:48061

Host: 172.17.1.1 ☑ Override

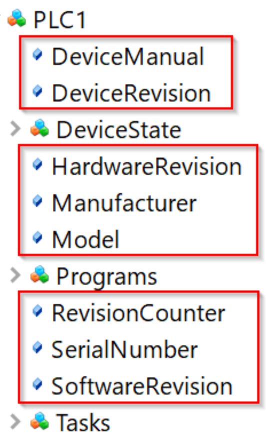
Port: 48061

Server Certificate: Length=1969, Content=308207ad30820 ... Load file...

In UA Expert, you can enable ReverseConnect in the **Advanced** tab and configure the hostname or IP address of the client PC, as well as the port to be opened by the client and the server certificate to establish trust.

4.1.14.5 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.



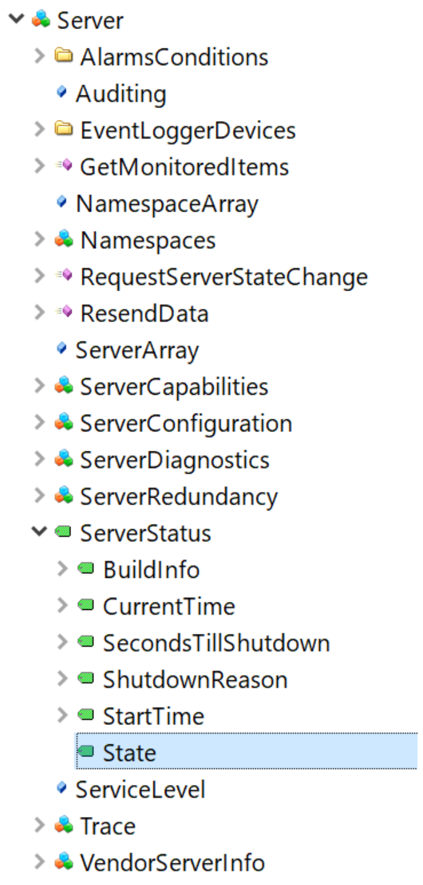
The following table provides more information about these nodes. The concrete value assignments of the individual nodes can be application-specific to a large extent, therefore only example values are used in the delivery state of the server. The server itself makes these nodes available in its namespace, 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.
SoftwareRevision	Contains the version or revision level of the software component, the firmware of a hardware component or even the firmware of the device.

4.1.14.6 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 [▶ 37].
Suspended	The server has not yet been completely booted, i.e. not all functions may be available yet.



4.1.14.7 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. This log file is usually only required for diagnostic purposes.

Logging is usually enabled/disabled via the Visual Studio Configurator [▶ 142] or Standalone Configurator [▶ 157].

The default path for the log files created is:

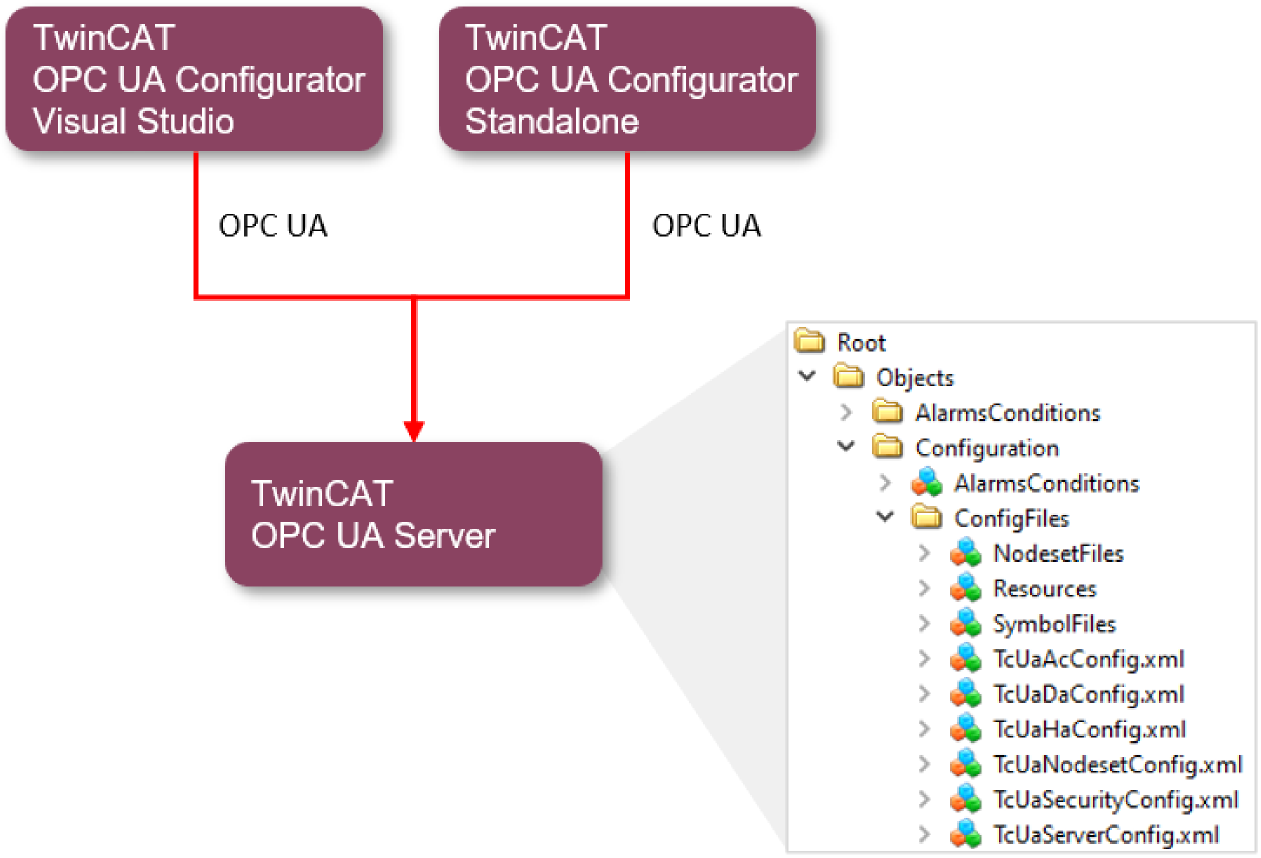
```
Windows: %ProgramData%\Beckhoff\TF6100-OPC-UA-Server\logs
TwinCAT/BSD: /var/log/TF6100-OPC-UA-Server
```

4.2 Configurator

The TwinCAT OPC UA Configurator offers a graphical front end for editing the TwinCAT OPC UA Server configuration files. It comes in two variants:

- as Visual Studio [▶ 116] project template
- as Standalone [▶ 144] application

Both variants are part of the TwinCAT OPC UA Configurator setup and enable online configuration of the TwinCAT OPC UA Server by connecting to the server via OPC UA and editing its configuration files. This relationship is illustrated once again in the following diagram.

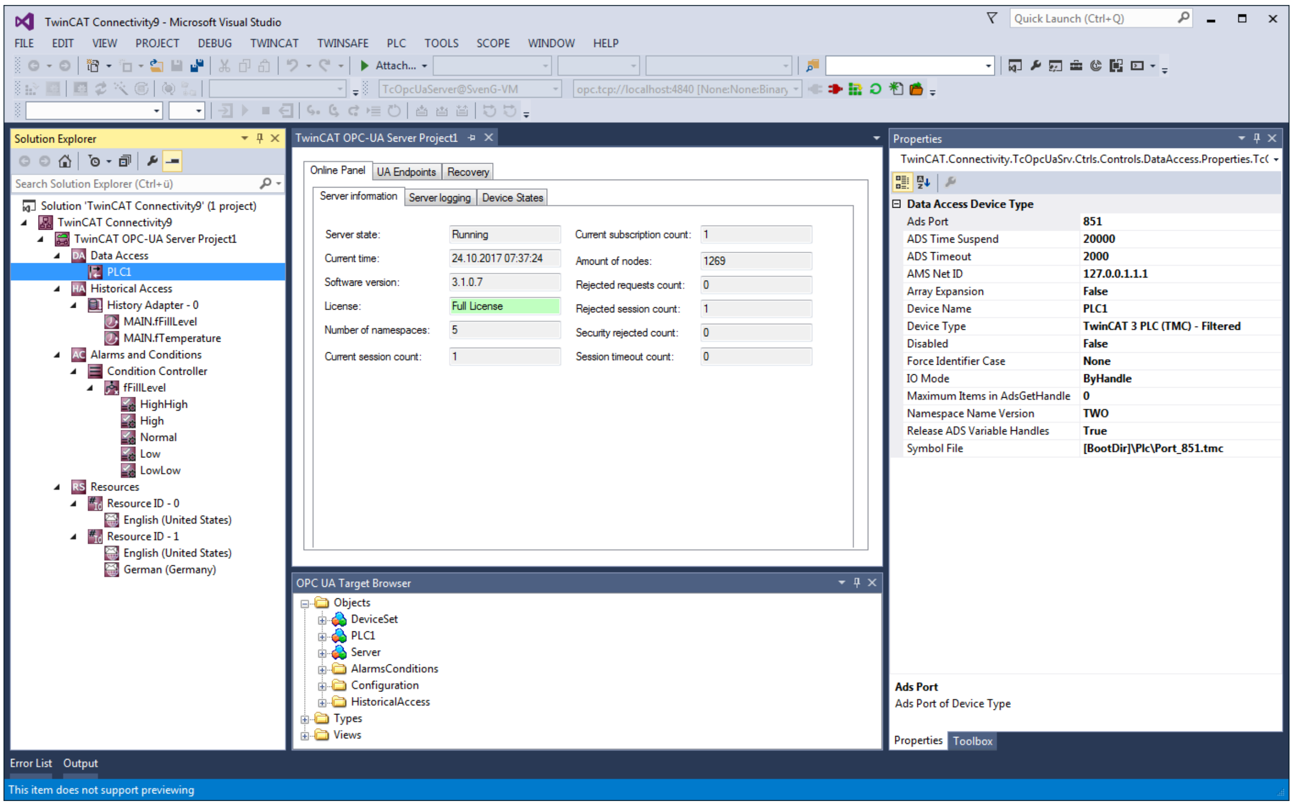


The basis for configuration via OPC UA is the so-called configuration namespace [▶ 110] of the TwinCAT OPC UA Server. Here the configuration files of the server are provided as OPC UA objects.

4.2.1 Visual Studio

4.2.1.1 Overview

The TF6100 setup (version 4.x.x and higher) contains the latest version of the OPC UA Server Configurator. This was integrated in Microsoft Visual Studio as a separate project type to provide an integrated and consistent engineering concept. You can configure all the different facets of the TwinCAT OPC UA Server and in doing so also use source control mechanisms such as Team Foundation Server or Subversion Integrations.



Server state:	Running	Current subscription count:	1
Current time:	24.10.2017 07:37:24	Amount of nodes:	1269
Software version:	3.1.0.7	Rejected requests count:	0
License:	Full License	Rejected session count:	1
Number of namespaces:	5	Security rejected count:	0
Current session count:	1	Session timeout count:	0

Data Access Device Type	
Ads Port	851
ADS Time Suspend	20000
ADS Timeout	2000
AMS Net ID	127.0.0.1.1.1
Array Expansion	False
Device Name	PLC1
Device Type	TwinCAT 3 PLC (TMC) - Filtered
Disabled	False
Force Identifier Case	None
IO Mode	ByHandle
Maximum Items in AdsGetHandle	0
Namespace Name Version	TWO
Release ADS Variable Handles	True
Symbol File	[BootDir]\Plc\Port_851.tmc

Ads Port
Ads Port of Device Type

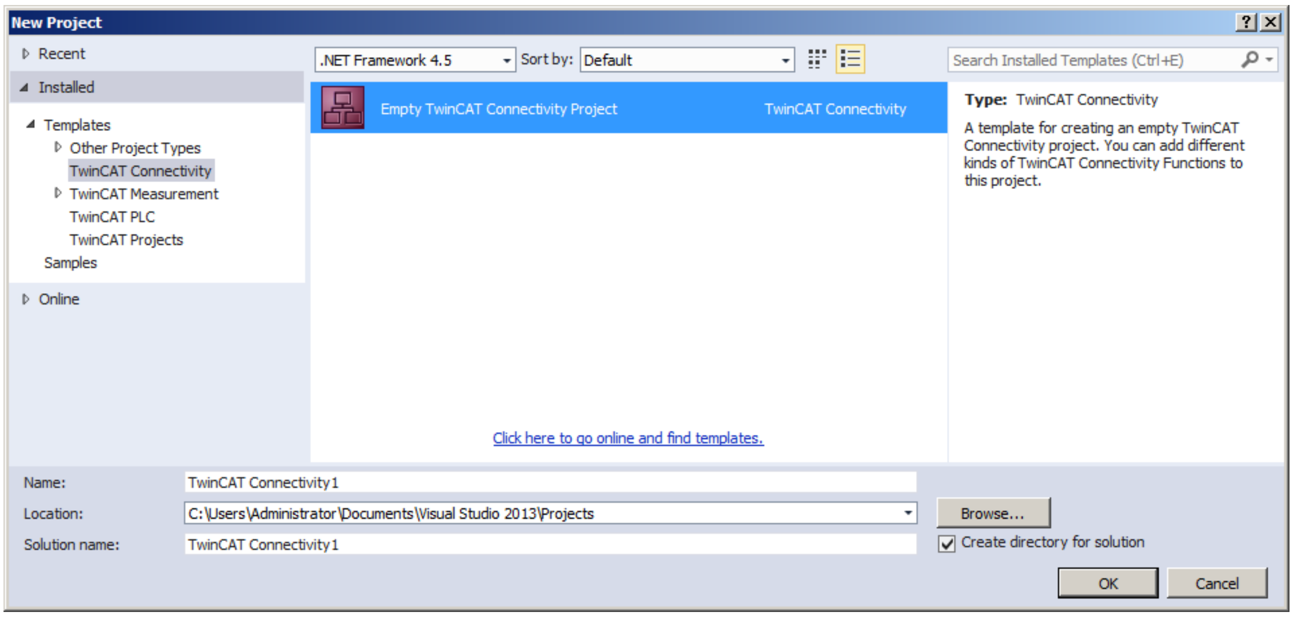
Requirements

Products	Setup versions	Target platform
TF6100	4.x.x	IPC or CX (x86, x64, ARM)

4.2.1.2 Creating a new project

The project package of the OPC UA Configurator integrates itself in the so-called connectivity package. You can select this when creating a new Visual Studio project.

Project template "TwinCAT Connectivity Project":



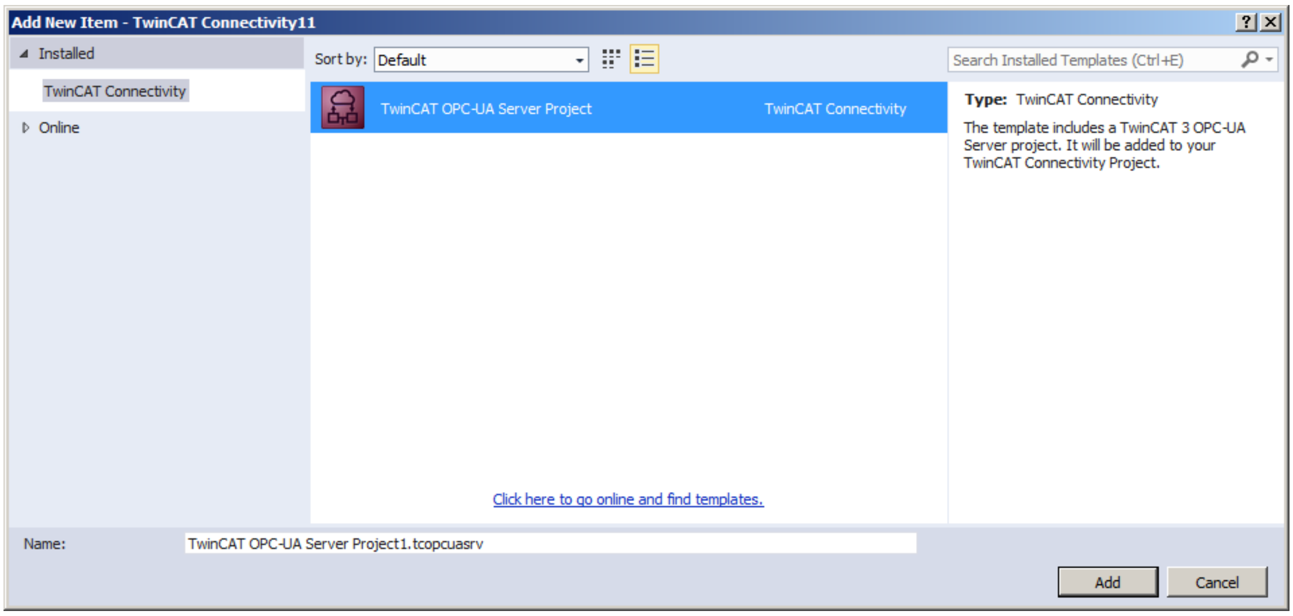
Type: TwinCAT Connectivity

A template for creating an empty TwinCAT Connectivity project. You can add different kinds of TwinCAT Connectivity Functions to this project.

Name:	TwinCAT Connectivity1
Location:	C:\Users\Administrator\Documents\Visual Studio 2013\Projects
Solution name:	TwinCAT Connectivity1

☑ Create directory for solution

Project template "TwinCAT OPC-UA Server Project":


Add New Item - TwinCAT Connectivity11

Installed

TwinCAT Connectivity

Online

Sort by: Default

Search Installed Templates (Ctrl+E)

TwinCAT OPC-UA Server Project TwinCAT Connectivity

Type: TwinCAT Connectivity

The template includes a TwinCAT 3 OPC-UA Server project. It will be added to your TwinCAT Connectivity Project.

Click here to go online and find templates.

Name: TwinCAT OPC-UA Server Project1.tcopcuasrv

Add Cancel

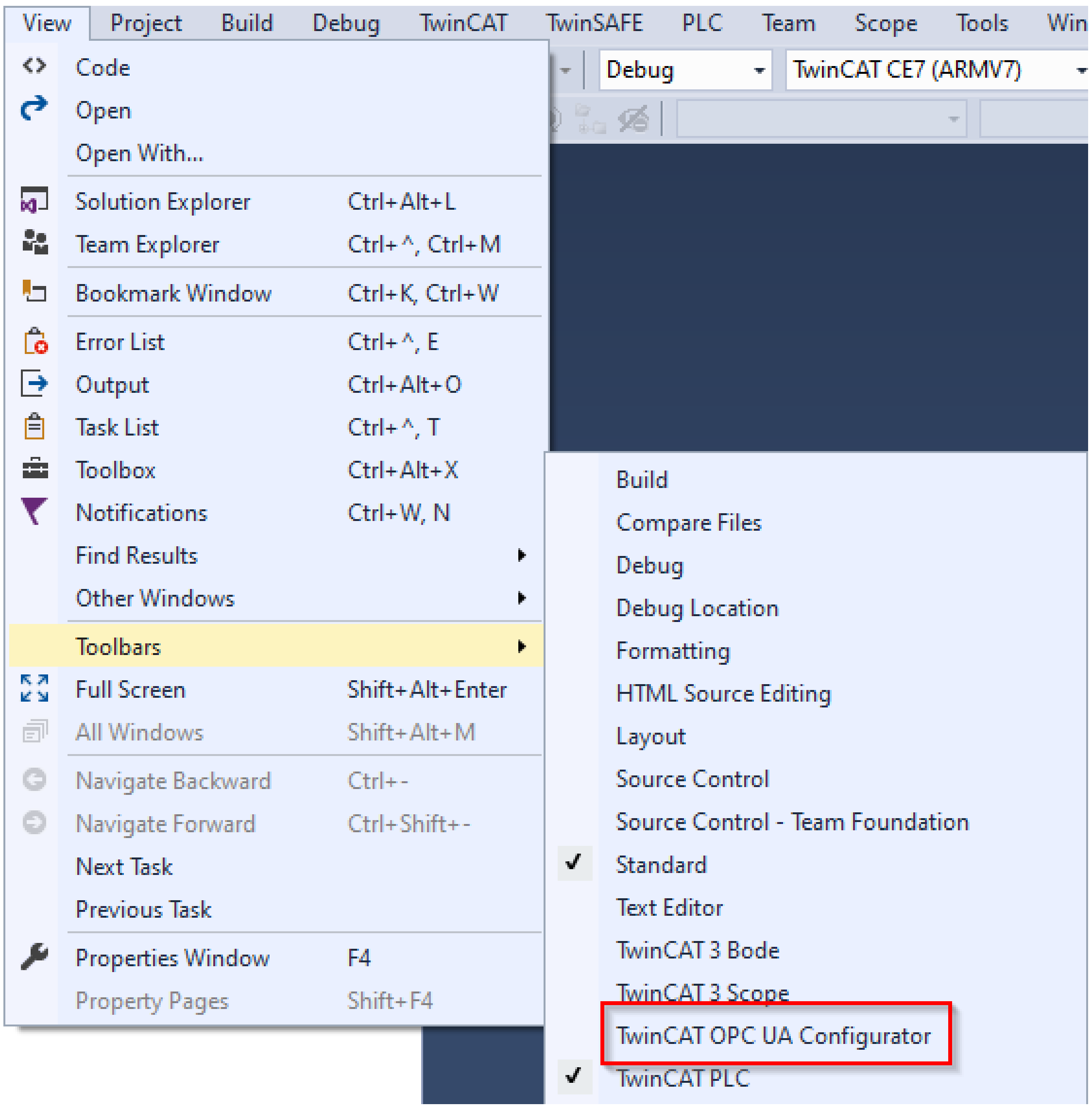
Requirements

Products	Setup versions	Target platform
TF6100	4.x.x	IPC or CX (x86, x64, ARM)

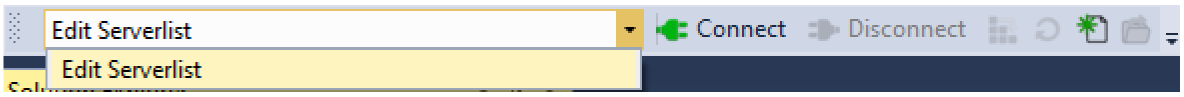
4.2.1.3 Connecting to a server

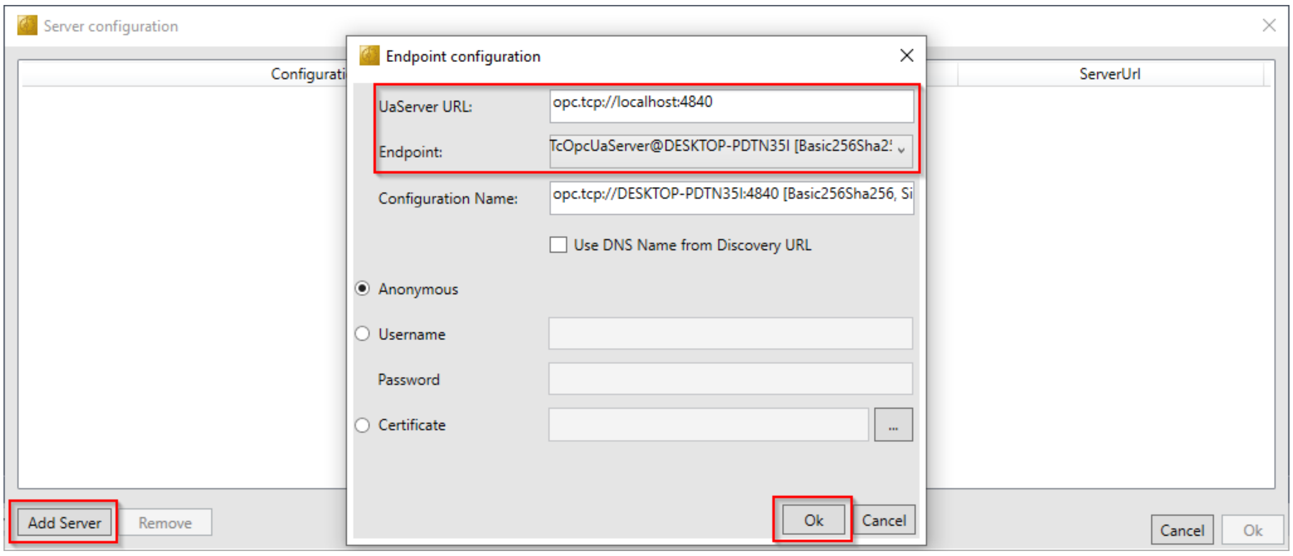
The OPC UA Configurator enables the complete parameterization of the Server via OPC UA. Similar to the TwinCAT XAE system, you can select an OPC UA Server to connect to via the toolbar.

To do this, first add the appropriate toolbar to your Visual Studio interface.



You can then add one or more server connections via the entry **Edit Serverlist** in the DropDownBox of the toolbar.





UaServer URL: opc.tcp://localhost:4840

Endpoint: TcOpcUaServer@DESKTOP-PDTN35I [Basic256Sha2!

Configuration Name: opc.tcp://DESKTOP-PDTN35I:4840 [Basic256Sha256, Si

☐ Use DNS Name from Discovery URL

● Anonymous

○ Username

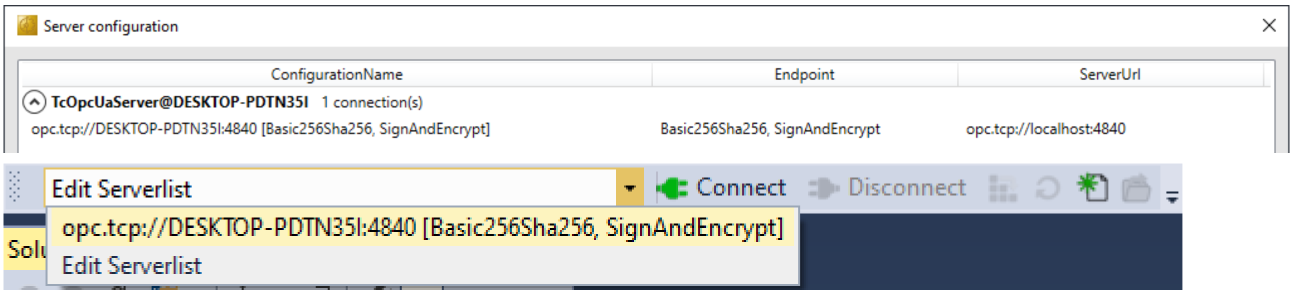
Password

○ Certificate ...

Ok Cancel

In the dialog **Endpoint configuration** you make all settings for the connection with the server, especially the server URL, the selection of an endpoint offered by the server and optionally also the IdentityToken (e.g. username/password) with which the configurator should connect to the server.

The server connection is then added to the server list under an automatically generated configuration name and can then be selected in the drop-down list of the toolbar.


Server configuration

ConfigurationName	Endpoint	ServerUrl
TcOpcUaServer@DESKTOP-PDTN35I 1 connection(s)		
opc.tcp://DESKTOP-PDTN35I:4840 [Basic256Sha256, SignAndEncrypt]	Basic256Sha256, SignAndEncrypt	opc.tcp://localhost:4840

Edit Serverlist
Connect
Disconnect
opc.tcp://DESKTOP-PDTN35I:4840 [Basic256Sha256, SignAndEncrypt]
Edit Serverlist

By clicking on the **Connect** button, a connection to the server can now be established and the server configured.

Online configuration

All settings that you make in your project are carried out for the connected TwinCAT OPC UA Server.

Initialization of the server

If the server is still in the (uninitialized) delivery state, you will receive a corresponding note for server initialization. This process is described in more detail in the article on performing the server initialization [▶ 120].

Requirements

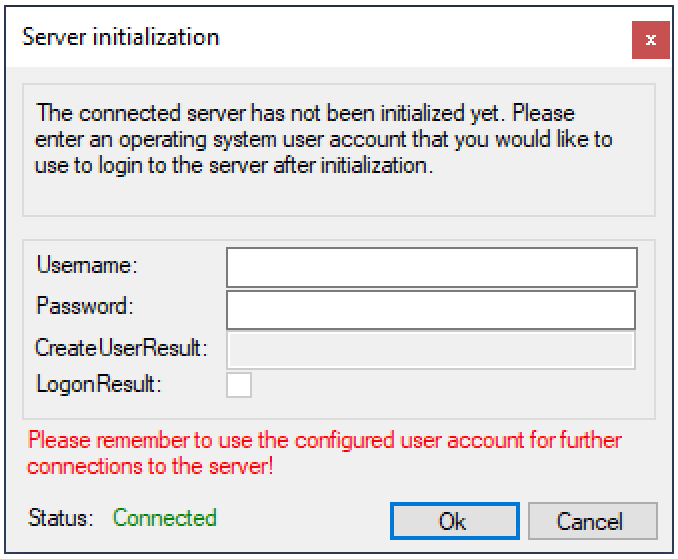
Products	Setup versions	Target platform
TF6100	4.x.x	IPC or CX (x86, x64, ARM)

Also see about this

- Namespace for configuration of the server [▶ 110]

4.2.1.4 Performing the server initialization

The TwinCAT OPC UA Server is delivered in an uninitialized mode, which is based on the so-called TOFU (Trust-On-First-Use) principle. Detailed information about this server feature and the corresponding background information can be found here [▶ 37]. The TwinCAT OPC UA Configurator enables the initialization of the server during the first connection establishment. A corresponding warning message indicates the uninitialized server and enables an appropriate initialization.



Server initialization

The connected server has not been initialized yet. Please enter an operating system user account that you would like to use to login to the server after initialization.

Username:	
Password:	
CreateUserResult:	
LogonResult:	

Please remember to use the configured user account for further connections to the server!

Status: Connected

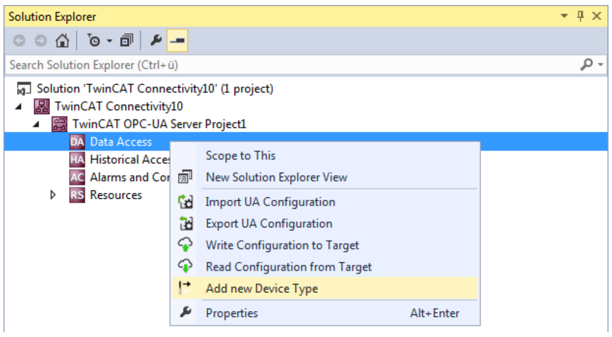
Also see about this

- Initialization [37]

4.2.1.5 Adding ADS devices

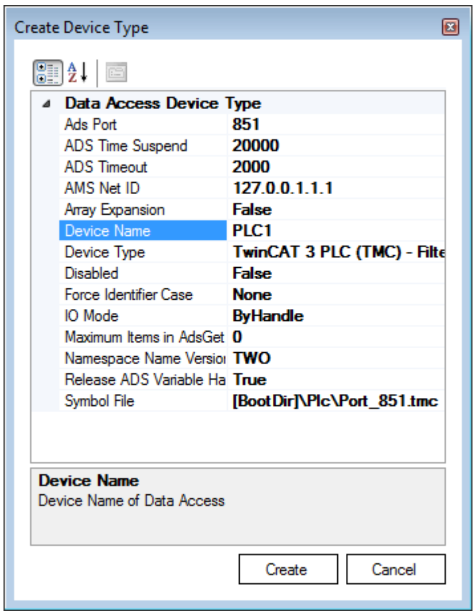
The OPC UA Server can "talk" to one of more ADS devices. To establish a connection, a route to the respective ADS device is required. In the OPC UA Configurator, ADS devices are created, configured and thus announced to the OPC UA Server in the **Data Access** facet.

New ADS devices are added to the configuration via the context menu command **Add new Device Type**.



Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TwinCAT Connectivity10' (1 project)
TwinCAT Connectivity10
TwinCAT OPC-UA Server Project1
Data Access
Historical Acce
Alarms and Co
Resources
Scope to This
New Solution Explorer View
Import UA Configuration
Export UA Configuration
Write Configuration to Target
Read Configuration from Target
Add new Device Type
Properties Alt+Enter

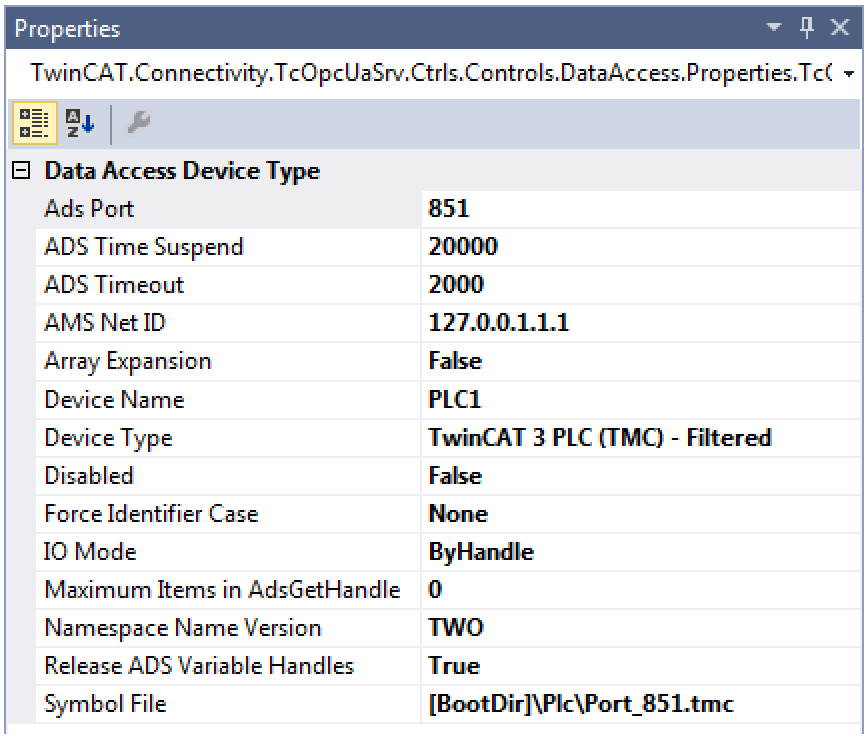
When the command is executed, a dialog box opens in which connection parameters can be configured for this device, e.g. AMS Net ID, ADS port or the symbol file.


Create Device Type

Data Access Device Type	
Ads Port	851
ADS Time Suspend	20000
ADS Timeout	2000
AMS Net ID	127.0.0.1.1.1
Array Expansion	False
Device Name	PLC1
Device Type	TwinCAT 3 PLC (TMC) - Filte
Disabled	False
Force Identifier Case	None
IO Mode	ByHandle
Maximum Items in AdsGet	0
Namespace Name Versio	TWO
Release ADS Variable Ha	True
Symbol File	[BootDir]\Plc\Port_851.tmc

Device Name
Device Name of Data Access

You can subsequently modify the connection parameters if necessary via the Properties window in Visual Studio.

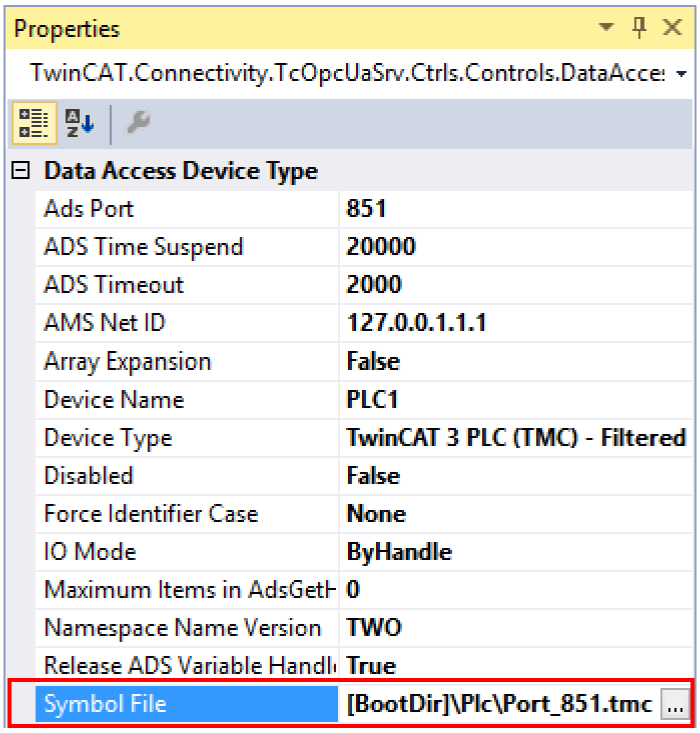

Properties

TwinCAT.Connectivity.TcOpcUaSrv.Ctrls.Controls.DataAccess.Properties.Tc(

Data Access Device Type	
Ads Port	851
ADS Time Suspend	20000
ADS Timeout	2000
AMS Net ID	127.0.0.1.1.1
Array Expansion	False
Device Name	PLC1
Device Type	TwinCAT 3 PLC (TMC) - Filtered
Disabled	False
Force Identifier Case	None
IO Mode	ByHandle
Maximum Items in AdsGetHandle	0
Namespace Name Version	TWO
Release ADS Variable Handles	True
Symbol File	[BootDir]\Plc\Port_851.tmc

Selecting the symbol file

Symbol files that are present on the selected target device can be imported directly. These symbol files can be stored either in the TwinCAT boot directory or in the symbol directory of the OPC UA Server. You can select the files via the corresponding dialog during the symbol file configuration.



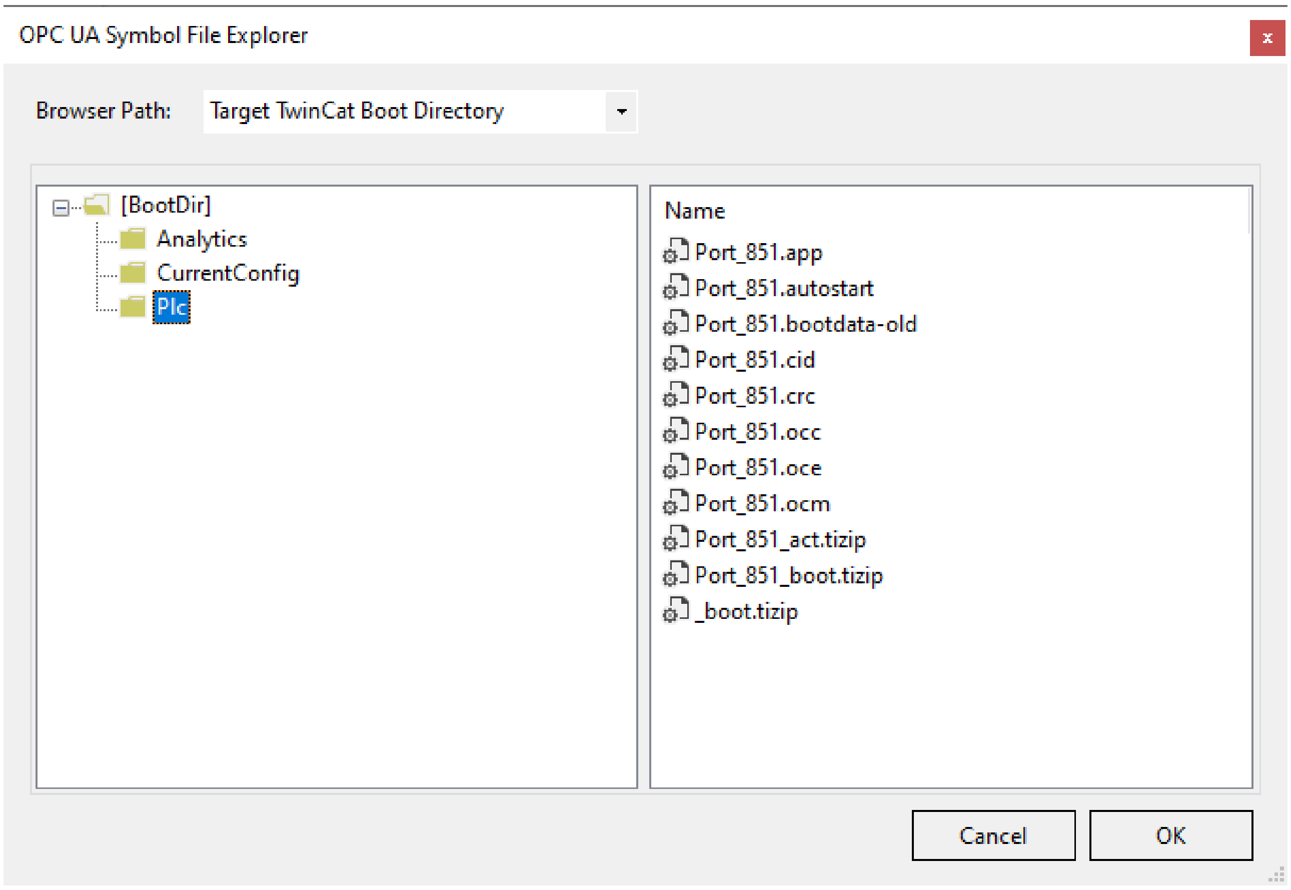
Properties

TwinCAT.Connectivity.TcOpcUaSrv.Ctrls.Controls.DataAcce:

Data Access Device Type

Ads Port	851
ADS Time Suspend	20000
ADS Timeout	2000
AMS Net ID	127.0.0.1.1.1
Array Expansion	False
Device Name	PLC1
Device Type	TwinCAT 3 PLC (TMC) - Filtered
Disabled	False
Force Identifier Case	None
IO Mode	ByHandle
Maximum Items in AdsGetH	0
Namespace Name Version	TWO
Release ADS Variable Handl	True
Symbol File	[BootDir]\Plc\Port_851.tmc

The TwinCAT OPC UA File Explorer can be connected to either the local TwinCAT directory or the remote boot directory. The latter can be read in via the configuration namespace of the server (see Namespace for configuration of the server [▶ 110]).



OPC UA Symbol File Explorer

Browser Path: Target TwinCat Boot Directory

[BootDir]
- Analytics
- CurrentConfig
- Plc

Name
- Port_851.app
- Port_851.autostart
- Port_851.bootdata-old
- Port_851.cid
- Port_851.crc
- Port_851.occ
- Port_851.oce
- Port_851.ocm
- Port_851_act.tizip
- Port_851_boot.tizip
- _boot.tizip

Requirements

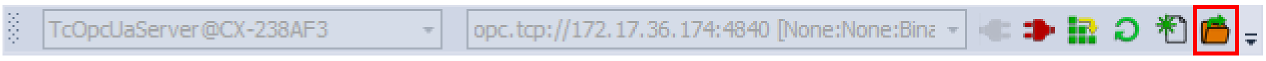
Products	Setup versions	Target platform
TF6100	4.x.x	IPC or CX (x86, x64, ARM)

4.2.1.6 Reading and writing the configuration

Via the configurator you can initiate the download/upload of complete server configurations as well as loading every single facet (data access, historical access, etc.) individually to the target device and opening it there. The functions necessary for this are integrated both in the toolbar and in the context menu of the respective facet.

Opening a configuration from the target device

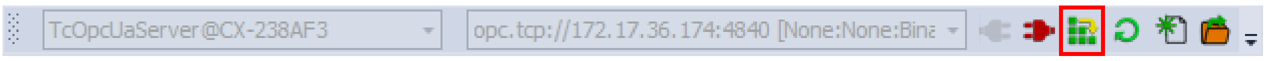
You can open the configuration of the selected target device via the corresponding button in the toolbar.



See also: Connecting to a server [▸ 118]

Activating the configuration on a target device

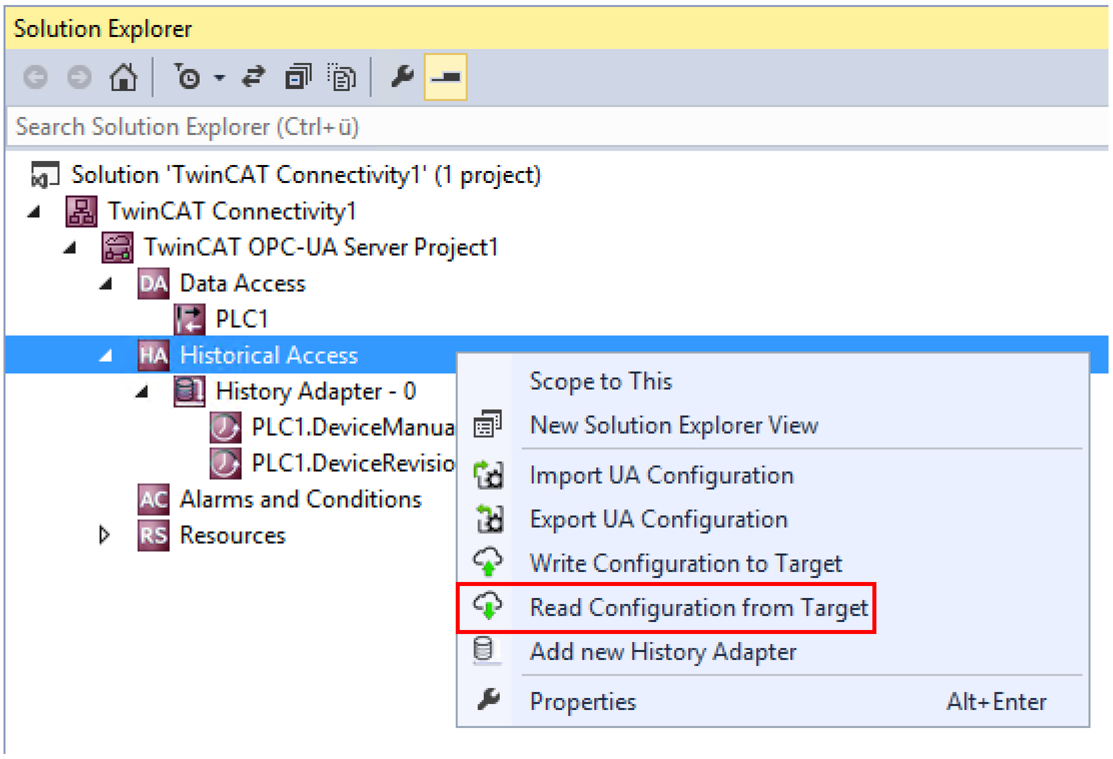
You can download the currently opened configuration to the selected target device using the corresponding button in the toolbar.



See also: Connecting to a server [▸ 118]

Opening a partial configuration

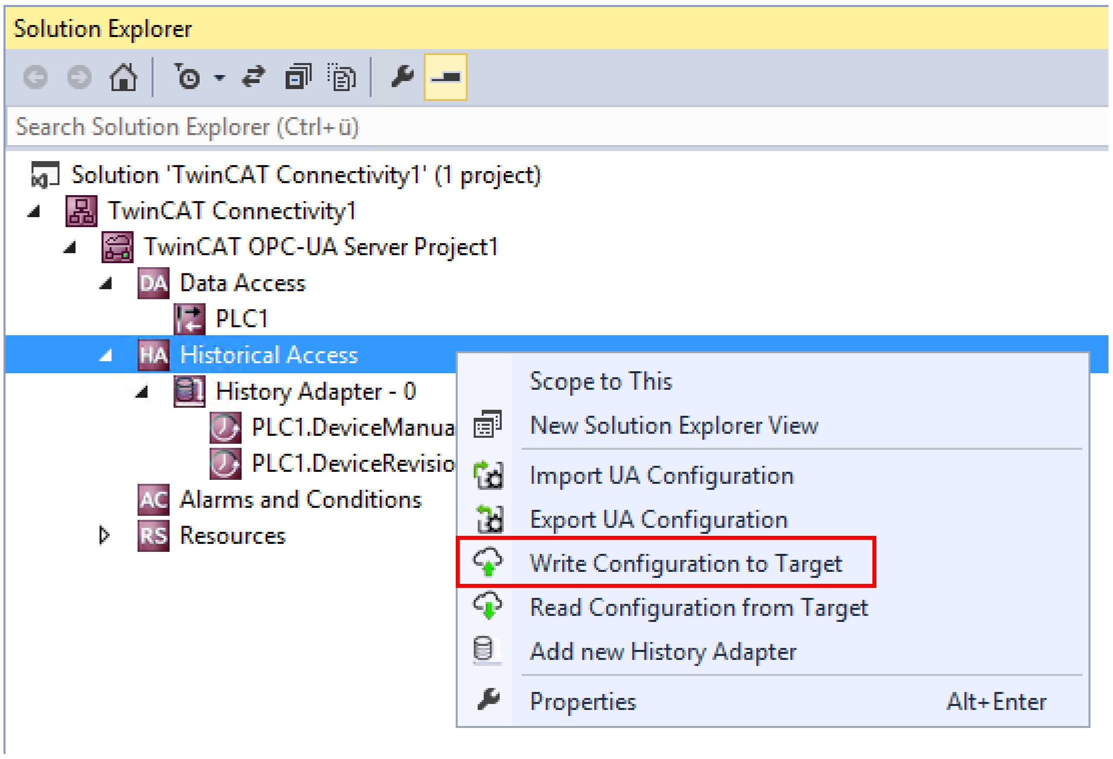
You can open the partial configuration of the selected target device using the command **Read Configuration from Target** in the context menu of a certain facet of the configuration.



See also: Connecting to a server [▸ 118]

Downloading a partial configuration

You can download the partial configuration to the selected target device using the command **Write Configuration to Target** in the context menu of a certain facet of the configuration.



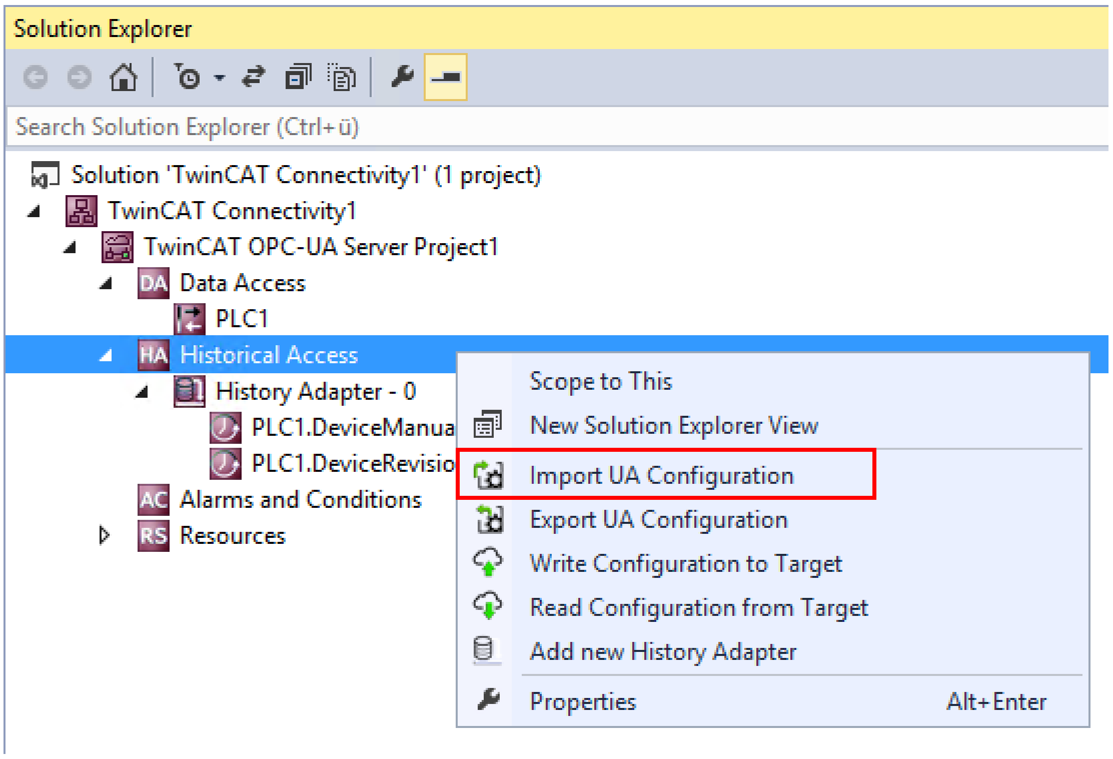
See also: Connecting to a server [▶ 118]

4.2.1.7 Importing and exporting configuration files

The context menu commands enable the import/export of configuration files of the OPC UA Server.

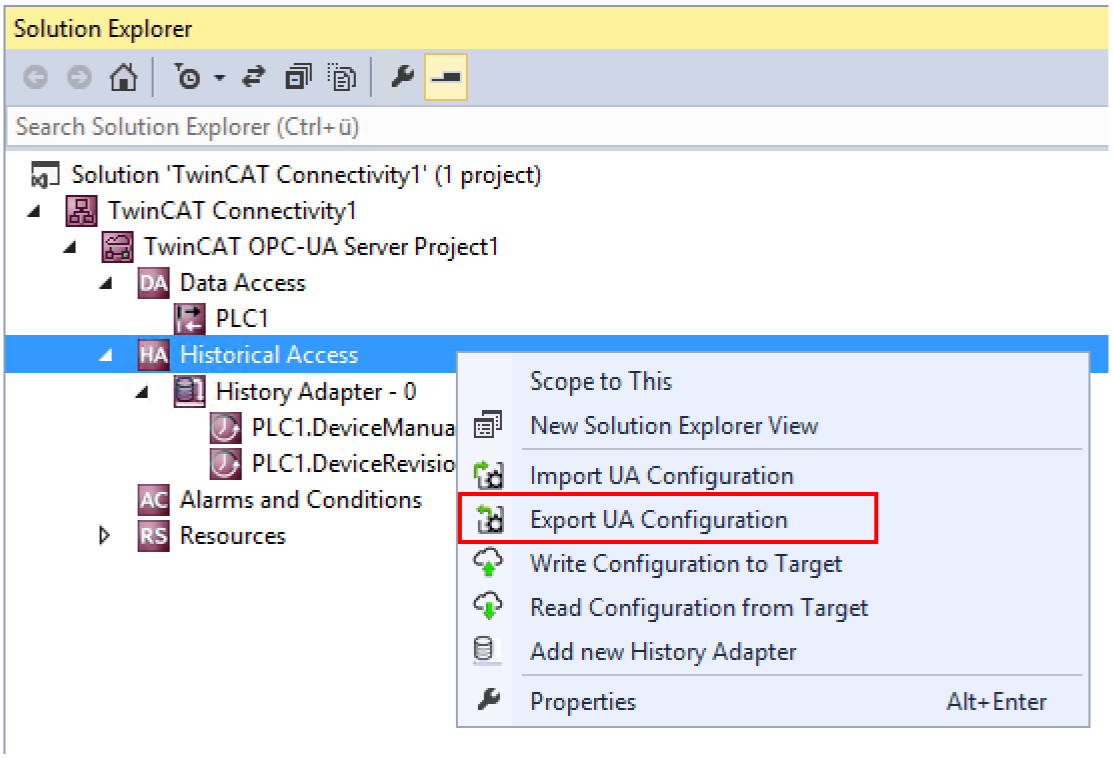
Importing a partial configuration

You can import the partial configuration (e.g. historical access) from an XML configuration file using the command **Import UA Configuration** in the context menu of a certain facet of the configuration.



Exporting a partial configuration

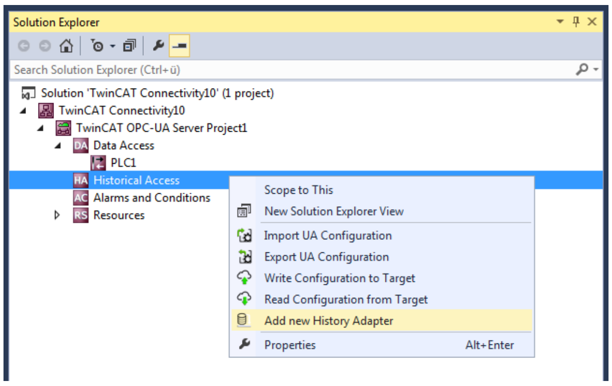
You can export the partial configuration (e.g. historical access) to an XML configuration file using the command **Export UA Configuration** in the context menu of a certain facet of the configuration.



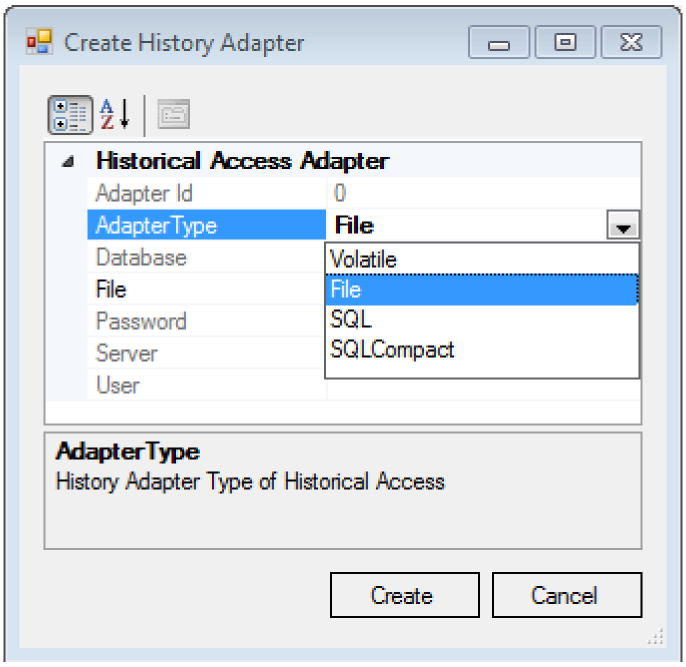
4.2.1.8 Configuring historical access

To configure Historical Access, you must first set up the History Adapters. These are the different locations for storing historical data, such as RAM, file, SQL Server.

History adapters are added to the configuration using the context menu command **Add new History Adapter**.



Depending on the adapter type you have to specify further parameters, e.g. the desired file storage path or the access data for the SQL Server.



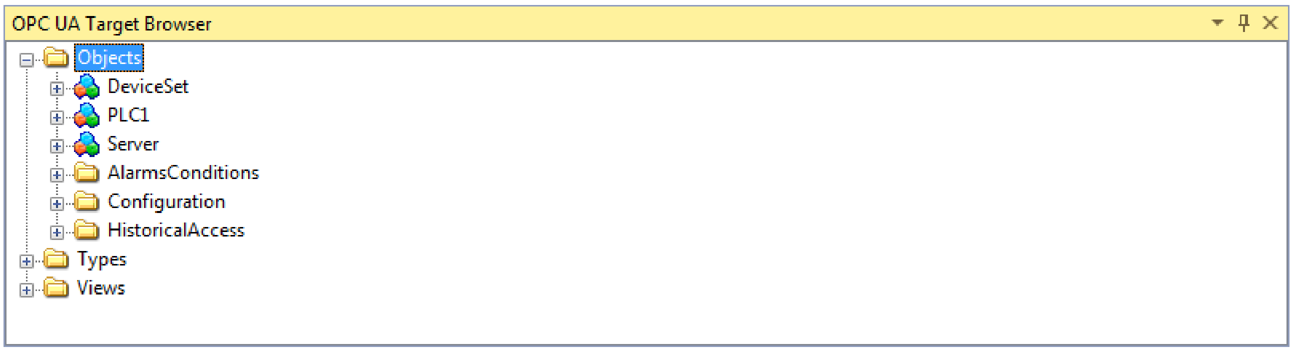
Historical Access Adapter

Adapter Id	0
AdapterType	File
Database	
File	
Password	
Server	
User	

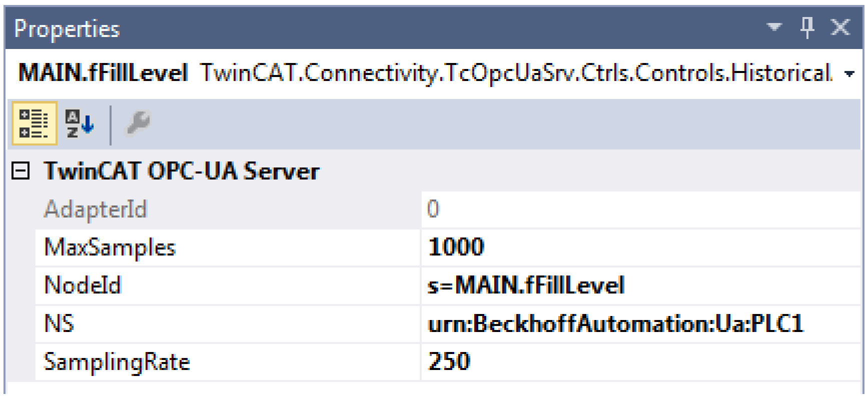
Volatile
File
SQL
SQLCompact

AdapterType
History Adapter Type of Historical Access

After you have created a history adapter you can add the desired variables to the adapter. These variables must already exist on the selected OPC UA Server when the engineering is implemented. You can use the integrated **OPC UA Target Browser** to select the variables and then add the variables from the target browser to the history adapter by drag & drop.



Additional parameters can be specified in the properties window of the newly added variable, e.g. the desired SamplingRate or the size of the ring buffer to be used in the History Adapter.



MAIN.fFillLevel TwinCAT.Connectivity.TcOpcUaSrv.Ctrls.Controls.Historical

TwinCAT OPC-UA Server

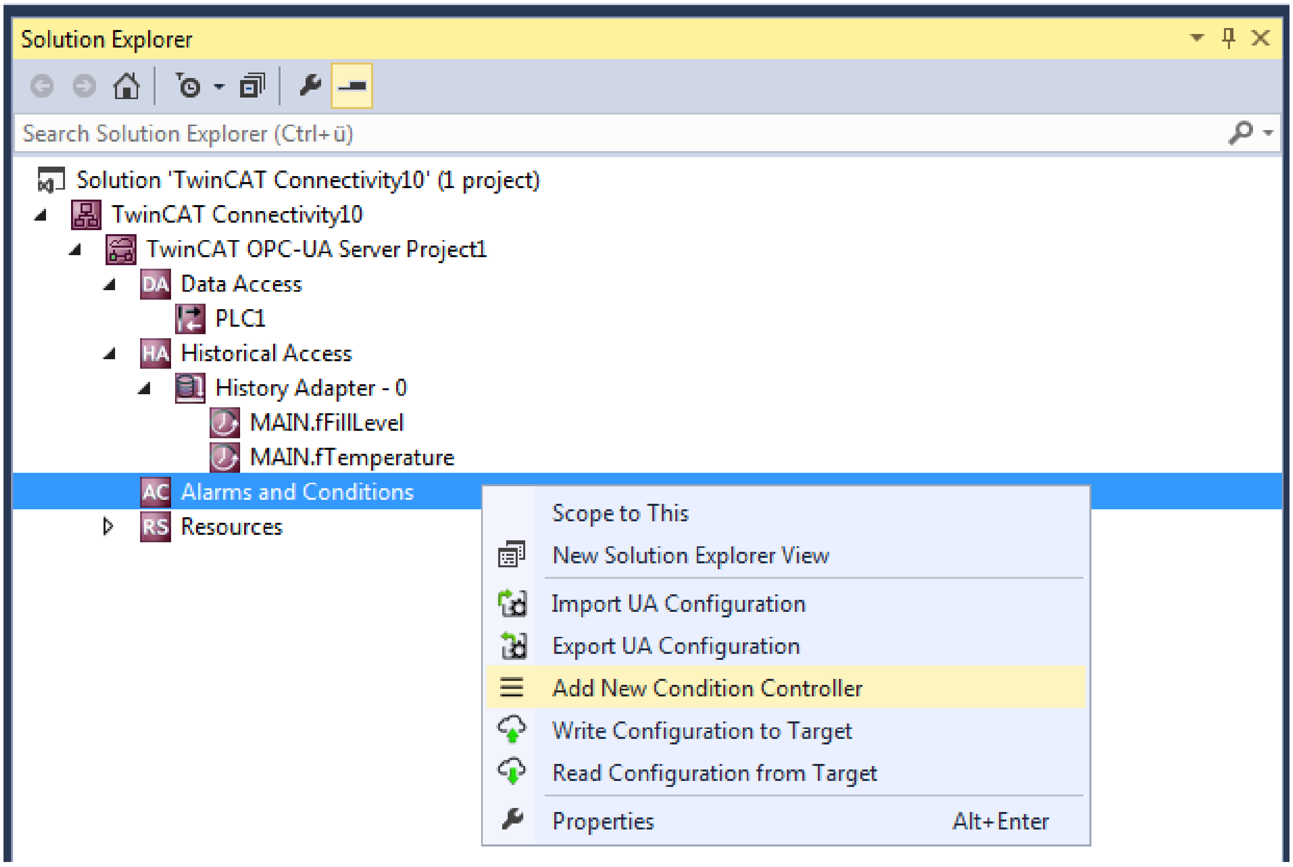
AdapterId	0
MaxSamples	1000
NodeId	s=MAIN.fFillLevel
NS	urn:BeckhoffAutomation:Ua:PLC1
SamplingRate	250

See also: Connecting to a server [▶ 118]

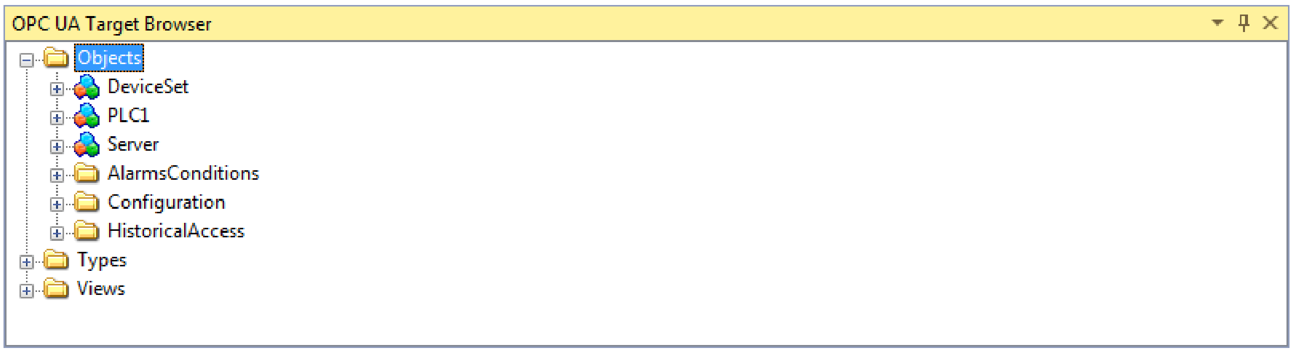
4.2.1.9 Configuring Alarms and Conditions

In order to configure Alarms and Conditions (A&C) you must first set up the Condition Controllers. These are container units that group together alarms.

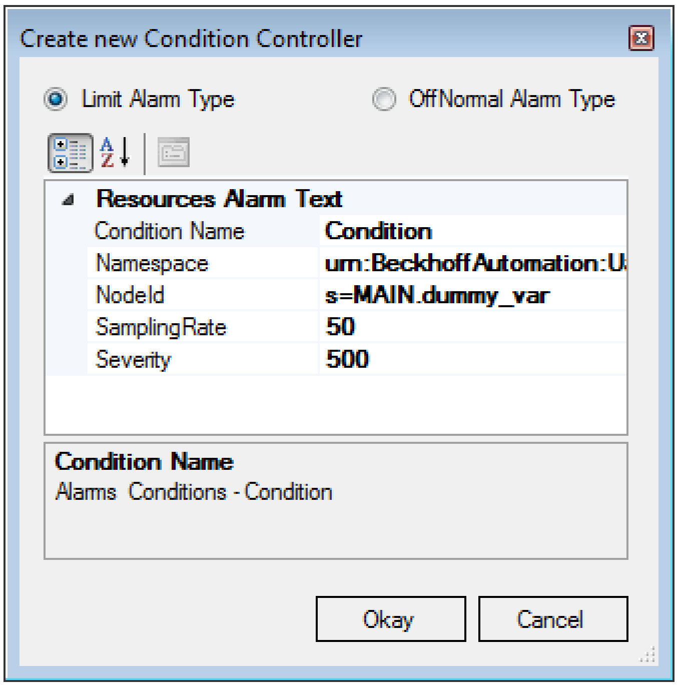
Condition Controllers are added to the configuration using the context menu command **Add New Condition Controller**.



After you have created a Condition Controller, you can add the desired variables to the controller and monitor them in the sense of alarms and conditions. A condition is created for each variable, which specifies the parameters for monitoring. These variables must already exist on the selected OPC UA Server when the engineering is implemented. You can use the integrated **OPC UA Target Browser** to select the variables and then add the variables from the target browser to the Condition Controller by drag & drop.



In the dialog window which then opens you can define the condition type and further parameters for the monitoring, e.g. SamplingRate and Severity.



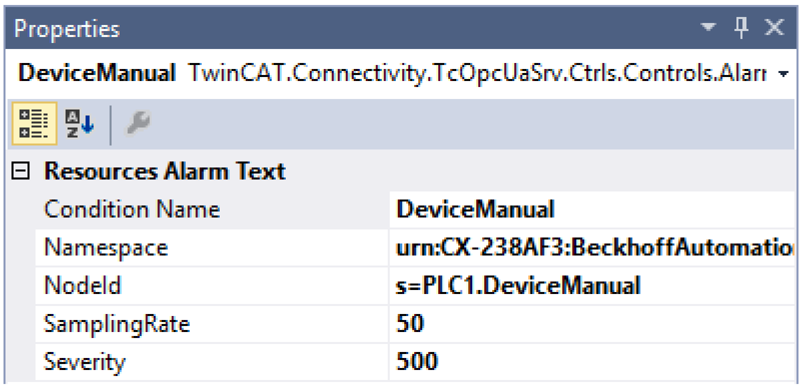
Create new Condition Controller

(● Limit Alarm Type) ○ OffNormal Alarm Type

Resources Alarm Text	
Condition Name	Condition
Namespace	urn:BeckhoffAutomation:U
NodeId	s=MAIN.dummy_var
SamplingRate	50
Severity	500

Condition Name
Alarms Conditions - Condition

Depending on the selected condition type you can specify additional parameters in the properties window of the condition. The threshold values for the respective condition type are displayed as individual entries in the tree view of the configuration. Here too, you can configure the corresponding parameters in the properties window.

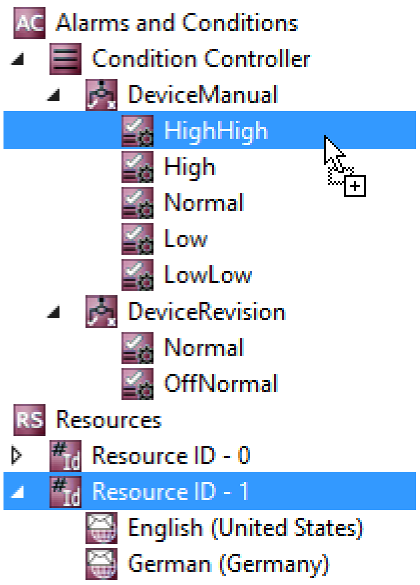


Properties

DeviceManual TwinCAT.Connectivity.TcOpcUaSrv.Ctrls.Controls.Aları

Resources Alarm Text	
Condition Name	DeviceManual
Namespace	urn:CX-238AF3:BeckhoffAutomatio
NodeId	s=PLC1.DeviceManual
SamplingRate	50
Severity	500

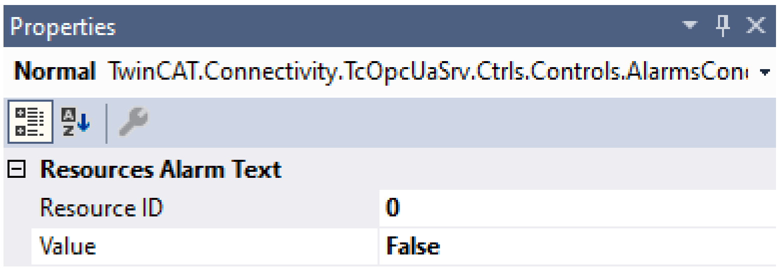
Subsequently you have to define the alarm texts that are to be sent to the OPC UA Client when a condition is triggered. The section Configuring alarm texts [▶ 131] describes how alarm texts are created. You can drag and drop the alarm texts onto the respective threshold value of a condition.



Alarm type OffNormal

An OffNormal alarm type is used to define which state of a Boolean variable is evaluated as normal. An alarm is triggered if the variable value deviates from this. The PLC must be used for working with value ranges (e.g., integer or double variables). Depending on the value, a corresponding TRUE or FALSE state is then passed to the OPC UA Server.

State	Value range
Normal	TRUE or FALSE, depending on the user's decision.
OffNormal	TRUE or FALSE, depending on the configuration of the normal state. Cannot be configured by the user.


Properties
Normal TwinCAT.Connectivity.TcOpcUaSrv.Ctrls.Controls.AlarmsCon
Resources Alarm Text

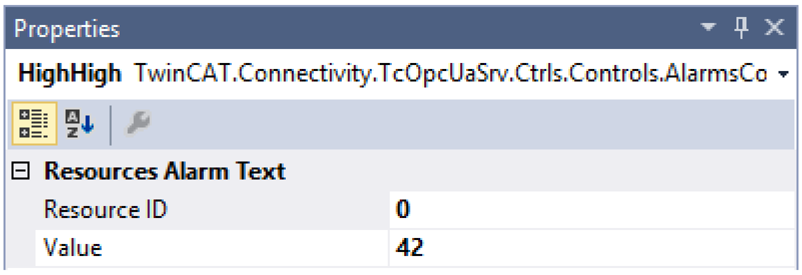
Resource ID	0
Value	False

The first step is to configure the normal state as described above. The user then defines an alarm text for the respective state (OffNormal and Normal) via Resources. This can be done either by drag & drop or by selecting from the **Resource ID** dropdown list.

Alarm type Limit

With an alarm type Limit you define different threshold values upon whose reaching an alarm is to be sent. The following table describes the different threshold values using an example configuration.

State	Example threshold values	Associated value range (INT)
HighHigh	5000	5000-32767
High	2000	2000-4999
Normal	-	1000-1999
Low	1000	500-999
LowLow	500	-32768-499


Properties
HighHigh TwinCAT.Connectivity.TcOpcUaSrv.Ctrls.Controls.AlarmsCo
Resources Alarm Text

Resource ID	0
Value	42

In the first step, the various threshold values are configured as described above. The user then defines an alarm text for the respective state (HighHigh, High, Normal, Low, LowLow) via Resources. This can be done either by drag & drop or by selecting from the **Resource ID** dropdown list.

Requirements

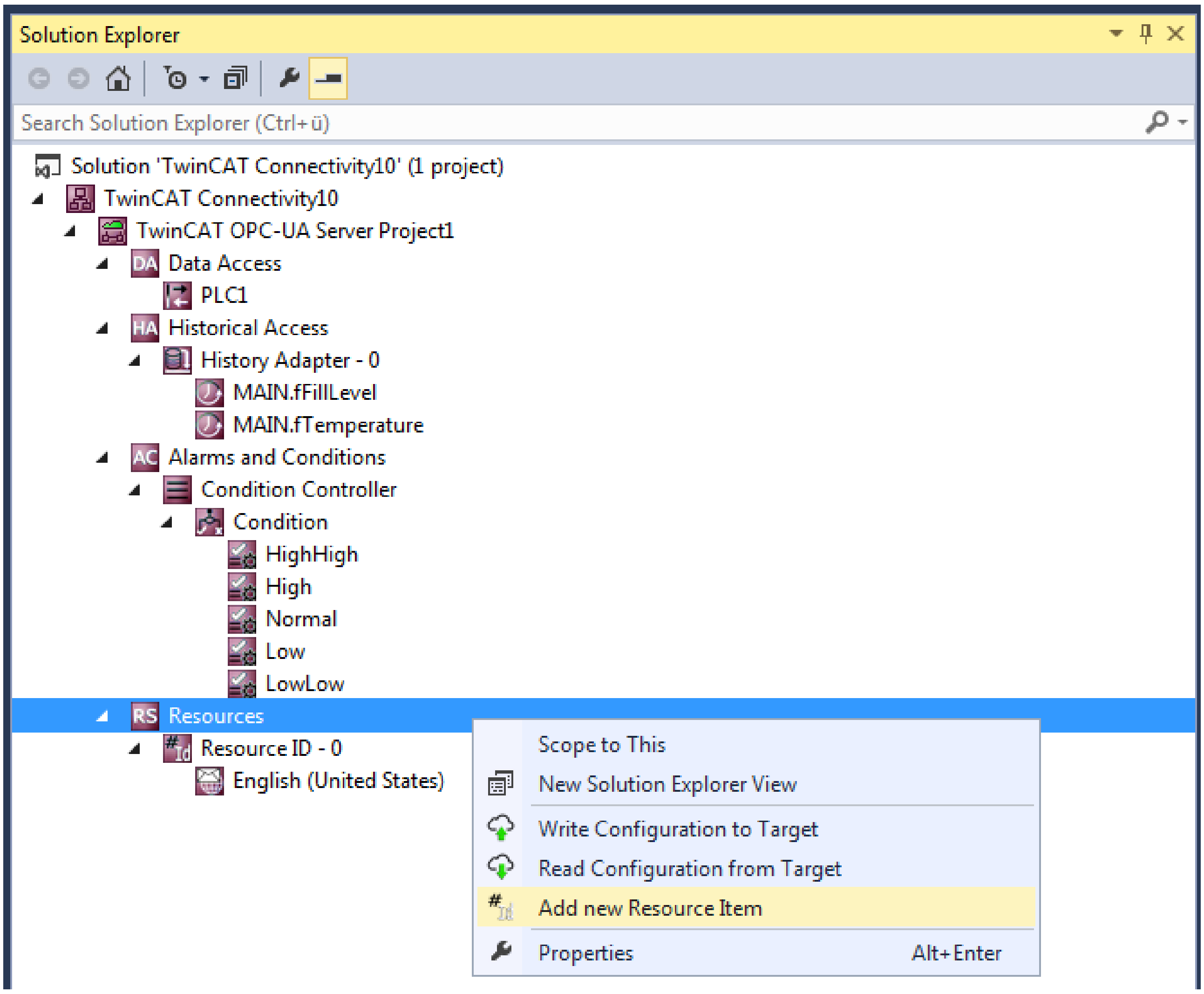
Products	Setup versions	Target platform
TF6100	4.x.x	IPC or CX (x86, x64, ARM)

See also: Connecting to a server [▸ 118]

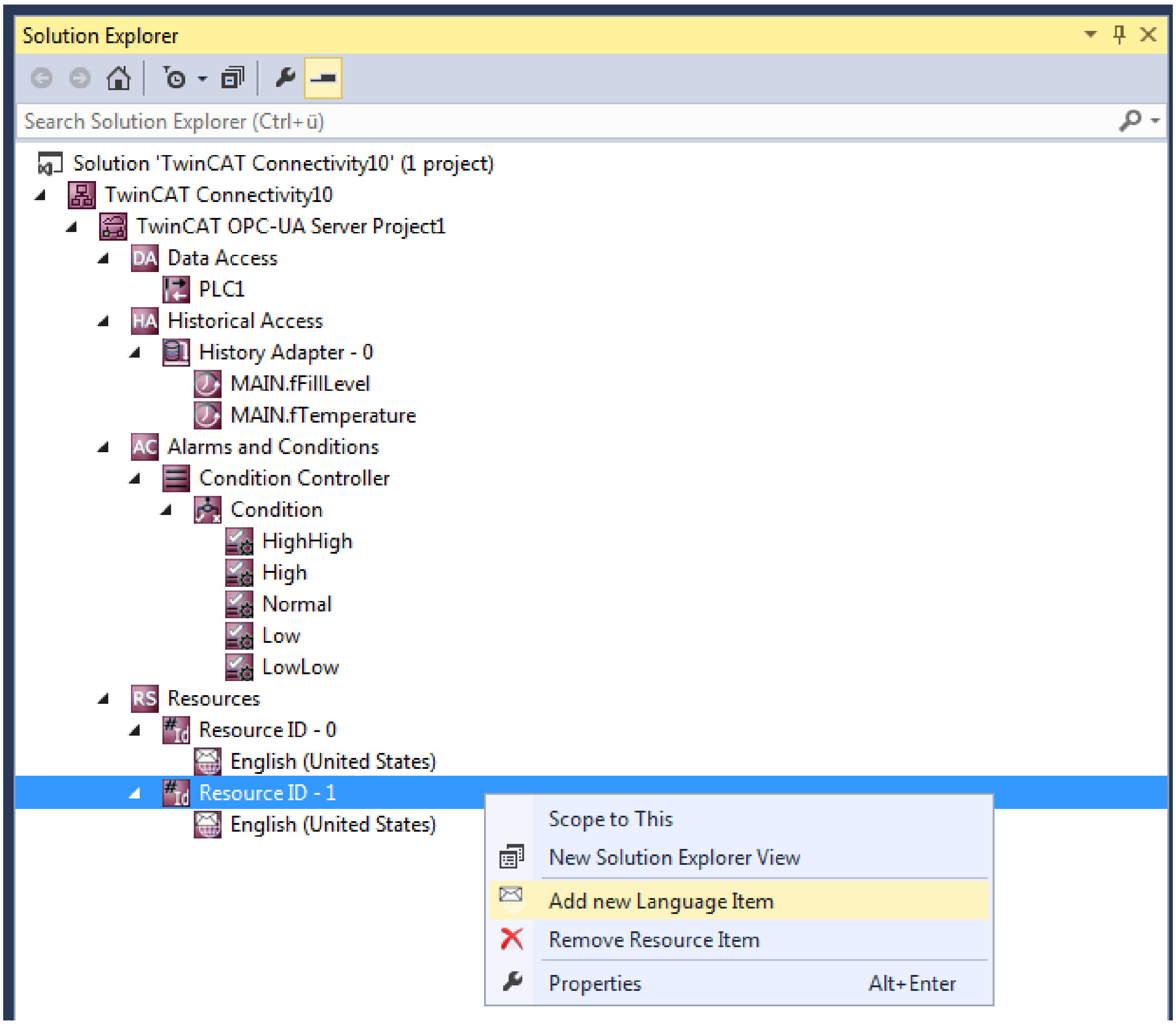
4.2.1.10 Configuring alarm texts

The OPC UA Configurator enables the (multilingual) management of alarm texts that are used, for example, with Alarms and Conditions [▸ 127]. The configuration of the alarm texts takes place in the **Resources** facet. Each alarm text is identified by a unique ID. Multiple language texts can then be assigned to this ID.

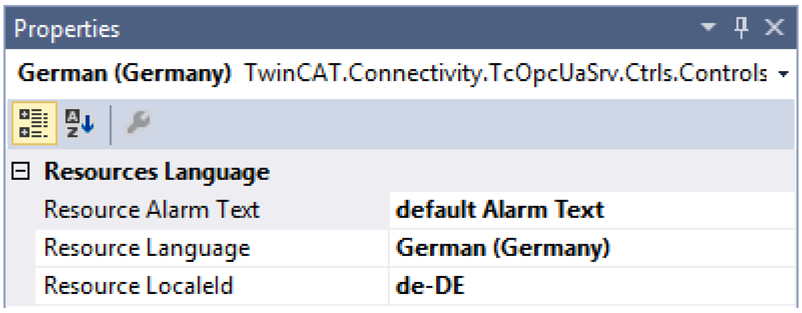
You can create so-called "resource items" using the context menu command **Add new Resource Item**.



You add new language items to a resource item using the command **Add new Language Item** in the resource item's context menu.



You can further parameterize a language item, e.g. the language text and the assigned language, in the properties window. When you define the language the associated LocaleID is automatically set. The LocaleID is requested by the OPC UA Client to indicate in which language it expects alarm texts.


Properties
German (Germany) TwinCAT.Connectivity.TcOpcUaSrv.Ctrls.Controls
Resources Language

Resource Alarm Text	default Alarm Text
Resource Language	German (Germany)
Resource LocaleId	de-DE

Requirements

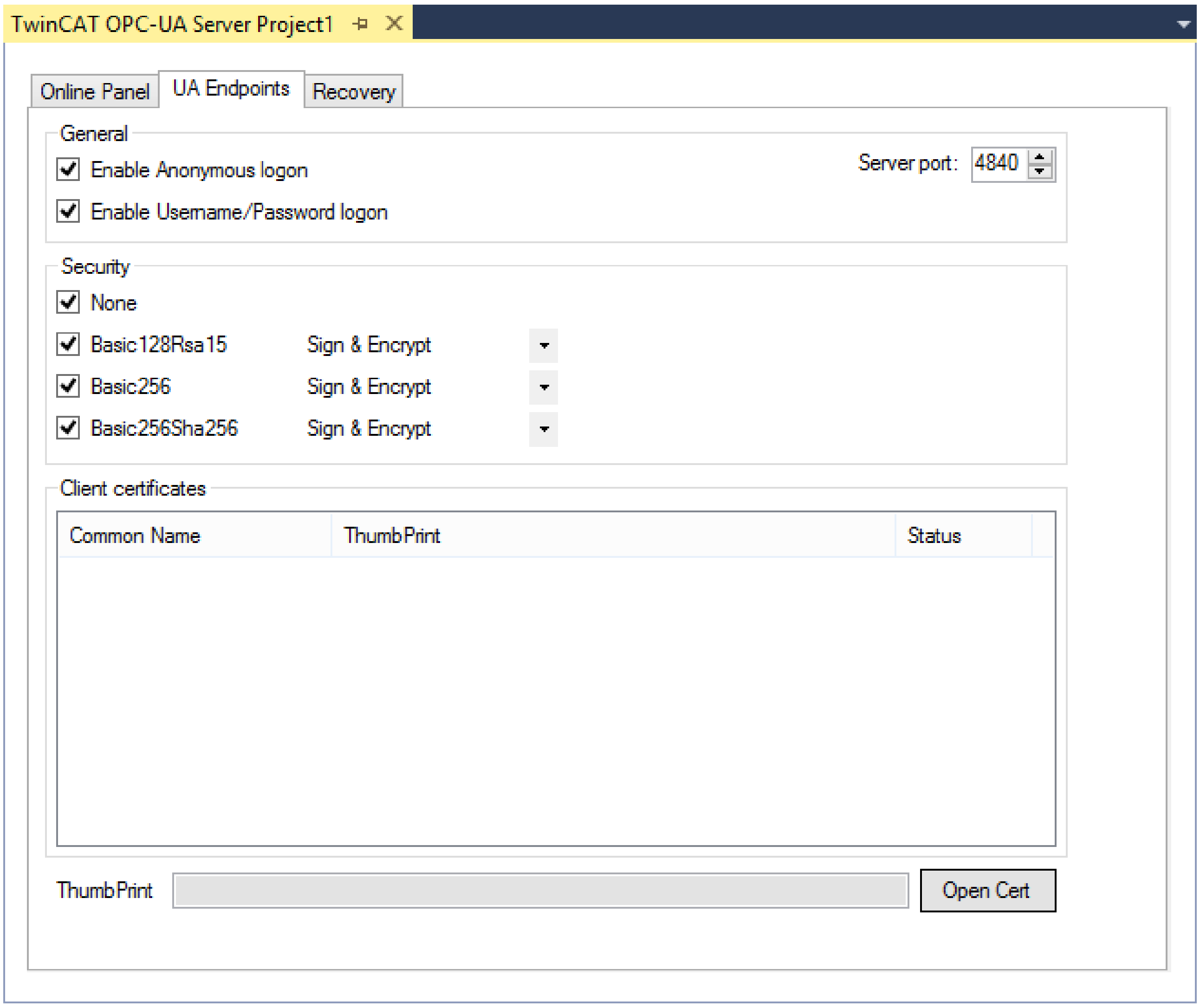
Products	Setup versions	Target platform
TF6100	4.x.x	IPC or CX (x86, x64, ARM)

4.2.1.11 Configuring endpoints

The endpoints of the OPC UA Server indicate which security mechanisms are to be used during the connection establishment of a client. These range from "unencrypted" to "encrypted and signed", based on different key strengths.

The endpoints can be activated and deactivated using the configurator. It may be useful to deactivate the unencrypted endpoint so that all clients can only connect themselves with valid certificates that are classified as trustworthy.

The endpoints are configured directly at the level of the OPC UA Server project. By double-clicking on the project you can make the corresponding settings on the **UA Endpoints** tab. The settings become effective after an activation of the configuration and a subsequent restart of the server (see Reading and writing the configuration [▶ 124] and Restarting the server [▶ 142]).



General
- [✓] Enable Anonymous logon
- [✓] Enable Username/Password logon

Server port: 4840

Security

[✓] None	
[✓] Basic128Rsa15	Sign & Encrypt
[✓] Basic256	Sign & Encrypt
[✓] Basic256Sha256	Sign & Encrypt

Client certificates

Common Name	ThumbPrint	Status

ThumbPrint

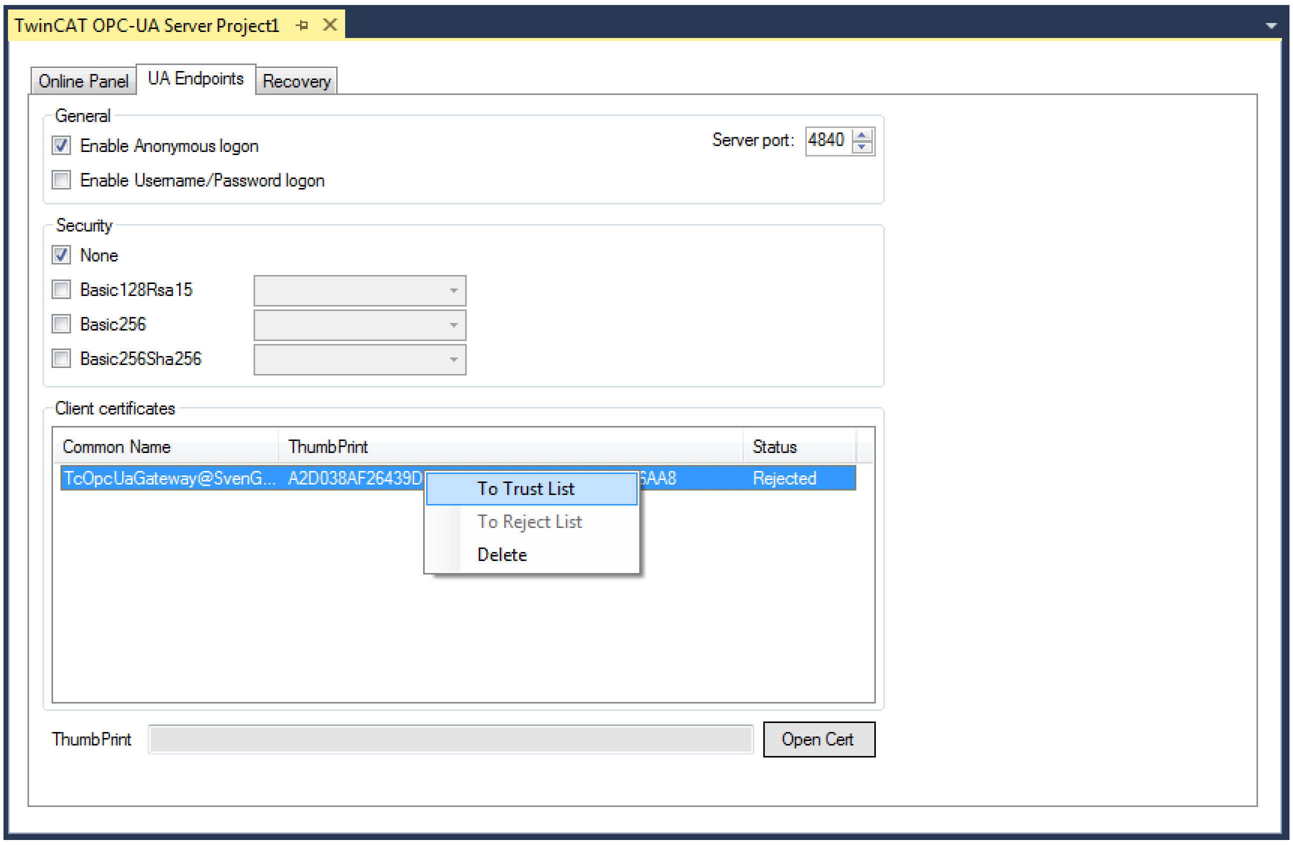
Requirements

Products	Setup versions	Target platform
TF6100	4.x.x	IPC or CX (x86, x64, ARM)

4.2.1.12 Trust relationship for certificates

The Configurator facilitates management of the client certificates on the server. In the project settings you can classify the certificates as trustworthy or refuse them on the **UA Endpoints** tab in the **Client certificates** area.

After an OPC UA Client has attempted to connect to a secure server endpoint for the first time, the client certificate is deposited on the server and declared "rejected". The server administrator can subsequently enable the certificate. A subsequent connection attempt of the client with a secured endpoint will then be successful.



General
- [✓] Enable Anonymous logon
- [ ] Enable Username/Password logon

Server port: 4840

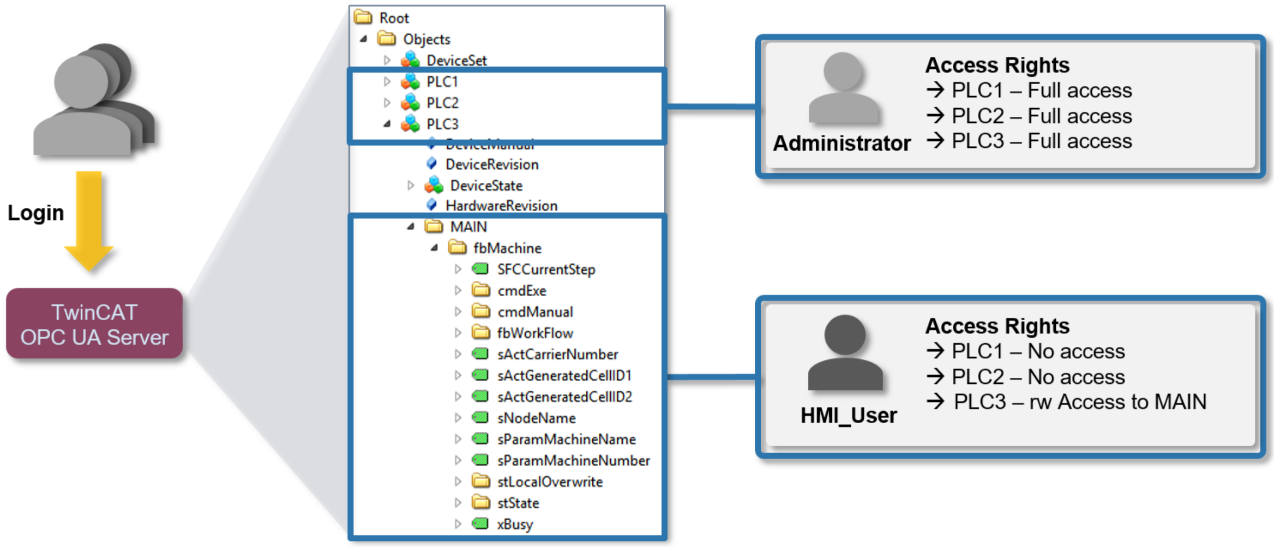
Security
- [✓] None
- [ ] Basic128Rsa15
- [ ] Basic256
- [ ] Basic256Sha256

Client certificates

Common Name	ThumbPrint	Status
TcOpcUaGateway@SvenG...	A2D038AF26439D... 5AA8	Rejected

4.2.1.13 Configuring security settings

The OPC UA Server enables the configuration of permissions at namespace and node level. This allows you to fine-granulate the access to ADS devices (for example, to different PLC runtimes) as well as variables. These security settings are available for all ADS devices that can be displayed in the server namespace.



Configuration

The permissions are configured on the basis of an XML-based configuration file (*TcUaSecurityConfig.xml*), which is located in the same directory as the server. The configuration file consists of the three areas "Users", "Groups" and "AccessInfos".

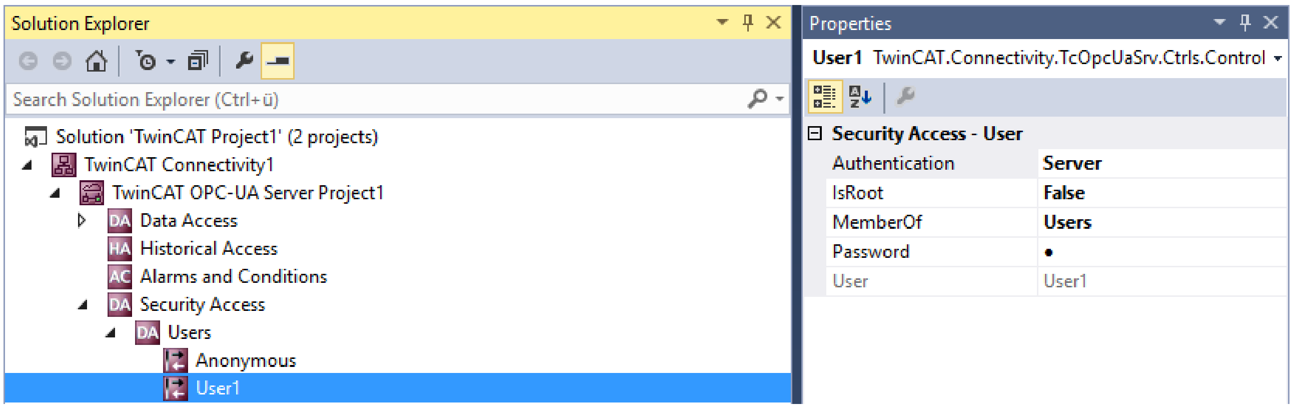
Users

In the "Users" area you can configure user accounts that are to be accepted by the OPC UA Server as logins. There are three different authentication methods:

OS (recommended authentication method)	The mechanisms of the operating system are used to validate user name and password. The user account is subject completely to the control of the operating system and/or domain.
Server (not recommended)	User name and password are known only to the OPC UA Server. Both pieces of information are stored in plain text in the XML file.
None	Only the user name of the server is evaluated, the password is ignored.

Users can be configured with a tag <DefaultAccess> that specifies the standard access of the user to a certain namespace.

Users can be members of one or more groups. You can specify this using the **MemberOf** attribute. In case of memberships of several groups, separate the groups by a semicolon.


Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TwinCAT Project1' (2 projects)
TwinCAT Connectivity1
TwinCAT OPC-UA Server Project1
Data Access
Historical Access
Alarms and Conditions
Security Access
Users
Anonymous
User1
Properties
User1 TwinCAT.Connectivity.TcOpcUaSrv.Ctrls.Control

Security Access - User

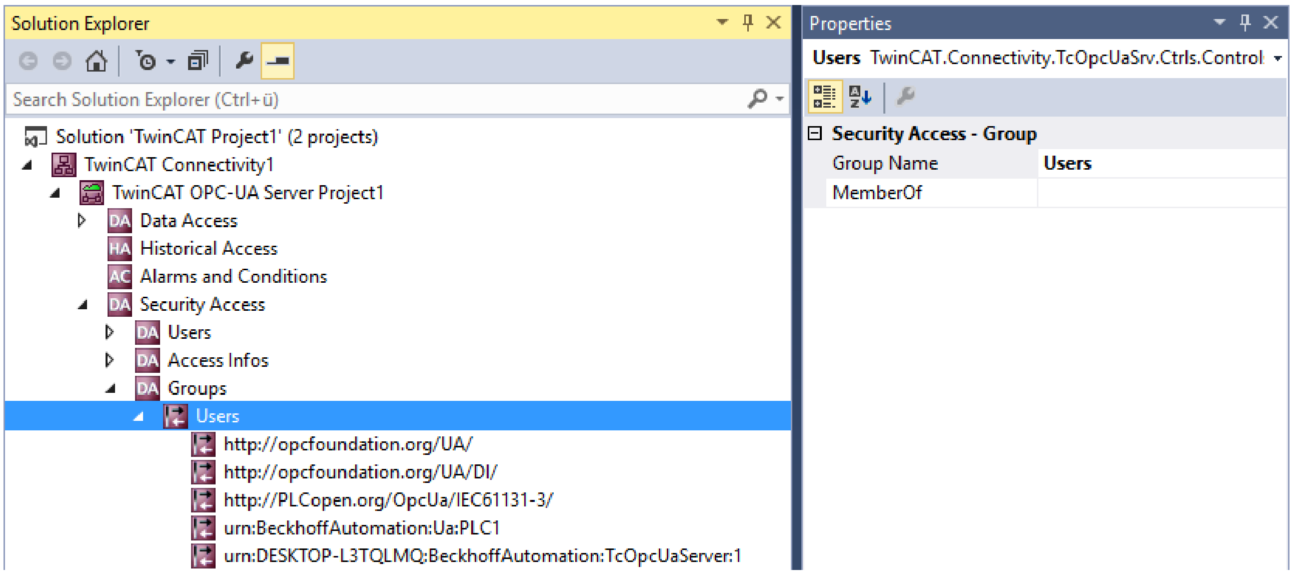
Authentication	Server
IsRoot	False
MemberOf	Users
Password	•
User	User1

Groups

In order to enable a simpler configuration with several user accounts, you can combine the users into groups.

Groups can also be configured with a tag <DefaultAccess>.

You can nest groups using the **MemberOf** attribute. In case of memberships of several groups, separate the groups by a semicolon.


Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TwinCAT Project1' (2 projects)
TwinCAT Connectivity1
TwinCAT OPC-UA Server Project1
Data Access
Historical Access
Alarms and Conditions
Security Access
Users
Access Infos
Groups
Users
http://opcfoundation.org/UA/
http://opcfoundation.org/UA/DI/
http://PLCopen.org/OpcUa/IEC61131-3/
urn:BeckhoffAutomation:Ua:PLC1
urn:DESKTOP-L3TQLMQ:BeckhoffAutomation:TcOpcUaServer:1
Properties
Users TwinCAT.Connectivity.TcOpcUaSrv.Ctrls.Control

Security Access - Group

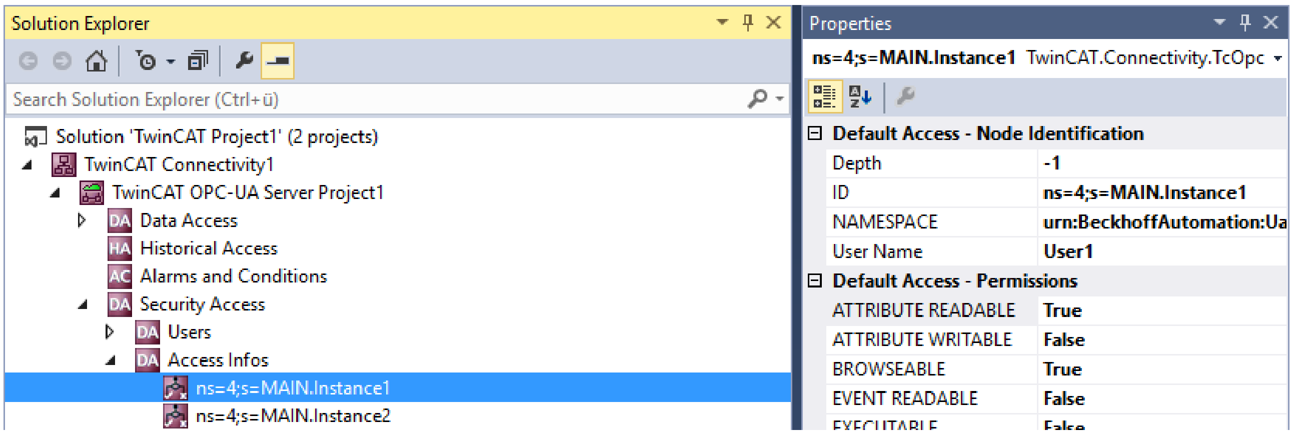
Group Name	Users
MemberOf	

AccessInfos

If a fine-granular setting of permissions at the node level is to be implemented, then AccessInfos can be configured additionally, which specify the access permissions on nodes. Access rights can be passed on to subelements. Although AccessInfos allow the most fine-grained configuration of permissions, such a configuration can quickly become confusing. Therefore, check whether configuring access rights at the namespace level (see above) is not sufficient.

The AccessInfo for a node contains the following settings:

NS	Configures the NamespaceName in which the node is localized
Id	Configures the identifier of the node, including the IdentifierType (e.g. s = String)
Depth	Inheritance depth of permissions (-1 for infinite)
User/Group	User or group that is to be given access to this node, including the AccessLevels

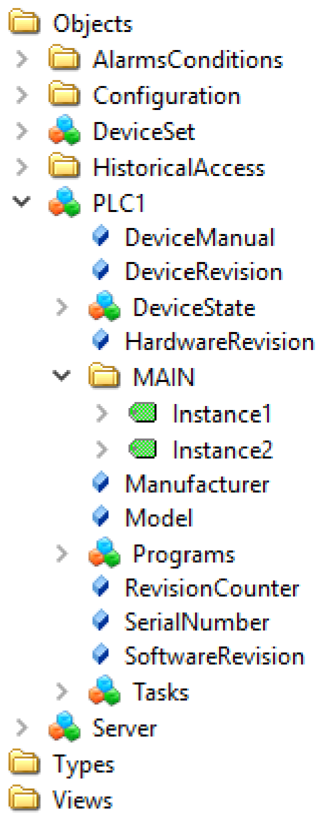


Default Access - Node Identification	
Depth	-1
ID	ns=4;s=MAIN.Instance1
NAMESPACE	urn:BeckhoffAutomation:Ua
User Name	User1
Default Access - Permissions	
ATTRIBUTE READABLE	True
ATTRIBUTE WRITABLE	False
BROWSEABLE	True
EVENT READABLE	False
EXECUTABLE	False

AccessInfos can be configured by dragging & dropping variables from the Target Browser. The configurable permissions are cumulative.

Sample configuration

Let's take the following simple control program. The variables are already published in the OPC UA namespace of the server. The OPC UA Server is initially in the delivery state.



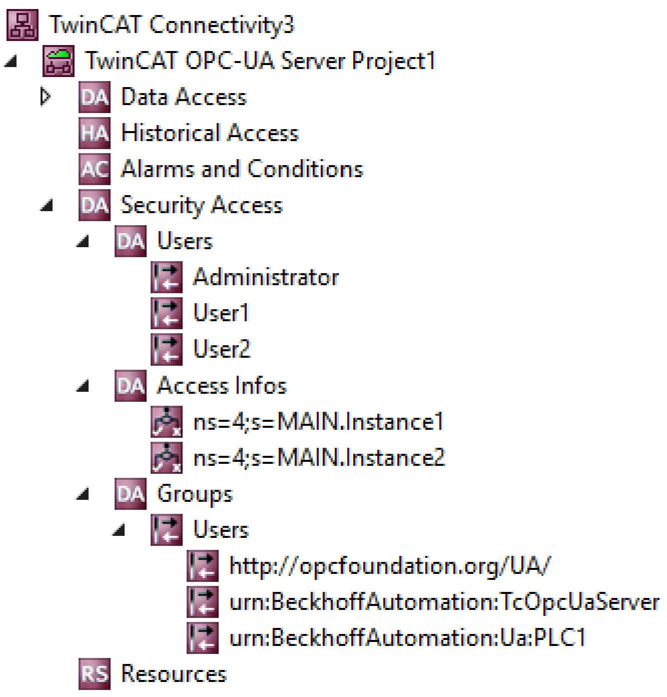
Access restrictions

Access to the server is to be restricted for clients as follows:

- Anonymous access is to be deactivated.
- There is to be a user - "Administrator" - who has full access to the complete server.
- There is to be a user - "User1" - who only has read access to MAIN.Instance1. The user should not come from the operating system here, but should only be used internally in the server.
- There is to be a user - "User2" - who only has read access to MAIN.Instance2. The user should not come from the operating system here, but should only be used internally in the server.
- General access permissions are to be configured for all users via a group called "Users".

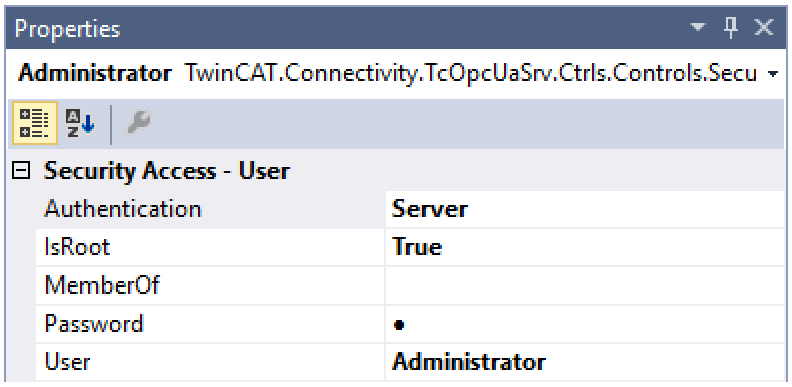
Settings

The configuration of the OPC UA Server is set as follows:



```
TwinCAT Connectivity3
  TwinCAT OPC-UA Server Project1
    Data Access
    Historical Access
    Alarms and Conditions
    Security Access
      Users
        Administrator
        User1
        User2
      Access Infos
        ns=4;s=MAIN.Instance1
        ns=4;s=MAIN.Instance2
      Groups
        Users
          http://opcfoundation.org/UA/
          urn:BeckhoffAutomation:TcOpcUaServer
          urn:BeckhoffAutomation:Ua:PLC1
    Resources
```

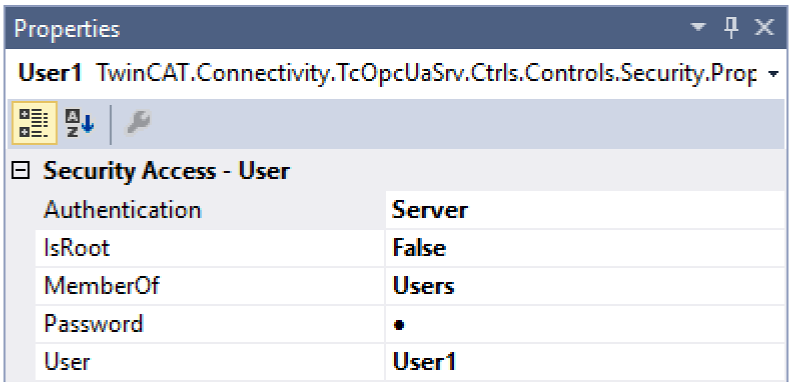
Settings for the user "Administrator":



Properties
Administrator TwinCAT.Connectivity.TcOpcUaSrv.Ctrls.Controls.Secu

Security Access - User	
Authentication	Server
IsRoot	True
MemberOf	
Password	●
User	Administrator

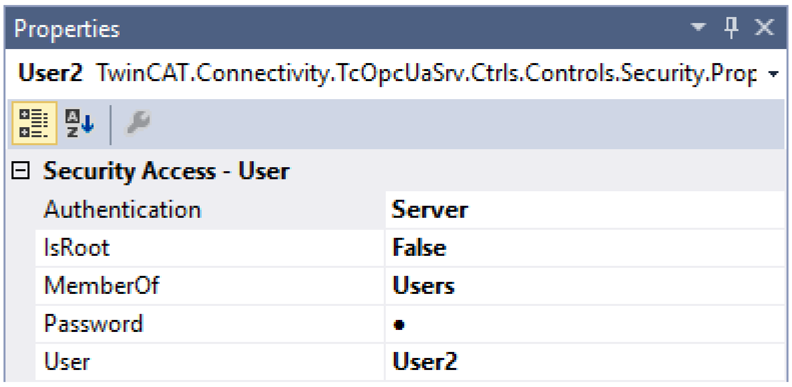
Settings for the user "User1":



Properties
User1 TwinCAT.Connectivity.TcOpcUaSrv.Ctrls.Controls.Security.Prop

Security Access - User	
Authentication	Server
IsRoot	False
MemberOf	Users
Password	●
User	User1

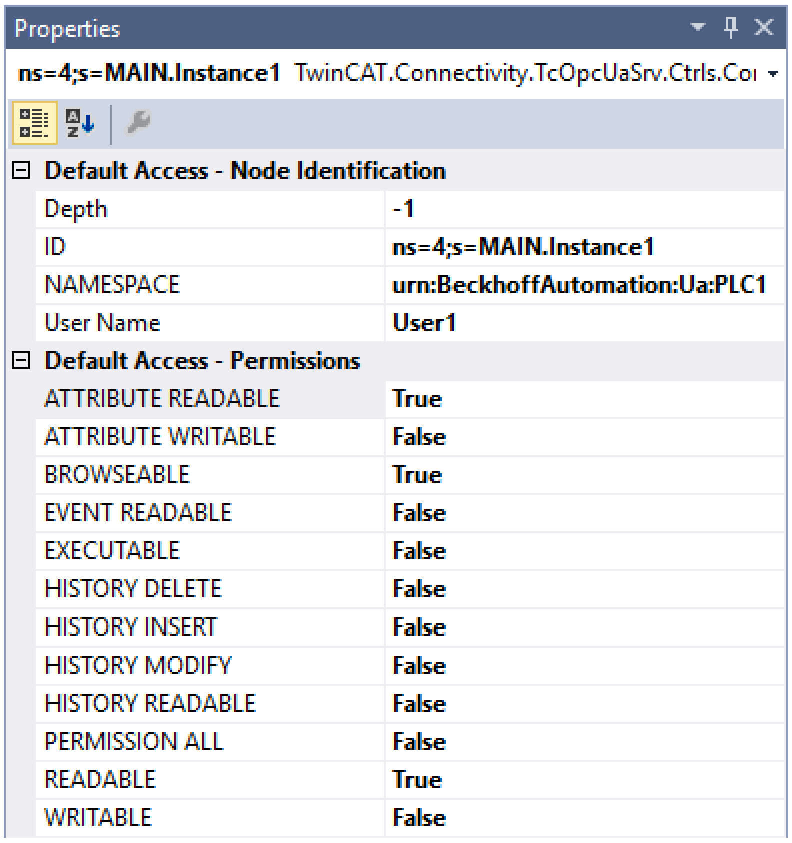
Settings for the user "User2":


Properties

User2 TwinCAT.Connectivity.TcOpcUaSrv.Ctrls.Controls.Security.Prop

Security Access - User	
Authentication	Server
IsRoot	False
MemberOf	Users
Password	●
User	User2

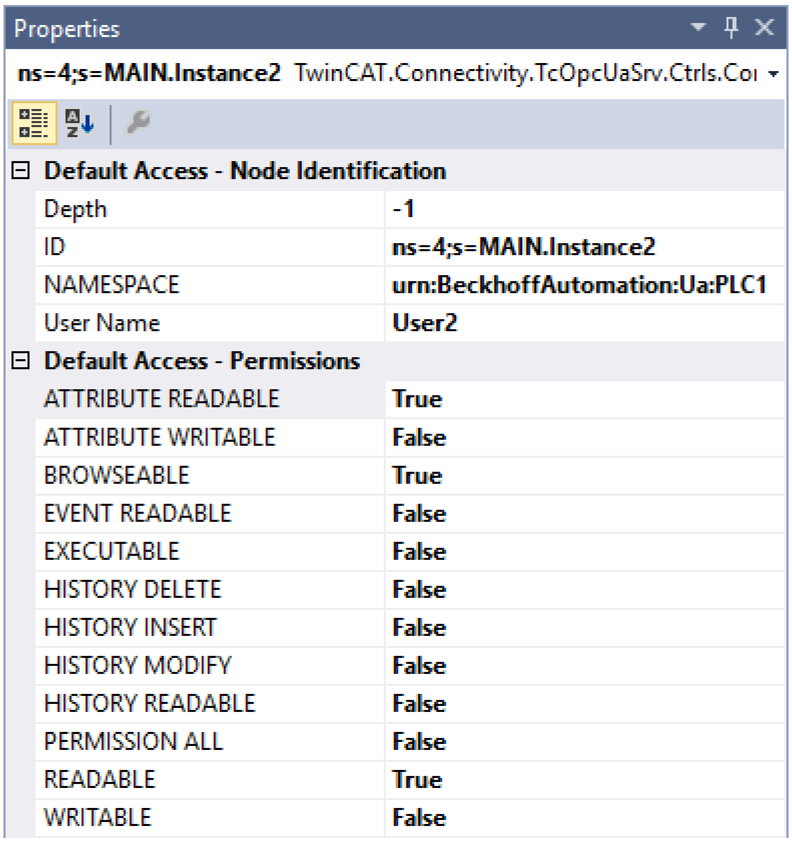
Settings for AccessInfos "MAIN.Instance1":


Properties

ns=4;s=MAIN.Instance1 TwinCAT.Connectivity.TcOpcUaSrv.Ctrls.Coı

Default Access - Node Identification	
Depth	-1
ID	ns=4;s=MAIN.Instance1
NAMESPACE	urn:BeckhoffAutomation:Ua:PLC1
User Name	User1
Default Access - Permissions	
ATTRIBUTE READABLE	True
ATTRIBUTE WRITABLE	False
BROWSEABLE	True
EVENT READABLE	False
EXECUTABLE	False
HISTORY DELETE	False
HISTORY INSERT	False
HISTORY MODIFY	False
HISTORY READABLE	False
PERMISSION ALL	False
READABLE	True
WRITABLE	False

Settings for AccessInfos "MAIN.Instance2":



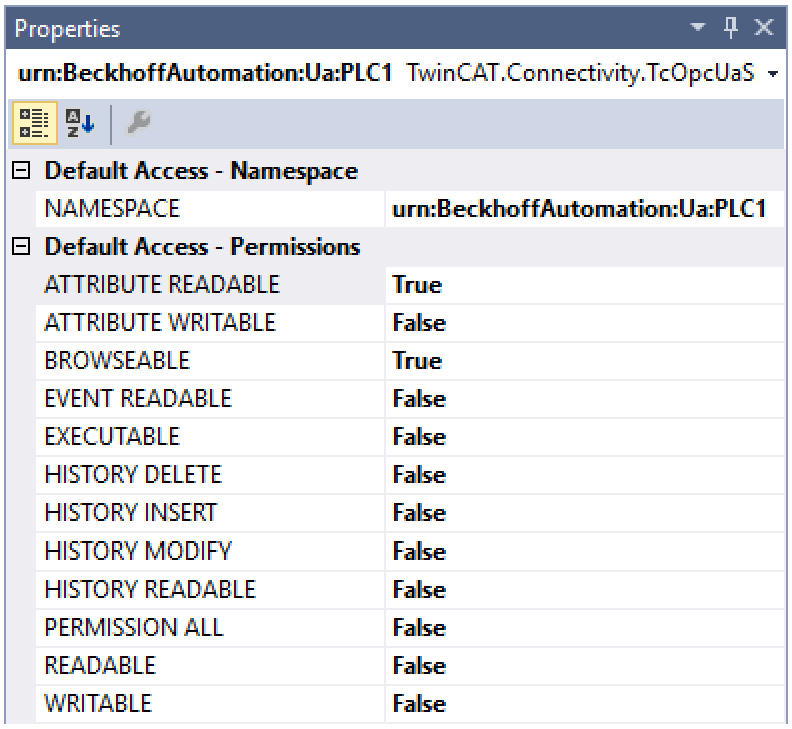
Properties

ns=4;s=MAIN.Instance2 TwinCAT.Connectivity.TcOpcUaSrv.Ctrls.Coı

Default Access - Node Identification	
Depth	**-1**
ID	**ns=4;s=MAIN.Instance2**
NAMESPACE	**urn:BeckhoffAutomation:Ua:PLC1**
User Name	**User2**
Default Access - Permissions	
ATTRIBUTE READABLE	**True**
ATTRIBUTE WRITABLE	**False**
BROWSEABLE	**True**
EVENT READABLE	**False**
EXECUTABLE	**False**
HISTORY DELETE	**False**
HISTORY INSERT	**False**
HISTORY MODIFY	**False**
HISTORY READABLE	**False**
PERMISSION ALL	**False**
READABLE	**True**
WRITABLE	**False**

Settings for the group "Users":

The user group is equipped both with basic access to required server and type system namespaces and with read and browse permissions to the PLC1 namespace.



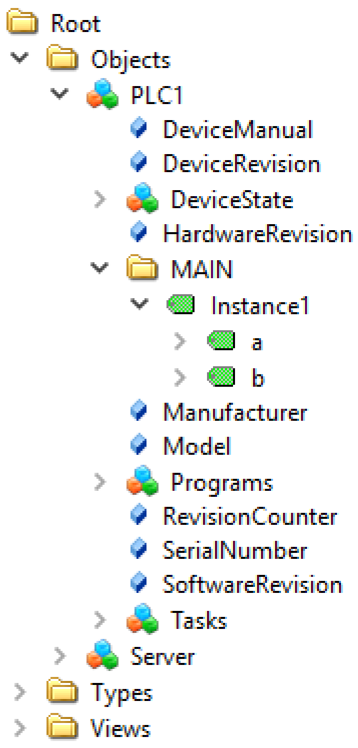
Properties

urn:BeckhoffAutomation:Ua:PLC1 TwinCAT.Connectivity.TcOpcUaS

Default Access - Namespace	
NAMESPACE	**urn:BeckhoffAutomation:Ua:PLC1**
Default Access - Permissions	
ATTRIBUTE READABLE	**True**
ATTRIBUTE WRITABLE	**False**
BROWSEABLE	**True**
EVENT READABLE	**False**
EXECUTABLE	**False**
HISTORY DELETE	**False**
HISTORY INSERT	**False**
HISTORY MODIFY	**False**
HISTORY READABLE	**False**
PERMISSION ALL	**False**
READABLE	**False**
WRITABLE	**False**

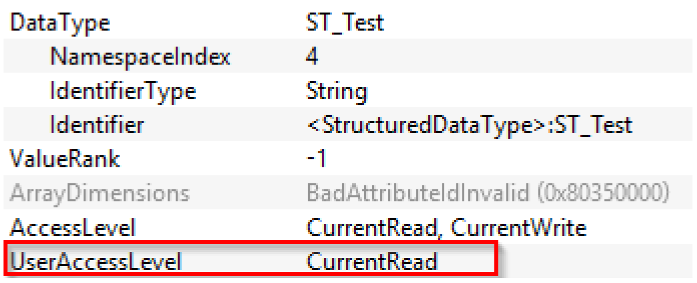
Result

Following activation of the configuration, the namespace of the server for "User1" looks like the following after establishment of a connection:



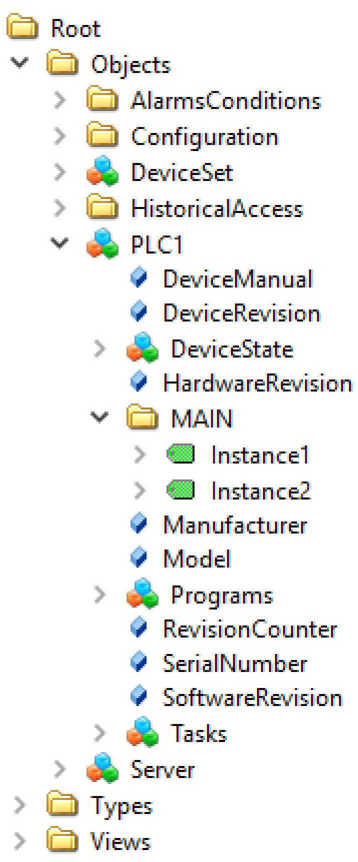
```
Root
  Objects
    PLC1
      DeviceManual
      DeviceRevision
      DeviceState
      HardwareRevision
      MAIN
        Instance1
          a
          b
      Manufacturer
      Model
      Programs
      RevisionCounter
      SerialNumber
      SoftwareRevision
      Tasks
    Server
  Types
  Views
```

The user has only read rights to the node "Instance1", which is clear from the attribute UserAccessLevel:



DataType	ST_Test
NamespaceIndex	4
IdentifierType	String
Identifier	<StructuredDataType>:ST_Test
ValueRank	-1
ArrayDimensions	BadAttributeIdInvalid (0x80350000)
AccessLevel	CurrentRead, CurrentWrite
UserAccessLevel	CurrentRead

The user "Administrator", conversely, has full access rights to all elements of the namespace:



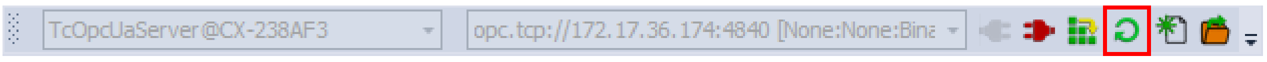
4.2.1.14 Restarting the server

The OPC UA Configurator enables the triggering of a restart of the OPC UA Server. This can be done locally or remotely and refers to the selected target device.

Loss of connection

A restart of the OPC UA Server always leads to a loss of the connection of all connected clients.

The restart is triggered via the toolbar.



Requirements

Products	Setup versions	Target platform
TF6100	4.x.x	IPC or CX (x86, x64, ARM)

See also: Connecting to a server [▶ 118]

4.2.1.15 Logging

For an advanced diagnostics you can activate the logging function of the OPC UA Server.

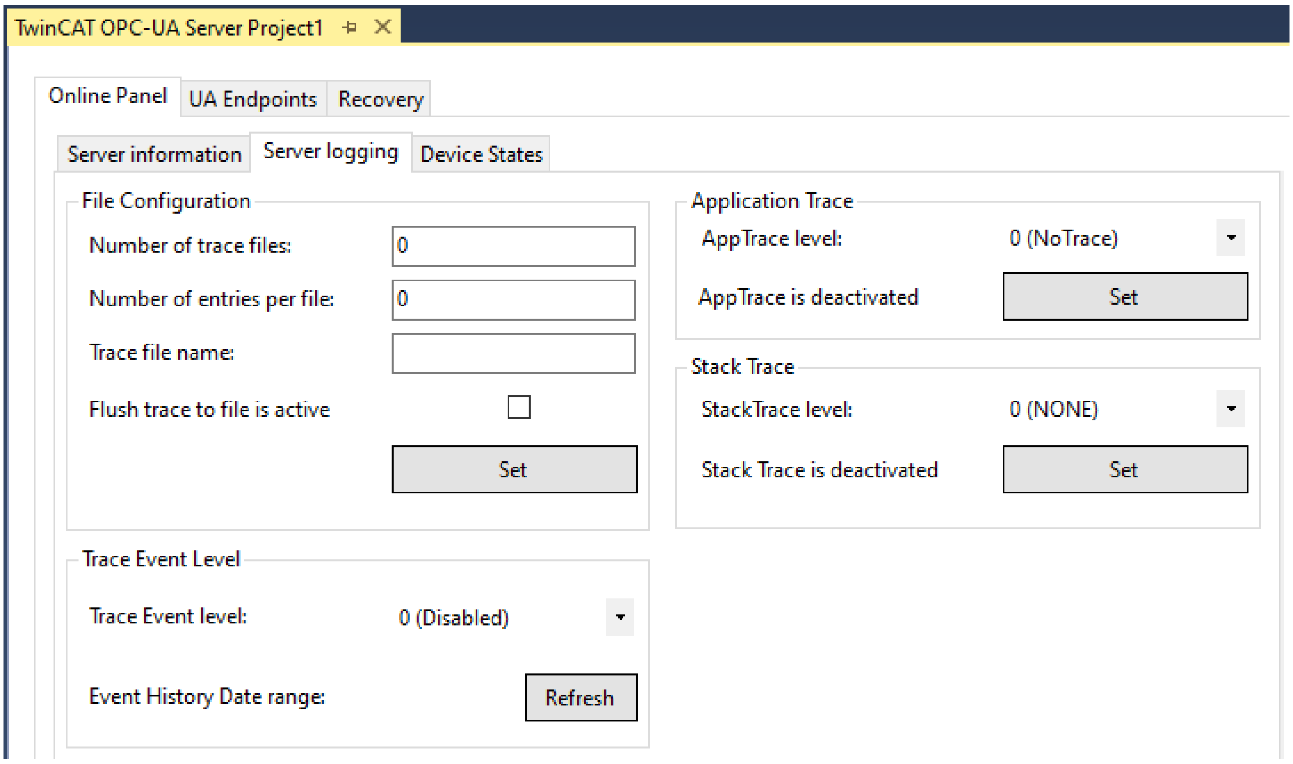
Writing the log file

Activating the logging function on the server causes a log file to be written on the file system. Make sure that there is sufficient storage space available and set the logging parameters accordingly (number of log files, size per log file).

Performance and timing behavior

Activation of the logging function will change the timing behavior of the OPC UA Server. As a result there may be losses of speed, depending on the platform and project.

The logging function is activated using the **Activate** button on the **Online Panel** tab in the project configurator.You can activate the function locally or remotely depending on the selected target device. The logging function remains active until it is deactivated again via the configurator or until the OPC UA Server is restarted.


TwinCAT OPC-UA Server Project1

Online Panel | UA Endpoints | Recovery

Server information | Server logging | Device States

File Configuration
Number of trace files: 0
Number of entries per file: 0
Trace file name:
Flush trace to file is active ☐
Set

Application Trace
AppTrace level: 0 (NoTrace)
AppTrace is deactivated
Set

Stack Trace
StackTrace level: 0 (NONE)
Stack Trace is deactivated
Set

Trace Event Level
Trace Event level: 0 (Disabled)
Event History Date range:
Refresh

Trace Level

In general, the higher the trace level, the more detailed (and more) data is written, but the more load is also placed on the server application, which changes the timing behavior accordingly. Please therefore only activate logging in the event of diagnostics and in consultation with Beckhoff Support.

Activate App Trace

In most cases it is sufficient to create a so-called "AppTrace". This logs information from the server application. To activate the AppTrace, please enter the number of TraceFiles and the number of entries per TraceFile in the corresponding text fields. Then select a trace level and click the button to activate the AppTrace. The values in the gray text boxes represent the current settings on the server.

Activate Stack Trace

In a few cases it is also necessary to create a so-called "StackTrace", whereby information from the OPC UA stack is logged. To activate the StackTrace please enter the number of TraceFiles as well as the number of entries per TraceFile into the corresponding text boxes. Then select a trace level and click on the button to activate the StackTrace. The values in the gray text boxes represent the current settings on the server.

Requirements

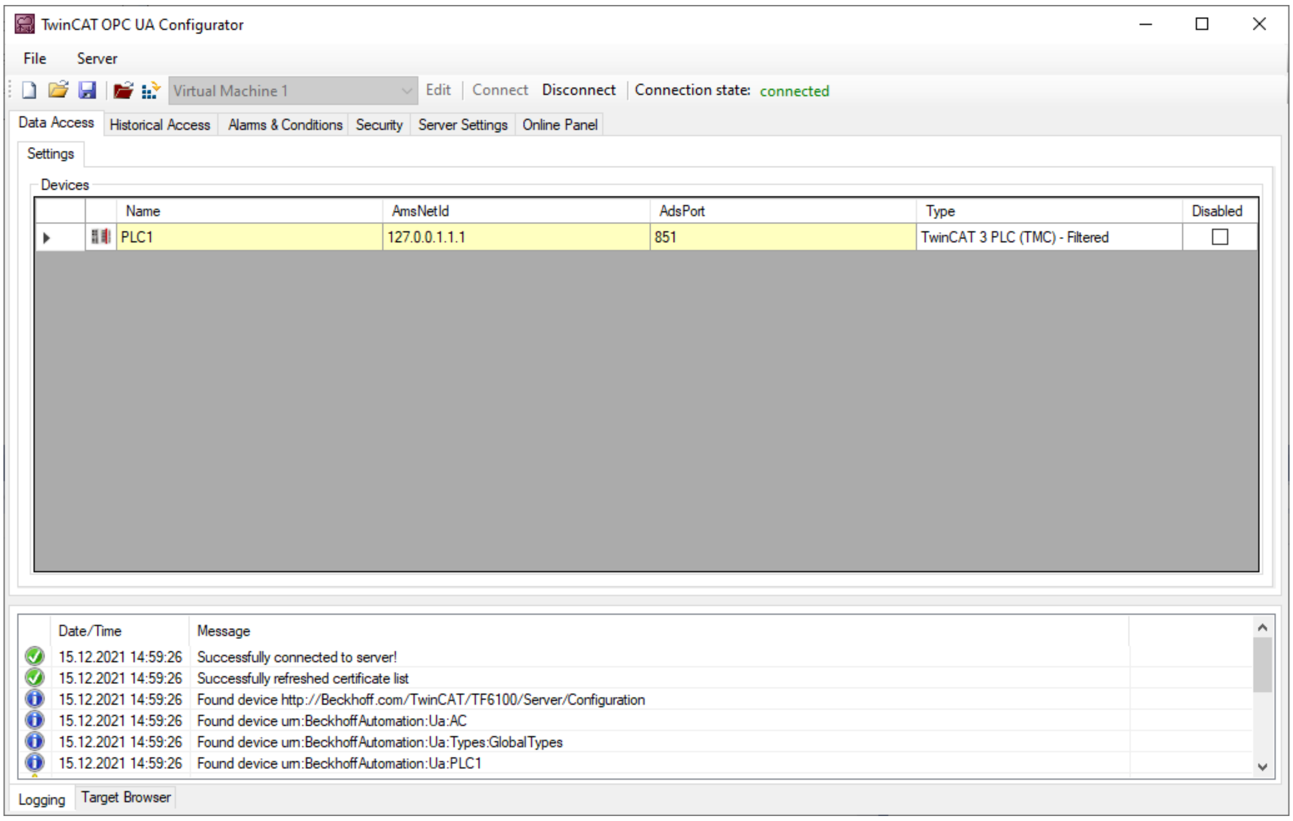
Products	Setup versions	Target platform
TF6100	4.x.x	IPC or CX (x86, x64, ARM)

See also: Selecting a target device [▶ 118]

4.2.2 Standalone

4.2.2.1 Overview

The standalone configurator enables parameterization of the TwinCAT OPC UA Server independently of Visual Studio. You can configure all the different features of the server.

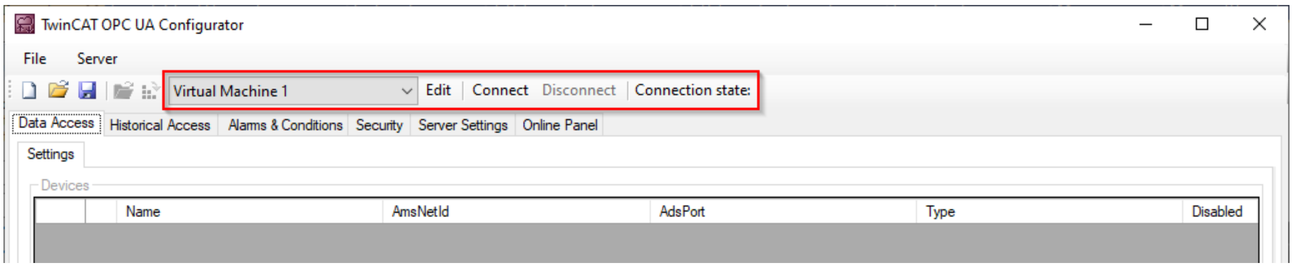


Name	AmsNetId	AdsPort	Type	Disabled
PLC1	127.0.0.1.1.1	851	TwinCAT 3 PLC (TMC) - Filtered	☐

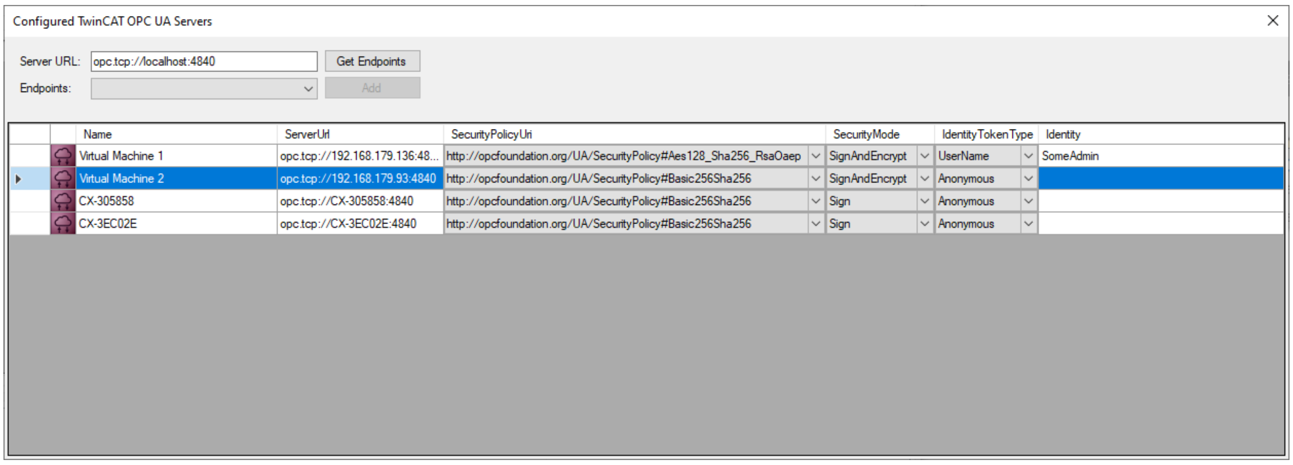
Date/Time	Message
15.12.2021 14:59:26	Successfully connected to server!
15.12.2021 14:59:26	Successfully refreshed certificate list
15.12.2021 14:59:26	Found device http://Beckhoff.com/TwinCAT/TF6100/Server/Configuration
15.12.2021 14:59:26	Found device urn:BeckhoffAutomation:Ua:AC
15.12.2021 14:59:26	Found device urn:BeckhoffAutomation:Ua:Types:GlobalTypes
15.12.2021 14:59:26	Found device urn:BeckhoffAutomation:Ua:PLC1

4.2.2.2 Connecting to a server

The OPC UA Configurator enables the complete parameterization of the Server via OPC UA. Similar to the TwinCAT XAE system, you can select an OPC UA Server to connect to via the toolbar.



Click on the **Edit** button to open the server list dialog. In this dialog you can add one or more server connections.



Server URL: opc.tcp://localhost:4840
Endpoints:

Name	ServerUrl	SecurityPolicyUri	SecurityMode	IdentityTokenType	Identity
Virtual Machine 1	opc.tcp://192.168.179.136:48...	http://opcfoundation.org/UA/SecurityPolicy#Aes128_Sha256_RsaOaep	SignAndEncrypt	UserName	SomeAdmin
Virtual Machine 2	opc.tcp://192.168.179.93:4840	http://opcfoundation.org/UA/SecurityPolicy#Basic256Sha256	SignAndEncrypt	Anonymous	
CX-305858	opc.tcp://CX-305858:4840	http://opcfoundation.org/UA/SecurityPolicy#Basic256Sha256	Sign	Anonymous	
CX-3EC02E	opc.tcp://CX-3EC02E:4840	http://opcfoundation.org/UA/SecurityPolicy#Basic256Sha256	Sign	Anonymous	

By entering a ServerURL and pressing the **Get Endpoints** button, a server connection can be added to the list. Any settings for the IdentityToken, e.g. whether the Configurator should connect as an anonymous user or with a user name/password combination, must be set manually.

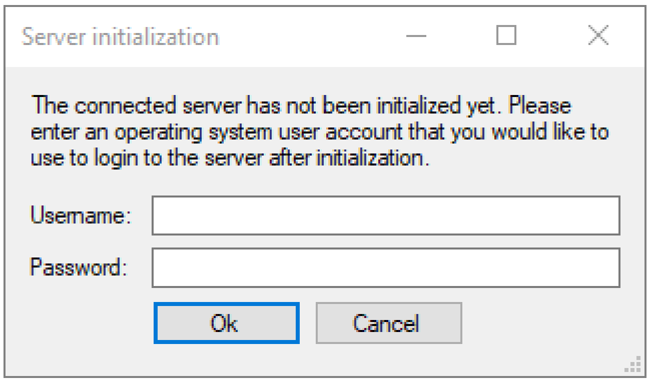
Confirming a configuration

Please always confirm changes to the entries with the **ENTER** key, as only then will they be automatically saved in the background.

After configuring a server connection, the corresponding entry is available in the DropDownBox and the connection can be established by clicking the **Connect** button.

4.2.2.3 Performing the server initialization

The TwinCAT OPC UA Server is delivered in an uninitialized mode, which is based on the so-called TOFU (Trust-On-First-Use) principle. Detailed information about this server feature and the corresponding background information can be found here [▸ 37]. The TwinCAT OPC UA Configurator enables the initialization of the server during the first connection establishment. A corresponding warning message indicates the uninitialized server and enables an appropriate initialization.



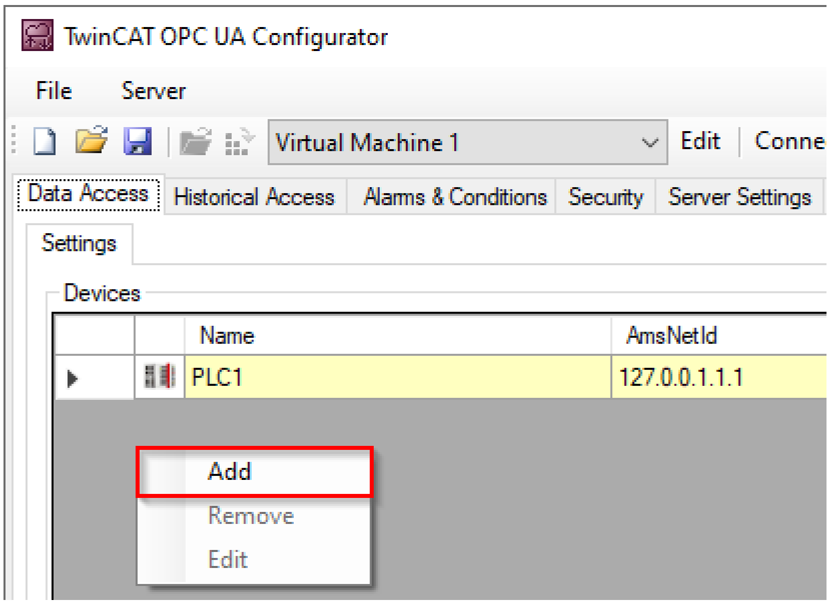
The connected server has not been initialized yet. Please enter an operating system user account that you would like to use to login to the server after initialization.

Username:

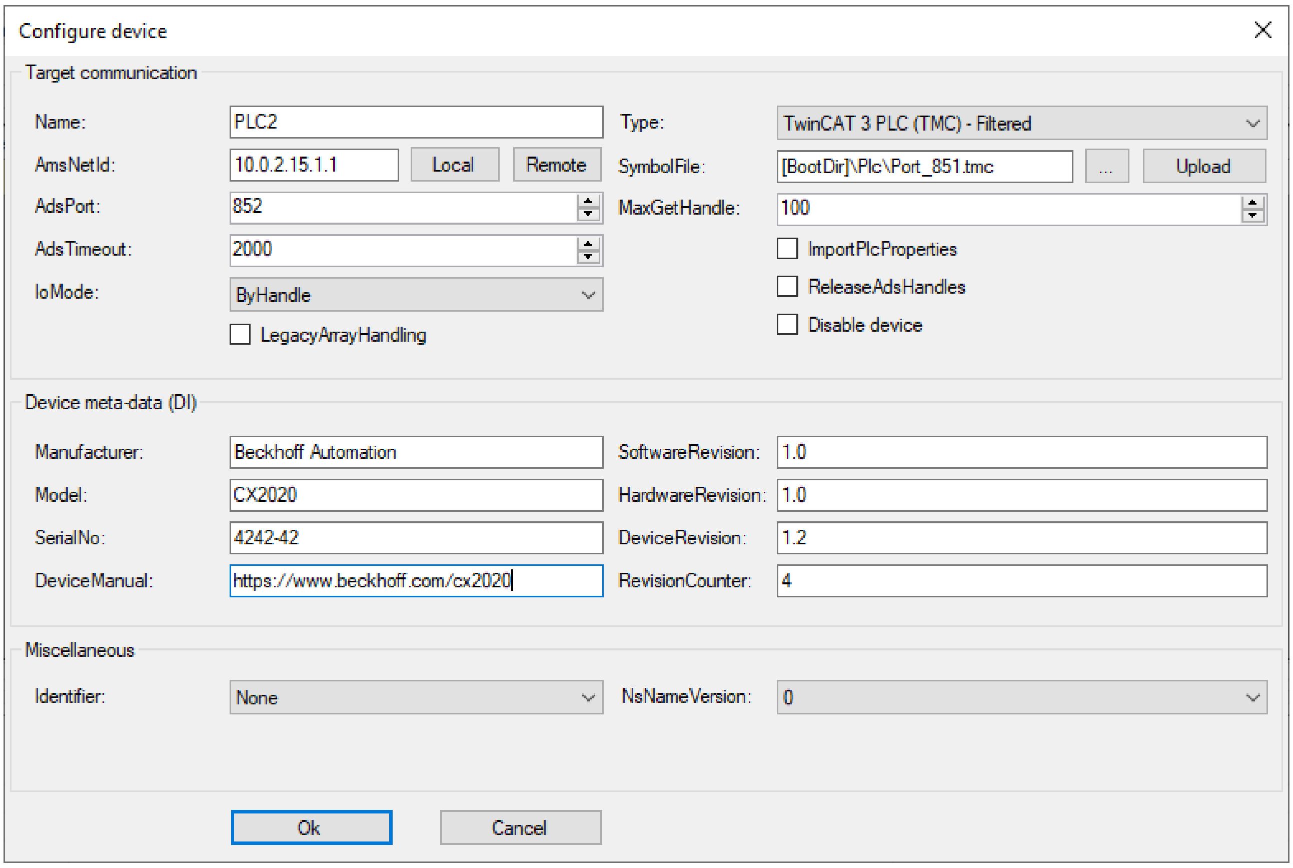
Password:

4.2.2.4 Adding ADS devices

ADS devices can be added to the TwinCAT OPC UA Server configuration via the **Data Access** tab. In the associated DataGrid, you can create a new device via the context menu.



In the subsequent dialog, you can set the device-specific parameters.



Configure device

Target communication

Name:	PLC2	Type:	TwinCAT 3 PLC (TMC) - Filtered
AmsNetId:	10.0.2.15.1.1	SymbolFile:	[BootDir]\Plc\Port_851.tmc
AdsPort:	852	MaxGetHandle:	100
AdsTimeout:	2000	ImportPlcProperties	☐
IoMode:	ByHandle	ReleaseAdsHandles	☐
LegacyArrayHandling	☐	Disable device	☐

Device meta-data (DI)

Manufacturer:	Beckhoff Automation	SoftwareRevision:	1.0
Model:	CX2020	HardwareRevision:	1.0
SerialNo:	4242-42	DeviceRevision:	1.2
DeviceManual:	https://www.beckhoff.com/cx2020	RevisionCounter:	4

Miscellaneous

Identifier:	None	NsNameVersion:	0

Selecting an AMS NetID

To select an AMS NetID, either the ADS devices from the local system or the connected TwinCAT OPC UA Server can be selected. An ADS device is a system that has an ADS route to the local system or server system. By clicking on the button **Local** the local ADS routes are displayed. By clicking the **Remote** button the ADS routes on the connected TwinCAT OPC UA Server are displayed.

Selecting a symbol file

The selection of a symbol file is always done from the local system. However, the symbol file can be uploaded to the connected TwinCAT OPC UA Server via the **Upload** button. The symbol file is stored in the subfolder "symbolfiles" of the TwinCAT OPC UA Server home directory and automatically referenced via a placeholder in the configuration file.

4.2.2.5 Reading and writing the configuration

The configurator enables both reading/writing of the configuration files from the TwinCAT OPC UA Server and loading/saving of the configuration files on the local system. These functionalities are available via the menu as well as the toolbar.

Local loading/saving

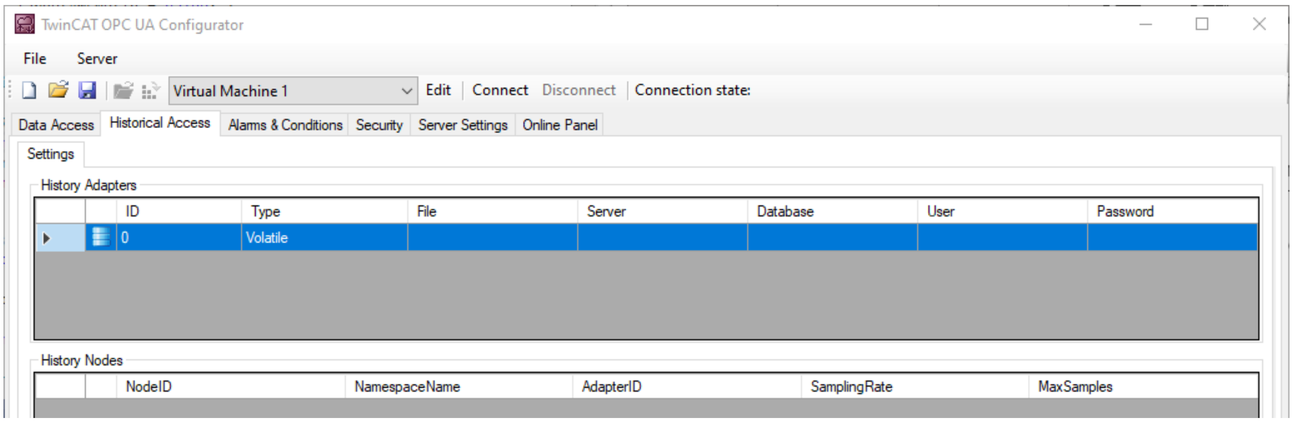
These functions are available via the **File** menu. The buttons available here **Open** and **Save** enable the configuration files to be loaded and saved. All configuration files are always loaded or saved.

Remote loading/saving

These functions are available from the **server** menu. The buttons **Open from target** and **Activate from target** available here enable loading and saving of the configuration files from the connected TwinCAT OPC UA Server. All configuration files are always loaded or saved.

4.2.2.6 Configuring historical access

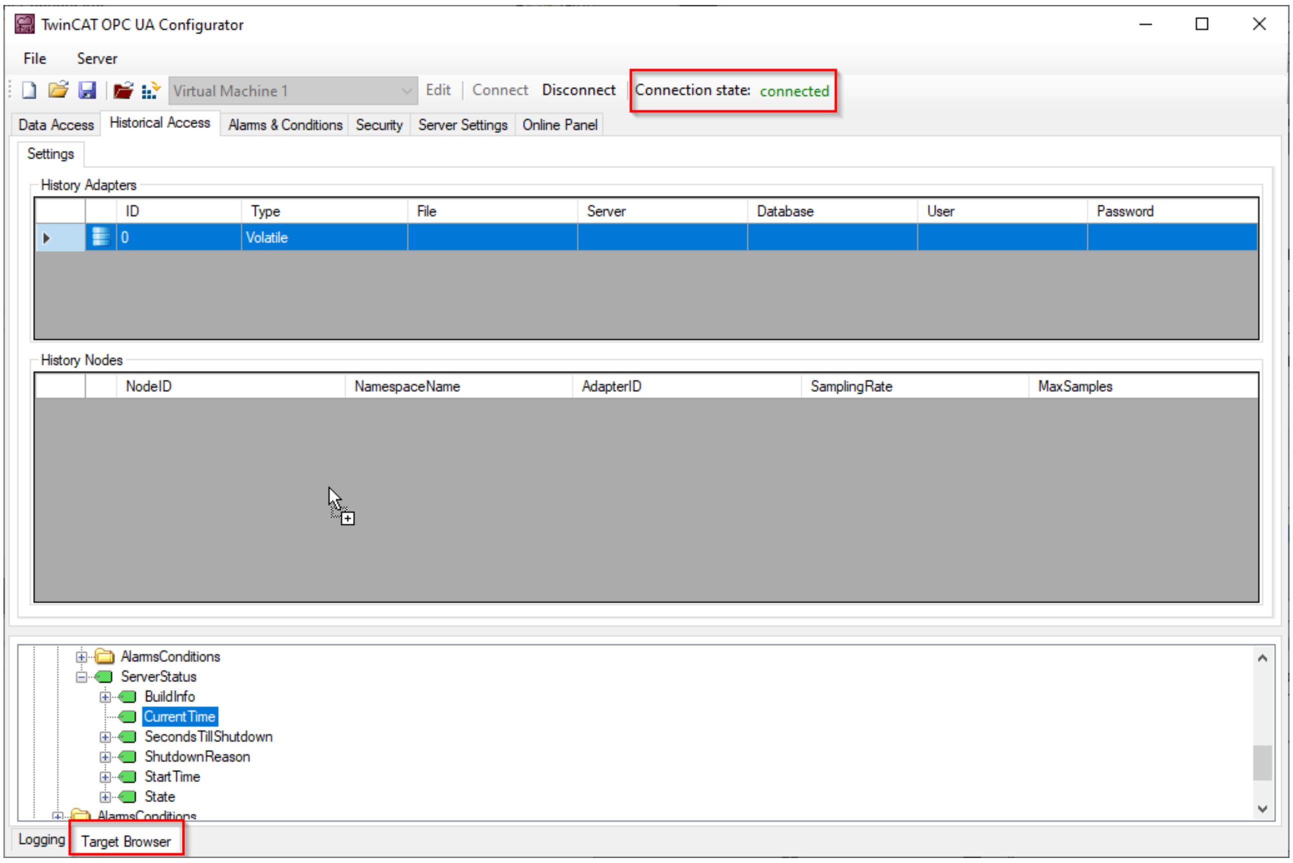
Use the **Historical Access** tab to configure both the **History Adapter** and the **History Nodes**. A **History Adapter** defines the type of data storage and a **History Node** the variable for which historical data should be saved in the data storage.



ID	Type	File	Server	Database	User	Password
0	Volatile					

NodeID	NamespaceName	AdapterID	SamplingRate	MaxSamples

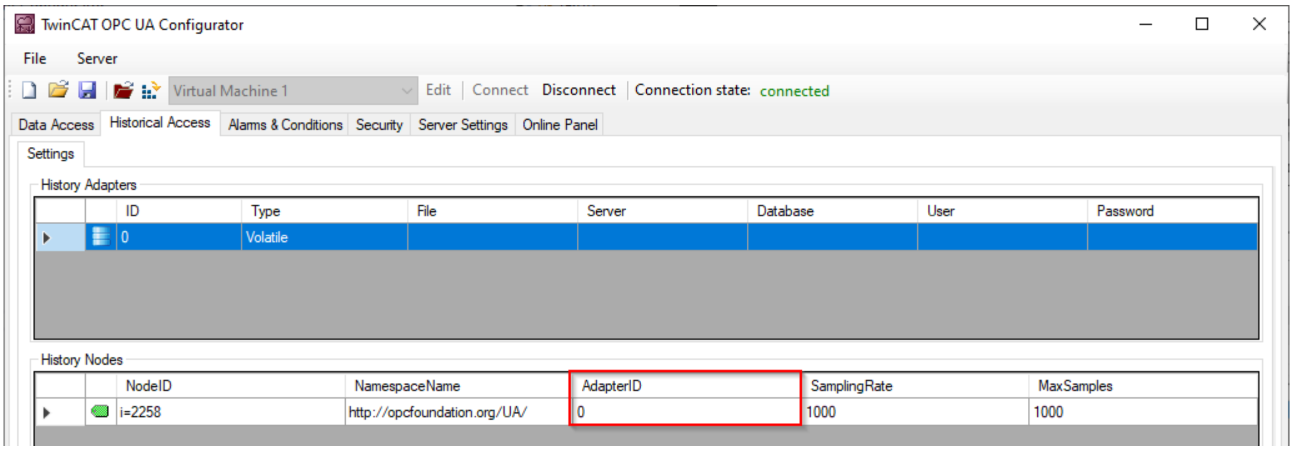
You can use the context menu to create both **History Adapter** and **History Nodes**. If you are connected to a TwinCAT OPC UA Server, you can also conveniently add the nodes to be configured via drag & drop from the **Target Browser** to the **History Nodes**.



ID	Type	File	Server	Database	User	Password
0	Volatile					

NodeID	NamespaceName	AdapterID	SamplingRate	MaxSamples

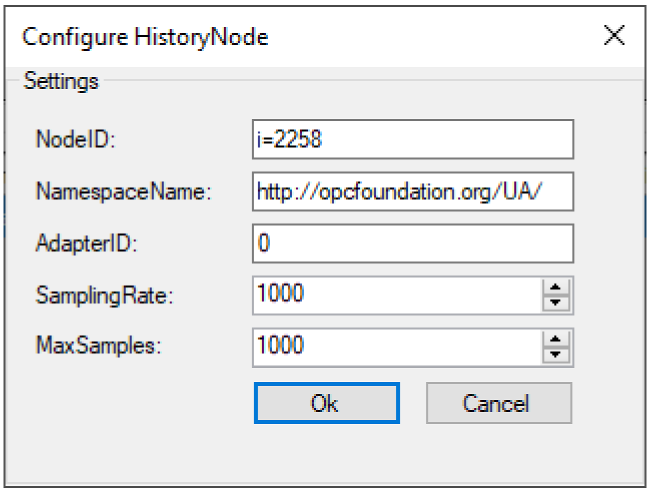
Subsequently, a **History Node** can be linked to the respective **History Adapter** via the AdapterID.



ID	Type	File	Server	Database	User	Password
0	Volatile					

NodeID	NamespaceName	AdapterID	SamplingRate	MaxSamples
i=2258	http://opcfoundation.org/UA/	0	1000	1000

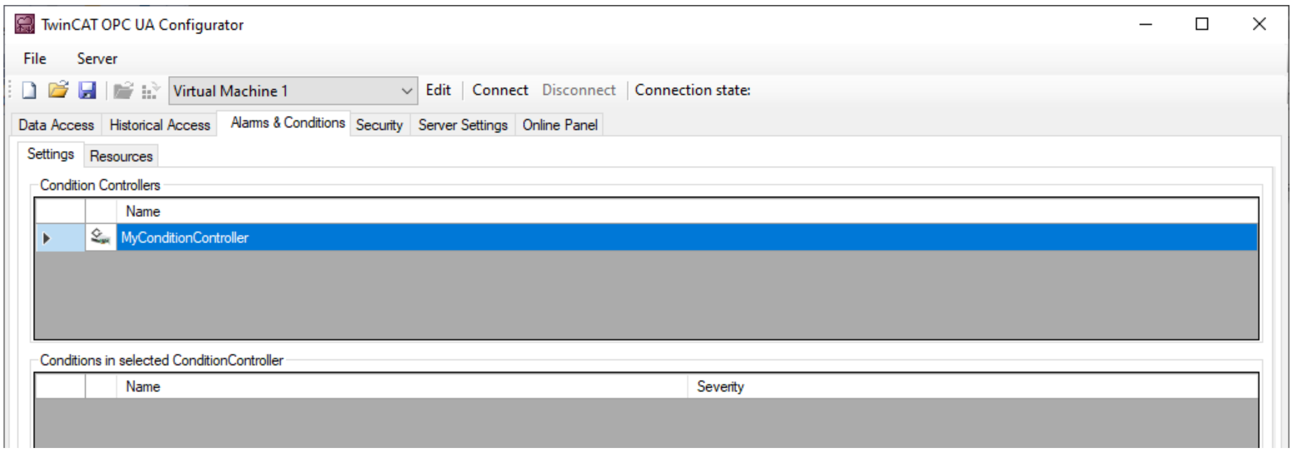
Double-click on the History Node to open the corresponding configuration dialog.



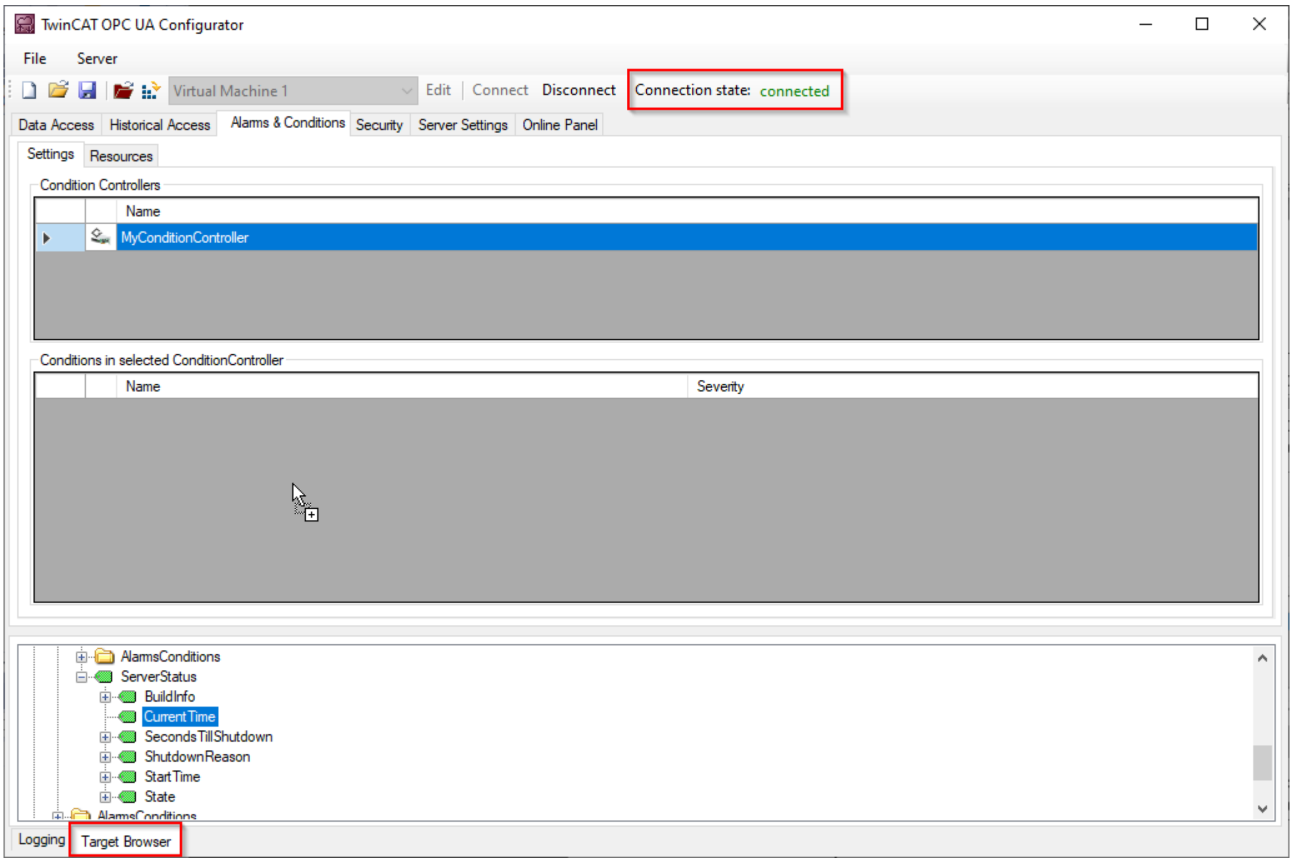
NodeID:	i=2258
NamespaceName:	http://opcfoundation.org/UA/
AdapterID:	0
SamplingRate:	1000
MaxSamples:	1000

4.2.2.7 Configuring Alarms and Conditions

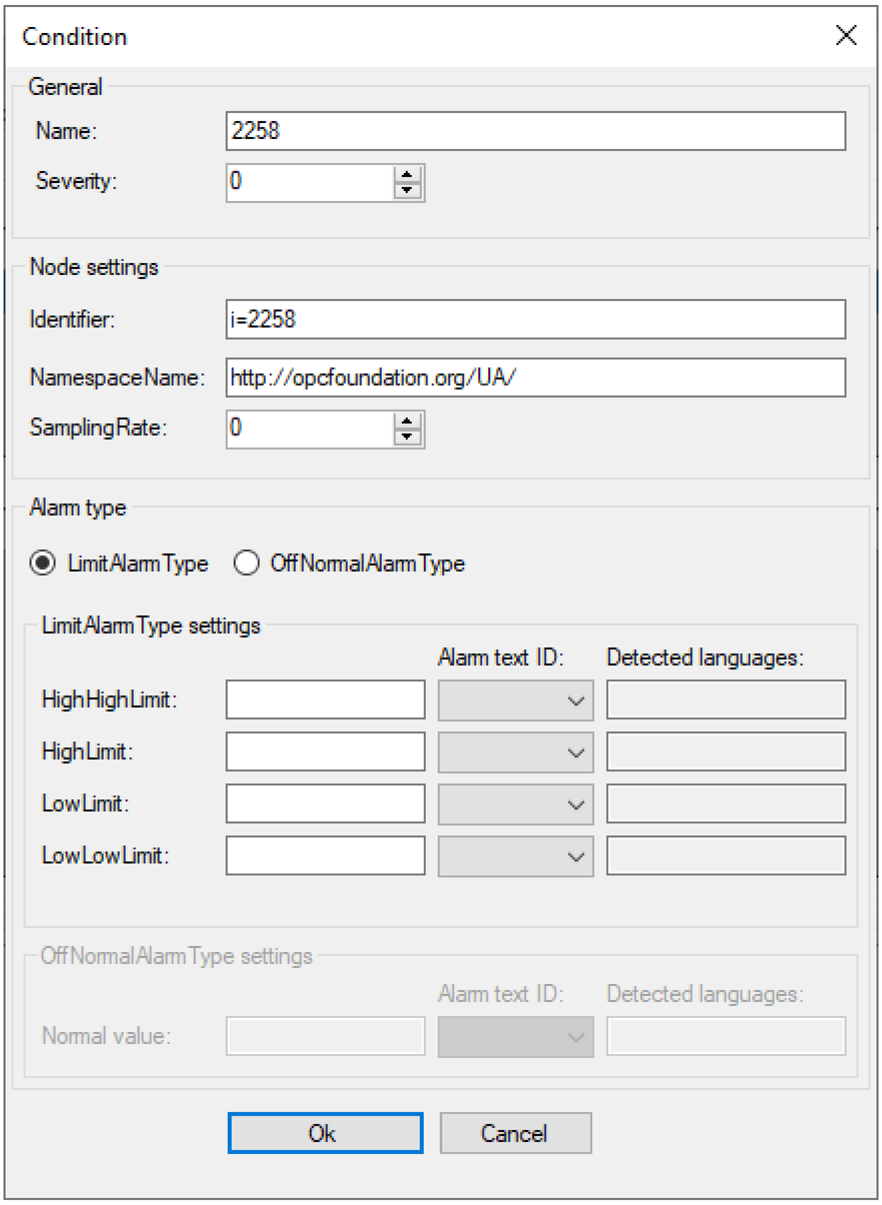
Use the **Alarms & Conditions** tab to configure both the **Condition Controller** and the **Conditions**. A **Condition Controller** is a management unit for organizing the individual **Conditions**. A **Condition** on the other hand reflects a variable which is to be monitored in the sense of **Alarms & Conditions** on the basis of configurable threshold values.



You can use the context menu to create both **Condition Controller** and **Conditions**. If you are connected to a TwinCAT OPC UA Server, you can also conveniently add the nodes to be configured to the **Conditions** via drag & drop from the **Target Browser**.



A **Condition** is always added to the currently selected **Condition Controller**. When using Drag&Drop, the configuration dialog of a **Condition** opens automatically.



Condition

General

Name: 2258

Severity: 0

Node settings

Identifier: i=2258

NamespaceName: http://opcfoundation.org/UA/

SamplingRate: 0

Alarm type

[✓ LimitAlarmType] OffNormalAlarmType

LimitAlarmType settings

		Alarm text ID:	Detected languages:
HighHighLimit:			
HighLimit:			
LowLimit:			
LowLowLimit:			

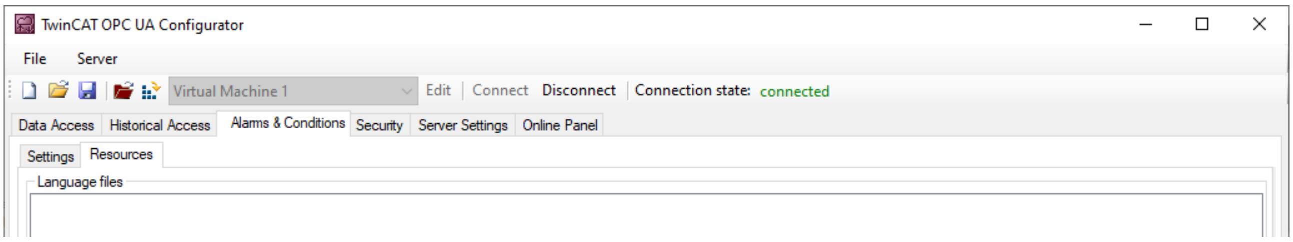
OffNormalAlarmType settings

		Alarm text ID:	Detected languages:
Normal value:			

The alarm texts to be configured when selecting the respective **AlarmType** can be selected via the corresponding drop-down boxes. Please note that the alarm texts must already be available. Read the article Configuring alarm texts [▶ 150] to learn more about this topic.

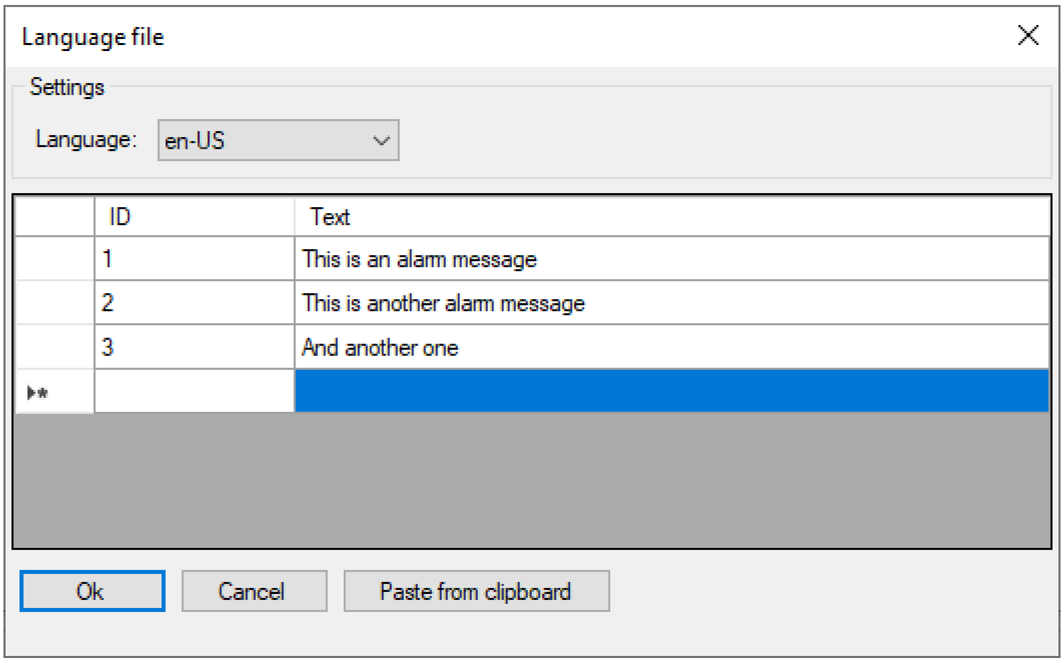
4.2.2.8 Configuring alarm texts

Within the **Alarms & Conditions** area, you can configure alarm texts via the **Resources** tab, which you can then use for a Condition.



Connection state: connected

You can add a new alarm text file via the context menu. These files are grouped according to the language for which the alarm texts are defined.



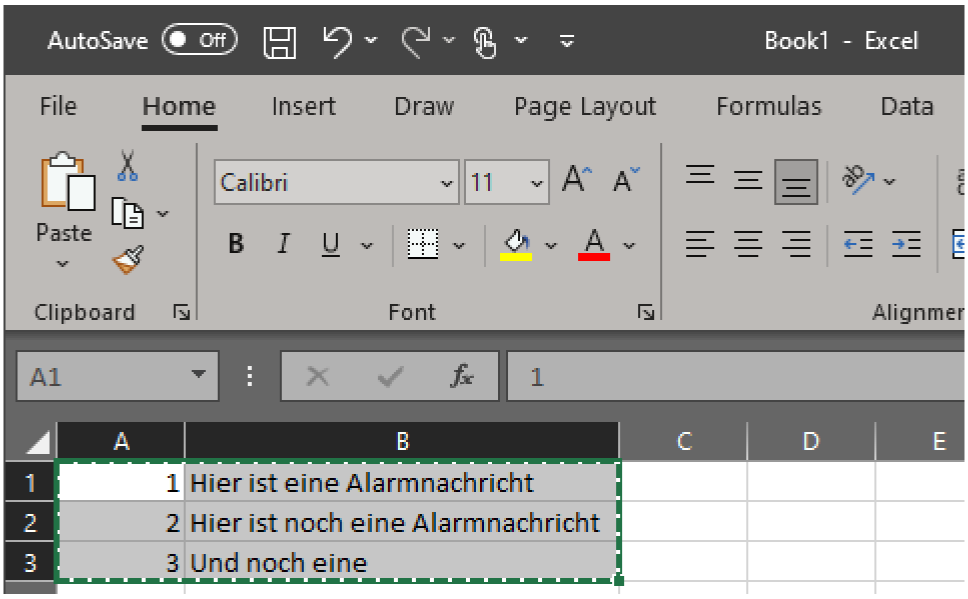
Language file

Settings

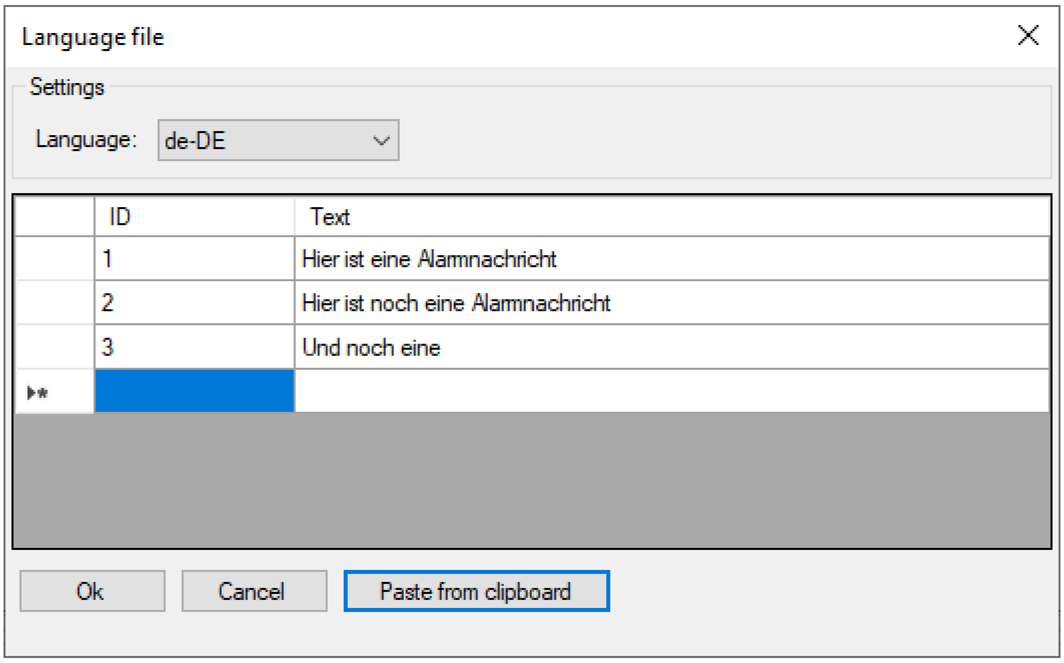
Language: en-US

	ID	Text
	1	This is an alarm message
	2	This is another alarm message
	3	And another one
▸*		

The **Paste from clipboard** button can be used to copy ID and text from an Excel spreadsheet by first copying them to the clipboard (CTRL+C) and then importing them via the button.



	A	B	C	D	E
1	1	Hier ist eine Alarmnachricht			
2	2	Hier ist noch eine Alarmnachricht			
3	3	Und noch eine			



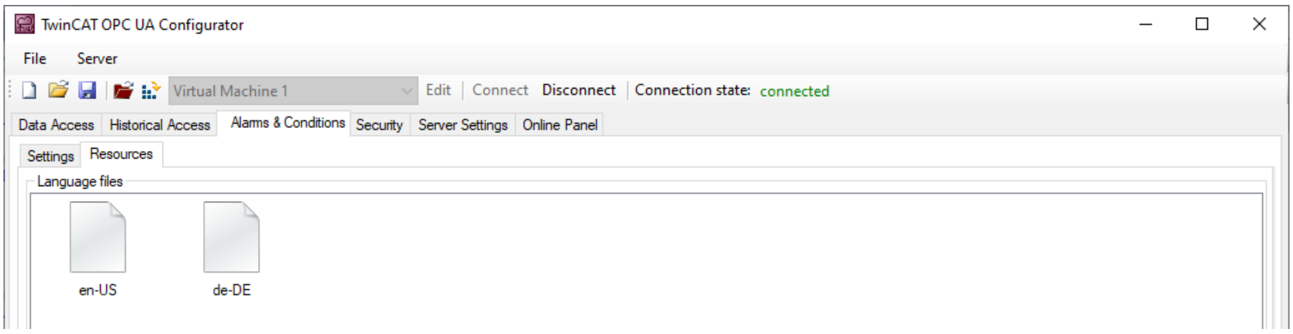
Language file

Settings

Language: de-DE

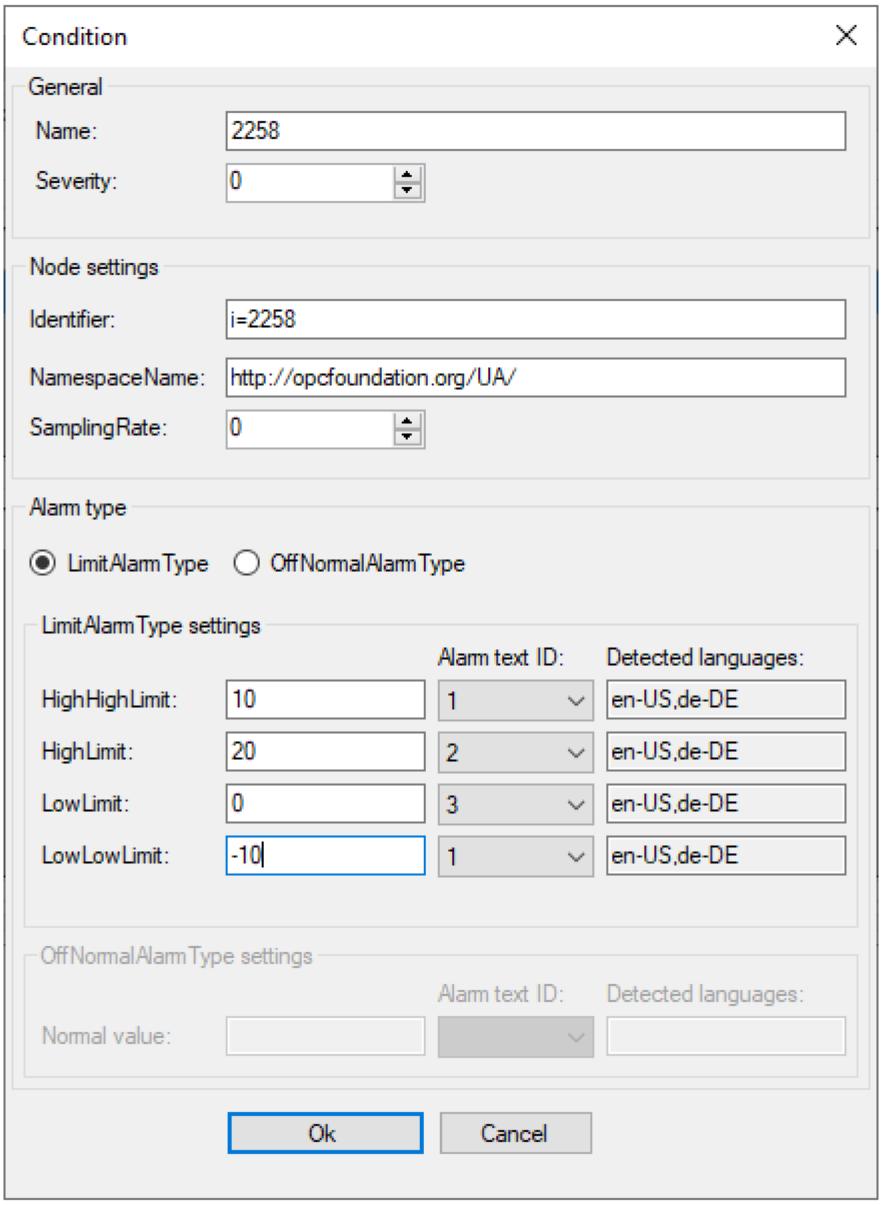
	ID	Text
	1	Hier ist eine Alarmnachricht
	2	Hier ist noch eine Alarmnachricht
	3	Und noch eine
▶*		

After configuring the language files, you can use the alarm texts on a Condition.



TwinCAT OPC UA Configurator

Virtual Machine 1 | Connection state: connected



Condition

General

Name:	2258
Severity:	0

Node settings

Identifier:	i=2258
NamespaceName:	http://opcfoundation.org/UA/
SamplingRate:	0

Alarm type

[✓ LimitAlarmType] OffNormalAlarmType

LimitAlarmType settings

		Alarm text ID:	Detected languages:
HighHighLimit:	10	1	en-US,de-DE
HighLimit:	20	2	en-US,de-DE
LowLimit:	0	3	en-US,de-DE
LowLowLimit:	-10	1	en-US,de-DE

OffNormalAlarmType settings

		Alarm text ID:	Detected languages:
Normal value:			

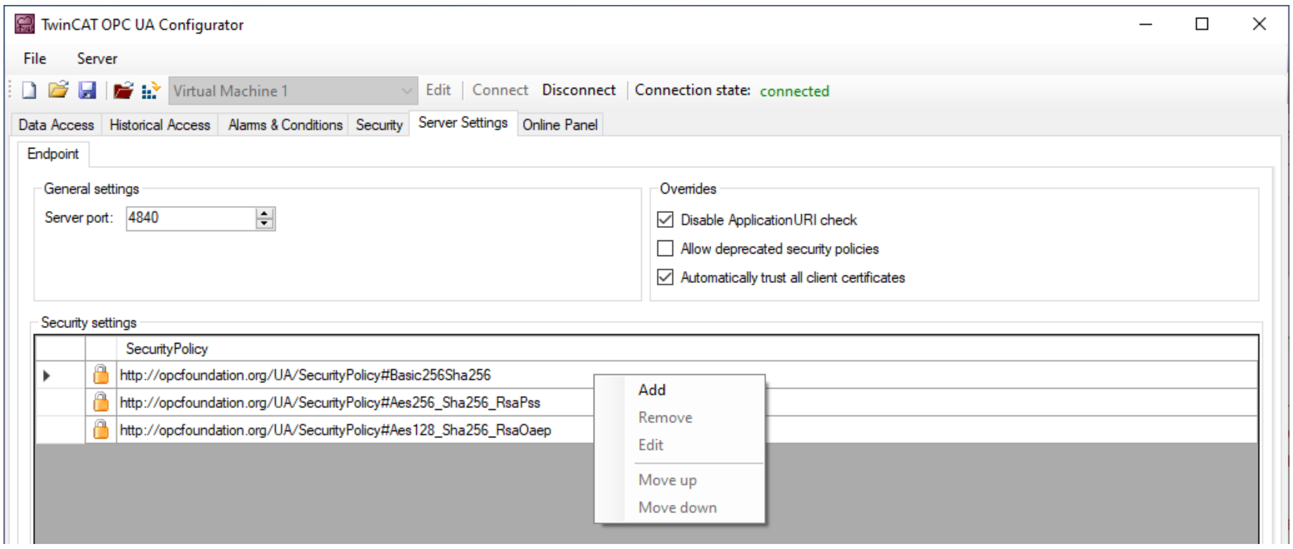
Using the **Detected languages** fields, you can quickly check whether you have defined the selected AlarmtextID for all languages, or whether a language may have been forgotten.

4.2.2.9 Configuring endpoints

The endpoints of the OPC UA Server indicate which security mechanisms are to be used during the connection establishment of a client. These range from "unencrypted" to "encrypted and signed", based on different key strengths.

The endpoints can be activated and deactivated using the configurator. It may be useful to deactivate the unencrypted endpoint so that all clients can only connect themselves with valid certificates that are classified as trustworthy.

The **Server Settings** tab allows you to configure the endpoints, as well as some additional parameters. The context menu can be used to add or remove endpoints from the configuration.



Connection state: connected

General settings

Server port: 4840

Overrides

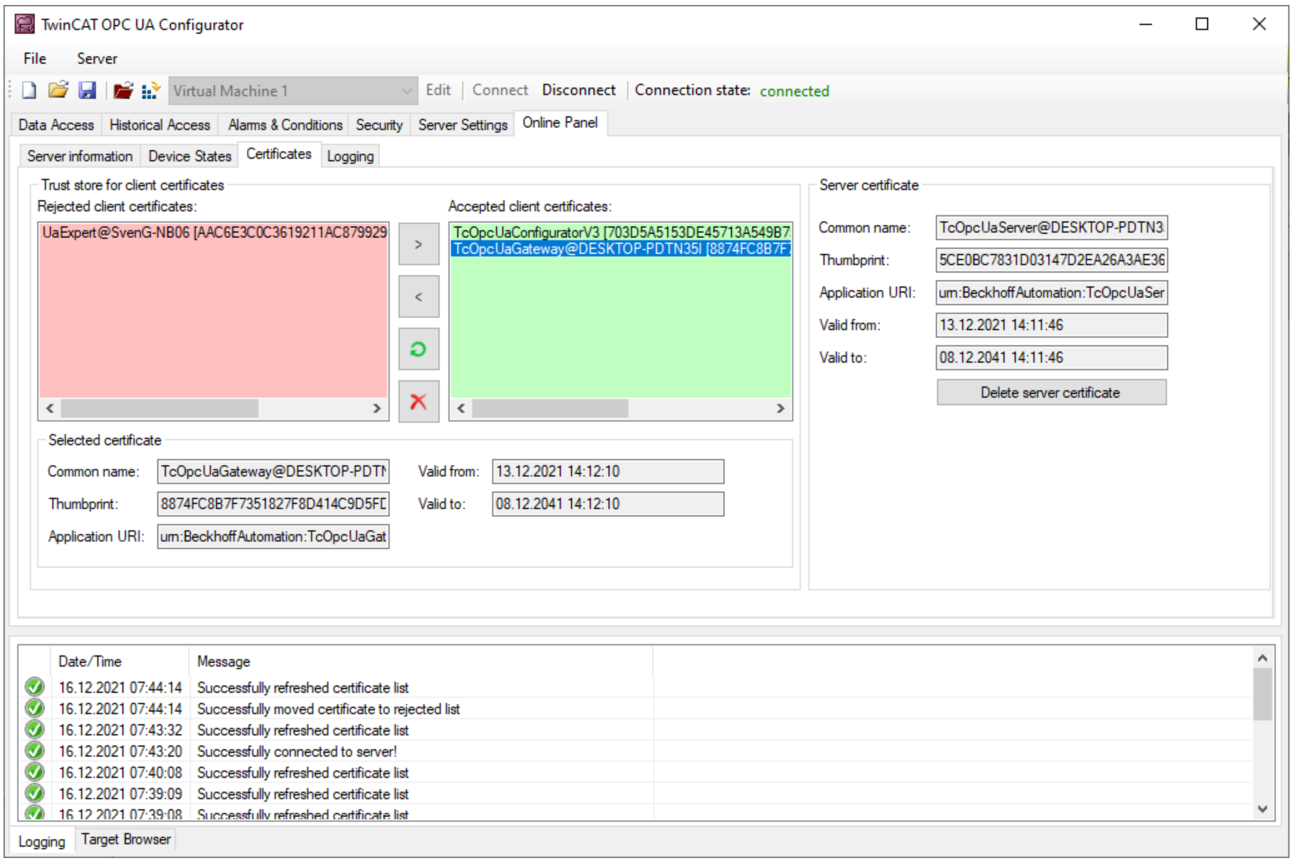
- [✓] Disable ApplicationURI check
- [ ] Allow deprecated security policies
- [✓] Automatically trust all client certificates

Security settings

SecurityPolicy
http://opcfoundation.org/UA/SecurityPolicy#Basic256Sha256
http://opcfoundation.org/UA/SecurityPolicy#Aes256_Sha256_RsaPss
http://opcfoundation.org/UA/SecurityPolicy#Aes128_Sha256_RsaOaep

4.2.2.10 Trust relationship for certificates

The trust relationships for client certificates on the TwinCAT OPC UA Server can be configured via the **Online Panel** tab and the **Certificates** section there. By selecting a client certificate in the respective TrustStore (Rejected/Accepted), certificate details can be displayed and moved between the TrustStores.



Connection state: connected

Trust store for client certificates

Rejected client certificates:
UaExpert@SvenG-NB06 [AAC6E3C0C3619211AC879929

Accepted client certificates:
TcOpcUaConfiguratorV3 [703D5A5153DE45713A549B7
TcOpcUaGateway@DESKTOP-PDTN35I [8874FC8B7F

Selected certificate

Common name:	TcOpcUaGateway@DESKTOP-PDTN	Valid from:	13.12.2021 14:12:10
Thumbprint:	8874FC8B7F7351827F8D414C9D5FD	Valid to:	08.12.2041 14:12:10
Application URI:	urn:BeckhoffAutomation:TcOpcUaGat		

Server certificate

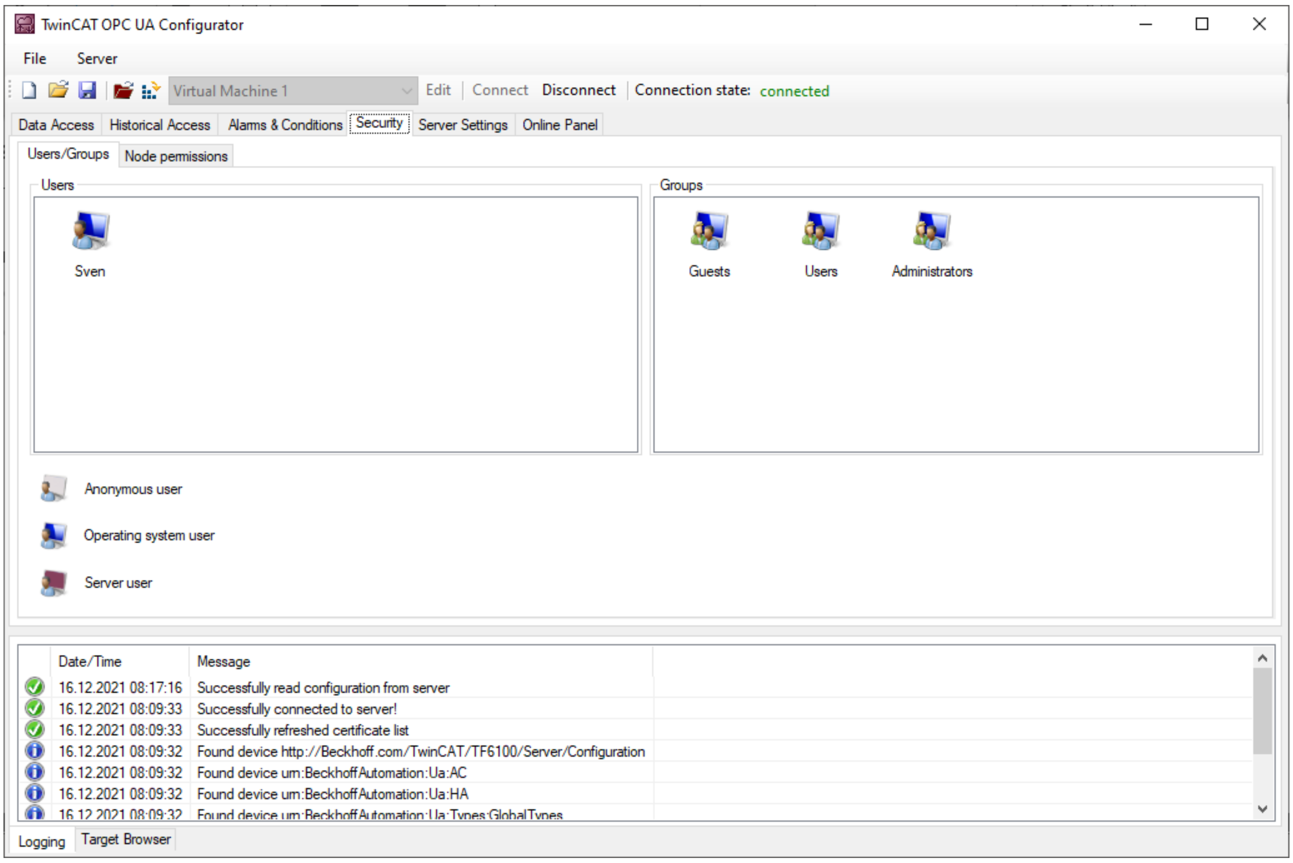
Common name:	TcOpcUaServer@DESKTOP-PDTN3
Thumbprint:	5CE0BC7831D03147D2EA26A3AE36
Application URI:	urn:BeckhoffAutomation:TcOpcUaSer
Valid from:	13.12.2021 14:11:46
Valid to:	08.12.2041 14:11:46

Date/Time	Message
16.12.2021 07:44:14	Successfully refreshed certificate list
16.12.2021 07:44:14	Successfully moved certificate to rejected list
16.12.2021 07:43:32	Successfully refreshed certificate list
16.12.2021 07:43:20	Successfully connected to server!
16.12.2021 07:40:08	Successfully refreshed certificate list
16.12.2021 07:39:09	Successfully refreshed certificate list
16.12.2021 07:39:08	Successfully refreshed certificate list

4.2.2.11 Configuring security settings

Security settings can be made on the server via the **Security** tab. These security settings may include the following items:

- Users and groups
- Access rights for groups to namespaces
- Access rights for groups to individual nodes



```
Date/Time            Message
16.12.2021 08:17:16  Successfully read configuration from server
16.12.2021 08:09:33  Successfully connected to server!
16.12.2021 08:09:33  Successfully refreshed certificate list
16.12.2021 08:09:32  Found device http://Beckhoff.com/TwinCAT/TF6100/Server/Configuration
16.12.2021 08:09:32  Found device urn:BeckhoffAutomation:Ua:AC
16.12.2021 08:09:32  Found device urn:BeckhoffAutomation:Ua:HA
16.12.2021 08:09:32  Found device urn:BeckhoffAutomation:Ua:Types:GlobalTypes
```

Users and groups

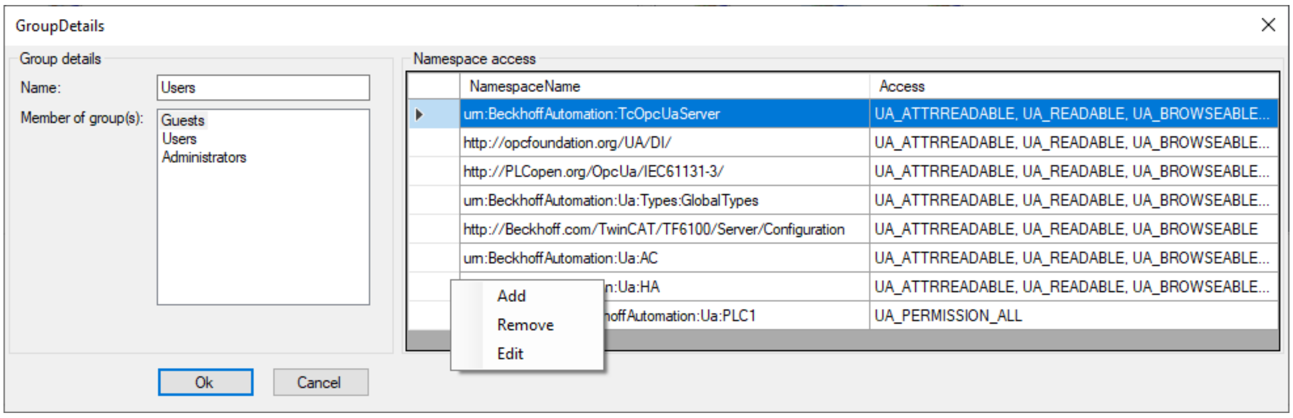
To configure access rights, users and user groups must first be created. Some groups are already predefined when the server is delivered. New users or groups can be added to the configuration via the context menu.

A user can be either the anonymous user, an operating system user or a server user. In any case, we recommend the configuration of operating system users.

A user group can have a so-called **default access** configured. These are access rights to a specific namespace.

Access rights to namespaces

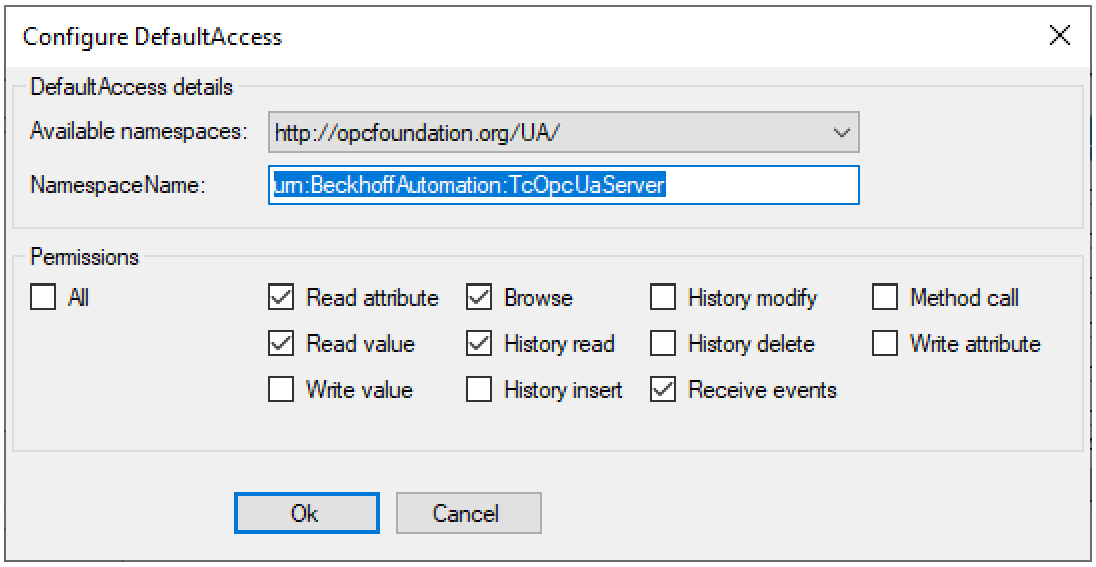
Access rights to certain namespaces can be defined at a user group. In the settings of the group there is a corresponding configuration area, which can be edited via the context menu.



Name: Users

Member of group(s): Guests, Users, Administrators

NamespaceName	Access
urn:BeckhoffAutomation:TcOpcUaServer	UA_ATTRREADABLE, UA_READABLE, UA_BROWSEABLE...
http://opcfoundation.org/UA/DI/	UA_ATTRREADABLE, UA_READABLE, UA_BROWSEABLE...
http://PLCopen.org/OpcUa/IEC61131-3/	UA_ATTRREADABLE, UA_READABLE, UA_BROWSEABLE...
urn:BeckhoffAutomation:Ua:Types:GlobalTypes	UA_ATTRREADABLE, UA_READABLE, UA_BROWSEABLE...
http://Beckhoff.com/TwinCAT/TF6100/Server/Configuration	UA_ATTRREADABLE, UA_READABLE, UA_BROWSEABLE
urn:BeckhoffAutomation:Ua:AC	UA_ATTRREADABLE, UA_READABLE, UA_BROWSEABLE...
n:Ua:HA	UA_ATTRREADABLE, UA_READABLE, UA_BROWSEABLE...
offAutomation:Ua:PLC1	UA_PERMISSION_ALL


Configure DefaultAccess

DefaultAccess details

Available namespaces: http://opcfoundation.org/UA/

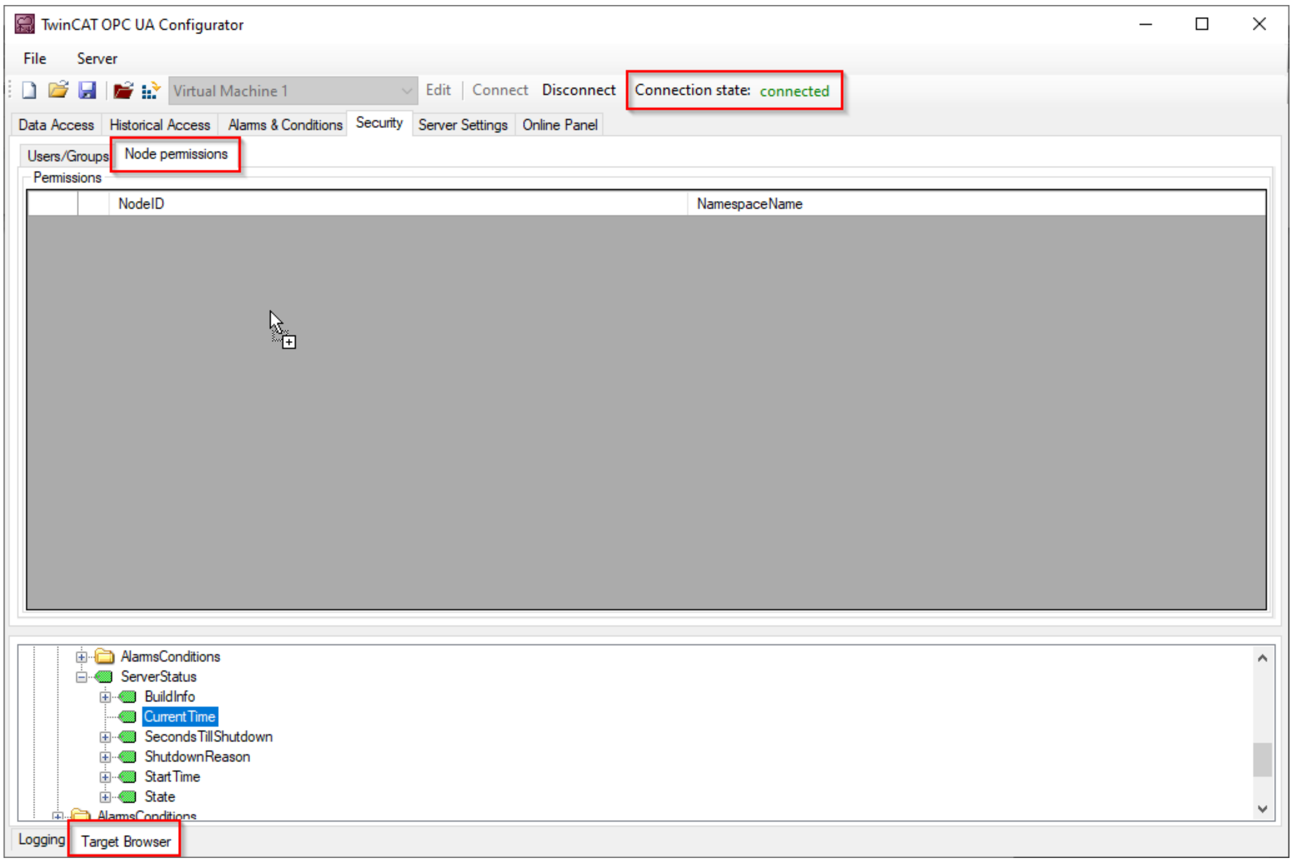
NamespaceName: urn:BeckhoffAutomation:TcOpcUaServer

Permissions

- [ ] All
- [x] Read attribute
- [x] Read value
- [ ] Write value
- [x] Browse
- [x] History read
- [ ] History insert
- [ ] History modify
- [ ] History delete
- [x] Receive events
- [ ] Method call
- [ ] Write attribute

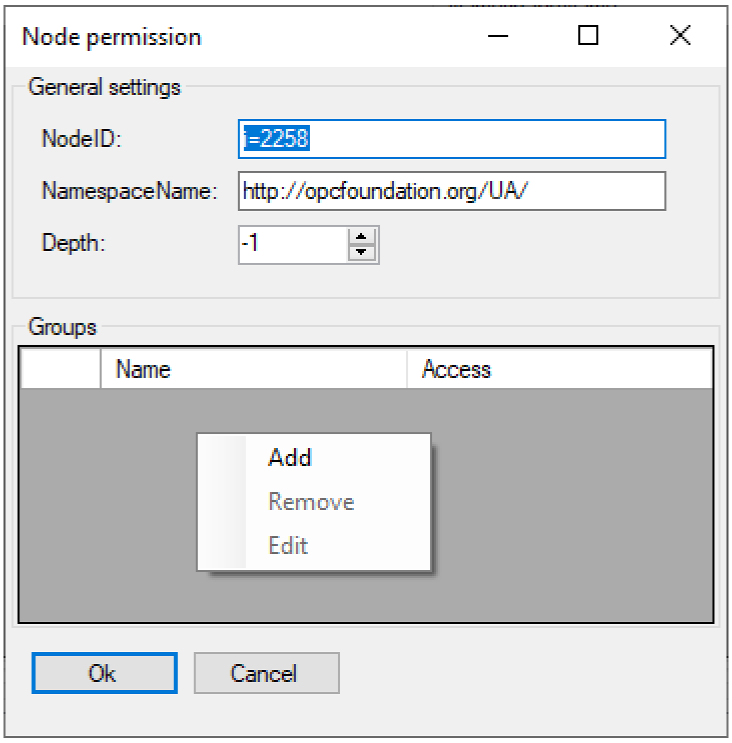
Access rights to individual nodes

The **Node permissions** tab can be used to define access rights to individual nodes and their child elements. You can configure the nodes manually via the context menu or conveniently add them to the configuration by dragging and dropping them from the **Target Browser**, provided you are connected to a server.



Connection state: connected

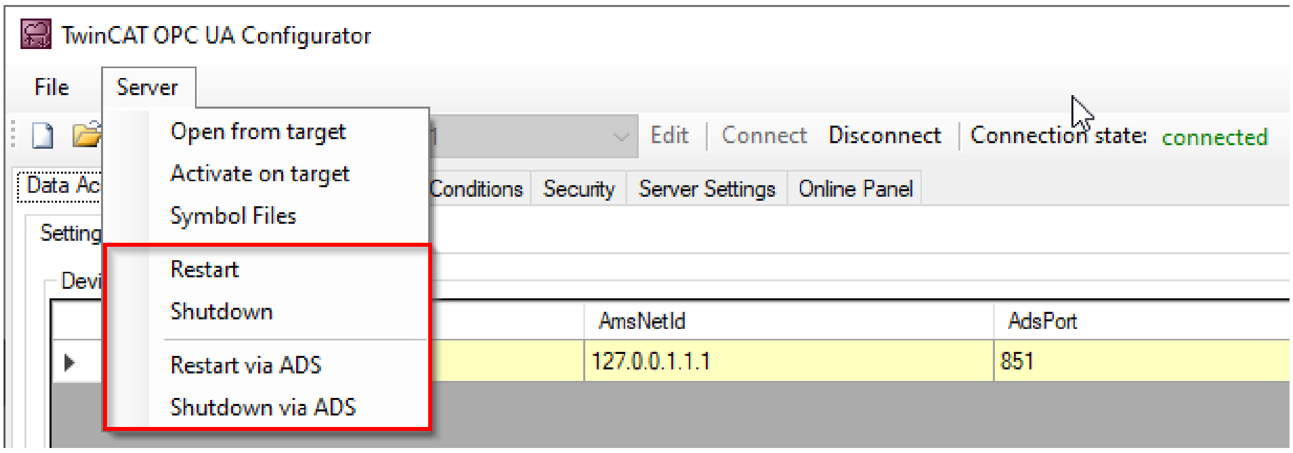
The user groups and access rights of the respective group can then be defined in the node configuration dialog.



You can use the parameter **Depth** to set whether the permissions should be inherited by child elements. The value "-1" indicates that all child elements should inherit the permissions.

4.2.2.12 Restarting the server

A TwinCAT OPC UA Server can be restarted via the **Server** menu. Usually you want to restart the server that is just connected via OPC UA. Alternatively, you can trigger the restart via ADS if you have established an ADS route to the server system.



4.2.2.13 Logging

For an advanced diagnostics you can activate the logging function of the OPC UA Server.

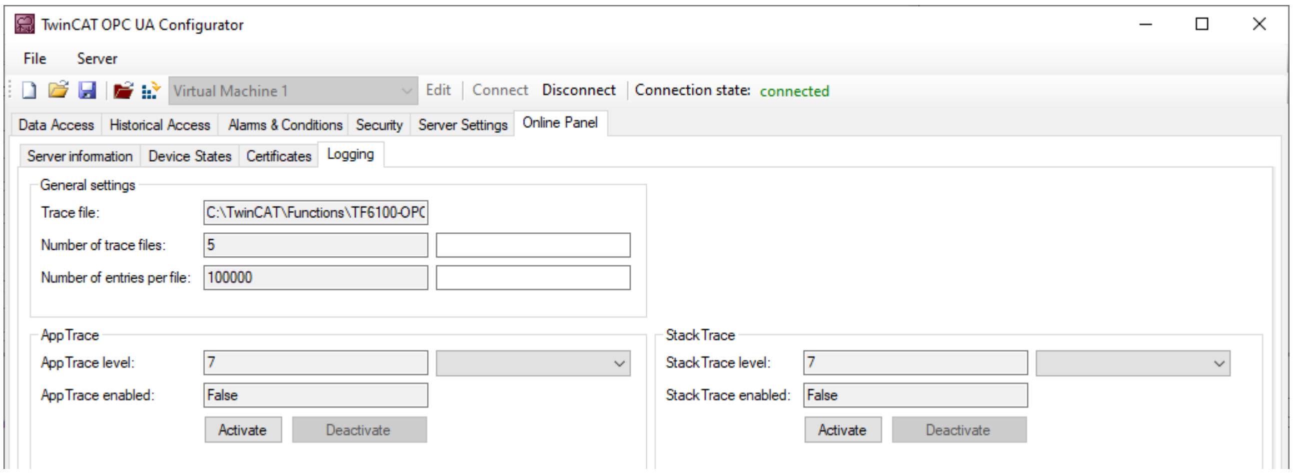
Writing the log file

Activating the logging function on the server causes a log file to be written on the file system. Make sure that there is sufficient storage space available and set the logging parameters accordingly (number of log files, size per log file).

Performance and timing behavior

Activation of the logging function will change the timing behavior of the OPC UA Server. As a result there may be losses of speed, depending on the platform and project.

The server logging functions can be activated via the **Online Panel** tab and the **Logging** section there.



TwinCAT OPC UA Configurator

Virtual Machine 1 | Edit | Connect | Disconnect | Connection state: connected

Data Access | Historical Access | Alarms & Conditions | Security | Server Settings | Online Panel

Server information | Device States | Certificates | Logging

General settings

Trace file: C:\TwinCAT\Functions\TF6100-OPC

Number of trace files: 5

Number of entries per file: 100000

AppTrace

AppTrace level: 7

AppTrace enabled: False

Activate | Deactivate

StackTrace

StackTrace level: 7

StackTrace enabled: False

Activate | Deactivate

Trace Level

In general, the higher the trace level, the more detailed (and more) data is written, but the more load is also placed on the server application, which changes the timing behavior accordingly. Please therefore only activate logging in the event of diagnostics and in consultation with Beckhoff Support.

Activate App Trace

In most cases it is sufficient to create a so-called "AppTrace". This logs information from the server application. To activate the AppTrace, please enter the number of TraceFiles and the number of entries per TraceFile in the corresponding text fields. Then select a trace level and click the button to activate the AppTrace. The values in the gray text boxes represent the current settings on the server.

Activate Stack Trace

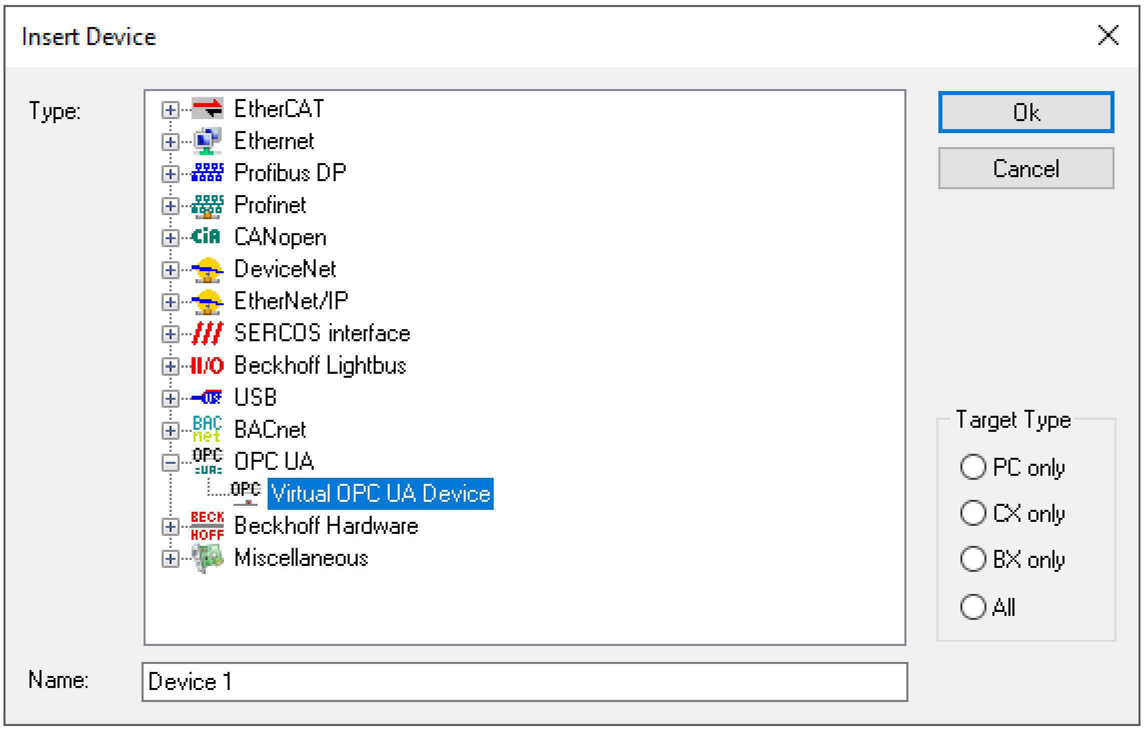
In a few cases it is also necessary to create a so-called "StackTrace", whereby information from the OPC UA stack is logged. To activate the StackTrace please enter the number of TraceFiles as well as the number of entries per TraceFile into the corresponding text boxes. Then select a trace level and click on the button to activate the StackTrace. The values in the gray text boxes represent the current settings on the server.

4.3 Client I/O

4.3.1 Overview

This function requires TwinCAT 3.1 Build 4022.4 or higher.

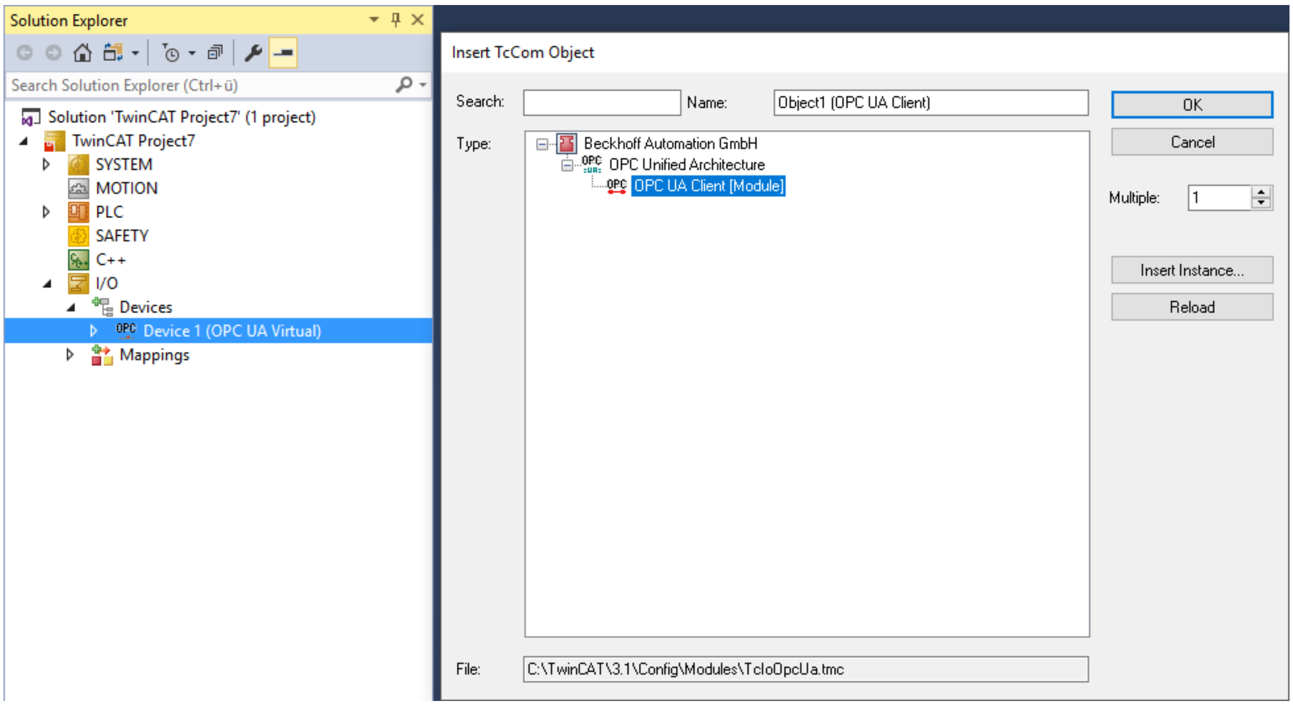
The TwinCAT OPC UA Client is also integrated as an I/O driver and is available as such in the TwinCAT I/O configuration. This not only shortens the engineering time required to establish the OPC UA connection to a remote server, it also enables the implementation of many application cases, such as the creation of protocol bridges that are simple to configure.


Insert Device

Type:
- EtherCAT
- Ethernet
- Profibus DP
- Profinet
- CANopen
- DeviceNet
- EtherNet/IP
- SERCOS interface
- Beckhoff Lightbus
- USB
- BACnet
- OPC UA
  - Virtual OPC UA Device
- Beckhoff Hardware
- Miscellaneous

Name: Device 1

First of all, add a virtual device container to the configuration and then an OPC UA Client object.


Solution Explorer

Search Solution Explorer (Ctrl+ü)

- Solution 'TwinCAT Project7' (1 project)
  - TwinCAT Project7
    - SYSTEM
    - MOTION
    - PLC
    - SAFETY
    - C++
    - I/O
      - Devices
        - Device 1 (OPC UA Virtual)
      - Mappings

Insert TcCom Object

Search:
Name: Object1 (OPC UA Client)

Type:
- Beckhoff Automation GmbH
  - OPC Unified Architecture
    - OPC UA Client [Module]

Multiple: 1

File: C:\TwinCAT\3.1\Config\Modules\TcIoOpcUa.tmc

After you have created the OPC UA I/O Client object, you can configure its connection to an OPC UA Target Server and add variables, methods and structures to the process image. The process of data exchange with a target server is reduced to a simple method of mapping process variables.

Requirements

Products	Setup versions	Target platform
TF6100	4.x.x	IPC or CX (x86, x64, ARM)

4.3.2 Quick start

The following instructions enable you to make a quick start with the OPC UA I/O Client. In the instructions a PLC project will be created that enables a variable via the OPC UA Server (for the simulation of a UA Server) and reads the enabled variable in again via the OPC UA I/O Client. This can of course be divided between two projects and two controllers.

Enabling variables for a server

For simulation, the OPC UA Server should be used in this quick start.

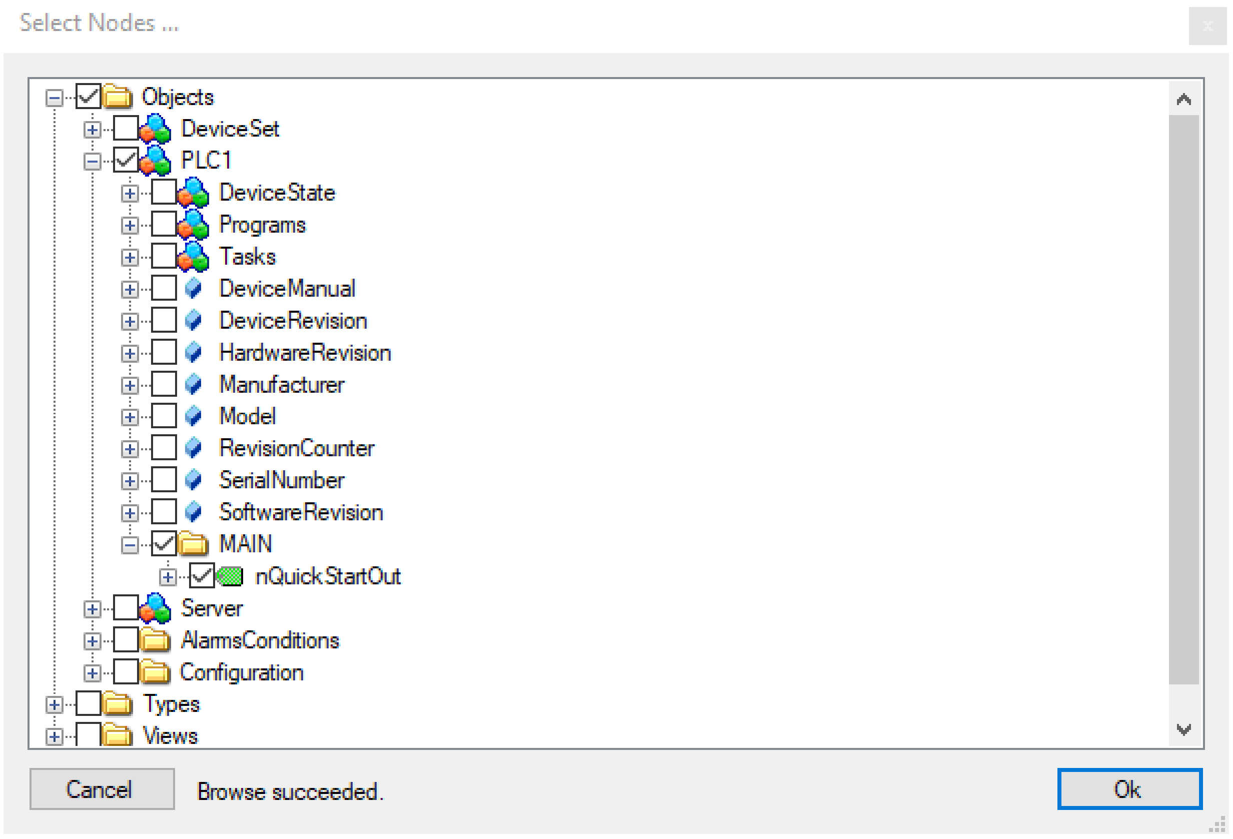
1. Create a new TwinCAT project and add a new PLC project to the project.
2. Create a new variable "nQuickStartOut" of data type INT in the PLC project. This should be the variable that is enabled via the OPC UA Server.
3. Set the OPC UA attribute [▸ 49] for this variable in order to be able to enable the variable via the OPC UA Server.
4. Activate the check box for the download of the TMC file.
5. Activate the configuration.

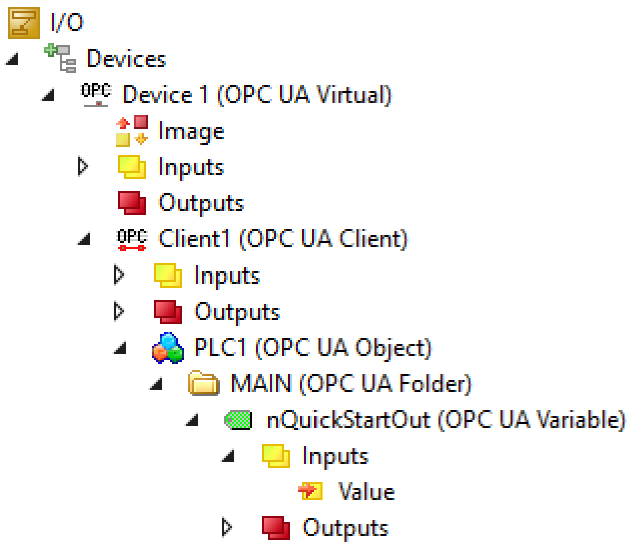
⇨ The variable is enabled for the server.

Configuring the I/O UA Client

1. In the same PLC project, create another variable "nQuickStartIn" of data type INT and define it as an input variable (AT%I*). This should be the variable that the OPC UA Client I/O device links to the server variable through mapping.
2. Compile the project so that the input variable is available in the PLC process image.
3. Add a new "Virtual OPC UA Device".
4. Add a new "OPC UA Client" to the device and open its settings.
5. Navigate to the **Settings** tab. Enter the server URL of the OPC UA Server. In this sample this is "opc.tcp://localhost:4840".
6. Click **Add Nodes**.

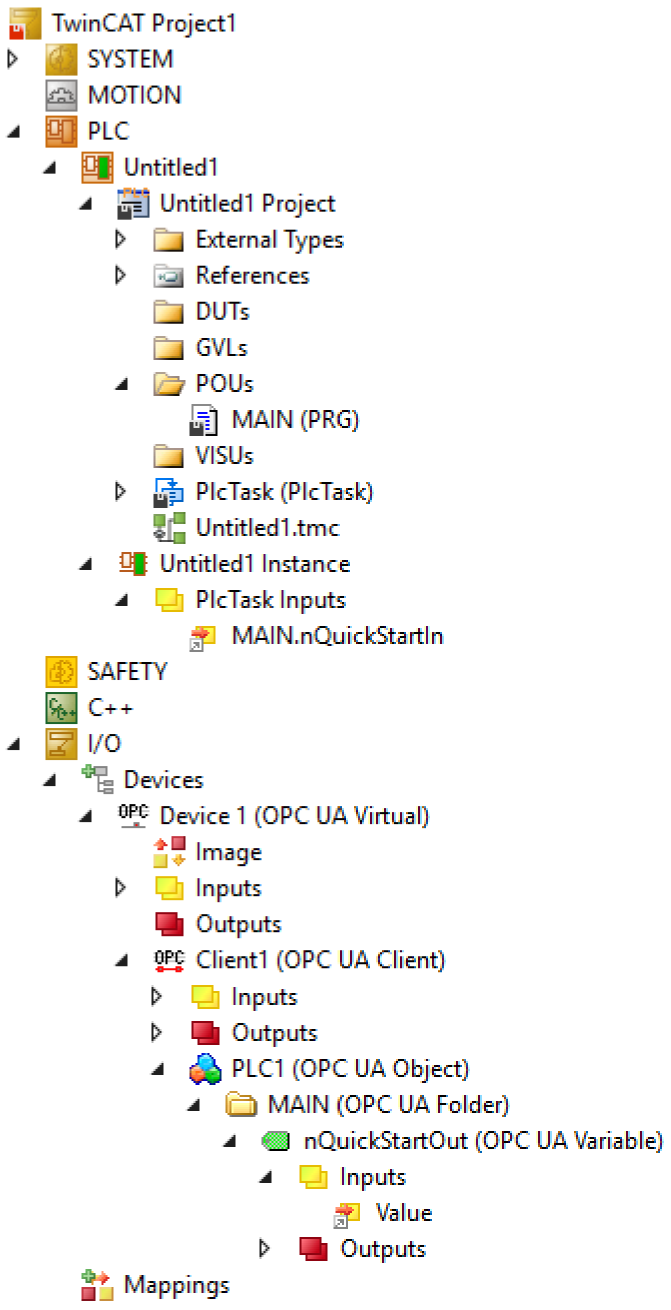
7. Navigate to the shared variable "nQuickStartOut" on the OPC UA Server (located under PLC1 > MAIN) and select it. The variable is now added to the process image of the "Virtual OPC UA Device".





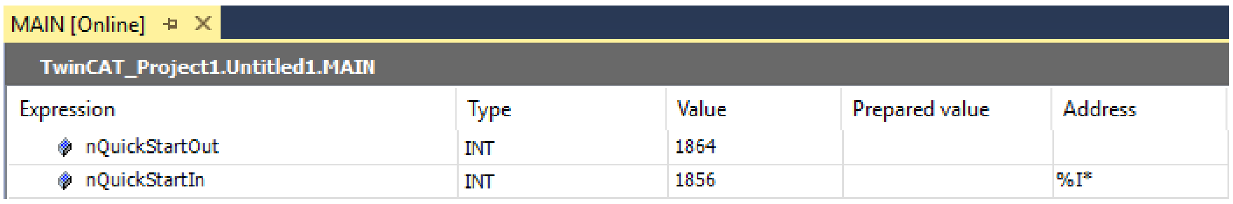
8. Expand the newly added variable in the process image and double-click on **Input > Value**. Link the variable to the variable "nQuickStartIn" from the PLC process image.
9. Activate the configuration.

⇨ You should now find the configuration shown below. The two process image variables nQuickStartIn and nQuickStartOut are linked to each other.



```
TwinCAT Project1
  SYSTEM
  MOTION
  PLC
    Untitled1
      Untitled1 Project
        External Types
        References
        DUTs
        GVLs
        POUs
          MAIN (PRG)
        VISUs
        PlcTask (PlcTask)
        Untitled1.tmc
      Untitled1 Instance
        PlcTask Inputs
          MAIN.nQuickStartIn
  SAFETY
  C++
  I/O
    Devices
      Device 1 (OPC UA Virtual)
        Image
        Inputs
        Outputs
        Client1 (OPC UA Client)
          Inputs
          Outputs
          PLC1 (OPC UA Object)
            MAIN (OPC UA Folder)
              nQuickStartOut (OPC UA Variable)
                Inputs
                  Value
                Outputs
    Mappings
```

10. After activating the configuration, log in to the PLC program.

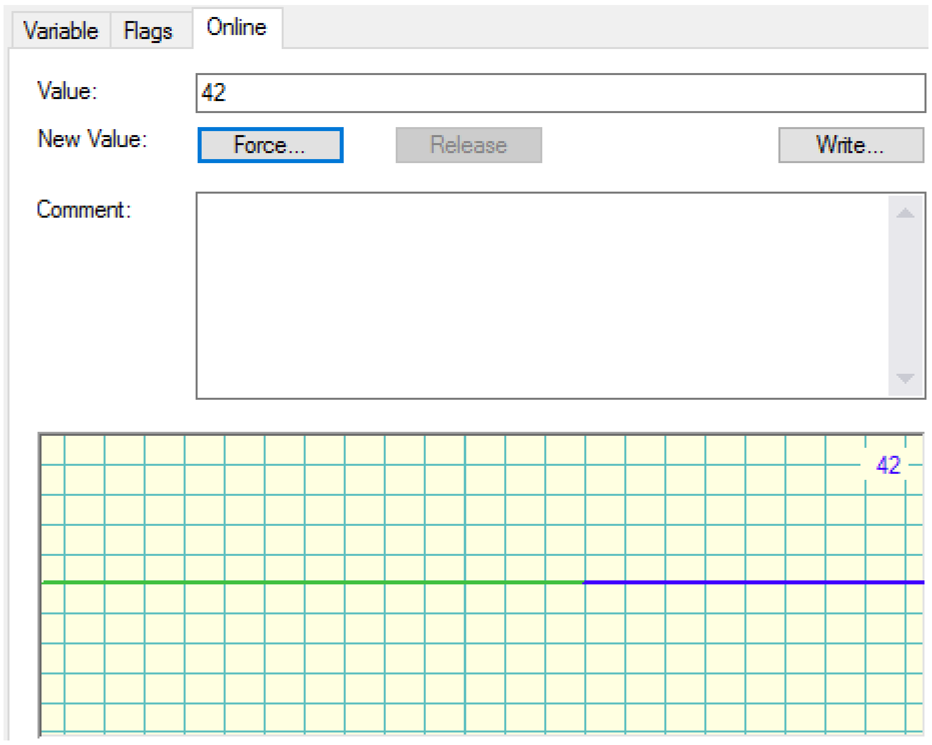


MAIN [Online]

TwinCAT_Project1.Untitled1.MAIN

Expression	Type	Value	Prepared value	Address
nQuickStartOut	INT	1864		
nQuickStartIn	INT	1856		%I*

You can also view the current value in the process image mapping.



Variable | Flags | Online

Value: 42

New Value: Force... Release Write...

Comment:

11. You can use the settings on the OPC UA Client to modify parameters such as the sampling rate.

Requirements

Products	Setup versions	Target platform
TF6100	4.x.x	IPC or CX (x86, x64, ARM)

4.3.3 Supported data types

The OPC UA Client enables direct access to an OPC UA Server from a real-time logic. For the reading and writing of data, the data type of the OPC UA node must be assigned to the TwinCAT environment (mapping). This assignment is described below.

Basic data types

The assignment of the basic data types is described in PLCopen OPC UA Information Model for IEC 61131-3.

OPC UA data type	PLC data type
Boolean	BOOL
SByte	SINT
Byte	USINT
Int16	INT
Int32	DINT
String	STRING
USint	BYTE
Float	REAL
Double	LREAL
UInt16	UINT
UInt32	UDINT
Int64	LINT
UInt64	ULINT
DateTime	DT

You can use the mapping both on the OPC UA Client PLCopen function block and on the OPC UA I/O Client.

Derived data types

OPC UA defines basic data types. Other data types are derived from these.

On the client side, access to all UA data types is only possible with PLC data types. Data types derived from the basic data types are also supported from TcOpcUaClient version 2.1.0.36 or higher.

Currently supported non-base data types on the client side are:

- Counter (use UDINT in PLC)

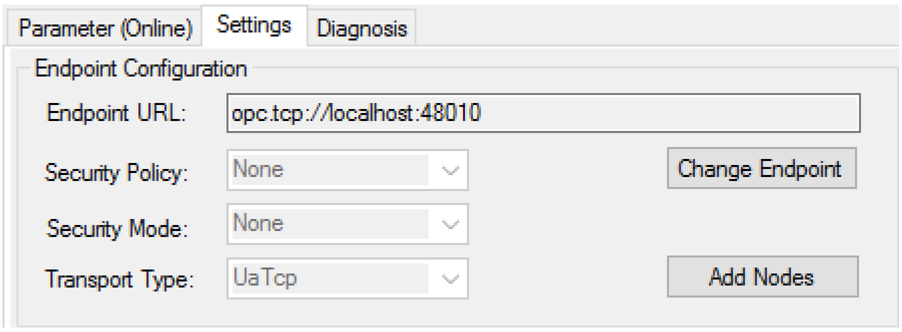
In addition, the OPC UA I/O Client supports structured data types (StructuredTypes) if the structured data type does not incorporate any non-supported data type (e.g. in a member variable). The following list provides an overview of the data types that are not currently supported:

- ByteString
- NodeID
- LocalizedText (and thus AnalogItemTypes, DiscreteItemTypes)

4.3.4 Adding nodes of a Server

To add nodes of an OPC UA Target Server to the process image of the OPC UA I/O Client, you can use the namespace browser that is integrated in the I/O Client.

Open the **Settings** tab of the client configuration and click on the **Add Nodes** button to open the namespace browser.



Endpoint Configuration

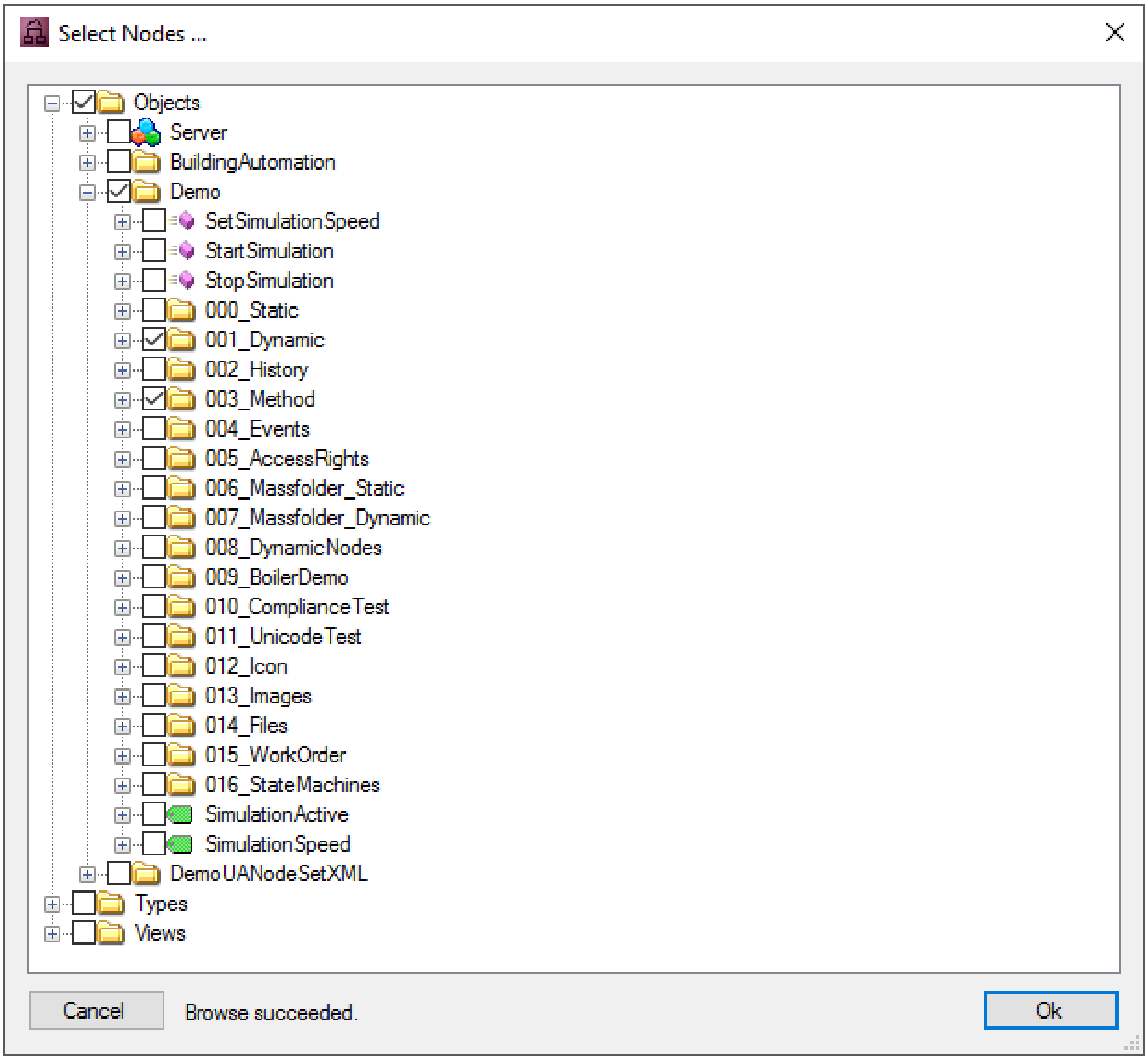
Endpoint URL: opc.tcp://localhost:48010

Security Policy: None

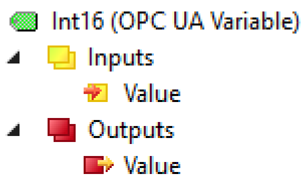
Security Mode: None

Transport Type: UaTcp

The namespace browser dialog automatically establishes a connection with the Target Server by using the specified connection parameters.



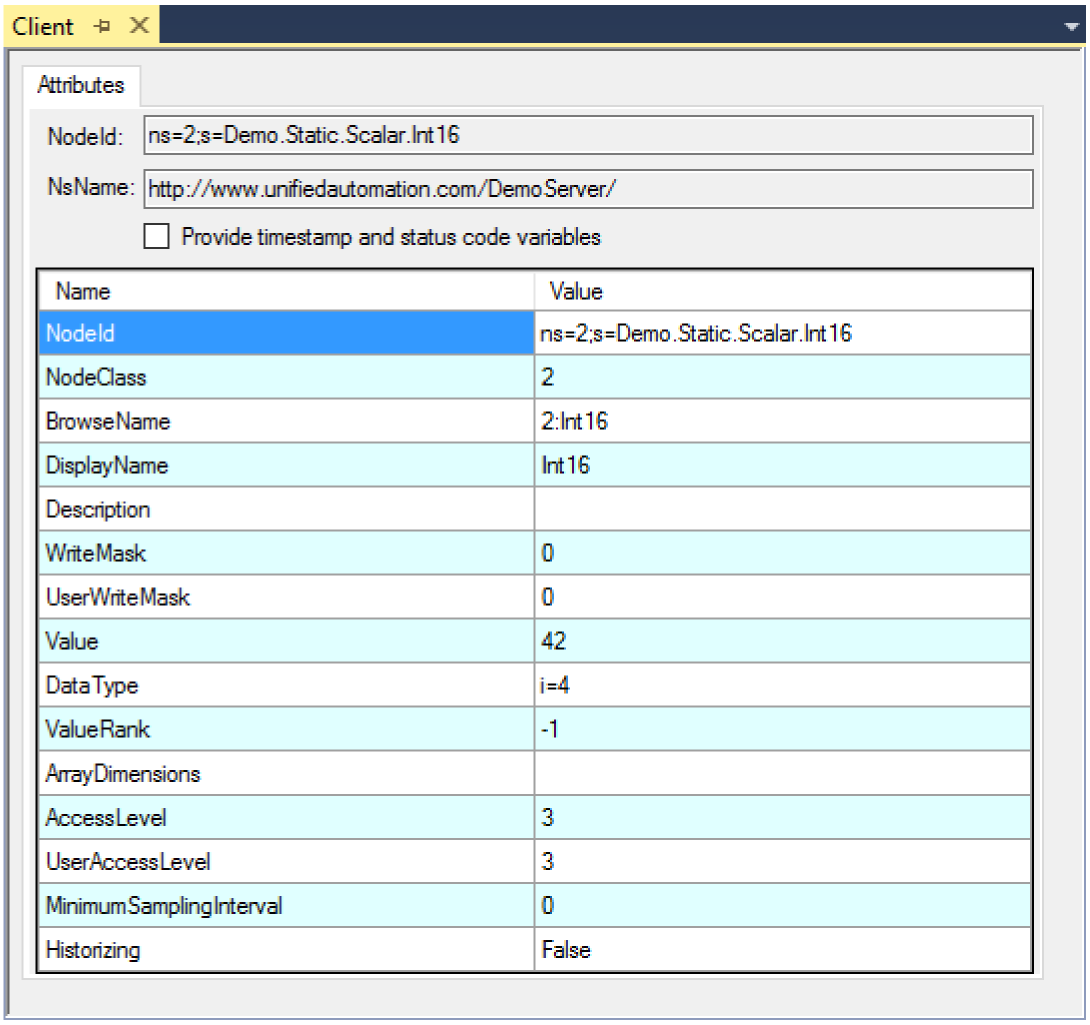
Within the namespace browser you can activate and deactivate the check boxes in order to add nodes to the process image or remove them from it. Each selected node is shown in the process image as an input or output variable.



The input variable reads data from the node and contains the last value received from the node. The output variable writes data to the node if a value has been set.

4.3.5 Node attributes

If you double-click on a node in the process image, you will see the UA attributes with their current values as they were at the time of opening the window.



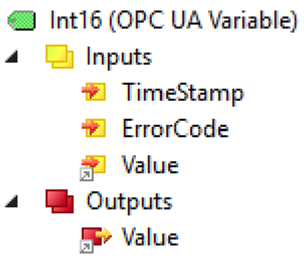
NodeId: ns=2;s=Demo.Static.Scalar.Int16

NsName: http://www.unifiedautomation.com/DemoServer/

☐ Provide timestamp and status code variables

Name	Value
NodeId	ns=2;s=Demo.Static.Scalar.Int16
NodeClass	2
BrowseName	2:Int16
DisplayName	Int16
Description	
WriteMask	0
UserWriteMask	0
Value	42
DataType	i=4
ValueRank	-1
ArrayDimensions	
AccessLevel	3
UserAccessLevel	3
MinimumSamplingInterval	0
Historizing	False

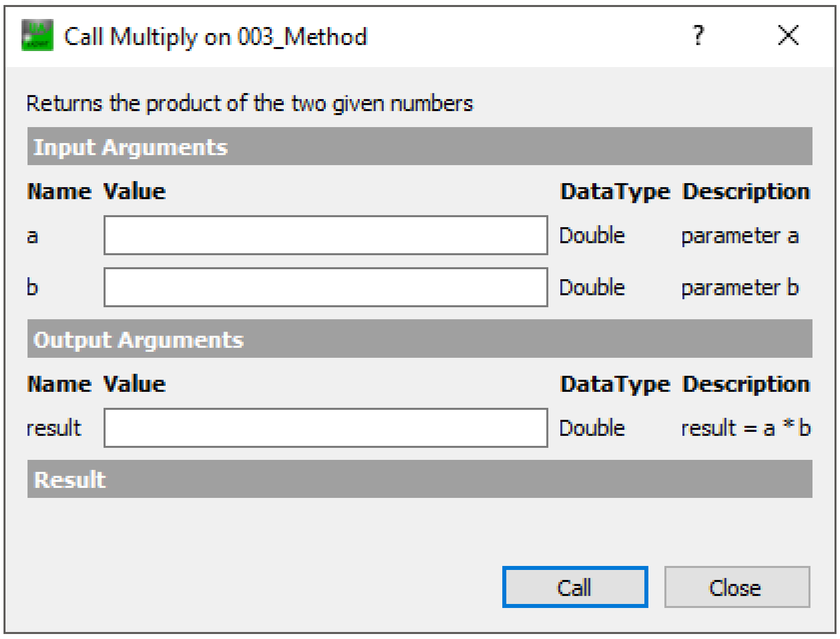
Further variables that can be used for diagnostic purposes are added to the process image using the check box **Provide timestamp and status code variables**.



4.3.6 Method call

The OPC UA I/O Client supports the call of server methods. You can add a method to the process image like any other variable. The "input arguments" of the method are then available as output variables in the process image, whereas the "output arguments" are added as input variables. Additional input and output variables, e.g. bExecute, bBusy, bError, are added to the process image so that the method can be called.

Sample: Method on the server



Call Multiply on 003_Method

Returns the product of the two given numbers

Input Arguments

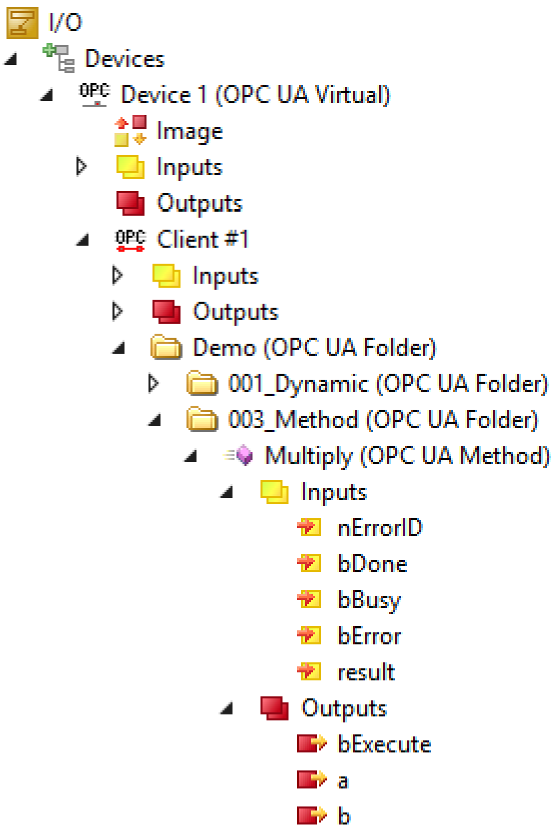
Name	Value	DataType	Description
a		Double	parameter a
b		Double	parameter b

Output Arguments

Name	Value	DataType	Description
result		Double	result = a * b

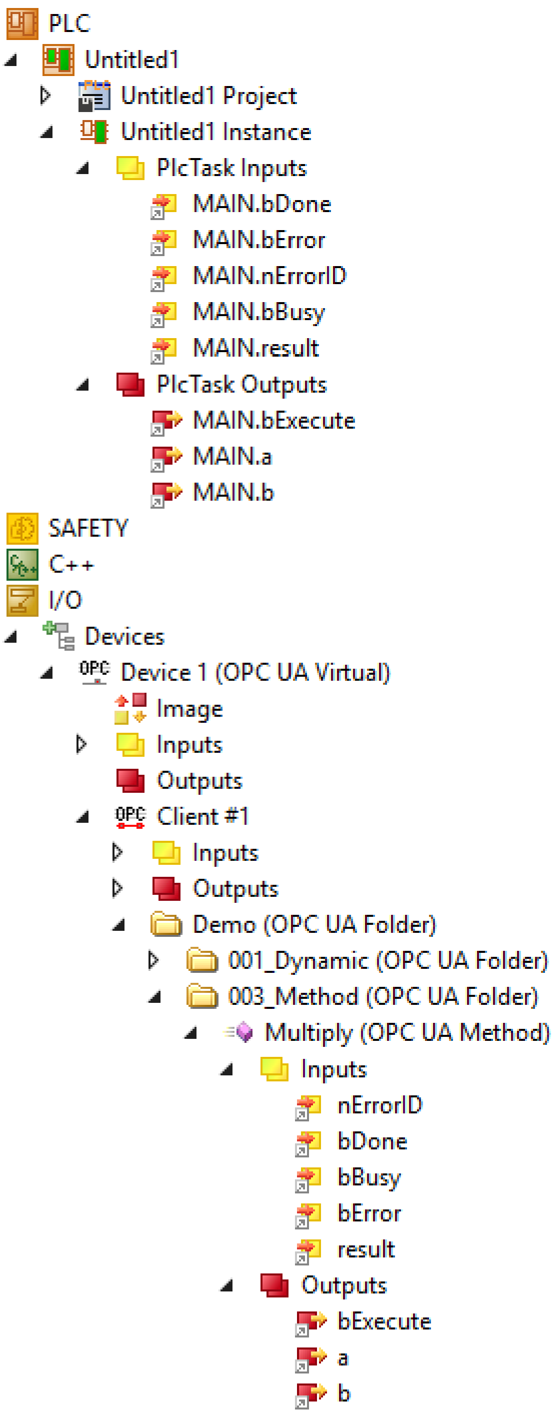
Result

Sample: Method after addition to the process image



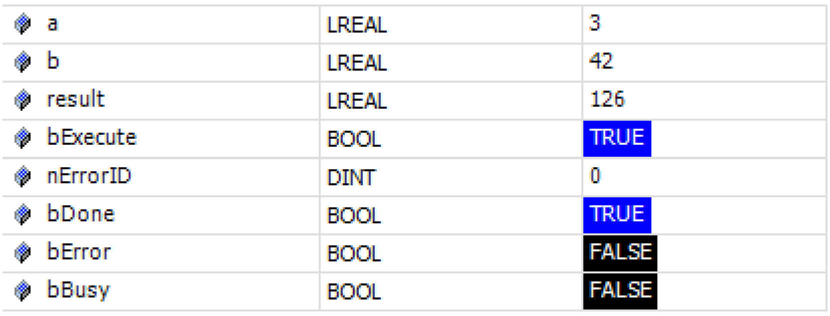
```
I/O
  Devices
    Device 1 (OPC UA Virtual)
      Image
      Inputs
      Outputs
      Client #1
        Inputs
        Outputs
        Demo (OPC UA Folder)
          001_Dynamic (OPC UA Folder)
          003_Method (OPC UA Folder)
            Multiply (OPC UA Method)
              Inputs
                nErrorID
                bDone
                bBusy
                bError
                result
              Outputs
                bExecute
                a
                b
```

You can then create a mapping between the input/output variables and the PLC variables.



Calling of a method

To call a method, set the output variable bExecute to TRUE. You can check whether the method call has been completed and whether it was successful via the input variables nErrorID, bDone, bBusy and bError.

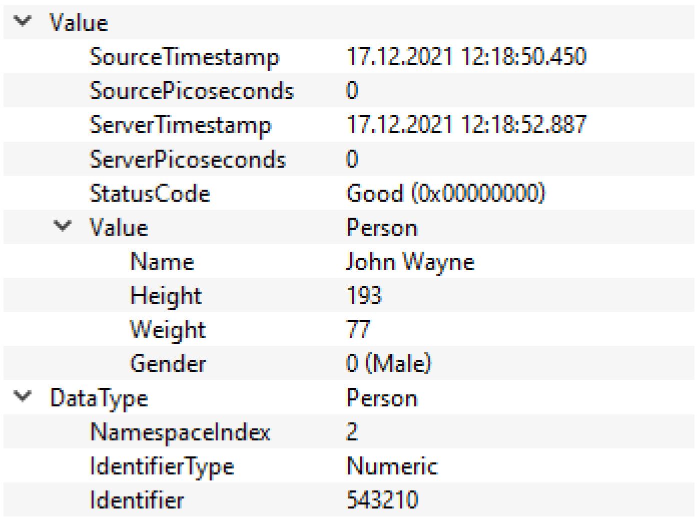


a	LREAL	3
b	LREAL	42
result	LREAL	126
bExecute	BOOL	TRUE
nErrorID	DINT	0
bDone	BOOL	TRUE
bError	BOOL	FALSE
bBusy	BOOL	FALSE

4.3.7 StructuredTypes

The OPC UA I/O Client supports read/write procedures with structured data types (StructuredTypes). You can also add StructuredTypes to the process image like any other variable. When adding a StructuredType to the process image, the type to be parsed is added to the TwinCAT type system so that, for example, it can simply be used by a PLC application.

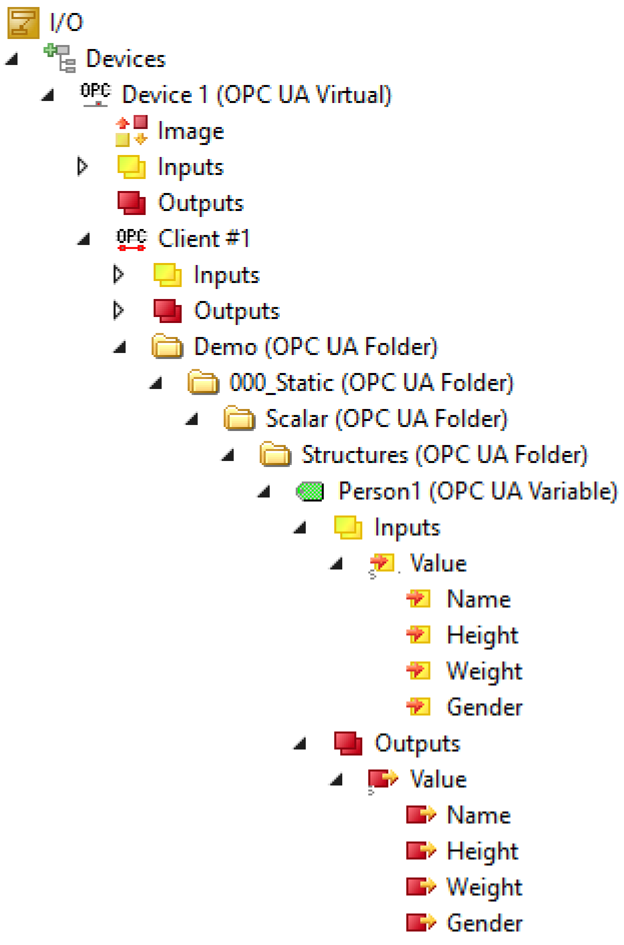
Example: StructuredType on the server



Value	
SourceTimestamp	17.12.2021 12:18:50.450
SourcePicoseconds	0
ServerTimestamp	17.12.2021 12:18:52.887
ServerPicoseconds	0
StatusCode	Good (0x00000000)
Value	Person
Name	John Wayne
Height	193
Weight	77
Gender	0 (Male)
DataType	Person
NamespaceIndex	2
IdentifierType	Numeric
Identifier	543210

In this example the server contains a node of the structured data type "Person", which contains various member variables (Name, Height, Weight, Gender).

Example: StructuredTypes in the process image

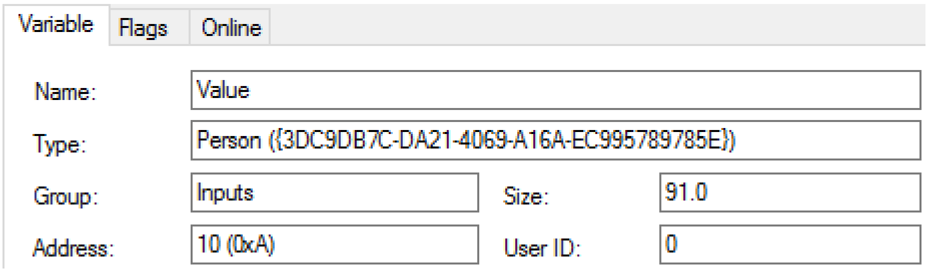


```
I/O
  Devices
    Device 1 (OPC UA Virtual)
      Image
      Inputs
      Outputs
      Client #1
        Inputs
        Outputs
        Demo (OPC UA Folder)
          000_Static (OPC UA Folder)
            Scalar (OPC UA Folder)
              Structures (OPC UA Folder)
                Person1 (OPC UA Variable)
                  Inputs
                    Value
                      Name
                      Height
                      Weight
                      Gender
                  Outputs
                    Value
                      Name
                      Height
                      Weight
                      Gender
```

After you have added a node to the process image, the process image contains the node and also the structural information of the type, e.g. whether individual member variables of the node should be read or written.

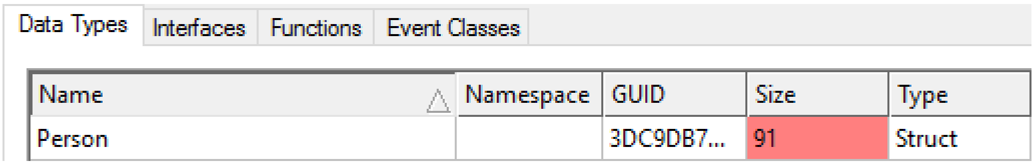
StructuredTypes in the TwinCAT type system

The data type is added to the type system of TwinCAT. The "Value" tree items then have this data type.



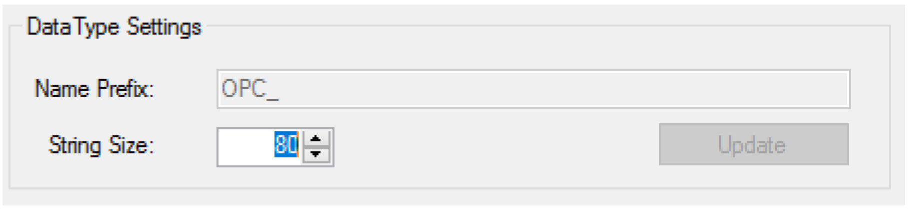
Name:	Value		
Type:	Person ({3DC9DB7C-DA21-4069-A16A-EC995789785E})		
Group:	Inputs	Size:	91.0
Address:	10 (0xA)	User ID:	0

You can also view the data type in the TwinCAT type system under SYSTEM > Type System.



Name	Namespace	GUID	Size	Type
Person		3DC9DB7...	91	Struct

To distinguish the data type from other data types you can add a prefix in the settings of the OPC UA Client.

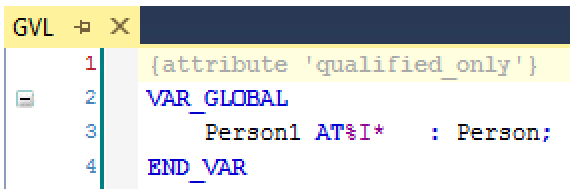


DataType Settings

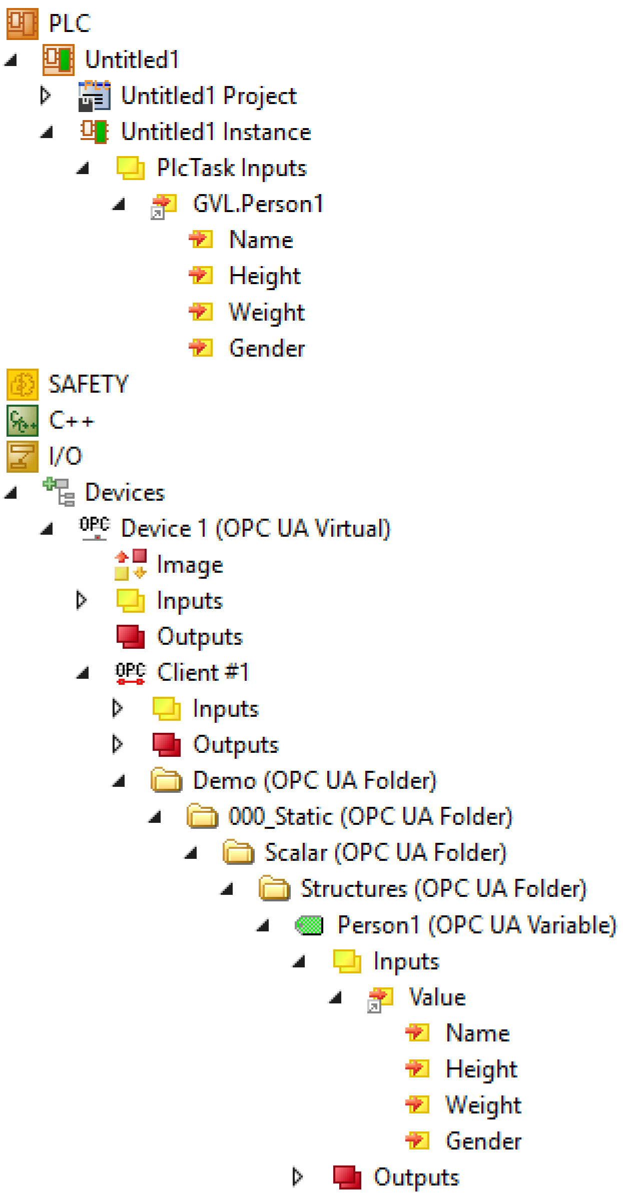
Name Prefix:	OPC_
String Size:	80

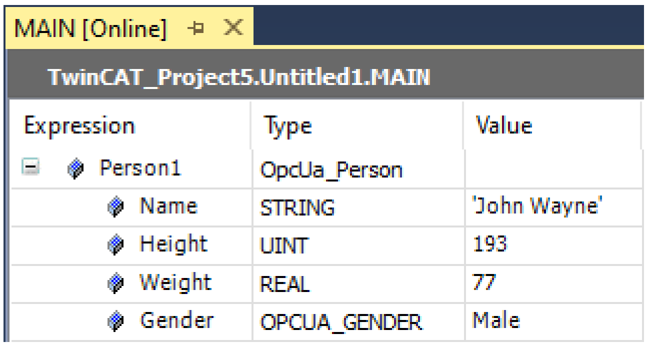
Mapping a StructuredType

Since every StructuredType is added to the TwinCAT type system, the mapping of the variables is simple. Create an input/output variable of this data type and subsequently a mapping.



```
{attribute 'qualified_only'}
VAR_GLOBAL
    Person1 AT%I*    : Person;
END_VAR
```




MAIN [Online]

TwinCAT_Project5.Untitled1.MAIN

Expression	Type	Value
Person1	OpcUa_Person	
Name	STRING	'John Wayne'
Height	UINT	193
Weight	REAL	77
Gender	OPCUA_GENDER	Male

4.3.8 Data recording

Data collection is configured on a per-device basis. The configuration can be opened by double-clicking on the OPC UA Client device. The **Process Data Configuration** area shows various parameters for setting the different modes of data collection from a server.

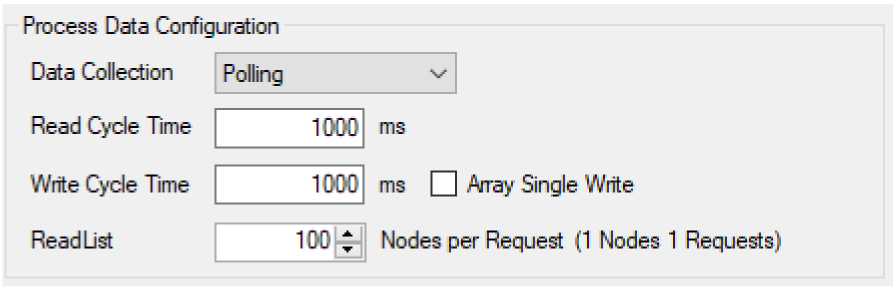
The OPC UA client device offers three different modes for data collection. In addition to cyclical reading of data, this includes reading via a trigger variable and the use of subscriptions.

Cyclic read/write

One of the possible types of data collection is cyclic reading and writing. Time intervals are defined for both reading and writing. You can also specify how many variables are to be read in one read command.

Writing variables in polling and subscription mode

When writing, please note that writing only takes place when the value changes. If no value change has taken place in the configured variables at the end of a cycle, no new value is written.


Process Data Configuration

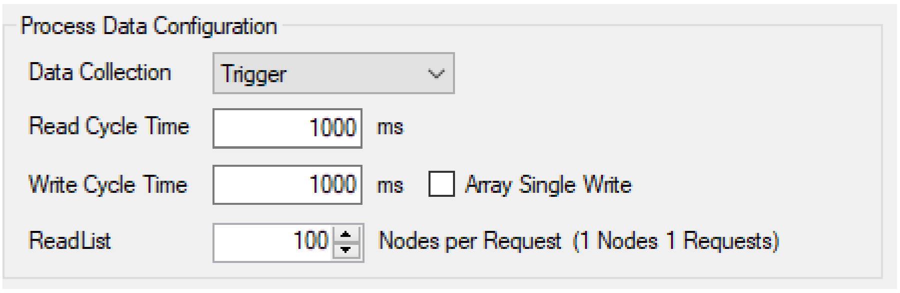
Data Collection: Polling
Read Cycle Time: 1000 ms
Write Cycle Time: 1000 ms ☐ Array Single Write
ReadList: 100 Nodes per Request (1 Nodes 1 Requests)

Parameter	Description
Read Cycle Time	Specifies how fast variables are cyclically read.
Write Cycle Time	Defines how often a write command is triggered on the OPC UA channel. If a variable value changes several times within a specific cycle time, only the last value is written to the OPC UA channel. If no configured value has changed in the cycle time, no write command is triggered.
ReadList	Read commands on the OPC UA channel are bundled to save bandwidth. This parameter specifies how many variables are collected in a single read command on the OPC UA channel. The labeling behind it indicates how many read commands arise from the current configuration.
Array Single Write	If a value in an array is changed, a write operation is only carried out for this value on the OPC UA channel when activated. If not activated, the entire array is always written.

Read and write via trigger variables

In addition, there is an option to trigger reading and writing via trigger variables. For each OPC UA Client device there is a trigger variable (it can be found under Outputs/Control/Execute) that can be connected to a variable from the PLC and set if required. This option is suitable, for example, if data is only to be read from an OPC UA Server when a certain event occurs in the PLC. If the trigger variable remains permanently set, the data collection type behaves in the same way as the cyclic configuration.

When writing, on the other hand, a value is written in each cycle if the trigger variable is set. No change in value is considered here.

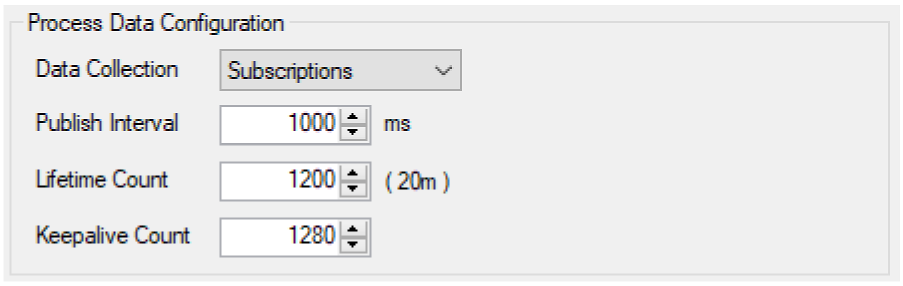

Process Data Configuration

Data Collection: Trigger
Read Cycle Time: 1000 ms
Write Cycle Time: 1000 ms ☐ Array Single Write
ReadList: 100 Nodes per Request (1 Nodes 1 Requests)

Reading and writing using subscriptions

The third and last way of data collection is to use subscriptions. The I/O client registers a subscription with the connected OPC UA Server. The parameters described below can be specified for Publish Interval, Lifetime Count and Keepalive Count.

Subscription mode is primarily intended for reading variables. If you write values in this mode, the same behavior applies as for cyclical writing (see above).


Process Data Configuration

Data Collection	Subscriptions	
Publish Interval	1000	ms
Lifetime Count	1200	(20m)
Keepalive Count	1280	

Parameter	Description
Publish Interval	After the specified time the connected OPC UA Server checks where there are new notification packets for the Client. If several value changes occur in a publishing interval, only the last value is transferred.
Lifetime Count	The OPC UA Client is responsible for sending a PublishRequest to the server. In the PublishResponse, the server returns the respective notification packages. The Lifetime Count indicates after how many failed PublishRequests from the client the server deletes the subscription. The calculated duration is shown in brackets (in the sample 1200 multiplied by 1000 ms = 20 minutes).
Keepalive Count	If the server does not have new notification packets for the client, it will not return any data. The Keepalive Count indicates after how many missed messages the server would send an empty message to the client to indicate that it is still active and the subscription is still in place.

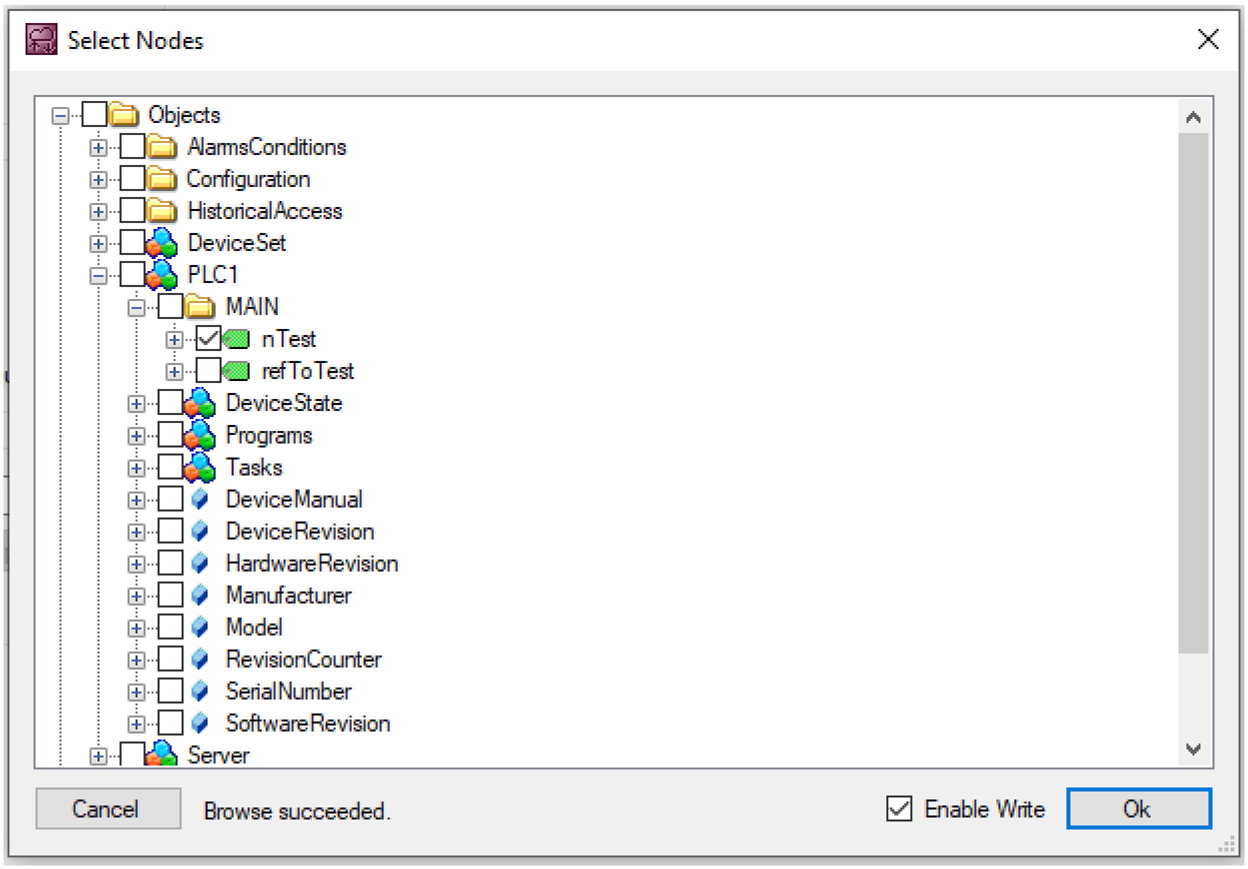
4.3.9 Writing variables

To enable the writing of variables, several conditions must be met:

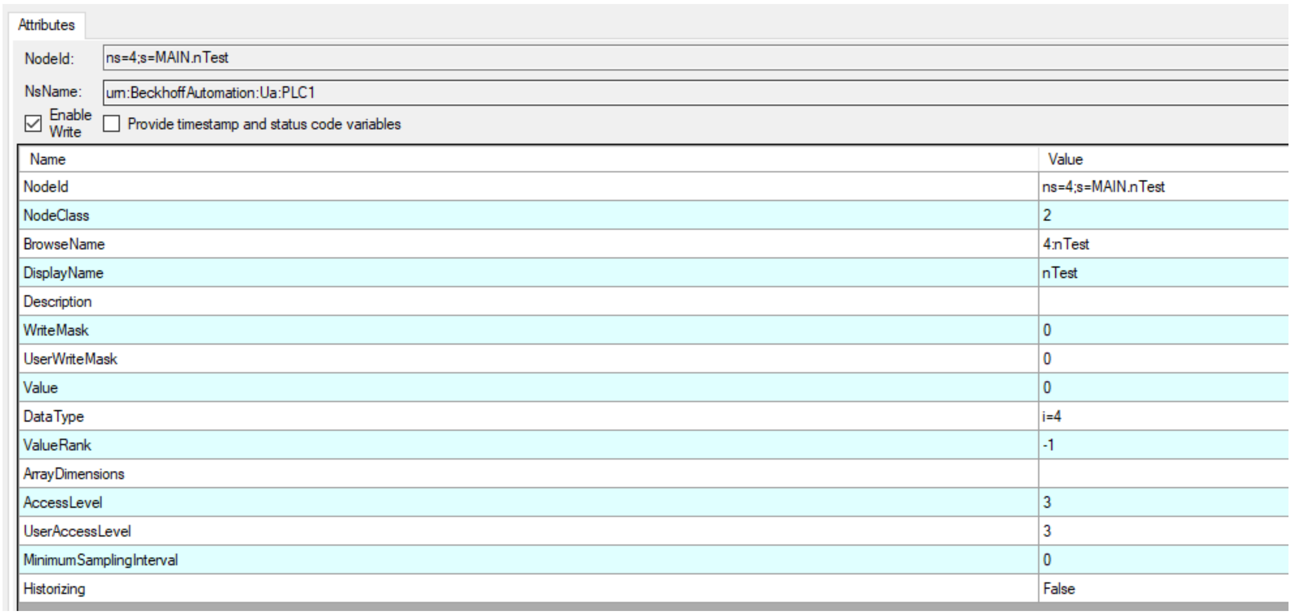
1. The flag "Enable Write" must be set for the variable. This can be done either during the adding process via the button **Add Nodes** or afterwards in the parameter settings of the variable.
2. Before a write command, the "Write Enable" output for the I/O client must be enabled globally. Only then are the write commands generated.
3. In the "Polling" and "Subscriptions" modes, writing only takes place after a value change within the I/O client. This is particularly important for server restarts. After a server restart, values written once in these modes are not automatically written again, as another OPC UA Client could have written a new value in the meantime and this would then be overwritten by an "old" value.

Setting Enable Write for a variable

In order to add not only an input (Read) but also an output (Write) element for a variable in the process image, it must be enabled explicitly. This can be done by using the **Add Nodes** dialog while adding the variables, for example:



Alternatively, this setting can be enabled/disabled at a later stage via the configuration parameters of the variables in the process image.


Attributes

NodeId: ns=4;s=MAIN.nTest

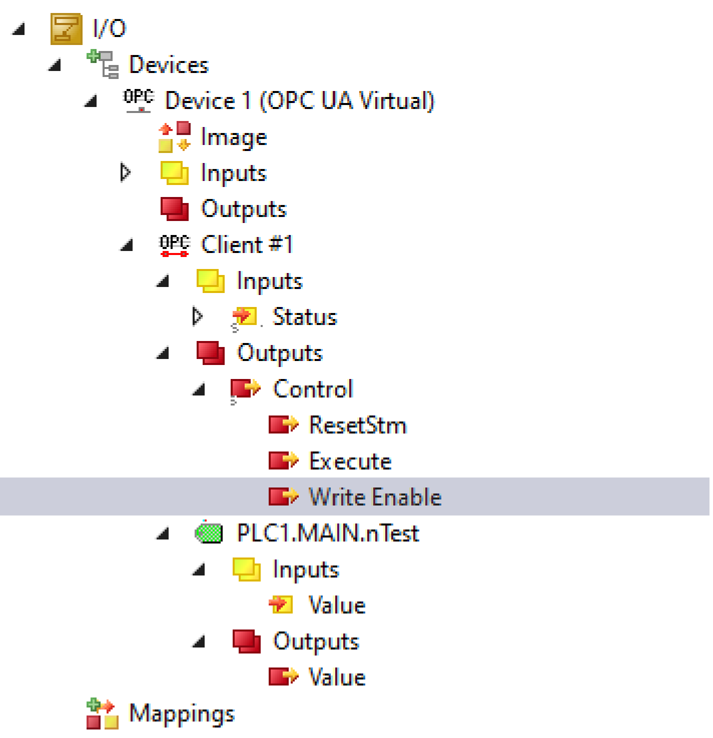
NsName: urn:BeckhoffAutomation:Ua:PLC1

☑ Enable Write ☐ Provide timestamp and status code variables

Name	Value
NodeId	ns=4;s=MAIN.nTest
NodeClass	2
BrowseName	4:nTest
DisplayName	nTest
Description	
WriteMask	0
UserWriteMask	0
Value	0
DataType	i=4
ValueRank	-1
ArrayDimensions	
AccessLevel	3
UserAccessLevel	3
MinimumSamplingInterval	0
Historizing	False

Enabling write access globally

Before write commands can be sent, they must be enabled globally. This is done by setting the output variable "Write Enable" for the I/O client:



4.3.10 Security

4.3.10.1 Overview

One of the reasons for the success of OPC UA as communication technology is the various integrated security mechanisms. OPC UA-based data communication can be secured on two levels:

1. Transport layer
2. Application level

Endpoints

A server offers the client a list of different endpoints [▸ 105] 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 user name/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 [▸ 176].

Application level

In addition to the transport layer, a communication connection can also be secured at the application layer. For this purpose, various authentication mechanisms [▸ 106] are available, which are offered by the server endpoint.

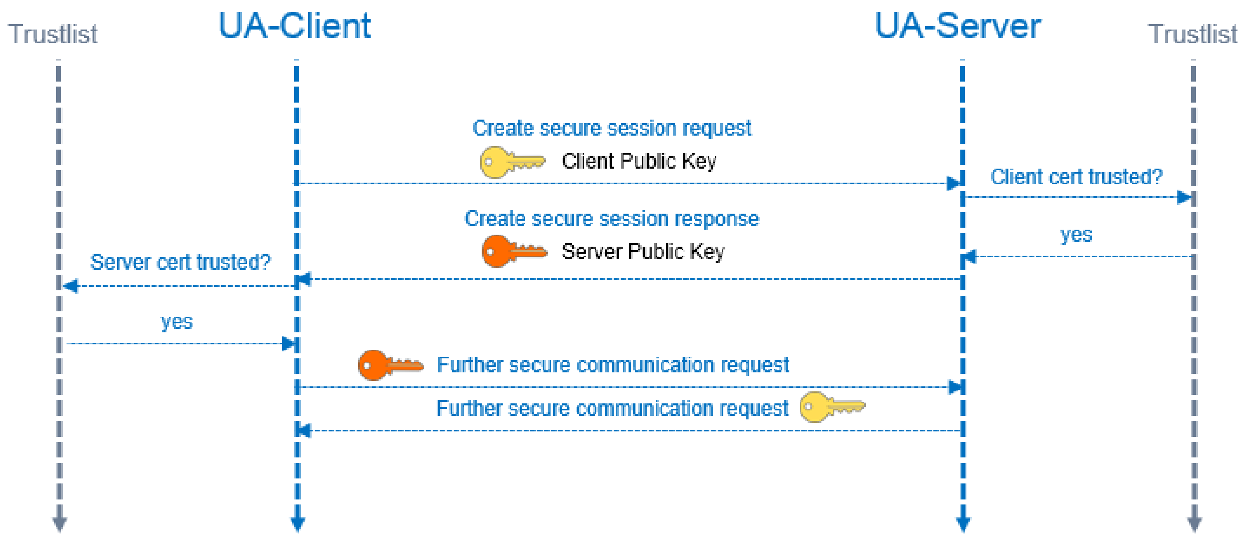
Also see about this

- Access rights [▸ 109]

4.3.10.2 Certificate exchange

To secure the communication connection at transport layer via a secure endpoint [▶ 105], it is necessary to establish a mutual trust between client and server.

By default, both the TwinCAT OPC UA Server and the TwinCAT OPC UA Client generate a machine-specific, self-signed certificate for authentication of the respective application at the first start.


Trustlist
UA-Client
UA-Server
Trustlist
Create secure session request
Client Public Key
Client cert trusted?
Create secure session response
yes
Server Public Key
Server cert trusted?
yes
Further secure communication request
Further secure communication request

Set up a trust relationship on the server

To establish a trust relationship between any OPC UA Client and the TwinCAT OPC UA Server, you need the public key of the client certificate. The server must trust this. This can be done via the file system, for example. The server manages the trust settings for client certificates in the PKI subdirectory.

- Trusted certificates: %InstallDir%\Server\PKI\CA\trusted\certs
- Untrusted certificates: %InstallDir%\Server\PKI\CA\rejected\certs

By moving client certificates between these directories, the trust settings can be adjusted accordingly. The public key of a client certificate is automatically stored in the above untrusted certificate directory the first time the client attempts to connect to a secure endpoint. By subsequently moving the public key to the trusted certificate directory, the client is trusted the next time it attempts to connect.

AutomaticallyTrustAllClientCertificates

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

Set up a trust relationship on the client

Depending on the OPC UA Client employed, different steps may need to be taken so that the OPC UA Client trusts the OPC UA Server. Typically, for client applications with a graphical user interface, a warning message is displayed the first time you connect to the server, whereby the server certificate can then be classified as trustworthy.

The following instruction is therefore only valid for the TwinCAT OPC UA Client.

The public key of the OPC UA Server is located as a DER file in the following directory: *%InstallDir%\Server\PKI\CA\own\certs*

In the case of the TwinCAT OPC UA Client, copy the file into the corresponding "Trusted" directory: *%InstallDir%\Client\PKI\CA\trusted\certs*

4.4 Client PLCopen

4.4.1 Overview

The TwinCAT OPC UA Client offers several options for communicating directly with one or more OPC UA Servers from the control logic. On the one hand, an OPC UA interface is available directly from the PLC, in which a connection to an OPC UA Server can be initiated via PLCopen standardized function blocks. On the other there is an OPC UA Client I/O device that offers a simple mapping-based interface.

PLC function blocks

The following table provides an overview of the functionalities offered:

Functionality	Description	Relevant function blocks
Connect/Disconnect	Establishment and disconnection of connections to OPC UA Servers.	UA_Connect UA_Disconnect
Polling (Read/Write)	Reading and writing of variables in the UA namespace in the polling mode. Contains no subscriptions.	UA_Connect UA_Disconnect UA_Read UA_Write UA_GetNamespaceIndex UA_NodeGetHandle UA_NodeReleaseHandle
Method Call	Execution of methods in the UA namespace.	UA_Connect UA_Disconnect UA_MethodGetHandle UA_MethodReleaseHandle UA_MethodCall
Security	Establishment of an encrypted connection to an OPC UA Server.	UA_Connect UA_Disconnect

The interfaces of each function block are described in the section PLC API [▸ 215].

I/O device

Further information on the TwinCAT OPC UA Client I/O device can be found in the section I/O > Quick start [▸ 160].

4.4.2 Supported data types

The OPC UA Client enables direct access to OPC UA Servers from the real-time logic. To read and write data, the data types must be assigned to both environments (mapping). This assignment is described below.

Basic data types

The assignment of the basic data types is described in PLCopen OPC UA Information Model for IEC 61131-3.

OPC UA data type	PLC data type
Boolean	BOOL
SByte	SINT
Byte	USINT
Int16	INT
Int32	DINT
String	STRING
USint	BYTE
Float	REAL
Double	LREAL
UInt16	UINT
UInt32	UDINT
Int64	LINT
UInt64	ULINT
DateTime	DT

Derived data types

OPC UA defines basic data types. Other data types are derived from these.

On the client side, access to all UA data types is only possible with PLC data types. Data types derived from the basic data types are also supported from TcOpcUaClient version 2.1.0.36 or higher.

Currently supported non-base data types on the client side are:

- Counter (use UDINT in PLC)

Access to arrays

When creating a node handle, the system checks the possibilities to access the node. Further checks are performed during reading and writing.

To access array nodes and input and output parameters of method calls, certain conditions must be fulfilled.

- For accessing nodes: The array dimension and data length must match the data provided when reading and writing.
- Reading strings: If just one string exceeds the specified length for PLC strings, the read process fails.
- Only array dimensions up to three levels are supported.

4.4.3 Best practice

4.4.3.1 How to determine communication parameters

In general a graphic OPC UA Client is used to determine the attributes of a node or methods that have to be used together with the PLC function blocks, e.g.:

- Node Identifier [▸ 179]
- Node namespace index and corresponding namespace URI [▸ 179]
- Node Data Type [▸ 180]
- Node ID and Object Node ID methods [▸ 181]

The following documentation uses the generic OPC UA Client UA Expert as an example. This client can be purchased from the Unified Automation website: www.unified-automation.com.

Node Identifier

A general task consists of reading or writing variables that are generally referred to as nodes in the context of OPC UA.

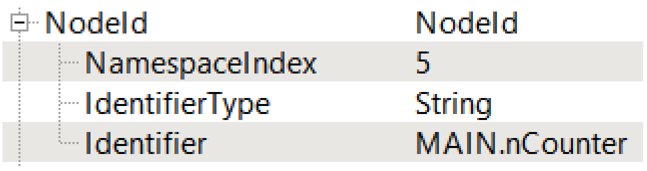
Nodes can be marked with the following three attributes:

- NamespaceIndex: The namespace in which the node is located, such as the PLC runtime.
- Identifier: Unique identifier of the node within its namespace
- IdentifierType: Type of node: String, Guid and Numeric

These attributes represent the so-called NodeID – the representation of a node on an OPC UA Server – and are required by some function blocks.

With the help of UA Expert you can simply determine the attributes of a node by establishing a connection to the OPC UA Server and browsing to the desired node. The attributes are then visible in the Attributes panel.

Sample:

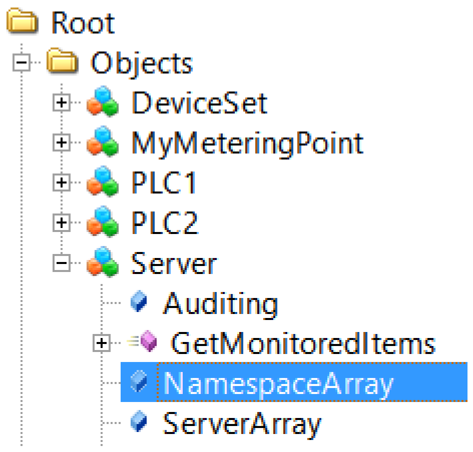


NodeId	NodeId
NamespaceIndex	5
IdentifierType	String
Identifier	MAIN.nCounter

Node NamespaceIndex and corresponding NamespaceURI

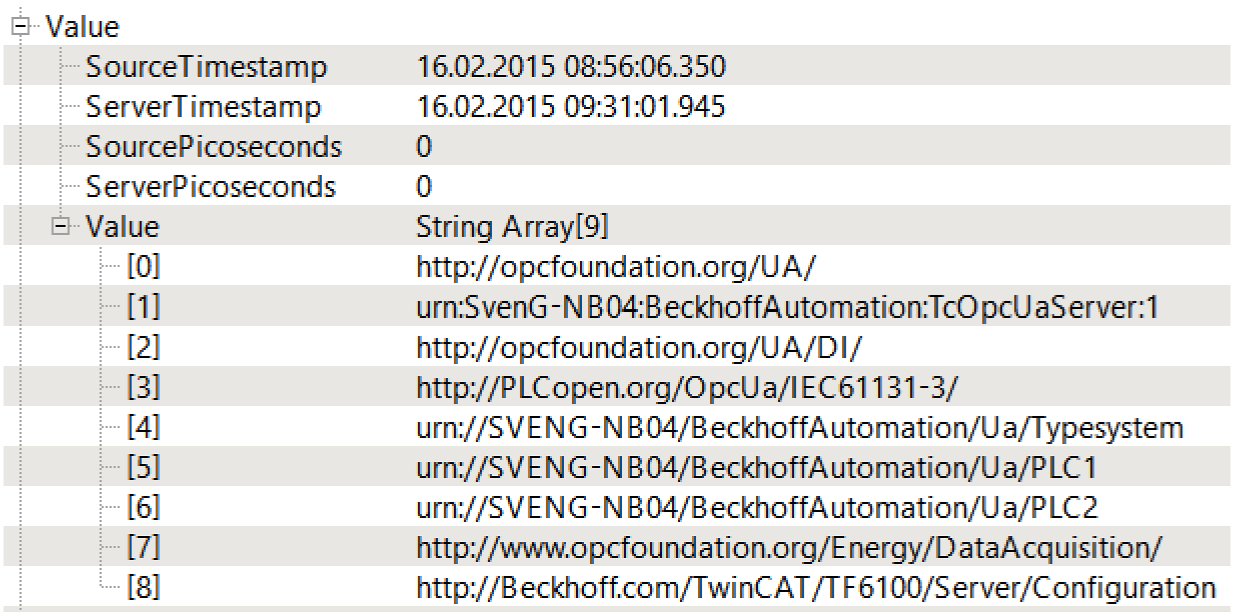
According to the OPC UA specification, the namespace index (as shown above) can be a dynamically generated value. Therefore, OPC UA Clients must always use the corresponding namespace URI to resolve the NamespaceIndex before a node handle is detected.

Use the UA_GetNamespaceIndex [▸ 237] function block to obtain the NamespaceIndex for a NamespaceURI. The NamespaceURI required for this can be determined with the help of UA Expert by establishing a connection to the OPC UA Server and browsing to the NamespaceArray node.



This node contains information about all namespaces registered on the OPC UA Server. The corresponding NamespaceURIs are visible in the Attributes panel.

Sample:



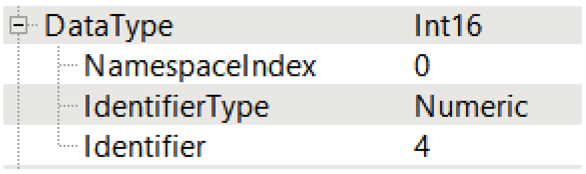
Value	
SourceTimestamp	16.02.2015 08:56:06.350
ServerTimestamp	16.02.2015 09:31:01.945
SourcePicoseconds	0
ServerPicoseconds	0
Value	String Array[9]
[0]	http://opcfoundation.org/UA/
[1]	urn:SvenG-NB04:BeckhoffAutomation:TcOpcUaServer:1
[2]	http://opcfoundation.org/UA/DI/
[3]	http://PLCopen.org/OpcUa/IEC61131-3/
[4]	urn://SVENG-NB04/BeckhoffAutomation/Ua/Typesystem
[5]	urn://SVENG-NB04/BeckhoffAutomation/Ua/PLC1
[6]	urn://SVENG-NB04/BeckhoffAutomation/Ua/PLC2
[7]	http://www.opcfoundation.org/Energy/DataAcquisition/
[8]	http://Beckhoff.com/TwinCAT/TF6100/Server/Configuration

A NodeID in which the NamespaceIndex is 5 is shown in the sample in the section Node Identifier [▸ 179]. According to the NamespaceArray shown in the figure, the corresponding NamespaceURI is urn://SVENG-NB04/BeckhoffAutomation/Ua/PLC1. This URI can now be used for the function block UA_GetNamespaceIndex. The OPC UA Server ensures that the URI always remains the same, even after a restart. As the NamespaceIndex shown can change, however, the NamespaceURI should always be used in combination with the UA_GetNamespaceIndex function block for later use with other function blocks, e.g. UA_Read [▸ 249], UA_Write [▸ 252], to resolve the correct NamespaceIndex.

Node DataType

The data type of a node is required in order to see which PLC data type needs to be used in order to assign a read value or write it to a node. With the help of UA Expert you can simply determine the data type of a node by establishing a connection to the OPC UA Server and browsing to the desired node. The data type is then visible in the Attributes panel.

Sample:



DataType	Int16
NamespaceIndex	0
IdentifierType	Numeric
Identifier	4

In this case the data type (DataType) is "Int16". This must be assigned to an equivalent data type in the PLC, e.g. "INT".

The PLCopen IEC61131 - to - OPC UA specification describes the defined data type mapping. The following table is an excerpt from this specification:

IEC61131 elementary data types	OPC UA built-in data types
BOOL	Boolean
SINT	SByte
USINT	Byte
INT	Int16
UINT	UInt16
DINT	Int32
UDINT	UInt32
LINT	Int64
ULINT	UInt64
BYTE	Byte
WORD	UInt16
DWORD	UInt32
LWORD	UInt64
REAL	Float
LREAL	Double
STRING	String
CHAR	Byte
WSTRING	String
WCHAR	UInt16
DT DATE_AND_TIME	DateTime
DATE	DateTime
TOD TIME_OF_DAY	DateTime
TIME	Double

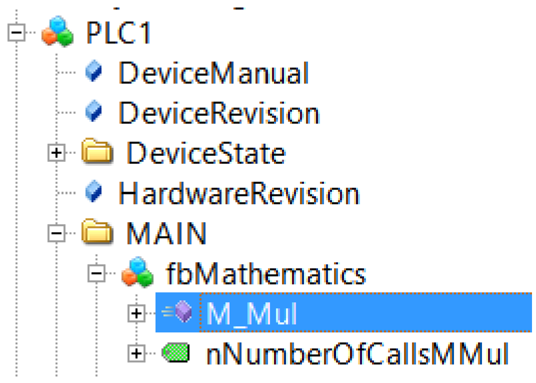
Method NodeID and Object NodeID

When calling methods from the OPC UA namespace, two identifiers are required if the method handle is get using the function block UA_MethodGetHandle [▶ 243]:

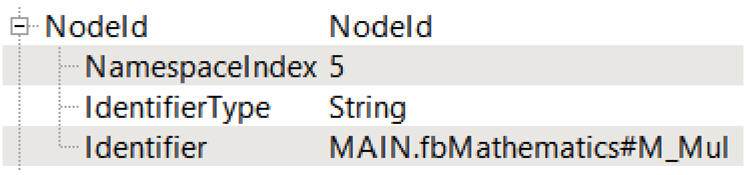
- ObjectNodeID: Identifies the UA object that contains the method
- MethodNodeID: Identifies the method itself

With the help of UA Expert you can simply determine both NodeIDs by establishing a connection to the OPC UA Server and browsing to the desired method or the desired UA object that contains the method.

Sample Method M_Mul:

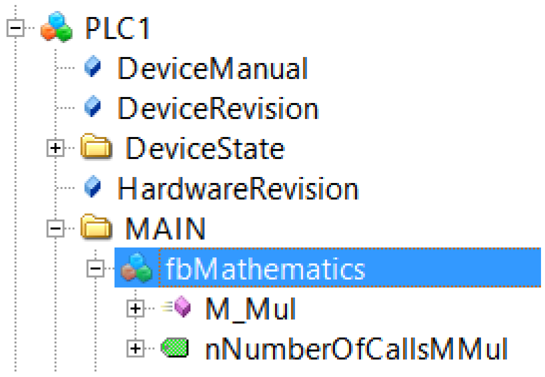


The method identifier is then visible in the Attributes panel.

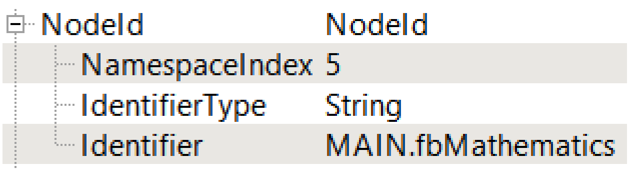


NodeId	NodeId
NamespaceIndex	5
IdentifierType	String
Identifier	MAIN.fbMathematics#M_Mul

Sample Object fbMathematics:



The object identifier is then visible in the Attributes panel.



NodeId	NodeId
NamespaceIndex	5
IdentifierType	String
Identifier	MAIN.fbMathematics

4.4.3.2 How to establish a connection

The following section describes how you use the TcX_PLCopen_OpcUa function block to establish a connection to a local or remote OPC UA Server. This connection can then be used to call other functions, such as read or write nodes, or call methods.

This section contains the following topics:

- Overview [▸ 182]
- Schematic workflow [▸ 182]
- General notes [▸ 183]
- Code snippet [▸ 183]

Overview

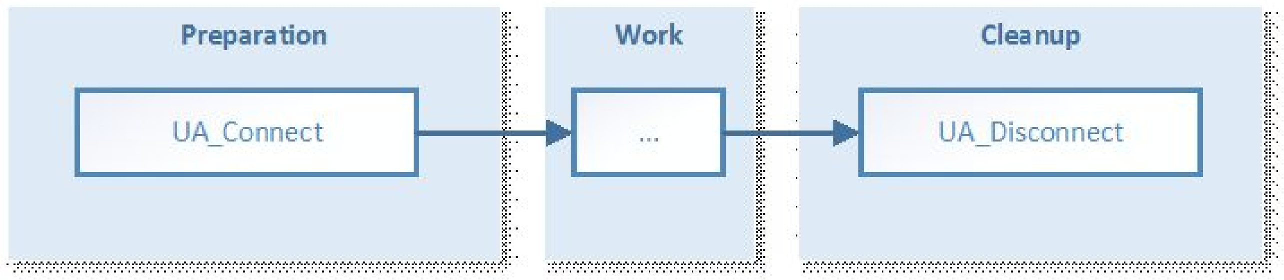
The following function blocks are required to establish a connection to an OPC UA Server and to interrupt the session later: UA_Connect [▸ 234], UA_Disconnect [▸ 236].

First read the section How to determine communication parameters [▸ 178] to better understand certain UA functionalities (e.g. how to determine node identifiers).

Schematic workflow

The schematic workflow of each TwinCAT OPC UA Client can be categorized into three different phases: Preparation, Work and Cleanup.

The use case described in this section can be visualized as follows:



General notes

The UA_Connect function block requires the following information in order to be able to establish a connection to a local or remote OPC UA Server:

- Server URL
- Session Connect Information

The server URL basically consists of a prefix, a hostname and a port. The prefix describes the OPC UA transport protocol that should be used for the connection, e.g."opc.tcp://" for a binary TCP connection (default). The host name or IP address part describes the address information of the OPC UA target server, e.g. "192.168.1.1" or "CX-12345". The port number is the target port of the OPC UA Server, e.g. "4840". Overall the server URL may then look like this: opc.tcp://CX-12345:4840.

Code snippet

Declaration:

```
(* Declarations for UA_Connect *)
fbUA_Connect : UA_Connect;
SessionConnectInfo : ST_UASessionConnectInfo;
nConnectionHdl : DWORD;

(* Declarations for UA_Disconnect *)
fbUA_Disconnect : UA_Disconnect;

(* Declarations for state machine and output handling *)
iState : INT;
bDone : BOOL;
bBusy : BOOL;
bError : BOOL;
nErrorID : DWORD;
```

Implementation:

```
CASE iState OF

  0:
      bError := FALSE;
      nErrorID := 0;
      SessionConnectInfo.tConnectTimeout := T#1M;
      SessionConnectInfo.tSessionTimeout := T#1M;
      SessionConnectInfo.sApplicationName := '';
      SessionConnectInfo.sApplicationUri := '';
      SessionConnectInfo.eSecurityMode := eUASecurityMsgMode_None;
      SessionConnectInfo.eSecurityPolicyUri := eUASecurityPolicy_None;
      SessionConnectInfo.eTransportProfileUri := eUATransportProfileUri_UATcp;
      stNodeAddInfo.nIndexRangeCount := nIndexRangeCount;
      stNodeAddInfo.stIndexRange := stIndexRange;
      iState := iState + 1;

  1:
    fbUA_Connect(
      Execute := TRUE,
      ServerURL := ‘opc.tcp://192.168.1.1:4840’,
      SessionConnectInfo := SessionConnectInfo,
      Timeout := T#5S,
      ConnectionHdl => nConnectionHdl);
    IF NOT fbUA_Connect.Busy THEN
      fbUA_Connect(Execute := FALSE);
      IF NOT fbUA_Connect.Error THEN
        iState := iState + 1;
      ELSE
```

```
        bError := TRUE;
        nErrorID := fbUA_Connect.ErrorID;
        nConnectionHdl  := 0;
        iState := 0;
      END_IF
    END_IF

  2:
    fbUA_Disconnect(
      Execute  := TRUE,
      ConnectionHdl  := nConnectionHdl);

    IF NOT fbUA_Disconnect.Busy THEN
      fbUA_Disconnect(Execute  := FALSE);
      IF NOT fbUA_Disconnect.Error THEN
        iState := 0;
      ELSE
        bError := TRUE;
        nErrorID := fbUA_Disconnect.ErrorID;
        iState := 0;
        nConnectionHdl  := 0;
      END_IF
    END_IF

END_CASE
```

4.4.3.3 How to read nodes

The following section describes how you use the TcX_PLCopen_OpcUa function block to read out an OPC UA node from a local or remote OPC UA Server.

This section contains the following topics:

- Overview [▶ 184]
- Schematic workflow [▶ 184]
- General notes [▶ 185]
- Code snippet [▶ 185]

Overview

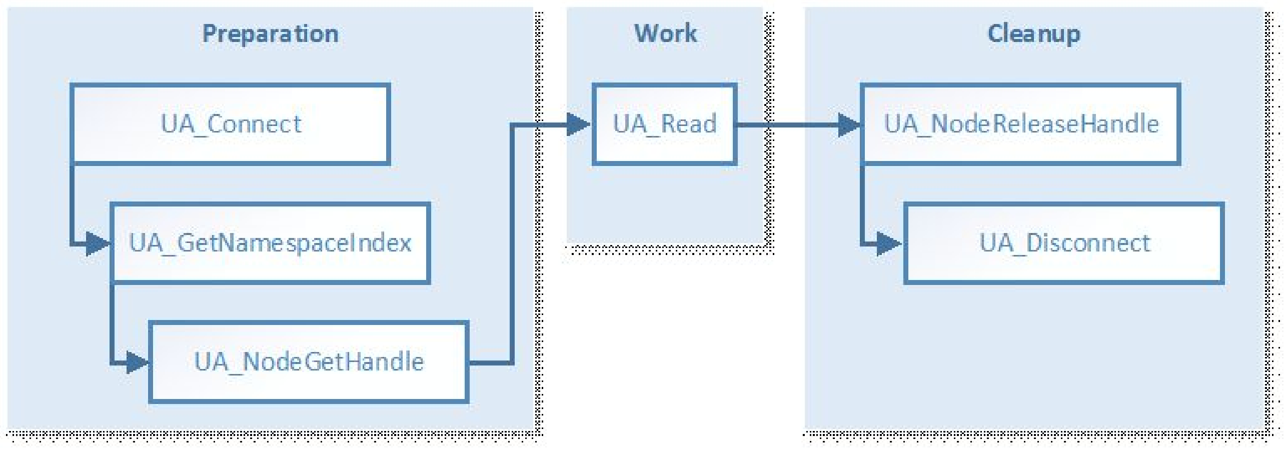
The following function blocks are required to establish a connection to an OPC UA Server, to read UA nodes and to interrupt the session later: UA_Connect [▶ 234], UA_GetNamespaceIndex [▶ 237], UA_NodeGetHandle [▶ 245], UA_Read [▶ 249], UA_NodeReleaseHandle [▶ 247], UA_Disconnect [▶ 236].

First of all, read the section How to determine communication parameters [▶ 178] so as to be able to understand certain UA functions better (e.g. how NodeIdentifiers can be determined) as well as the section How to establish a connection [▶ 182].

Schematic workflow

The schematic workflow of each TwinCAT OPC UA Client can be categorized into three different phases: Preparation, Work and Cleanup.

The use case described in this section can be visualized as follows:



General notes

- The UA_Connect function block requires the following information to establish a connection to a local or remote OPC UA Server (see also How to establish a connection [▶ 182]):
  - Server URL
  - Session Connect Information
- The UA_GetNamespaceIndex function block requires a connection handle (from UA_Connect) and a NamespaceURI for resolution in a NamespaceIndex, which will be used later by UA_NodeGetHandle to obtain a node handle (see also How to determine communication parameters [▶ 178]).
- The UA_NodeGetHandle function block requires a connection handle (from UA_Connect) and the NodeID (from a ST_UANodeID) in order to obtain a node handle (see also How to determine communication parameters [▶ 178]).
- The UA_Read function block requires a connection handle (from UA_Connect), a node handle (from UA_NodeGetHandle) and a pointer to the target variable (where the read value is to be saved). Make sure that the target variable has the correct data type (see How to determine communication parameters [▶ 178]).
- The UA_NodeReleaseHandle function block requires a connection handle (from UA_Connect) and a node handle (from UA_NodeGetHandle).

Code snippet

Declaration:

```
(* Declarations for UA_GetNamespaceIndex *)
fbUA_GetNamespaceIndex : UA_GetNamespaceIndex;
nNamespaceIndex : UINT;

(* Declarations for UA_NodeGetHandle *)
fbUA_NodeGetHandle : UA_NodeGetHandle;
NodeID : ST_UANodeID;
nNodeHdl : DWORD;

(* Declarations for UA_Read *)
fbUA_Read : UA_Read;
stIndexRange : ARRAY [1..nMaxIndexRange] OF ST_UAIndexRange;
nIndexRangeCount : UINT;
stNodeAddInfo : ST_UANodeAdditionalInfo;
sNodeIdentifier : STRING(MAX_STRING_LENGTH) := 'MAIN.nCounter';
nReadData : INT;
cbDataRead : UDINT;

(* Declarations for UA_NodeReleaseHandle *)
fbUA_NodeReleaseHandle : UA_NodeReleaseHandle;
```

Implementation:

```
CASE iState OF
  0:
    [...]

  2: (* GetNS Index *)
```

```
    fbUA_GetNamespaceIndex(
      Execute := TRUE,
      ConnectionHdl := nConnectionHdl,
      NamespaceUri := sNamespaceUri,
      NamespaceIndex => nNamespaceIndex
      );
    IF NOT fbUA_GetNamespaceIndex.Busy THEN
      fbUA_GetNamespaceIndex(Execute := FALSE);
      IF NOT fbUA_GetNamespaceIndex.Error THEN
        iState := iState + 1;
      ELSE
        bError := TRUE;
        nErrorID := fbUA_GetNamespaceIndex.ErrorID;
        iState := 6;
      END_IF
    END_IF

  3: (* UA_NodeGetHandle *)
    NodeID.eIdentifierType := eUAIdentifierType_String;
    NodeID.nNamespaceIndex := nNamespaceIndex;
    NodeID.sIdentifier := sNodeIdentifier;
    fbUA_NodeGetHandle(
      Execute := TRUE,
      ConnectionHdl := nConnectionHdl,
      NodeID := NodeID,
      NodeHdl => nNodeHdl);
    IF NOT fbUA_NodeGetHandle.Busy THEN
      fbUA_NodeGetHandle(Execute := FALSE);
      IF NOT fbUA_NodeGetHandle.Error THEN
        iState := iState + 1;
      ELSE
        bError := TRUE;
        nErrorID := fbUA_NodeGetHandle.ErrorID;
        iState := 6;
      END_IF
    END_IF

  4: (* UA_Read *)
    fbUA_Read(
      Execute := TRUE,
      ConnectionHdl := nConnectionHdl,
      NodeHdl := nNodeHdl,
      cbData := SIZEOF(nReadData),
      stNodeAddInfo := stNodeAddInfo,
      pVariable := ADR(nReadData));
    IF NOT fbUA_Read.Busy THEN
      fbUA_Read( Execute := FALSE, cbData_R => cbDataRead);
      IF NOT fbUA_Read.Error THEN
        iState := iState + 1;
      ELSE
        bError := TRUE;
        nErrorID := fbUA_Read.ErrorID;
        iState := 6;
      END_IF
    END_IF

  5: (* Release Node Handle *)
    fbUA_NodeReleaseHandle(
      Execute := TRUE,
      ConnectionHdl := nConnectionHdl,
      NodeHdl := nNodeHdl);
    IF NOT fbUA_NodeReleaseHandle.Busy THEN
      fbUA_NodeReleaseHandle(Execute := FALSE);
      IF NOT fbUA_NodeReleaseHandle.Error THEN
        iState := iState + 1;
      ELSE
        bError := TRUE;
        nErrorID := fbUA_NodeReleaseHandle.ErrorID;
        iState := 6;
      END_IF
    END_IF

  6:
    [...]

END_CASE
```

4.4.3.4 How to write nodes

The following section describes how you use the TcX_PLCopen_OpcUa function block to write values in an OPC UA node from a local or remote OPC UA Server.

This section contains the following topics:

Overview

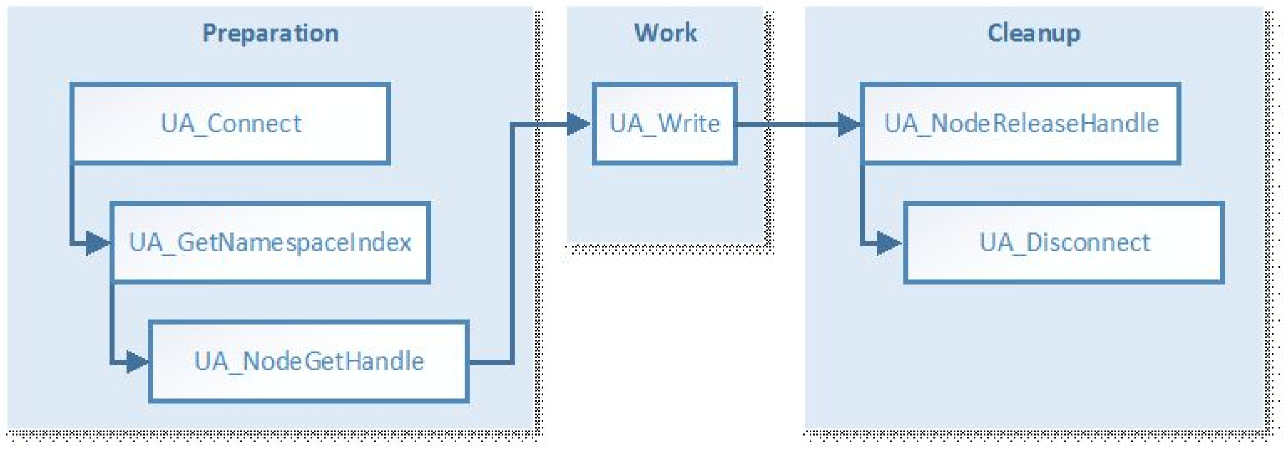
The following function blocks are required to establish a connection to an OPC UA Server, write UA nodes and subsequently interrupt the session: UA_Connect [▶ 234], UA_GetNamespaceIndex [▶ 237], UA_NodeGetHandle [▶ 245], UA_Write [▶ 252], UA_NodeReleaseHandle [▶ 247], UA_Disconnect [▶ 236].

First of all, read the section How to determine communication parameters [▶ 178] so as to be able to understand certain UA functions better (e.g. how NodeIdentifiers can be determined) as well as the section How to establish a connection [▶ 182].

Schematic workflow

The schematic workflow of each TwinCAT OPC UA Client can be categorized into three different phases: Preparation, Work and Cleanup.

The use case described in this section can be visualized as follows:



General notes

- The UA_Connect function block requires the following information to establish a connection to a local or remote OPC UA Server (see also How to establish a connection [▶ 182]):
  - Server URL
  - Session Connect Information
- The UA_GetNamespaceIndex function block requires a connection handle (from UA_Connect) and a NamespaceURI for resolution in a NamespaceIndex, which will be used later by UA_NodeGetHandle to obtain a node handle (see also How to determine communication parameters [▶ 178]).
- The UA_NodeGetHandle function block requires a connection handle (from UA_Connect) and the NodeID (from a ST_UANodeID) in order to obtain a node handle (see also How to determine communication parameters [▶ 178]).

- The UA_Write function block requires a connection handle (from UA_Connect), a node handle (from UA_NodeGetHandle) and a pointer to a variable containing the value that is to be written. Make sure that the target variable has the correct data type (see How to determine communication parameters [▸ 178]).
- The UA_NodeReleaseHandle function block requires a connection handle (from UA_Connect) and a node handle (from UA_NodeGetHandle).

Code snippet

Declaration:

```
(* Declarations for UA_GetNamespaceIndex *)
fbUA_GetNamespaceIndex : UA_GetNamespaceIndex;
nNamespaceIndex : UINT;

(* Declarations for UA_NodeGetHandle *)
fbUA_NodeGetHandle : UA_NodeGetHandle;
NodeID : ST_UANodeID;
nNodeHdl : DWORD;

(* Declarations for UA_Write *)
fbUA_Write : UA_Write;
stIndexRange : ARRAY [1..nMaxIndexRange] OF ST_UAIndexRange;
nIndexRangeCount : UINT;
stNodeAddInfo : ST_UANodeAdditionalInfo;
sNodeIdentifier: STRING(MAX_STRING_LENGTH) := 'MAIN.nNumber';
nWriteData: INT := 42;

(* Declarations for UA_NodeReleaseHandle *)
fbUA_NodeReleaseHandle : UA_NodeReleaseHandle;
```

Implementation:

```
CASE iState OF
  0:
    [...]

  2: (* GetNS Index *)
    fbUA_GetNamespaceIndex(
      Execute := TRUE,
      ConnectionHdl := nConnectionHdl,
      NamespaceUri := sNamespaceUri,
      NamespaceIndex => nNamespaceIndex
      );
    IF NOT fbUA_GetNamespaceIndex.Busy THEN
      fbUA_GetNamespaceIndex(Execute := FALSE);
      IF NOT fbUA_GetNamespaceIndex.Error THEN
        iState := iState + 1;
      ELSE
        bError := TRUE;
        nErrorID := fbUA_GetNamespaceIndex.ErrorID;
        iState := 6;
      END_IF
    END_IF

  3: (* UA_NodeGetHandle *)
    NodeID.eIdentifierType := eUAIdentifierType_String;
    NodeID.nNamespaceIndex := nNamespaceIndex;
    NodeID.sIdentifier := sNodeIdentifier;
    fbUA_NodeGetHandle(
      Execute := TRUE,
      ConnectionHdl := nConnectionHdl,
      NodeID := NodeID,
      NodeHdl => nNodeHdl);
    IF NOT fbUA_NodeGetHandle.Busy THEN
      fbUA_NodeGetHandle(Execute := FALSE);
      IF NOT fbUA_NodeGetHandle.Error THEN
        iState := iState + 1;
      ELSE
        bError := TRUE;
        nErrorID := fbUA_NodeGetHandle.ErrorID;
        iState := 6;
      END_IF
    END_IF

  4: (* UA_Write *)
    fbUA_Write(
      Execute := TRUE,
      ConnectionHdl := nConnectionHdl,
      NodeHdl := nNodeHdl,
      stNodeAddInfo := stNodeAddInfo,
      cbData := SIZEOF(nWriteData),
      pVariable := ADR(nWriteData));
    IF NOT fbUA_Write.Busy THEN
      fbUA_Write(
        Execute := FALSE,
        pVariable := ADR(nWriteData));
      IF NOT fbUA_Write.Error THEN
        iState := iState + 1;
      ELSE
        bError := TRUE;
        nErrorID := fbUA_Write.ErrorID;
        iState := 6;
      END_IF
    END_IF

  5: (* Release Node Handle *)
    fbUA_NodeReleaseHandle(
      Execute := TRUE,
      ConnectionHdl := nConnectionHdl,
      NodeHdl := nNodeHdl);
    IF NOT fbUA_NodeReleaseHandle.Busy THEN
      fbUA_NodeReleaseHandle(Execute := FALSE);
      IF NOT fbUA_NodeReleaseHandle.Error THEN
        iState := iState + 1;
      ELSE
        bError := TRUE;
        nErrorID := fbUA_NodeReleaseHandle.ErrorID;
        iState := 6;
      END_IF
    END_IF

  6:
    [...]

END_CASE
```

4.4.3.5 How to call methods

The following section describes how you use the TcX_PLCopen_OpcUa function block to call methods on a local or remote OPC UA Server.

This section contains the following topics:

- Overview [▶ 189]
- Schematic workflow [▶ 189]
- General notes [▶ 190]
- Code snippet [▶ 190]

Overview

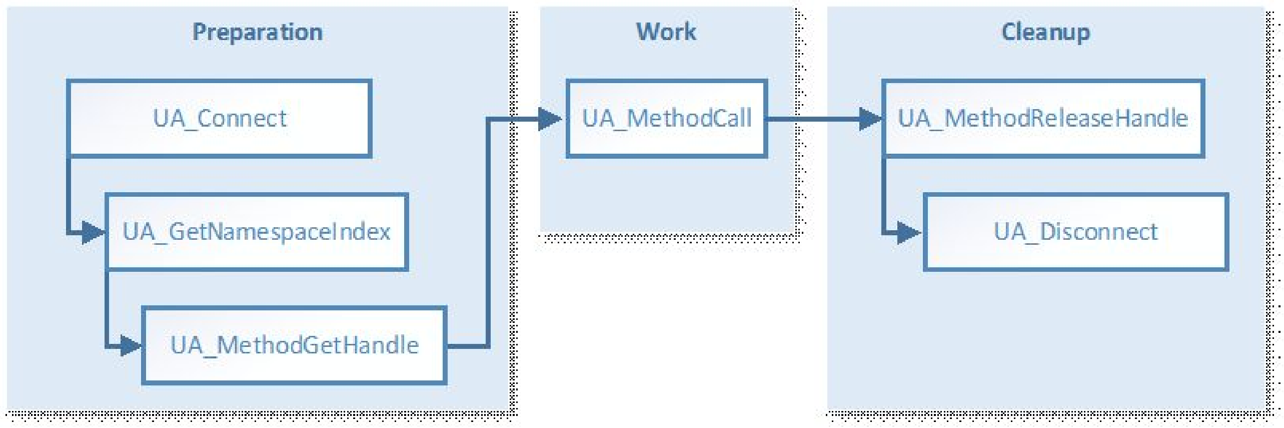
The following function blocks are required to connect to an OPC UA Server, call UA methods, and subsequently interrupt the session: UA_Connect [▶ 234], UA_GetNamespaceIndex [▶ 237], UA_MethodGetHandle [▶ 243], UA_MethodCall [▶ 240], UA_MethodReleaseHandle [▶ 244], UA_Disconnect [▶ 236].

First of all, read the section How to determine communication parameters [▶ 178] so as to be able to understand certain UA functions better (e.g. how MethodIdentifier can be determined) as well as the section How to establish a connection [▶ 182].

Schematic workflow

The schematic workflow of each TwinCAT OPC UA Client can be categorized into three different phases: Preparation, Work and Cleanup.

The use case described in this section can be visualized as follows:



General notes

- The UA_Connect function block requires the following information to establish a connection to a local or remote OPC UA Server (see also How to establish a connection [▶ 182]):
    - Server URL
    - Session Connect Information
- The UA_GetNamespaceIndex function block requires a connection handle (from UA_Connect) and a NamespaceURI for resolution in a NamespaceIndex, which will be used later by UA_NodeGetHandle to obtain a node handle (see also How to determine communication parameters [▶ 178]).
- The UA_MethodGetHandle function block requires a connection handle (from UA_Connect), an ObjectNodeID and a MethodNodeID in order to obtain a method handle (see also How to determine communication parameters [▶ 178]).
- The UA_MethodCall function block requires a connection handle (from UA_Connect), a method handle (from UA_MethodGetHandle) and information about the input and output arguments of the method that is to be called. Information about the input arguments is represented by the input parameters pInputArgInfo and pInputArgData of UA_MethodCall. Information about the output parameters is represented by the pOutputArgInfo and pOutputArgData input parameters of UA_MethodCall. The input parameter pOutputArgInfoAndData then represents a pointer to a structure containing the results of the method call, including all output parameters. In the following code snippet the pInputArgInfo and pInputArgData parameters are calculated and created in the M_Init method.
- The UA_NodeReleaseHandle function block requires a connection handle (from UA_Connect) and a method handle (from UA_MethodGetHandle).

Code snippet

M_Init initialization method of the function block containing the UA method call

```
MEMSET(ADR(nInputData),0,SIZEOF(nInputData));
nArg := 1;

(********** Input parameter 1 **********)
InputArguments[nArg].DataType := eUAType_Int16;
InputArguments[nArg].ValueRank := -1;  (* Scalar = -1 or Array *)
InputArguments[nArg].ArrayDimensions[1]  := 0;  (* Number of Dimension in case its an array *)
InputArguments[nArg].nLenData := SIZEOF(numberIn1);  (* Length if its a STRING *)
IF nOffset + SIZEOF(numberIn1) > nInputArgSize THEN
  bInputDataError := TRUE;
  RETURN;
ELSE
  MEMCPY(ADR(nInputData)+nOffset,ADR(numberIn1),SIZEOF(numberIn1));  (* VALUE in BYTES FORM *)
  nOffset := nOffset + SIZEOF(numberIn1);
END_IF
nArg := nArg + 1;

(********** Input parameter 2 **********)
InputArguments[nArg].DataType := eUAType_Int16;
InputArguments[nArg].ValueRank := -1;  (* Scalar = -1 or Array *)
InputArguments[nArg].ArrayDimensions[1]  := 0;  (* Number of Dimension in case its an array *)
InputArguments[nArg].nLenData := SIZEOF(numberIn2);  (* Length if its a STRING *)
IF nOffset + SIZEOF(numberIn2) > nInputArgSize THEN
  bInputDataError := TRUE;
  RETURN;
ELSE
  MEMCPY(ADR(nInputData)+nOffset,ADR(numberIn2),SIZEOF(numberIn2));(* VALUE in BYTES FORM *)
  nOffset := nOffset + SIZEOF(numberIn2);
END_IF

cbWriteData := nOffset;
```

Declaration:

```
(* Declarations for UA_GetNamespaceIndex *)
fbUA_GetNamespaceIndex : UA_GetNamespaceIndex;
nNamespaceIndex : UINT;

(* Declarations for UA_MethodGetHandle *)
fbUA_MethodGetHandle: UA_MethodGetHandle;
ObjectNodeID: ST_UANodeID;
MethodNodeID: ST_UANodeID;
nMethodHdl: DWORD;

(* Declarations for UA_MethodCall *)
fbUA_MethodCall: UA_MethodCall;
sObjectNodeIdIdentifier : STRING(MAX_STRING_LENGTH) := 'MAIN.fbMathematics';
sMethodNodeIdIdentifier : STRING(MAX_STRING_LENGTH) := 'MAIN.fbMathematics#M_Mul';
nAdrWriteData: PVOID;
numberIn1: INT := 42; // change according to input value and data type
numberIn2: INT := 42; // change according to input value and data type
numberOutPro: DINT; // result (output parameter of M_Mul())
cbWriteData: UDINT; // calculated automatically by M_Init()
InputArguments: ARRAY[1..2] OF ST_UAMethodArgInfo; // change according to input parameters
stOutputArgInfo: ARRAY[1..1] OF ST_UAMethodArgInfo; // change according to output parameters
stOutputArgInfoAndData: ST_OutputArgInfoAndData;
nInputData: ARRAY[1..4] OF BYTE; // numberIn1(INT16)(2) + numberIn2(INT16)(2)
nOffset: UDINT; // calculated by M_Init()
nArg: INT; // used by M_Init()

(* Declarations for UA_MethodReleaseHandle *)
fbUA_MethodReleaseHandle: UA_MethodReleaseHandle;
```

Implementation:

```
CASE iState OF
  0:
    [...]

  2: (* GetNS Index *)
    fbUA_GetNamespaceIndex(
      Execute := TRUE,
      ConnectionHdl := nConnectionHdl,
      NamespaceUri := sNamespaceUri,
      NamespaceIndex => nNamespaceIndex);
    IF NOT fbUA_GetNamespaceIndex.Busy THEN
      fbUA_GetNamespaceIndex(Execute := FALSE);
      IF NOT fbUA_GetNamespaceIndex.Error THEN
        iState := iState + 1;
      ELSE
        bError := TRUE;
        nErrorID := fbUA_GetNamespaceIndex.ErrorID;
        iState := 7;
      END_IF
    END_IF

  3: (* Get Method Handle *)
    ObjectNodeID.eIdentifierType := eUAIdentifierType_String;
    ObjectNodeID.nNamespaceIndex := nNamespaceIndex;
    ObjectNodeID.sIdentifier := sObjectNodeIdIdentifier;
    MethodNodeID.eIdentifierType := eUAIdentifierType_String;
    MethodNodeID.nNamespaceIndex := nNamespaceIndex;
    MethodNodeID.sIdentifier := sMethodNodeIdIdentifier;

    M_Init();

    IF bInputDataError = FALSE THEN
      iState := iState + 1;
    ELSE
      bBusy := FALSE;
      bError := TRUE;
      nErrorID := 16#70A; //out of memory
    END_IF
  4: (* Method Get Handle *)
    fbUA_MethodGetHandle(
      Execute := TRUE,
      ConnectionHdl := nConnectionHdl,
      ObjectNodeID := ObjectNodeID,
      MethodNodeID := MethodNodeID,
      MethodHdl => nMethodHdl);
    IF NOT fbUA_MethodGetHandle.Busy THEN
      fbUA_MethodGetHandle(Execute := FALSE);
      IF NOT fbUA_MethodGetHandle.Error THEN
        iState := iState + 1;
      ELSE
        bError := TRUE;
        nErrorID := fbUA_MethodGetHandle.ErrorID;
        iState := 6;
      END_IF
    END_IF

  5: (* Method Call *)
    stOutputArgInfo[1].nLenData := SIZEOF(stOutputArgInfoAndData.pro);
    fbUA_MethodCall(
      Execute := TRUE,
      ConnectionHdl := nConnectionHdl,
      MethodHdl := nMethodHdl,
      nNumberOfInputArguments := nNumberOfInputArguments,
      pInputArgInfo := ADR(InputArguments),
      cbInputArgInfo := SIZEOF(InputArguments),
      pInputArgData := ADR(nInputData),
      cbInputArgData := cbWriteData,
      pInputWriteData := 0,
      cbInputWriteData := 0,
      nNumberOfOutputArguments := nNumberOfOutputArguments,
      pOutputArgInfo := ADR(stOutputArgInfo),
      cbOutputArgInfo := SIZEOF(stOutputArgInfo),
      pOutputArgInfoAndData := ADR(stOutputArgInfoAndData),
      cbOutputArgInfoAndData := SIZEOF(stOutputArgInfoAndData));
    IF NOT fbUA_MethodCall.Busy THEN
      fbUA_MethodCall(Execute := FALSE);
      IF NOT fbUA_MethodCall.Error THEN
        iState := iState + 1;
        numberOutPro := stOutputArgInfoAndData.pro;
      ELSE
        bError := TRUE;
        nErrorID := fbUA_MethodCall.ErrorID;
        iState := 6;
      END_IF
    END_IF

  6: (* Release Method Handle *)
    fbUA_MethodReleaseHandle(
      Execute := TRUE,
      ConnectionHdl := nConnectionHdl,
      MethodHdl := nMethodHdl);
    IF NOT fbUA_MethodReleaseHandle.Busy THEN
      fbUA_MethodReleaseHandle(Execute := FALSE);
      bBusy := FALSE;
      IF NOT fbUA_MethodReleaseHandle.Error THEN
        iState := 7;
      ELSE
        bError := TRUE;
        nErrorID := fbUA_MethodReleaseHandle.ErrorID;
        iState := 7;
      END_IF
    END_IF

  7:
    [...]

END_CASE
```

4.4.4 Security

4.4.4.1 Overview

One of the reasons for the success of OPC UA as communication technology is the various integrated security mechanisms. OPC UA-based data communication can be secured on two levels:

1. Transport layer
2. Application level

Endpoints

A server offers the client a list of different endpoints [▸ 105] 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 user name/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 [▸ 193].

Application level

In addition to the transport layer, a communication connection can also be secured at the application layer. For this purpose, various authentication mechanisms [▸ 106] are available, which are offered by the server endpoint.

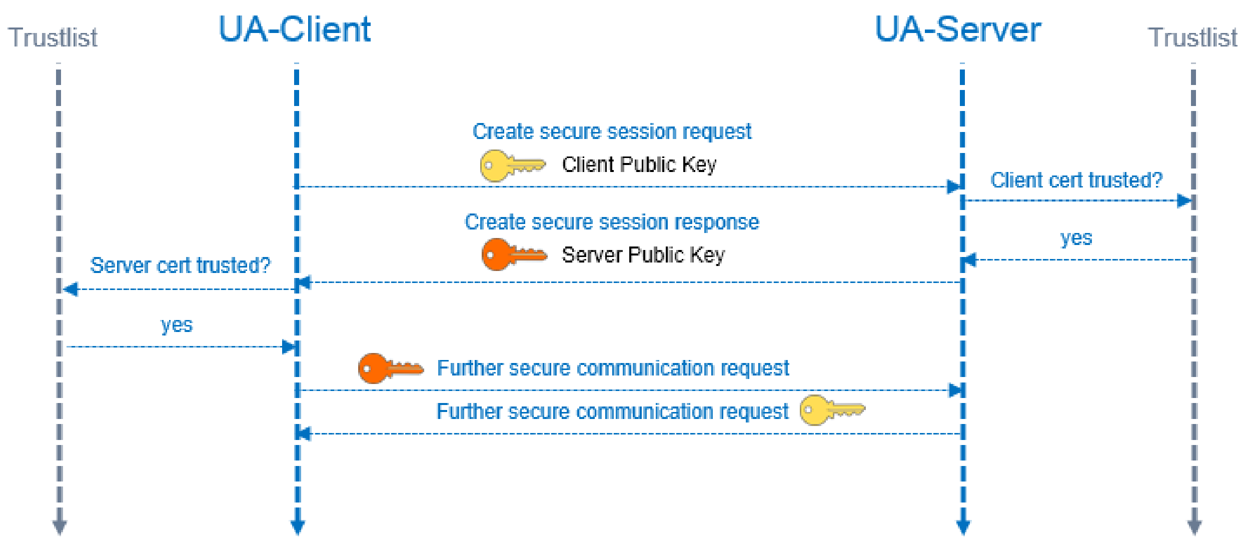
Also see about this

Access rights [▸ 109]

4.4.4.2 Certificate exchange

To secure the communication connection at transport layer via a secure endpoint [▸ 105], it is necessary to establish a mutual trust between client and server.

By default, both the TwinCAT OPC UA Server and the TwinCAT OPC UA Client generate a machine-specific, self-signed certificate for authentication of the respective application at the first start.


Trustlist
UA-Client
UA-Server
Trustlist
Create secure session request
Client Public Key
Client cert trusted?
Create secure session response
yes
Server Public Key
Server cert trusted?
yes
Further secure communication request
Further secure communication request

Set up a trust relationship on the server

To establish a trust relationship between any OPC UA Client and the TwinCAT OPC UA Server, you need the public key of the client certificate. The server must trust this. This can be done via the file system, for example. The server manages the trust settings for client certificates in the PKI subdirectory.

- Trusted certificates: %InstallDir%\Server\PKI\CA\trusted\certs

- Untrusted certificates: %InstallDir%\Server\PKI\CA\rejected\certs

By moving client certificates between these directories, the trust settings can be adjusted accordingly. The public key of a client certificate is automatically stored in the above untrusted certificate directory the first time the client attempts to connect to a secure endpoint. By subsequently moving the public key to the trusted certificate directory, the client is trusted the next time it attempts to connect.

AutomaticallyTrustAllClientCertificates

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

Set up a trust relationship on the client

Depending on the OPC UA Client employed, different steps may need to be taken so that the OPC UA Client trusts the OPC UA Server. Typically, for client applications with a graphical user interface, a warning message is displayed the first time you connect to the server, whereby the server certificate can then be classified as trustworthy.

The following instruction is therefore only valid for the TwinCAT OPC UA Client.

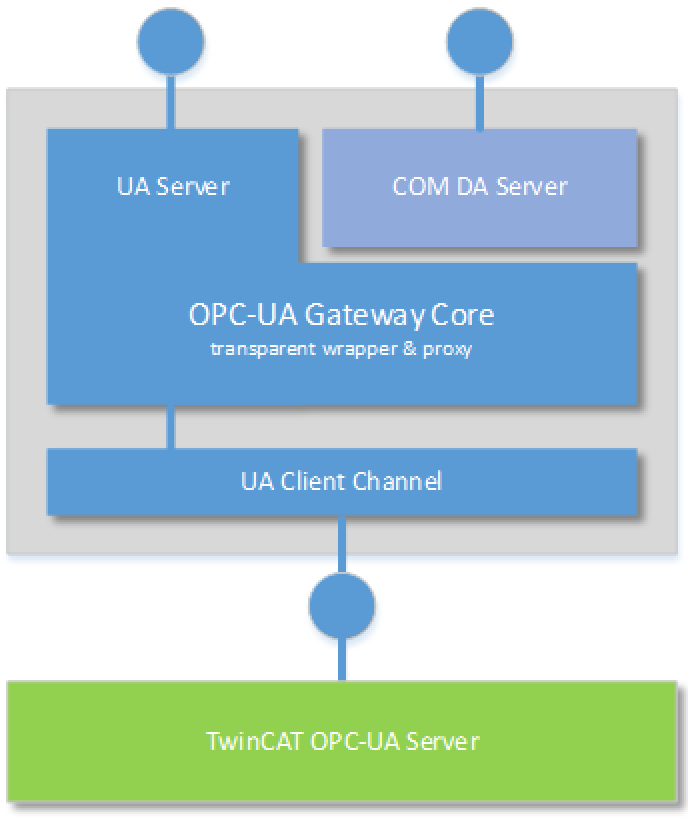
The public key of the OPC UA Server is located as a DER file in the following directory: *%InstallDir%\Server\PKI\CA\own\certs*

In the case of the TwinCAT OPC UA Client, copy the file into the corresponding "Trusted" directory: *%InstallDir%\Client\PKI\CA\trusted\certs*

4.5 Gateway

4.5.1 Overview

The TwinCAT OPC UA Gateway is the latest addition to the TS6100/TF6100 software product. It not only includes a conventional OPC DA interface for connecting older OPC COM DA applications to the TwinCAT OPC UA Server and can therefore be regarded as the successor of the old TwinCAT OPC DA Server (TS6120/TF6120), it also offers an OPC UA interface for converting several basic TwinCAT OPC UA Servers to a central OPC UA Server.



4.5.2 Quick start

The TwinCAT OPC UA Gateway is available for download as a separate setup. The setup automatically configures access to a TwinCAT OPC UA Server running on the same computer as the gateway.

If more than one OPC UA Server is added to the gateway, or if the server is running on a different computer, the standard configuration has to be modified. Use the Configurator [▸ 199] to configure these settings.

Configuration of the TwinCAT OPC UA Server

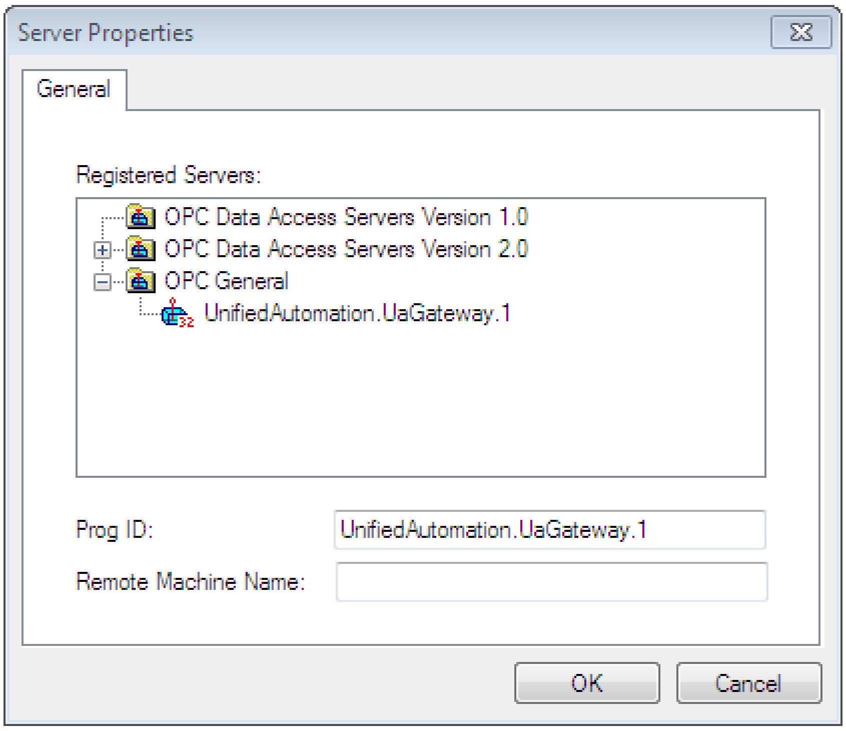
Check the configuration of the OPC UA Server and make sure that it is operating as expected before continuing.

For further information regarding the configuration of the OPC UA Server, read the Quick start [▸ 35] in the section "OPC UA Server".

Quick start – OPC COM DA

To connect an OPC COM DA Client to the gateway, start the client and establish a connection to the following ProgId:

```
UnifiedAutomation.UaGateway.1
```


Server Properties

General

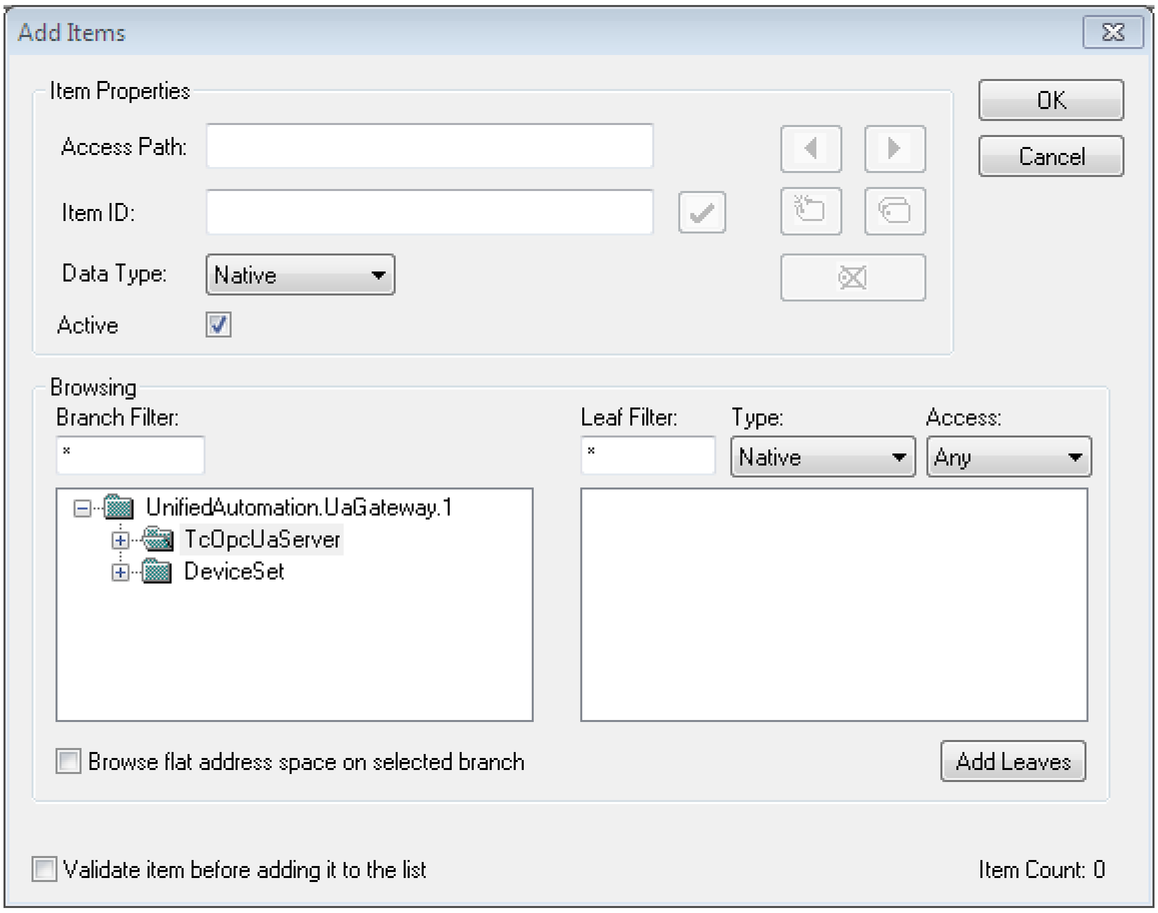
Registered Servers:

- OPC Data Access Servers Version 1.0
- OPC Data Access Servers Version 2.0
- OPC General
  - UnifiedAutomation.UaGateway.1

Prog ID: UnifiedAutomation.UaGateway.1

Remote Machine Name:

When browsing the gateway, one or more OPC UA Servers will be visible in the namespace of the gateway.


Add Items

Item Properties

Access Path:

Item ID:

Data Type: Native

Active ☑

Browsing

Branch Filter: *

- UnifiedAutomation.UaGateway.1
  - TcOpcUaServer
  - DeviceSet

Leaf Filter: *

Type: Native

Access: Any

☐ Browse flat address space on selected branch

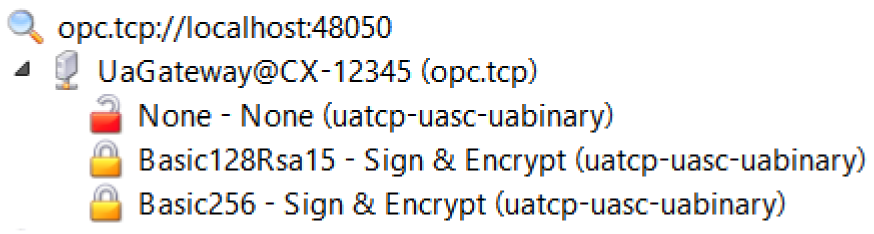
☐ Validate item before adding it to the list

Item Count: 0

Quick start – OPC UA

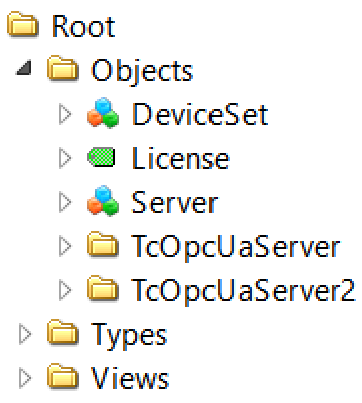
The gateway not only offers an OPC COM DA interface, but also allows the aggregation of one or more OPC UA Servers. The gateway also opens an OPC UA interface for this purpose. The gateway can be accessed via the following OPC UA Server URL:

```
opc.tcp://[HostnameOrIpAddressOrLocalhost]:48050
```



opc.tcp://localhost:48050
UaGateway@CX-12345 (opc.tcp)
None - None (uatcp-uasc-uabinary)
Basic128Rsa15 - Sign & Encrypt (uatcp-uasc-uabinary)
Basic256 - Sign & Encrypt (uatcp-uasc-uabinary)

The namespace of the gateway then contains all underlying TwinCAT OPC UA Servers.



Root
Objects
DeviceSet
License
Server
TcOpcUaServer
TcOpcUaServer2
Types
Views

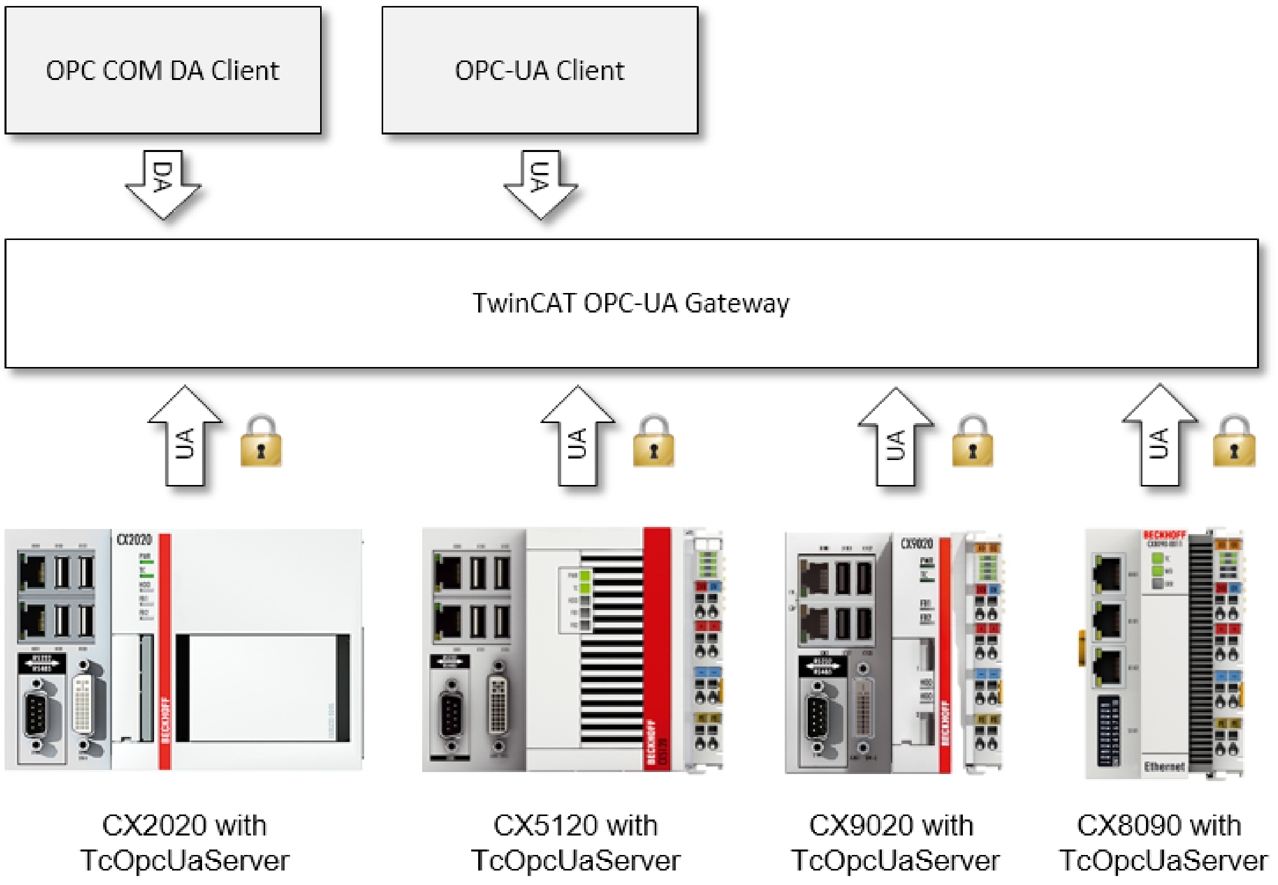
4.5.3 Licensing

The TwinCAT OPC UA Gateway is supplied free of charge. No further license purchase is required.

Note that the gateway component can only be used for connecting to TwinCAT OPC UA Servers. The software prevents connections to third-party UA Servers. The Unified Automation UA Gateway is recommended where the environment necessitates the connection with a UA Server from a third party. It can be purchased from http://www.unified-automation.com.

4.5.4 Scenarios

On account of the open and flexible PC-based automation technology from Beckhoff, the OPC UA Gateway can be operated and installed in different ways. The following section describes the various setup scenarios and explains the advantages and disadvantages of each configuration.


OPC COM DA Client
OPC-UA Client
DA
UA
TwinCAT OPC-UA Gateway
UA
UA
UA
UA
CX2020 with TcOpcUaServer
CX5120 with TcOpcUaServer
CX9020 with TcOpcUaServer
CX8090 with TcOpcUaServer

Gateway and UA Server on the same computer

In this scenario, the gateway and the UA Server are installed on the same computer. The gateway is configured with the standard settings in order to establish a connection with the local OPC UA Server with the following Server URL: opc.tcp://localhost:4840.

This scenario only works on non-Windows CE devices.

The TwinCAT OPC UA Server is also available for Windows CE, but the gateway is only available for "big" Windows platforms.

Gateway and UA Server on different computers

In this scenario, the gateway and the UA Server are installed on different computers. The gateway is configured for the establishment of a connection with the remote OPC UA Server by defining the latter's corresponding Server URL, e.g. opc.tcp://192.168.1.1:4840.

The OPC UA Server may be installed on a small embedded device (e.g. a CX8090 with Windows CE), while the gateway component is installed on a separate big Windows platform, e.g. a central server device.

Gateway connected to multiple UA Server devices

Please note that, regardless of the scenarios described above, it is also possible to connect other TwinCAT OPC UA Servers to the gateway. For example, the gateway can be configured to access the following servers:

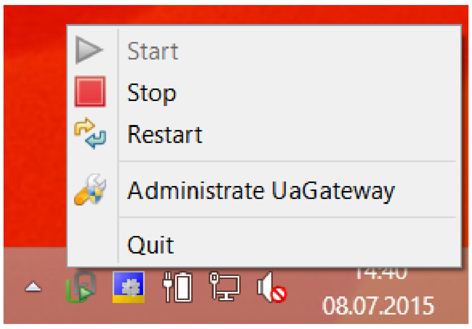
- the local TwinCAT OPC UA Server (e.g. opc.tcp://localhost:4840)
- a remote TwinCAT OPC UA Server (e.g. opc.tcp://192.168.1.1:4840)
- another remote TwinCAT OPC UA Server (e.g. opc.tcp://192.168.1.21:4841)
- ...

4.5.5 Configurator

4.5.5.1 Overview

The UA Gateway Administration Tool is a graphic user interface for the configuration of the gateway.

The tool is opened via the context menu of the gateway symbol in the Windows taskbar. After starting the administration tool via the command **Administrate UaGateway**, the graphic user interface appears.

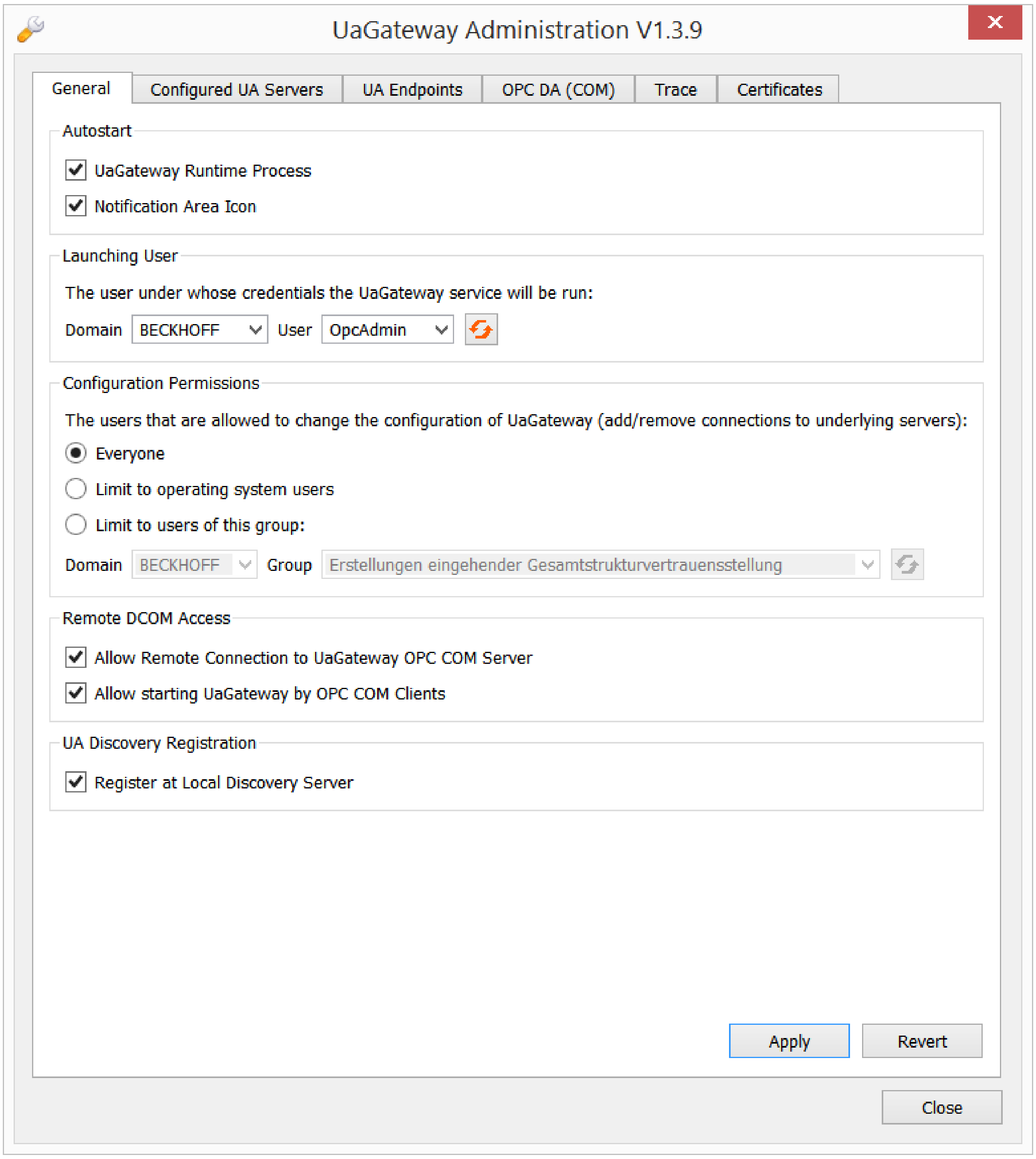


The interface offers several configuration options:

- General settings [▶ 199]
- Additional UA Servers [▶ 201]
- Additional endpoints [▶ 202]
- OPC COM DA settings [▶ 203]

4.5.5.2 General settings

The General tab displays general settings of the UA Gateway.


UaGateway Administration V1.3.9

General | Configured UA Servers | UA Endpoints | OPC DA (COM) | Trace | Certificates

Autostart
- [✓] UaGateway Runtime Process
- [✓] Notification Area Icon

Launching User

The user under whose credentials the UaGateway service will be run:

Domain: BECKHOFF   User: OpcAdmin

Configuration Permissions

The users that are allowed to change the configuration of UaGateway (add/remove connections to underlying servers):
- [✓ Everyone]
- Limit to operating system users
- Limit to users of this group:

Domain: BECKHOFF   Group: Erstellungen eingehender Gesamtstrukturvertrauensstellung

Remote DCOM Access
- [✓] Allow Remote Connection to UaGateway OPC COM Server
- [✓] Allow starting UaGateway by OPC COM Clients

UA Discovery Registration
- [✓] Register at Local Discovery Server

Autostart

In this area you can configure the autostart behavior of the UA Gateway.

Activate **UaGateway Runtime Process** to start the UA Gateway Service automatically when the computer is switched on.

Activate the **Notification Area Icon** to start the symbol of the notification area when a user logs on.

Launching User

The UA Gateway is executed as a Windows NT service. This service is assigned a specific user context so that COM/DCOM can be properly configured. The user you select is assigned to the UA Gateway service. In addition, the user is granted a LogOnAsService right (so he/she can start the service) and is added to a local user group ("UaGatewayUsers"). This group is added to the Access Control List (ACL) of the local machine. For proper COM/DCOM configuration you must add to this group all the users who are permitted to start and access the UA Gateway.

Configuration Permissions

It is possible to allow only certain users to change the configuration of the UA Gateway, i.e. to add or remove connections to basic servers. You can choose from the following settings:

Everyone	Any user (including users anonymously logged on to UA) who can contact the UA Gateway can change the configuration.
Limit to operating system users	Only local users and users within the same domain can change the configuration.
Limit to users of this group	Only users within a specific group to change the configuration. If not all available groups are displayed in the **Group** drop-down list (or a newly created group is missing), use the **Refresh** button to read this group again.

Remote DCOM Access

When **Allow Remote Connection to UaGateway OPC COM Server** is enabled, DCOM port 135 and the executable UA Gateway are added to the firewall exception list.

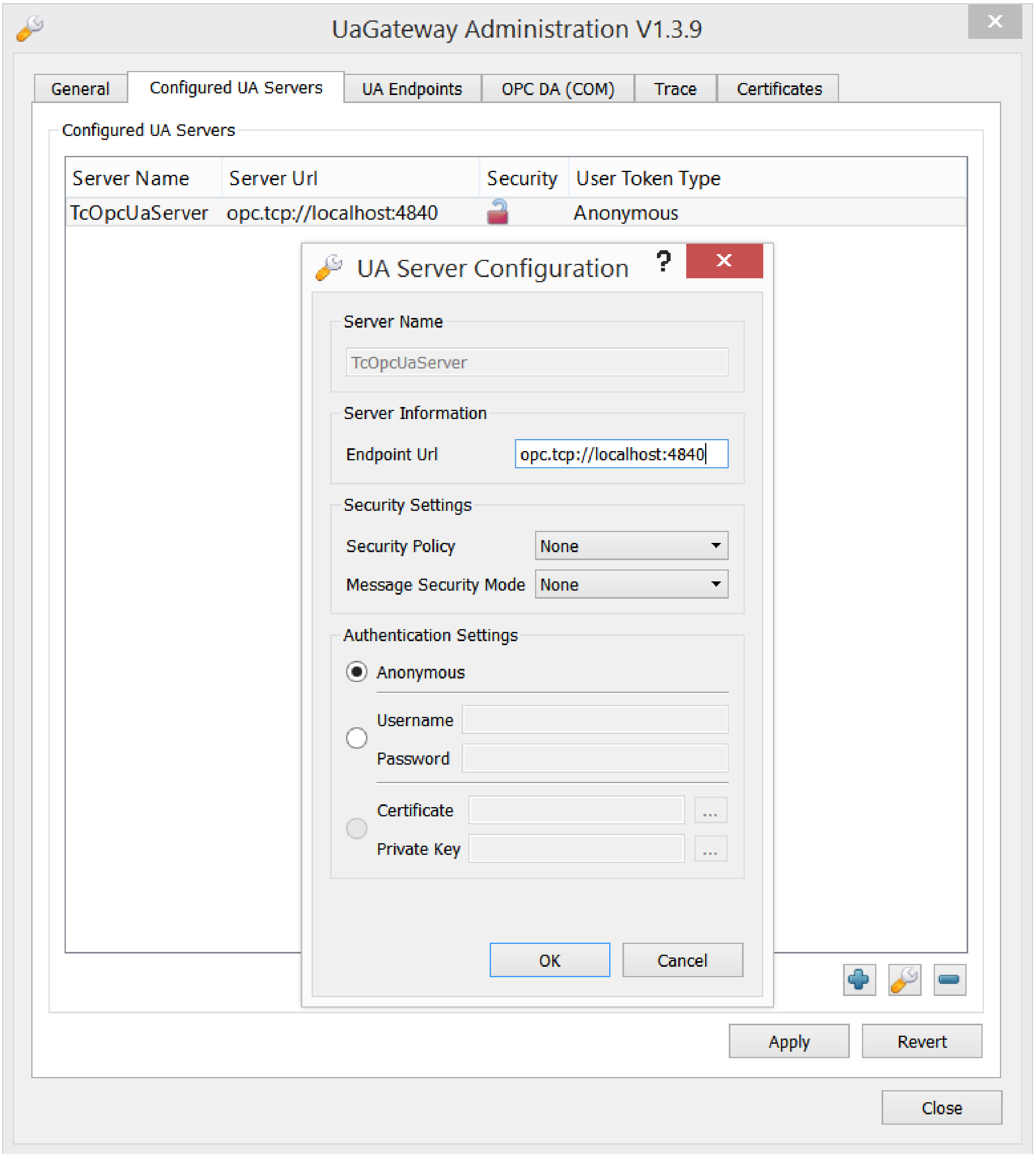
If **Allow starting UaGateway by DCOM Clients** is disabled, DCOM clients cannot start the UA Gateway. In this case, UA Gateway can still be started or stopped using the Notification Area Icon or the Start menu entries.

UA Discovery Registration (UA Local Discovery Server)

Activate **Register at Local Discovery Server** if the UA Gateway is to be registered with the OPC UA LDS (Local Discovery Server), if one is installed.

4.5.5.3 Additional UA Servers

The **Configured UA Servers** tab offers options for the configuration of the underlying OPC UA Servers. By default the gateway already establishes a connection with the local OPC UA Server (which is running on the same computer).


UaGateway Administration V1.3.9

Configured UA Servers

Server Name	Server Url	Security	User Token Type
TcOpcUaServer	opc.tcp://localhost:4840	🔓	Anonymous

UA Server Configuration

Server Name: TcOpcUaServer

Server Information
Endpoint Url: opc.tcp://localhost:4840

Security Settings
Security Policy: None
Message Security Mode: None

Authentication Settings
● Anonymous
○ Username / Password
○ Certificate / Private Key

To configure or remove further OPC UA Servers from the configuration, click on the Plus and Minus buttons in the lower right-hand corner and then **Apply** to save the changes.

4.5.5.4 Additional endpoints

The **UA Endpoints** tab shows the settings for the UA endpoint configuration.

The UA endpoint is the connection information that a UA Client requires to connect to the gateway.

General

Use the checkboxes to specify the logon methods that a client can use to connect to your UA Gateway.

Endpoints

Here you can define all settings required for different UA endpoints. The endpoint is configured with default settings as standard. These represent a single UA endpoint offering two security options: None and Basic128RSsa15.

The None security option allows every UA Client to connect to the UA Gateway. This configuration is only recommended during commissioning and testing. In a production environment this configuration should be switched off.

The various configuration elements are described in the following sections.

Network configuration

Endpoint URL	This is the endpoint URL of the UA Gateway as seen in FindServers and GetEndpoint calls.
Protocol	This is the protocol used for this endpoint.
Host name/IP	This is the host name of the UA Gateway (it can also be the IP address of the PC running the UA Gateway).
Network adapter	This is the network adapter to be used for binding. The available options are:
All	Binding is to be applied to all IP addresses of the computer. The endpoint will be accessible via the given port on all IP addresses.
Network adapter	Select a network adapter and an IP address (below) to bind only to that address. The endpoint will only be accessible to clients that establish a connection with the selected IP address.
Local only	With this selection, the UA Gateway only establishes a binding with the loopback adapter. The endpoint can only be reached by clients running on the same machine as the UA Gateway.
Port	This is the TCP port of the endpoint (normally 48050).

Security

In this area you can configure the supported security settings of the endpoint. Select the checkboxes for the security options you want to apply to a specific endpoint. For options other than "None", the available message security mode(s) must be specified. Signing ensures that messages cannot be changed and that they are exchanged between applications that have established a connection. Encryption guarantees that no one can read the messages.

4.5.5.5 OPC COM DA settings

The **OPC DA (COM)** tab shows the settings for the configuration of the COM DA Server of the UA Gateway.

The following section describes how to configure the COM DA Server of the UA Gateway using the administration tool.

General

ItemIDs of the COM DA Server are formed from the URI namespace and the identifier of the variable node in the OPC UA address space. The namespace part can be omitted in the case of a single namespace.

In the **Default Name Space** drop-down field you can specify the default namespace with the namespace of a basic OPC server. The ItemIDs of this particular namespace can then be reached by specifying the identifier only, because the default namespace is automatically added internally when an element is accessed. This feature can be used to reconfigure all ItemIDs in the client that accesses the UA Gateway server, if the latter serves as a tunnel solution for a basic COM DA Server, while maintaining the ItemIDs of the original COM DA Server.

In the second drop-down field the **Timestamp Source** can be defined. The following options are available:

Internal	The time stamps are generated by the OPC COM DA Server.
SourceTimestamp	The SourceTimestamps are used as time stamps provided by the OPC COM DA Server.
ServerTimestamp	The ServerTimestamps are used as time stamps provided by the OPC COM DA Server.

Properties mapping from UA to COM DA

When connecting to the UA Gateway's OPC COM DA Server, all six standard properties (DataType, Value, Quality, TimeStamp, AccessRights and ScanRate) are automatically assigned. Underlying OPC Servers can provide further properties (e.g. user-defined properties, DI properties, etc.). These properties can be assigned to vendor-specific properties (PropertyID □ 5000) in the COM DA Server of the UA Gateway.

These vendor-specific PropertyIDs are automatically assigned when the properties are requested for the first time. This dialog allows you to change the assigned PropertyIDs or configure how the OPC UA properties in the UA Gateway address space are assigned to the vendor-specific COM DA properties. You have to define the property name on the UA side and the namespace of the property in the UA Gateway and assign it to the COM DA PropertyIDs. When connecting to the UA Gateway's COM DA server, you can navigate through the available properties (QueryAvailableProperties) of a single OPCItem and then you will be able to see the associated properties as they have been configured (in the range of vendor-specific PropertyIDs above 5000).

Press the **[+]** or **[-]** key respectively to add or remove a certain property. To change the contents of a particular field, double-click it and enter the required values. Double-clicking a value in the **UA Property NameSpace URI** column displays a drop-down menu where you can make a selection.

If you add a new property by pressing **[+]**, the values of the last entry are copied to the new line and the PropertyID is automatically incremented.

4.5.6 Migrating from Tx6120

One of the primary purposes of the UA Gateway is to provide a sustainable connectivity in order to replace the Tx6120 OPC DA supplement/function. Observe the following notes if you wish to migrate Tx6120 OPC DA to UA Gateway.

Standard configuration

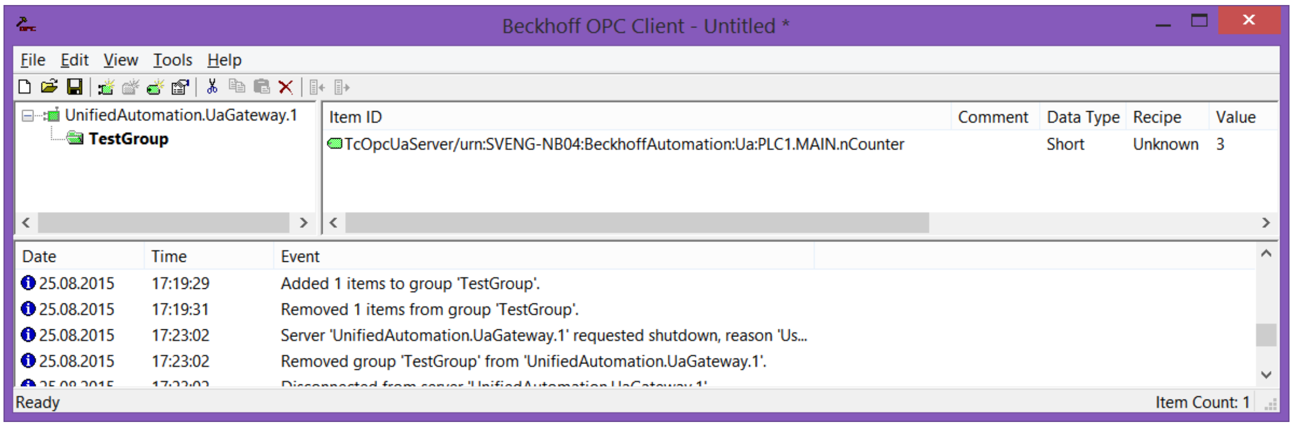
The standard configuration of the UA Gateway automatically establishes a connection with the local OPC UA Server and offers the OPC DA Clients an OPC DA interface. For a connection based on this standard configuration, the OPC DA clients must consider the following points:

- The default ProgID of the UA Gateway is "UnifiedAutomation.Gateway.1". The TwinCAT OPC DA Server uses a different ProgID ("Beckhoff.TwinCATOpcServerDA").
- The UA Gateway always uses a ProgID instead of multiple clones.
- The ItemIdentifier of an OPC symbol is generated differently in the UA Gateway. This behavior can be changed.

Changing the syntax of an ItemIdentifier

The syntax used by the UA Gateway for ItemIdentifier can be changed so that the latter corresponds more to the type of the TwinCAT OPC DA Server. By default, the UA Gateway uses a different syntax to that of the TwinCAT OPC DA Server when creating its identifiers.

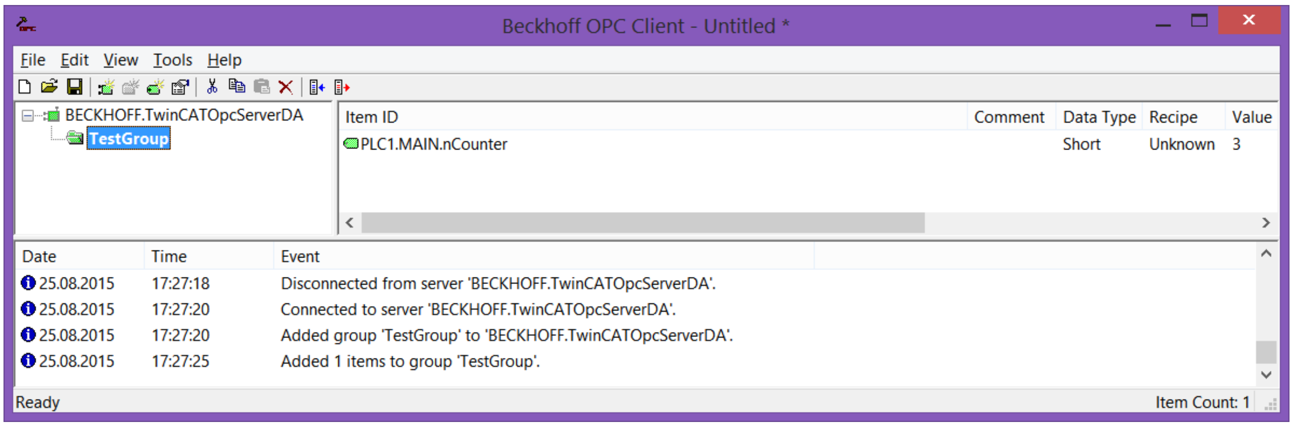
UA Gateway sample:


Beckhoff OPC Client - Untitled *
File Edit View Tools Help
UnifiedAutomation.UaGateway.1
TestGroup
Ready
Item Count: 1

Item ID	Comment	Data Type	Recipe	Value
TcOpcUaServer/urn:SVENG-NB04:BeckhoffAutomation:Ua:PLC1.MAIN.nCounter		Short	Unknown	3

Date	Time	Event
25.08.2015	17:19:29	Added 1 items to group 'TestGroup'.
25.08.2015	17:19:31	Removed 1 items from group 'TestGroup'.
25.08.2015	17:23:02	Server 'UnifiedAutomation.UaGateway.1' requested shutdown, reason 'Us...
25.08.2015	17:23:02	Removed group 'TestGroup' from 'UnifiedAutomation.UaGateway.1'.
25.08.2015	17:23:02	Disconnected from server 'UnifiedAutomation.UaGateway.1'

Sample TwinCAT OPC DA Server:


Beckhoff OPC Client - Untitled *
File Edit View Tools Help
BECKHOFF.TwinCATOpcServerDA
TestGroup
Ready
Item Count: 1

Item ID	Comment	Data Type	Recipe	Value
PLC1.MAIN.nCounter		Short	Unknown	3

Date	Time	Event
25.08.2015	17:27:18	Disconnected from server 'BECKHOFF.TwinCATOpcServerDA'.
25.08.2015	17:27:20	Connected to server 'BECKHOFF.TwinCATOpcServerDA'.
25.08.2015	17:27:20	Added group 'TestGroup' to 'BECKHOFF.TwinCATOpcServerDA'.
25.08.2015	17:27:25	Added 1 items to group 'TestGroup'.

The UA Gateway uses a prefix so that the underlying OPC UA Client from which the variable originates can be clearly identified.

The following steps are required to configure the UA Gateway so that it forms its identifiers in roughly the same way as the TwinCAT OPC DA Server. The functionality has been implemented to simplify the migration process.

1. Open the UA Gateway configuration file
   C:\Program Files (x86)\UnifiedAutomation\UaGateway\bin\uagateway.config.xml
2. Look for the following XML tags in the XML file:

```
<OpcServerConfig>
  <ComDaServerConfig>
    <ComDaNamespaceUseAlias>false</ComDaNamespaceUseAlias>
  </ComDaServerConfig>
</OpcServerConfig>
```

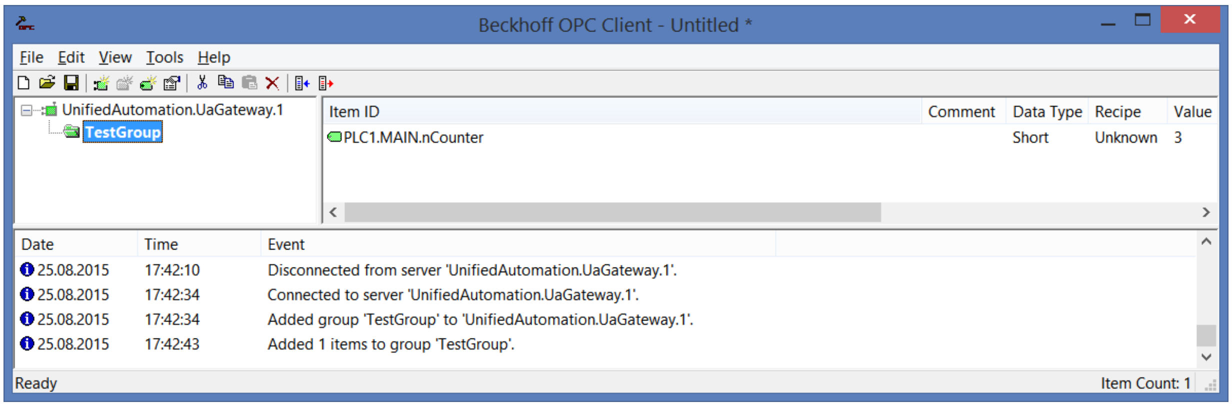
1. If the XML tag ComDaNamespaceUseAlias is set to "true", user-defined prefixes can be specified. To do this, look for the following XML tag in the same XML file:

```
<OpcServerConfig>
  <UaServerConfig>
    <ConfiguredNamespaces>
      ...
    </ConfiguredNamespaces
  </UaServerConfig>
</OpcServerConfig>
```

1. In this XML structure, identify the TwinCAT OPC UA Server namespace. By default, it should read as follows:

```
<OpcServerConfig>
  <UaServerConfig>
    <ConfiguredNamespaces>
      ...
      <Namespace>
        <Index>...</Index>
        <Uri>TcOpcUaServer/urn:Hostname:BeckhoffAutomation:Ua:PLC1</Uri>
        <AllowRenameUri>false</AllowRenameUri>
        <UniqueId>TcOpcUaServer#TcOpcUaServer/urn:Hostname:BeckhoffAutomation:Ua:PLC1</UniqueId>
        <ComAlias>...</ComAlias>
      </Namespace>
      ...
    </ConfiguredNamespaces
  </UaServerConfig>
</OpcServerConfig>
```

1. On your computer, the placeholder "..." may look different. Set <ComAlias> to your preferred prefix, for example "PLC1". The identifiers are then created with the prefix "PLC1".


Beckhoff OPC Client - Untitled *
File Edit View Tools Help
UnifiedAutomation.UaGateway.1
TestGroup
Ready
Item Count: 1

Item ID	Comment	Data Type	Recipe	Value
PLC1.MAIN.nCounter		Short	Unknown	3

Date	Time	Event
25.08.2015	17:42:10	Disconnected from server 'UnifiedAutomation.UaGateway.1'.
25.08.2015	17:42:34	Connected to server 'UnifiedAutomation.UaGateway.1'.
25.08.2015	17:42:34	Added group 'TestGroup' to 'UnifiedAutomation.UaGateway.1'.
25.08.2015	17:42:43	Added 1 items to group 'TestGroup'.

4.5.7 Security

4.5.7.1 Overview

One of the reasons for the success of OPC UA as communication technology is the various integrated security mechanisms. OPC UA-based data communication can be secured on two levels:

1. Transport layer
2. Application level

Endpoints

A server offers the client a list of different endpoints [▸ 105] 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 user name/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 [▸ 206].

Application level

In addition to the transport layer, a communication connection can also be secured at the application layer. For this purpose, various authentication mechanisms [▸ 106] are available, which are offered by the server endpoint.

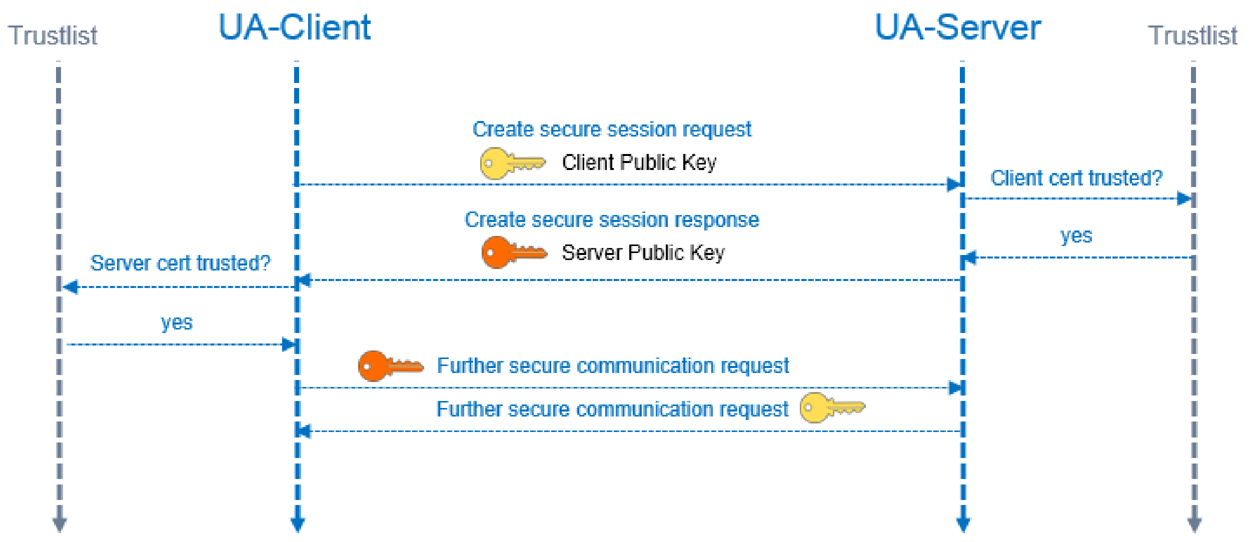
Also see about this

- Access rights [▸ 109]

4.5.7.2 Certificate exchange

To secure the communication connection at transport layer via a secure endpoint [▸ 105], it is necessary to establish a mutual trust between client and server.

By default, both the TwinCAT OPC UA Server and the TwinCAT OPC UA Client generate a machine-specific, self-signed certificate for authentication of the respective application at the first start.


Trustlist
UA-Client
UA-Server
Trustlist
Create secure session request
Client Public Key
Client cert trusted?
Create secure session response
Server Public Key
yes
Server cert trusted?
yes
Further secure communication request
Further secure communication request

Set up a trust relationship on the server

To establish a trust relationship between any OPC UA Client and the TwinCAT OPC UA Server, you need the public key of the client certificate. The server must trust this. This can be done via the file system, for example. The server manages the trust settings for client certificates in the PKI subdirectory.

- Trusted certificates: %InstallDir%\Server\PKI\CA\trusted\certs
- Untrusted certificates: %InstallDir%\Server\PKI\CA\rejected\certs

By moving client certificates between these directories, the trust settings can be adjusted accordingly. The public key of a client certificate is automatically stored in the above untrusted certificate directory the first time the client attempts to connect to a secure endpoint. By subsequently moving the public key to the trusted certificate directory, the client is trusted the next time it attempts to connect.

AutomaticallyTrustAllClientCertificates

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

Set up a trust relationship on the client

Depending on the OPC UA Client employed, different steps may need to be taken so that the OPC UA Client trusts the OPC UA Server. Typically, for client applications with a graphical user interface, a warning message is displayed the first time you connect to the server, whereby the server certificate can then be classified as trustworthy.

The following instruction is therefore only valid for the TwinCAT OPC UA Client.

The public key of the OPC UA Server is located as a DER file in the following directory: *%InstallDir%\Server\PKI\CA\own\certs*

In the case of the TwinCAT OPC UA Client, copy the file into the corresponding "Trusted" directory: *%InstallDir%\Client\PKI\CA\trusted\certs*

4.6 Sample Client

4.6.1 Overview

As of version 1.6.80 of the TwinCAT OPC UA Server, a small "UA Sample Client" program is automatically installed. The program enables you to browse the OPC UA namespace and to test the UA Server installation. It is located in the Windows Start menu and in the installation directory of the Supplement/ Function. You can run the program both directly on the UA Server and on a computer in your network.

The UA Sample Client currently offers the following features:

- Connecting to the OPC UA Server
- Establishing a secure connection with OPC UA Server (see Establishing a secure connection to OPC UA Server [▸ 209])
- Browsing the UA namespace of an OPC UA Server (see Browsing the UA namespace [▸ 212])
- Adding a UA node from the namespace to the watchlist, which reads the value of the node regularly (see Using the Watchlist [▸ 213])

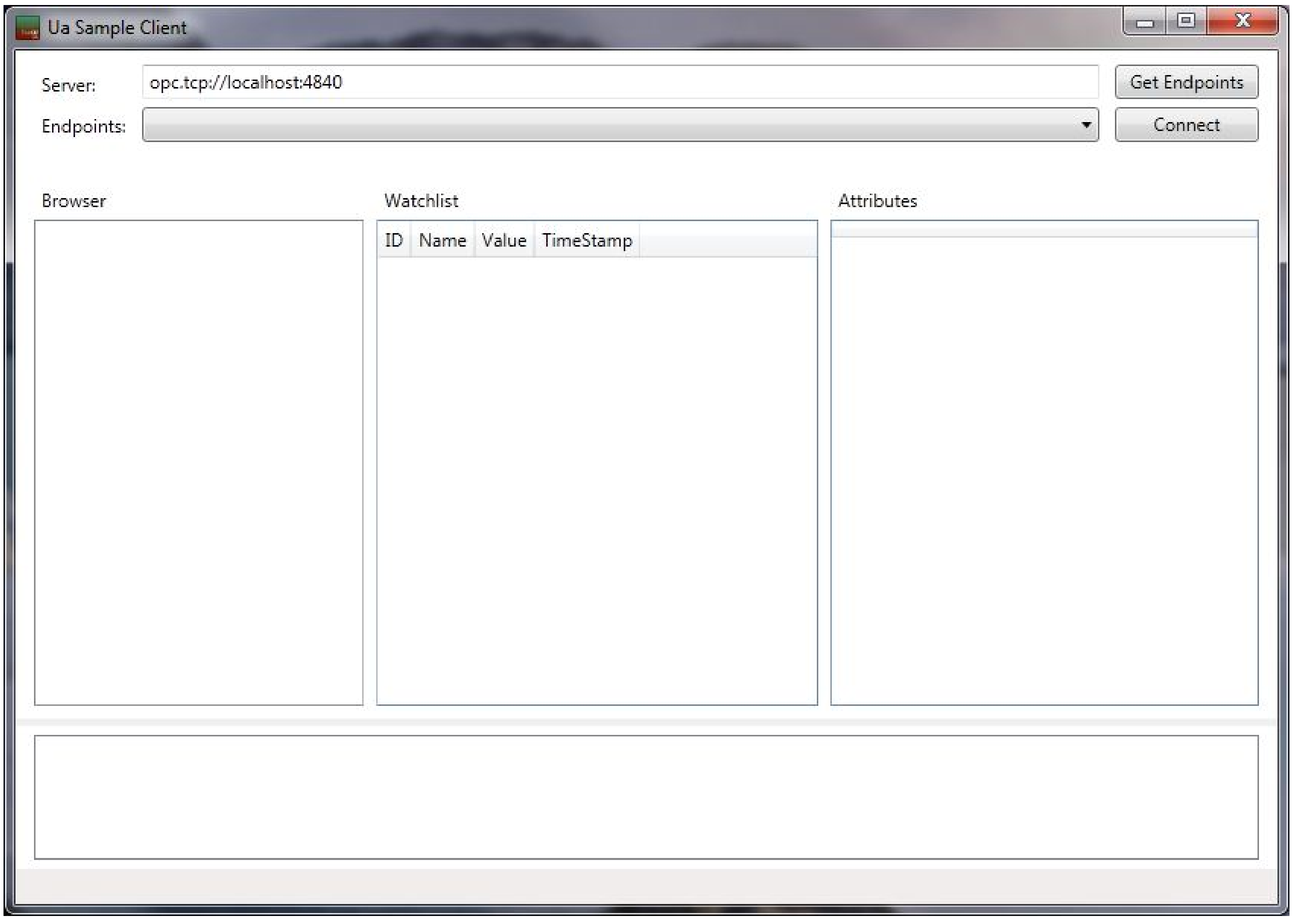
This application is only an OPC UA Sample Client. It does not offer any sophisticated functionalities, but has been developed to provide users with an easy-to-use interface for carrying out initial tests on the OPC UA Server

To start the application, run the *UA SampleClient.exe* file with the **Run as Admin** option.

Endpoints of the OPC UA Server

The UA Sample Client first connects itself to a specified server URL. The client acquires all endpoints of the OPC UA Server (see **Endpoints** drop-down list). The list returned to the server then contains more information about all the available endpoints the client can connect to. Each endpoint can contain the host name of the OPC UA Server instead of the IP address. The client then uses the information from the endpoint to connect to the server.

If the name solution does not work on the user's network, the client cannot connect. If the endpoint to which you want the client to connect contains the host name of the server, make sure that the name solution works on your network and that the host name is accessible on the server.

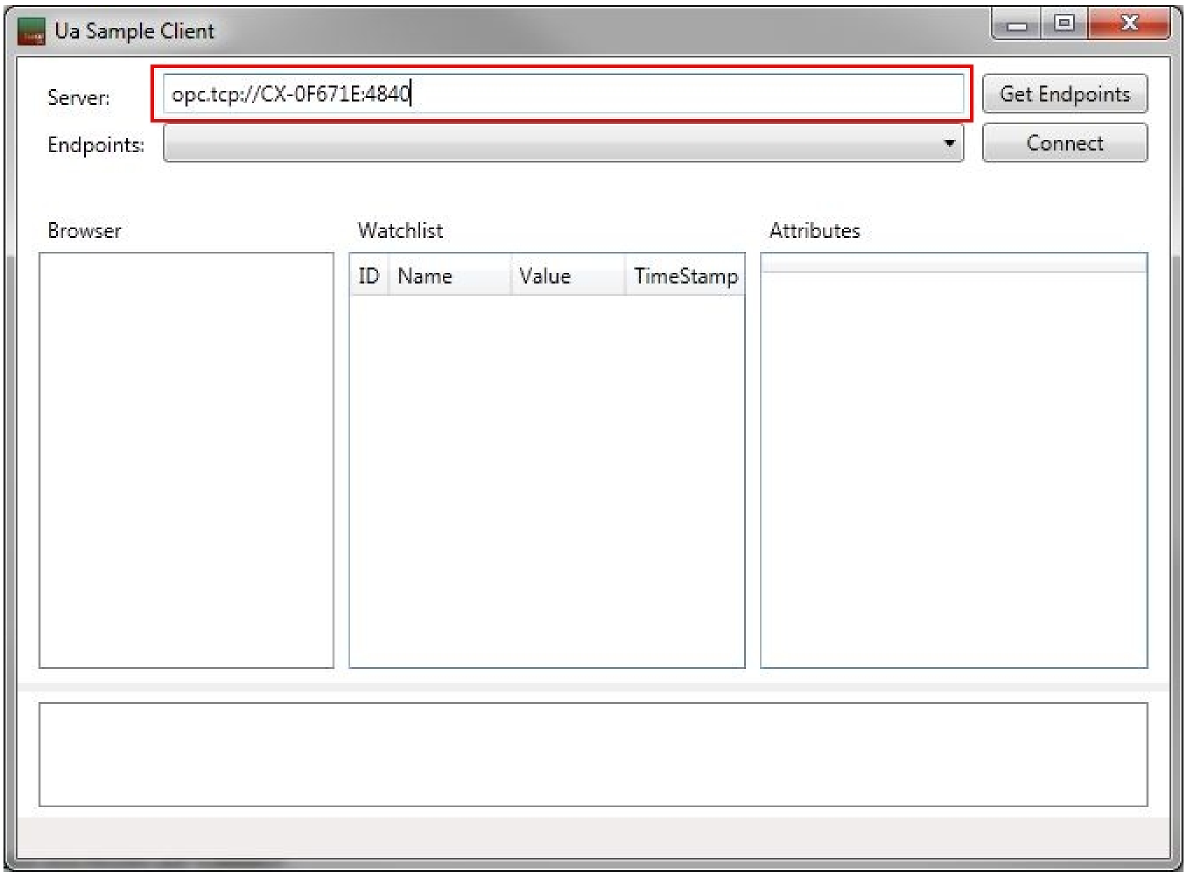


opc.tcp://localhost:4840

4.6.2 Establishing a secure connection to OPC UA Server

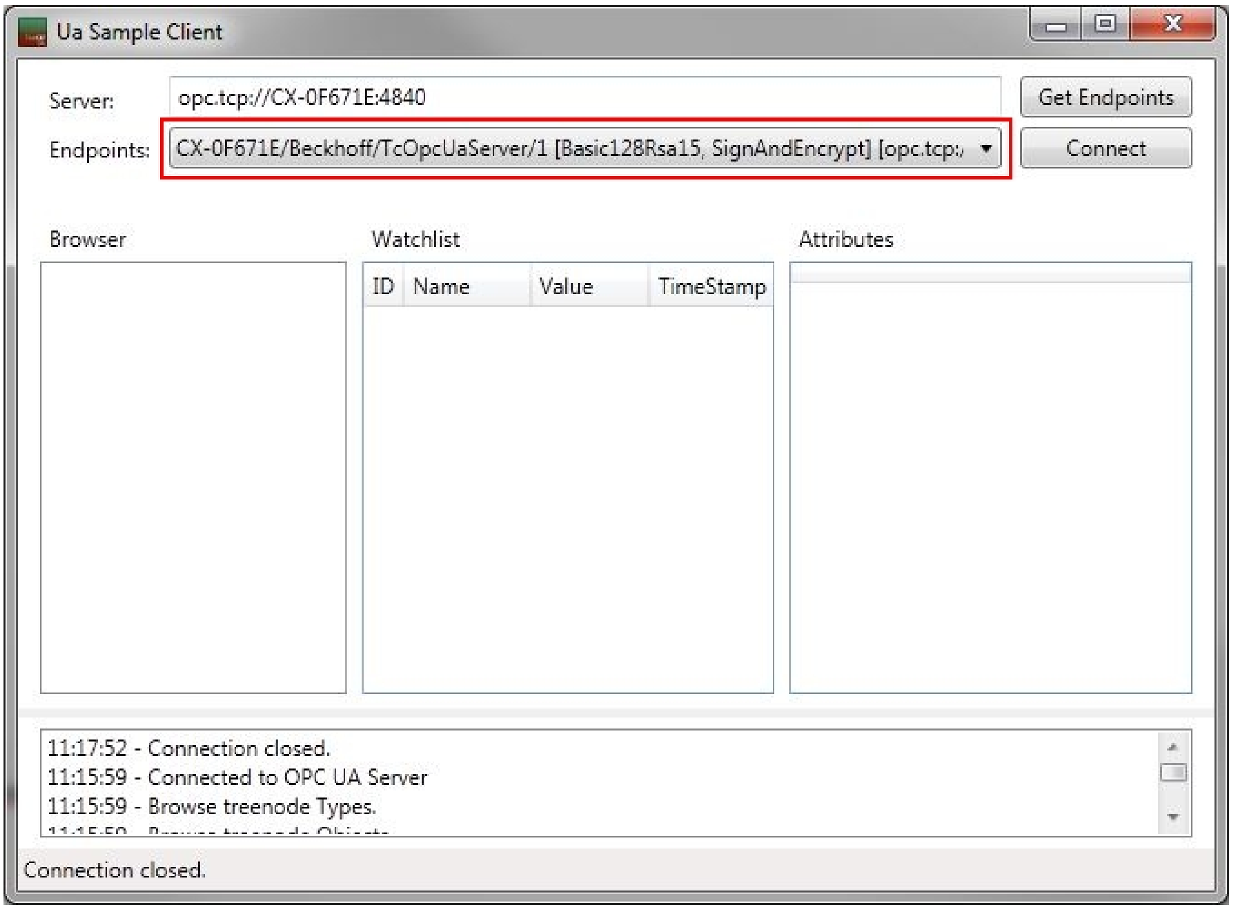
1. Enter the URL of an OPC UA Server in the upper text field of the UA Sample Client.
2. Click the **Get Endpoints** button.

⇨ The endpoints provided by the UA Server are then displayed in the **Endpoints** drop-down list.



3. In this sample, select the entry "<SomeName>/Beckhoff/TcOpcUaServer/1[Basic128Rsa15, SignAndEncrypt] [opc.tcp://<SomeName>:4840]" and click **Connect**.
You must copy the public key from the certificate of the UA Sample Client to the UA Server so that it "trusts" the Sample Client. Otherwise, the connection attempt is rejected by the UA Server via the secure

channel ("BadSecureChannelClosed"). Further information on the certificate management with the OPC UA Server can be found in the section Certificate exchange [▸ 206].



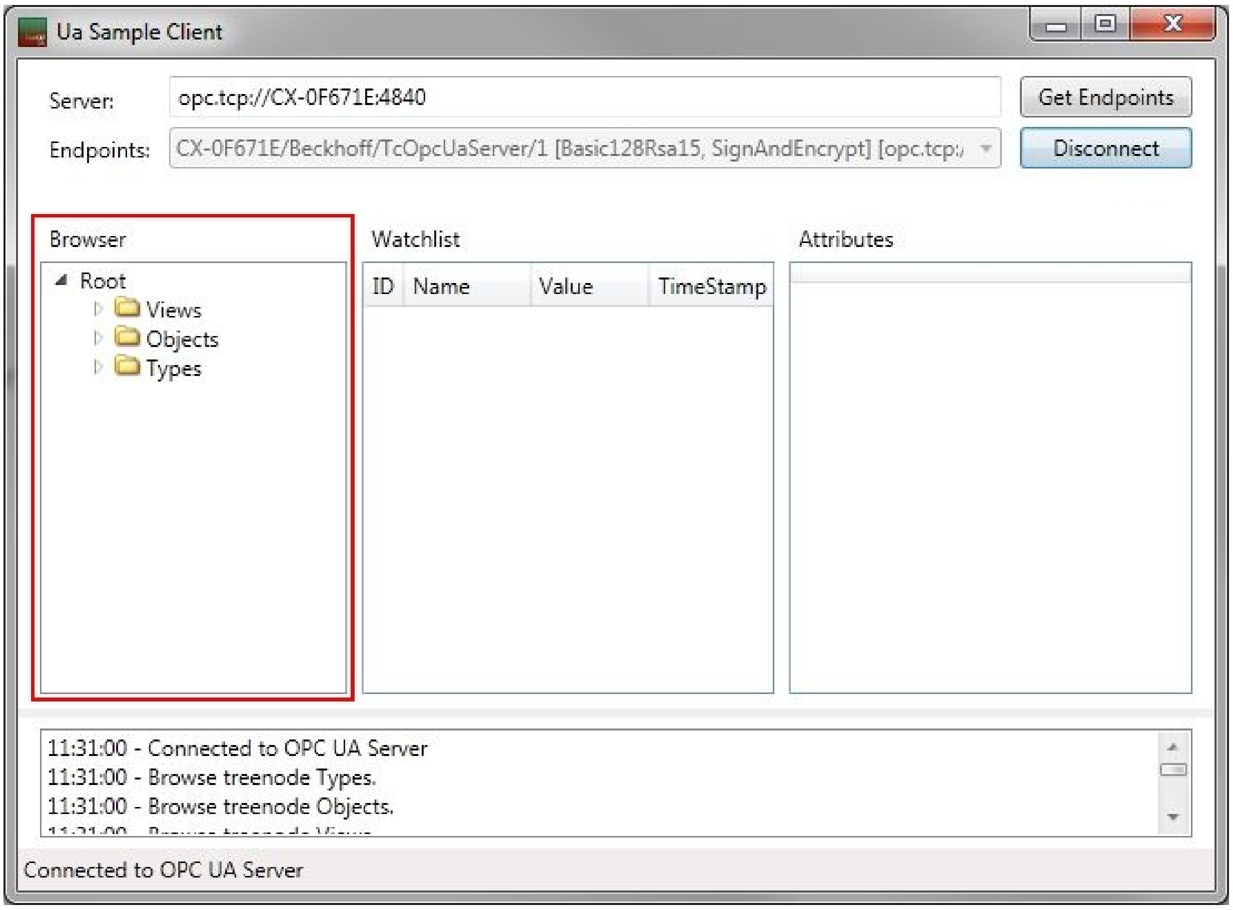
Server: opc.tcp://CX-0F671E:4840

Endpoints: CX-0F671E/Beckhoff/TcOpcUaServer/1 [Basic128Rsa15, SignAndEncrypt] [opc.tcp:/

```
11:17:52 - Connection closed.
11:15:59 - Connected to OPC UA Server
11:15:59 - Browse treenode Types.
```

Connection closed.

⇨ You can now use the **Browser** in the left half of the window to navigate through the UA namespace.



Server: opc.tcp://CX-0F671E:4840

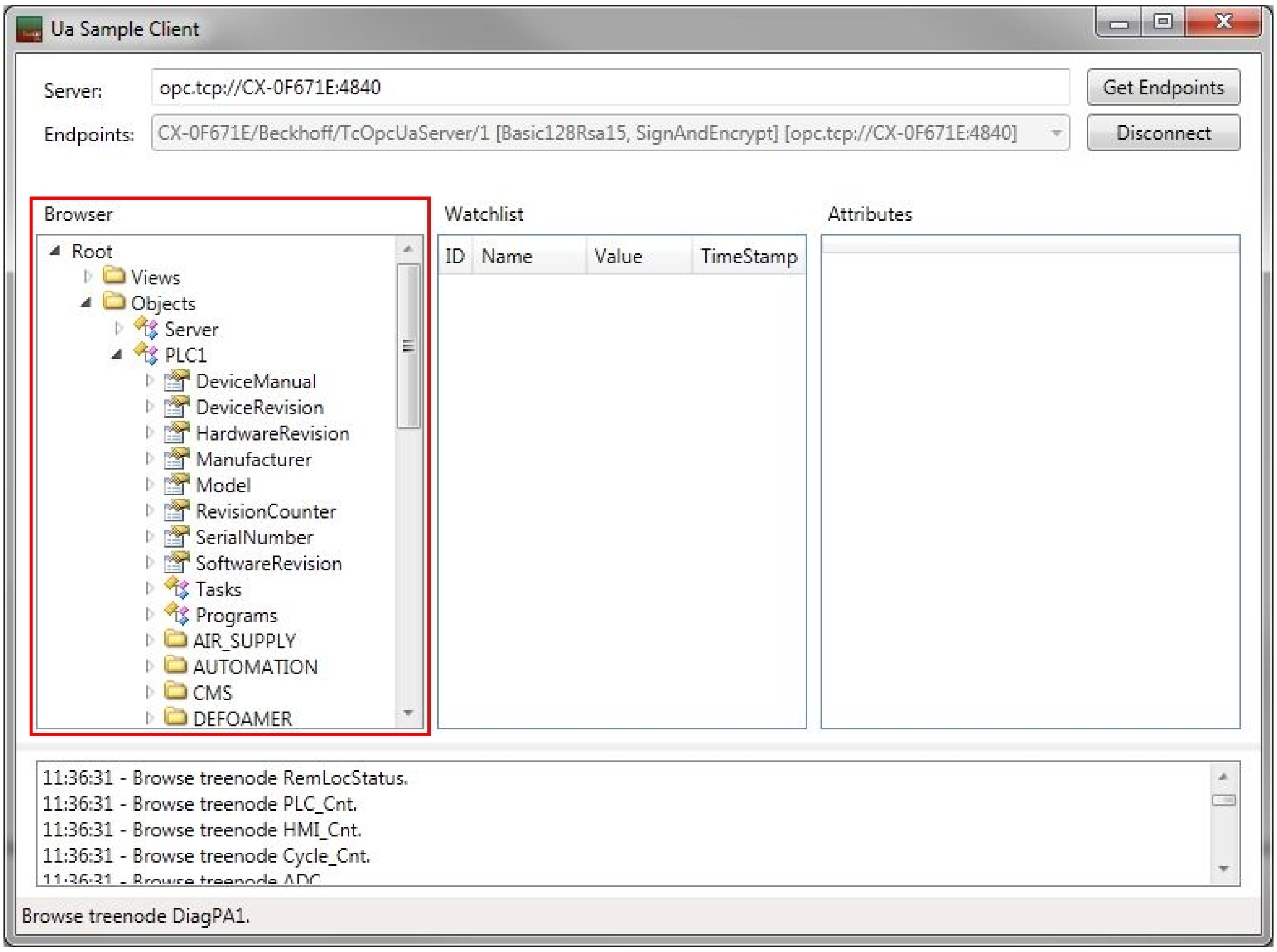
Endpoints: CX-0F671E/Beckhoff/TcOpcUaServer/1 [Basic128Rsa15, SignAndEncrypt] [opc.tcp:/

```
11:31:00 - Connected to OPC UA Server
11:31:00 - Browse treenode Types.
11:31:00 - Browse treenode Objects.
```

Connected to OPC UA Server

4.6.3 Browsing the UA namespace

When a successful connection has been established you can use the **Browser** in the left half of the UA Sample Client to navigate through the UA namespace. Below the node **PLC1** you will find the currently running PLC program, and you can display the variables declared there and released for UA.



Server: opc.tcp://CX-0F671E:4840

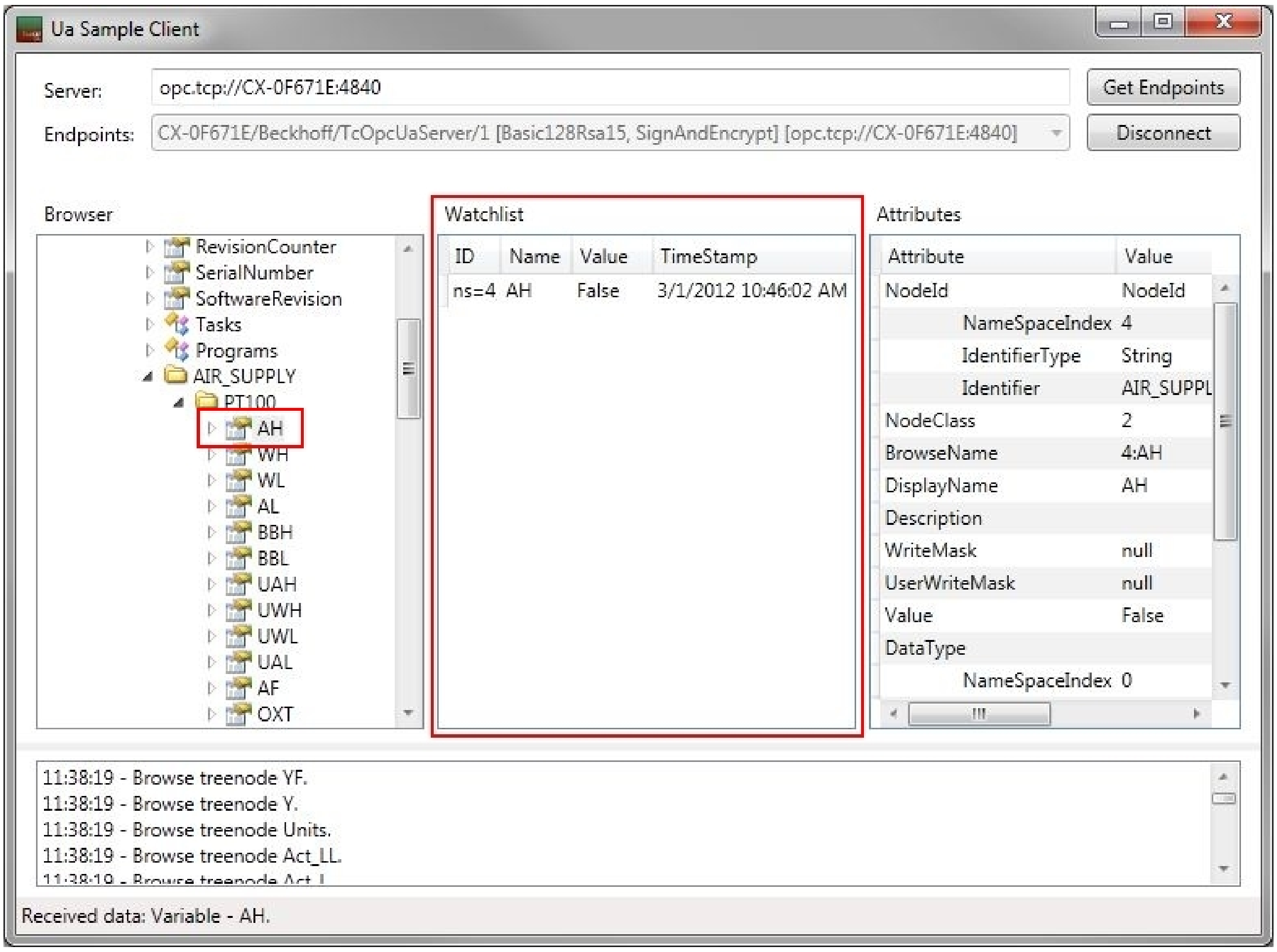
Endpoints: CX-0F671E/Beckhoff/TcOpcUaServer/1 [Basic128Rsa15, SignAndEncrypt] [opc.tcp://CX-0F671E:4840]

```
11:36:31 - Browse treenode RemLocStatus.
11:36:31 - Browse treenode PLC_Cnt.
11:36:31 - Browse treenode HMI_Cnt.
11:36:31 - Browse treenode Cycle_Cnt.
11:36:31 - Browse treenode ADC
```

Browse treenode DiagPA1.

4.6.4 Using the Watchlist

You can insert PLC variables from the UA namespace into a watchlist, for example to have their values read cyclically by the UA Sample Client. To do this, open the context menu of a variable and select **Add to Watchlist**. The variable is then transferred to the watchlist and its values are automatically read out cyclically from the PLC.



Server: opc.tcp://CX-0F671E:4840

Endpoints: CX-0F671E/Beckhoff/TcOpcUaServer/1 [Basic128Rsa15, SignAndEncrypt] [opc.tcp://CX-0F671E:4840]

Browser:
- RevisionCounter
- SerialNumber
- SoftwareRevision
- Tasks
- Programs
- AIR_SUPPLY
  - PT100
    - AH
    - WH
    - WL
    - AL
    - BBH
    - BBL
    - UAH
    - UWH
    - UWL
    - UAL
    - AF
    - OXT

Watchlist:

ID	Name	Value	TimeStamp
ns=4	AH	False	3/1/2012 10:46:02 AM

Attributes:

Attribute	Value
NodeId	NodeId
NameSpaceIndex	4
IdentifierType	String
Identifier	AIR_SUPPL
NodeClass	2
BrowseName	4:AH
DisplayName	AH
Description	
WriteMask	null
UserWriteMask	null
Value	False
DataType	
NameSpaceIndex	0

```
11:38:19 - Browse treenode YF.
11:38:19 - Browse treenode Y.
11:38:19 - Browse treenode Units.
11:38:19 - Browse treenode Act_LL.
11:38:19 - Browse treenode Act_L
```

Received data: Variable - AH.

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 [▸ 215].

5.1.1.3 E_OpcUAServerStatus

E_OpcUAServerStatus represents the runtime status of a TwinCAT OPC UA Server.

Syntax

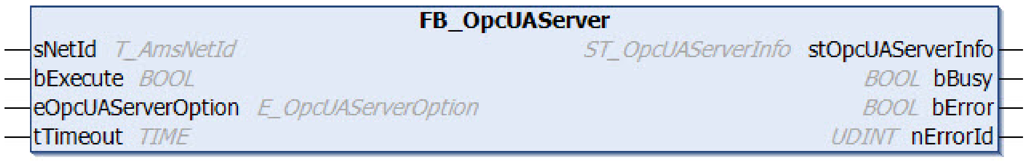
```
TYPE E_OpcUAServerStatus
(
    eOPCUAServerStatus_None,
    eOPCUAServerStatus_Alive,
    eOPCUAServerStatus_NotResponding
);
END_TYPE
```

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
```

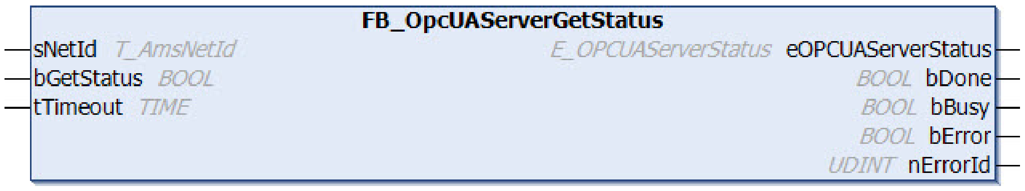
Inputs

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 [▸ 215]	Specifies the operation to be performed.
tTimeout	TIME	ADS Timeout

Outputs

Name	Type	Description
stOpcUAServerInfo	ST_OpcUAServerInfo [▸ 215]	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 [▶ 216]	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_PLCopen_OpcUa

5.2.1 Data types

5.2.1.1 E_UAAttributeID

Syntax

```
TYPE E_UAAttributeID:
(
    eUAAI_NodeID          := 1,
    eUAAI_NodeClass       := 2,
    eUAAI_BrowseName      := 3,
    eUAAI_DisplayName     := 4,
    eUAAI_Description     := 5,
    eUAAI_WriteMask       := 6,
    eUAAI_UserWriteMask   := 7,
    eUAAI_IsAbstract      := 8,
    eUAAI_Symmetric       := 9,
    eUAAI_InverseName     := 10,
    eUAAI_ContainsNoLoops := 11,
    eUAAI_EventNotifier   := 12,
    eUAAI_Value           := 13,
    eUAAI_DataType        := 14,
    eUAAI_ValueRank       := 15,
    eUAAI_ArrayDimensions := 16
)DINT;
END_TYPE
```

Values

Name	Description
NodeID	OPC UA NodeID
NodeClass	OPC UA NodeClass
BrowseName	OPC UA BrowseName
DisplayName	OPC UA DisplayName
Description	OPC UA Description
WriteMask	OPC UA WriteMask
UserWriteMask	OPC UA UserWriteMask
IsAbstract	OPC UA IsAbstract
Symmetric	OPC UA Symmetric
InverseName	OPC UA InverseName
ContainsNoLoops	OPC UA ContainsNoLoops
EventNotifier	OPC UA EventNotifier
Value	OPC UA Value
DataType	OPC UA DataType
ValueRank	OPC UA ValueRank
ArrayDimensions	OPC UA ArrayDimensions

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.2 E_UABrowseDirection

Syntax

```
TYPE E_UABrowseDirection:
(
    eUABD_Forward   := 0,
    eUABD_Inverse   := 1,
    eUABD_Both      := 2
)DINT;
END_TYPE
```

Values

Name	Description
eUABD_Forward	Forward references
eUABD_Inverse	Inverse references
eUABD_Both	Forward and inverse references

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.3 E_UABrowseResultMask

Syntax

```
TYPE E_UABrowseResultMask:
(
    eUABRM_ReferenceTypeId   := 1,
    eUABRM_IsForward         := 2,
    eUABRM_ReferenceTypeInfo := 3,
    eUABRM_NodeClass         := 4,
    eUABRM_BrowseName        := 8,
    eUABRM_DisplayName       := 16,
    eUABRM_TypeDefinition    := 32,
    eUABRM_TargetInfo        := 60,
    eUABRM_All               := 63
)DINT;
END_TYPE
```

Values

Name	Description
eUABRM_ReferenceTypeId	ReferenceTypeId
eUABRM_IsForward	IsForward
eUABRM_ReferenceTypeInfo	ReferenceTypeInfo
eUABRM_NodeClass	NodeClass
eUABRM_BrowseName	BrowseName
eUABRM_DisplayName	DisplayName
eUABRM_TypeDefinition	TypeDefinition
eUABRM_TargetInfo	TargetInfo
eUABRM_All	All

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.4 E_UAConnectionStatus

Syntax

```
TYPE E_UAConnectionStatus:
(
    Connected       := 0
    ConnectionError := 1,
    Shutdown        := 2
)DINT;
END_TYPE
```

Values

Name	Description
Connected	The connection has been established.
ConnectionError	An error occurred while establishing the connection.
Shutdown	The connection was disconnected.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

PLC library	Required version
Tc3_PLCopen_OpcUa	>= 3.2.11.0

5.2.1.5 E_UADataType

Syntax

```
TYPE E_UADataType:
(
    eUAType_Undefinied      := -1,
    eUAType_Null            := 0,
    eUAType_Boolean         := 1,
    eUAType_SByte           := 2,
    eUAType_Byte            := 3,
    eUAType_Int16           := 4,
    eUAType_UInt16          := 5,
    eUAType_Int32           := 6,
    eUAType_UInt32          := 7,
    eUAType_Int64           := 8,
    eUAType_UInt64          := 9,
    eUAType_Float           := 10,
    eUAType_Double          := 11,
    eUAType_String          := 12,
    eUAType_DateTime        := 13,
    eUAType_Guid            := 14,
    eUAType_ByteString      := 15,
    eUAType_XmlElement      := 16,
    eUAType_NodeId          := 17,
    eUAType_ExpandedNodeId  := 18,
    eUAType_StatusCode      := 19,
    eUAType_QualifiedName   := 20,
    eUAType_LocalizedText   := 21,
    eUAType_ExtensionObject := 22,
    eUAType_DataValue       := 23,
    eUAType_Variant         := 24,
    eUAType_DiagnosticInfo  := 25
)DINT;
END_TYPE
```

Values

Name	Description
eUAType_Undefinied	Undefinied
eUAType_Null	Zero
eUAType_Boolean	Boolean
eUAType_SByte	SByte
eUAType_Byte	Byte
eUAType_Int16	Int16
eUAType_UInt16	UInt16
eUAType_Int32	Int32
eUAType_UInt32	UInt32
eUAType_Int64	Int64
eUAType_UInt64	UInt64
eUAType_Float	Float
eUAType_Double	Double
eUAType_String	String
eUAType_DateTime	DateTime
eUAType_Guid	Guid
eUAType_ByteString	ByteString
eUAType_XmlElement	XmlElement
eUAType_NodeId	NodeId
eUAType_ExpandedNodeId	ExpandedNodeId
eUAType_StatusCode	StatusCode
eUAType_QualifiedName	QualifiedName
eUAType_LocalizedText	LocalizedText
eUAType_ExtensionObject	ExtensionObject
eUAType_DataValue	DataValue
eUAType_Variant	Variant
eUAType_DiagnosticInfo	DiagnosticInfo

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.6 E_UAIdentifierType

Syntax

```
TYPE E_UAIdentifierType:
(
    eUAIdentifierType_String  := 1,
    eUAIdentifierType_Numeric := 2,
    eUAIdentifierType_GUID    := 3,
    eUAIdentifierType_Opaque  := 4
)DINT;
END_TYPE
```

Values

Name	Description
eUAIdentifierType_String	String
eUAIdentifierType_Numeric	Numeric
eUAIdentifierType_GUID	GUID
eUAIdentifierType_Opaque	Opaque

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.7 E_UANodeClassMask

Syntax

```
TYPE E_UANodeClassMask:
(
    eUANCM_Unspecified   := 0,
    eUANCM_Object        := 1,
    eUANCM_Variable      := 2,
    eUANCM_Method        := 4,
    eUANCM_ObjectType    := 8,
    eUANCM_VariableType  := 16,
    eUANCM_ReferenceType := 32,
    eUANCM_DataType      := 64,
    eUANCM_View          := 128,
    eUANCM_All           := 255
)DINT;
END_TYPE
```

Values

Name	Description
eUANCM_Unspecified	Unspecified
eUANCM_Object	Object
eUANCM_Variable	Variable
eUANCM_Method	Method
eUANCM_ObjectType	ObjectType
eUANCM_VariableType	VariableType
eUANCM_ReferenceType	ReferenceType
eUANCM_DataType	DataType
eUANCM_View	View
eUANCM_All	All

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.8 E_UASecurityMsgMode

Syntax

```
TYPE E_UASecurityMsgMode:
(
    eUASecurityMsgMode_BestAvailable := 0,
    eUASecurityMsgMode_None          := 1,
    eUASecurityMsgMode_Sign          := 2,
    eUASecurityMsgMode_Sign_Encrypt  := 3
)DINT;
END_TYPE
```

Values

Name	Description
eUASecurityMsgMode_BestAvailable	Highest available security
eUASecurityMsgMode_None	No security
eUASecurityMsgMode_Sign	Signing
eUASecurityMsgMode_Sign_Encrypt	Signing and encryption

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.9 E_UASecurityPolicy

Syntax

```
TYPE E_UASecurityPolicy:
(
    eUASecurityPolicy_BestAvailable := 0
    eUASecurityPolicy_None          := 1,
    eUASecurityPolicy_Basic128      := 2,
    eUASecurityPolicy_Basic128Rsa15 := 3,
    eUASecurityPolicy_Basic256      := 4
)DINT;
END_TYPE
```

Values

Name	Description
BestAvailable	Highest available security.
None	Guideline for configurations with minimal security requirements.
Basic128	Guideline for configurations with low to medium security requirements.
Basic128Rsa15	Defines a security guideline for configurations with moderate to high security requirements.
Basic256	Defines a security policy for configurations with high security requirements.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.10 E_UAServerState

Syntax

```
TYPE E_UAServerState:
(
    Running             := 0
    Failed              := 1,
    NoConfiguration     := 2,
    Suspended           := 3,
    Shutdown            := 4,
    Test                := 5,
    CommunicationFault  := 6,
    Unknown             := 7
)DINT;
END_TYPE
```

Values

Name	Description
Running	Running
Failed	Failed
NoConfiguration	NoConfiguration
Suspended	Suspended
Shutdown	Shutdown
Test	Test
CommunicationFault	CommunicationFault
Unknown	Unknown

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

PLC library	Required version
Tc3_PLCopen_OpcUa	>= 3.2.11.0

5.2.1.11 E_UATransportProfile

Syntax

```
TYPE E_UATransportProfile:
(
    eUATransportProfileUri_UATcp             := 1,
    eUATransportProfileUri_WSHttpBinary      := 2,
    eUATransportProfileUri_WSHttpXmlOrBinary := 3,
    eUATransportProfileUri_WSHttpXml         := 4
)DINT;
END_TYPE
```

Values

Name	Description
eUATransportProfileUri_UATcp	UATcp
eUATransportProfileUri_WSHttpBinary	WSHttpBinary
eUATransportProfileUri_WSHttpXmlOrBinary	WSHttpXmlOrBinary
eUATransportProfileUri_WSHttpXml	WSHttpXml

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.12 E_UAUserIdentityTokenType

Syntax

```
TYPE E_UAUserIdentityTokenType:
(
    eUAUITT_Anonymous       := 0,
    eUAUITT_Username        := 1,
    eUAUITT_x509            := 2,
    eUAUITT_IssuedToeken    := 3
)DINT;
END_TYPE
```

Values

Name	Description
eUAUITT_Anonymous	Anonymous user.
eUAUITT_Username	Log in by user name.
eUAUITT_x509	Certificate file for logging in.
eUAUITT_IssuedToeken	Log in via token.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.13 ST_UABrowseDescription

Syntax

```
TYPE ST_UABrowseDescription:
STRUCT
    stStartingNodeId   : ST_UANodeId;
    eDirection         : E_UABrowseDirection;
    stReferenceTypeId  : ST_UANodeId;
    bIncludeSubtypes   : BOOL;
    eNodeClass         : E_UANodeClassMask;
    eResultMask        : E_UABrowseResultMask;
END_STRUCT
END_TYPE
```

Values

Name	Description
stStartingNodeId	Default Starting Node: ObjectRoot
eDirection	Default Browse Direction: Forward
stReferenceTypeId	Default ReferenceType: Hierarchical
bIncludeSubtypes	Default IncludeSubtypes: TRUE
eNodeClass	Default NodeClassMask: All
eResultMask	Default BrowseResultMask: All

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.14 ST_UAExpandedNodeID

Syntax

```
TYPE ST_UAExpandedNodeID:
STRUCT
    nServerIndex  : UDINT;
    sNamespaceURI : STRING(MAX_STRING_LENGTH);
    stNodeID      : ST_UANodeID;
END_STRUCT
END_TYPE
```

Values

Name	Description
nServerIndex	ServerIndex
sNamespaceURI	NamespaceName
stNodeID	NodeID (ST_UANodeID [▶ 228])

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.15 ST_UASessionConnectInfo

Syntax

```
TYPE ST_UASessionConnectInfo:
STRUCT
    sApplicationName      : STRING(MAX_STRING_LENGTH);

    eSecurityMode         : E_UASecurityMsgMode;
    eSecurityPolicyUri    : E_UASecurityPolicy;
    eTransportProfileUri  : E_UATransportProfile;

    tSessionTimeout       : TIME;
    tConnectTimeout       : TIME;
END_STRUCT
END_TYPE
```

Values

Name	Description
sApplicationUri (obsolete)	Application Uri maximum string length 255. From TcUAClient 2.0.0.14 or higher this is automatically specified by the certificate, as defined in the PLCOpen specification. Therefore no longer used in current library versions.
sApplicationName	Application name with a maximum string length of 255.
eSecurityMode	Security message mode. For available modes see E_UASecurityMsgMode [▸ 223].
eSecurityPolicyUri	Security policy Uri. For available security policy Uri see E_UASecurityPolicy [▸ 223].
eTransportProfileUri	Transport profile Uri. For available transport profile Uri see E_UATransportProfile [▸ 224];
stUserIdentTokenType	Structure with authentication data for logging on to the OPC UA Server. Full description under ST_UAUserIdentityTokenType [▸ 230].
tSessionTimeout	Session timeout value.
tConnectTimeout	Value for the connection timeout. This must be set at the UA_Connect function block to match the ADS timeout. The rule of thumb is: ADS Timeout > 2 * ConnectionTimeout.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.16 ST_UAIndexRange

Syntax

```
TYPE ST_UAIndexRange:
STRUCT
    nStartIndex : UDINT;
    nEndIndex   : UDINT;
END_STRUCT
END_TYPE
```

Values

Name	Description
nStartIndex	Start index of the data.
nEndIndex	End index of the data.

For all dimensions:

- StartIndex and EndIndex must be assigned.
- StartIndex must be smaller than EndIndex.
- To be able to access all elements in a dimension, StartIndex and EndIndex must be assigned in the dimension depending on the total number of elements.
- Individual elements of a dimension can be selected by specifying the same StartIndex and EndIndex.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.17 ST_UALocalizedText

Syntax

```
TYPE ST_UALocalizedText:
STRUCT
    sLocale : STRING(6);
    sText   : STRING(MAX_STRING_LENGTH);
END_STRUCT
END_TYPE
```

Values

Name	Description
sLocale	Language identifier of the LocalizedText
sText	Text

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.18 ST_UAMethodArgInfo

Syntax

```
TYPE ST_UAMethodArgInfo:
STRUCT
    DataType        : E_UADataType := -1;
    ValueRank       : DINT := 2147483647;
    ArrayDimensions : ARRAY[1..3] OF UDINT := [0,0,0];
    nLenData        : DINT;
END_STRUCT
END_TYPE
```

Values

Name	Description
DataType	Defines the UA data type for the method parameter. (Type: E_UADataType [▶ 220])
ValueRank	Determines whether the parameter is scalar (-1) or array.
ArrayDimensions	If the parameter is an array, it specifies the dimensions of the array. Each element determines the length per dimension.
nLenData	Specifies the length of the argument. For output information STRUCT only requests this element.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.19 ST_UANodeID

Syntax

```
TYPE ST_UANodeID:
STRUCT
    nNamespaceIndex  : UINT;
    nReserved        : ARRAY [1..2] OF BYTE; //fill bytes
    sIdentifier      : STRING(MAX_STRING_LENGTH);
    eIdentifierType  : E_UAIdentifierType;
END_STRUCT
END_TYPE
```

Values

Name	Description
nNamespaceIndex	Namespace index under which the node is available. Can be determined with the function block UA_GetNamespaceIndex [▸ 237].
nReserved	Placeholder
sIdentifier	Identifier as shown in the UA namespace (attribute 'Identifier').
eIdentifierType	Variable type, described by E_UAIdentifierType [▸ 222].

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.20 ST_UANodeAdditionalInfo

Syntax

```
TYPE ST_UANodeAdditionalInfo:
STRUCT
    eAttributeID      : E_UAAttributeID;
    nIndexRangeCount : UINT;
    nReserved         : ARRAY[1..2] OF BYTE; // fill bytes
    stIndexRange      : ARRAY[1..nMaxIndexRange] OF ST_UAIndexRange;
END_STRUCT
END_TYPE
```

Values

Name	Description
eAttributeID	Specifies the ID of the OPC UA attribute. eUAAI_Value is used by default. (Type: E_UAAttributeID [▸ 218]).
nIndexRangeCount	Determines how many index ranges are used in stIndexRange.
nReserved	Placeholder
stIndexRange	Specifies an index range for reading values from an array. (Type: ST_UAIndexRange [▸ 227]).

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.21 ST_UAReferenceDescription

Syntax

```
TYPE ST_UAReferenceDescription:
STRUCT
    stReferenceTypeId : ST_UANodeId;
    bIsForward        : BOOL;
    stNodeId          : ST_UAExpandedNodeId;
    stBrowseName      : STRING(MAX_STRING_LENGTH);
    stDisplayName     : ST_UALocalizedText;
    eNodeClass        : E_UANodeClassMask;
    stTypeDefinition  : ST_UAExpandedNodeId;
END_STRUCT
END_TYPE
```

Values

Name	Description
stReferenceTypeId	NodeId of the reference type (e.g. Organizes, HasChild, HasTypeDefinition, ...) as data type ST_UANodeId [▸ 228].
bIsForward	Indicates whether the reference is a forward or backward reference.
stNodeId	NodeId as data type ST_UAExpandedNodeId [▸ 226].
stBrowseName	BrowseName of the reference.
stDisplayName	DisplayName of the reference (ST_UALocalizedText [▸ 228]).
eNodeClass	NodeClass of the reference (E_UANodeClassMask [▸ 222]).
stTypeDefinition	Type definition (HasTypeDefinition) (ST_UAExpandedNodeId [▸ 226]).

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.22 ST_UAUserIdentityTokenType

Syntax

```
TYPE ST_UAUserIdentityTokenType:
STRUCT
    eUserIdentTokenType   : E_UAUserIdentityTokenType;
    sTokenParam1          : STRING(MAX_STRING_LENGTH);
    sTokenParam2          : STRING(MAX_STRING_LENGTH);
END_STRUCT
END_TYPE
```

Values

Name	Description
eUserIdentTokenType	Type of user, described using E_UAUserIdentityTokenType [▸ 225]..
sTokenParam1	User name for logging on to the OPC UA Server.
sTokenParam2	Password for logging on to the OPC UA Server.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.1.23 UAHADataValue

This function block acts as a data object. An instance represents a value for the OPC UA Historical Access function. A whole field of these values is transferred to the UA_HistoryUpdate [▸ 238] function block on calling.

Syntax

```
aDataValues : ARRAY [1..50] OF UAHADataValue(ValueSize:=SIZEOF(LREAL));
```

Each data object is initialized with the expected size (in bytes) of the value.

Properties

Name	Type	Access	Initial value	Description
Value	PVOID	Set	-	Specifies the address of a variable containing the desired value. This is usually assigned with the help of the operator ADR(). The value itself is hereby assigned at the same time and copied into the data object.
StatusCode	UAHAUpdateStatusCode [▶ 231]	Get, Set	UAHAUpdateStatusCode.HistorianRaw	Indicates the status code of the value.
SourceTimeStamp	ULINT	Get, Set	0	Indicates the timestamp of the source in UTC format. This can be determined with the help of the function F_GetSystemTime (Tc2_System PLC library).
ServerTimeStamp	ULINT	Get, Set	0	Indicates the timestamp of the OPC UA Server in UTC format. This function is not currently supported.

Data type size of the value

The size of the data type used is already indicated and thus defined in the declaration of the data object. This size is taken as the basis when assigning a value later.

Values of the type STRING are accordingly also saved and transmitted with a fixed initialized size. An indication of the current text length cannot be made.

Sample

```
{attribute 'OPC.UA.DA' := '1'}
fMyValue    : LREAL;        // Variable for HistorcalAccess
aDataValues : ARRAY [1..50] OF UAHADataValue(ValueSize:=SIZEOF(LREAL));
```

```
fMyValue := 27.75;
aDataValues[1].Value           := ADR(fMyValue);
aDataValues[1].StatusCode      := UAHAUpdateStatusCode.HistorianRaw;
aDataValues[1].SourceTimeStamp := F_GetSystemTime();
```

In this sample a field of 50 values is defined, of which each is represented by a data object. The current content of the variable fMyValue (= 27.75) is assigned to the first value.

The field can now be filled by means of further assignments in subsequent PLC cycles.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1 >= 4024.1	Win32, Win64, WinCE-x86	Tc3_PLCopen_OpcUa >= v3.1.9.0

5.2.1.24 UAHAUpdateStatusCode

A status code is assigned to each data value transferred using the OPC UA Historical Access function. This is a property of the object UAHADataValue [▶ 230].

Syntax

```
{attribute 'qualified_only'}
TYPE UAHAUpdateStatusCode :
(
    HistorianRaw           := 0,          // A raw data value.
    HistorianCalculated    := 1,   // A data value which was calculated.
    HistorianInterpolated := 2, // A data value which was interpolated.
    Reserved               := 3,              // Undefined.
    HistorianPartial       := 4,      // A data value which was calculated with an incomplete interva
l.
    HistorianExtraData     := 8,    // A raw data value that hides other data at the same timestamp.
    HistorianMultiValue    := 16   // Multiple values match the Aggregate criteria (i.e. multiple min
imum values at different timestamps within the same interval).
) UDINT;
END_TYPE
```

Values

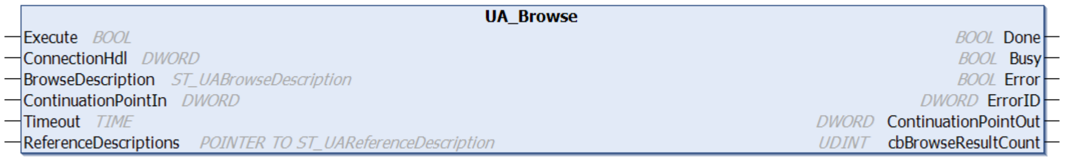
Name	Description
HistorianRaw	HistorianRaw
HistorianCalculated	HistorianCalculated
HistorianInterpolated	HistorianInterpolated
Reserved	Reserved
HistorianPartial	HistorianPartial
HistorianExtraData	HistorianExtraData
HistorianMultiValue	HistorianMultiValue

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1 >= 4024.1	Win32, Win64, WinCE-x86	Tc3_PLCopen_OpcUa >= v3.1.9.0

5.2.2 Function blocks

5.2.2.1 UA_Browse


UA_Browse
Execute BOOL
ConnectionHdl DWORD
BrowseDescription ST_UABrowseDescription
ContinuationPointIn DWORD
Timeout TIME
ReferenceDescriptions POINTER TO ST_UAReferenceDescription
BOOL Done
BOOL Busy
BOOL Error
DWORD ErrorID
DWORD ContinuationPointOut
UDINT cbBrowseResultCount

This function block allows browsing through the namespace of a server. Starting from a start node, its references are read and returned accordingly.

Inputs

```
VAR_INPUT
    Execute             : BOOL;
    ConnectionHdl       : DWORD;
    BrowseDescription   : ST_UABrowseDescription;
    ContinuationPointIn : DWORD;
    Timeout             : TIME;
END_VAR
```

Name	Type	Description
Execute	BOOL	The command is triggered by a rising edge at this input.
ConnectionHdl	DWORD	Connection handle previously output by the function block UA_Connect.
BrowseDescription	ST_UABrowseDescription [▶ 225]	The address information for the node to be read is specified here.
ContinuationPointIn	DWORD	If a previous call of the function block returned a value as ContinuationPointOut, this value can be created here to get further data from the server.
Timeout	TIME	Time until the function is aborted.

Inputs/outputs

```
VAR_IN_OUT
    ReferenceDescriptions : POINTER TO ST_UAReferenceDescriptions;
END_VAR
```

Name	Type	Description
ReferenceDescriptions	POINTER TO ST_UAReferenceDescriptions	Contains the list of ReferenceDescriptions returned by the server, i.e. the result of the UA_Browse call. The contained ReferenceDescriptions can then be used for further UA_Browse calls in the BrowseDescription, e.g. to navigate deeper into the namespace.

Outputs

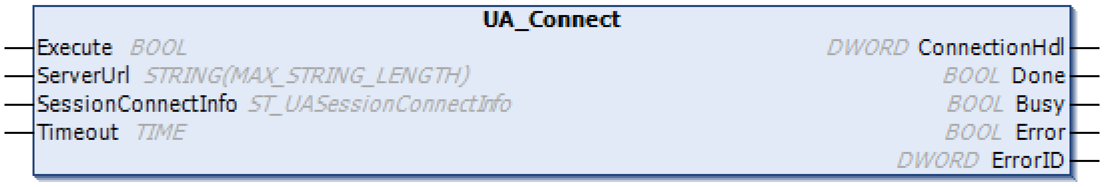
```
VAR_OUTPUT
    Done                    : BOOL;
    Busy                    : BOOL;
    Error                   : BOOL;
    ErrorID                 : DWORD;
    ContinuationPointOut : DWORD;
    cbBrowseResultCnt     : UDINT;
END_VAR
```

Name	Type	Description
Done	BOOL	Switches to TRUE if the function block was executed successfully.
Busy	BOOL	TRUE until the function block has executed a command, at the most for the duration of the "Timeout" at the input. The inputs accept no new command as long as Busy = TRUE. It is not the connection time that is monitored but the reception time.
Error	BOOL	Switches to TRUE if an error occurs while executing a command. The command-specific error code is included in ErrorID.
ErrorID	DWORD	Contains the command-specific error code of the most recently executed command.
ContinuationPointOut	DWORD	If the server returns data batch-wise (ContinuationPointOut != 0), the value of ContinuationPointOut can be used as ContinuationPointIn at the next call of the function block to get the further data.
cbBrowseResultCnt	UDINT	Number of ReferenceDescriptions.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.2.2 UA_Connect



This function block establishes an OPC UA Remote connection to another OPC UA Server, which is specified via ServerUrl and SessionConnectInfo. The function block returns a connection handle that can be used for other function blocks, such as UA_Read.

Inputs

```
VAR_INPUT
    Execute            : BOOL;
    ServerUrl          : STRING(MAX_STRING_LENGTH);
    SessionConnectInfo : ST_UASessionConnectInfo;
    Timeout            : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
Execute	BOOL	The command is triggered by a rising edge at this input.
ServerUrl	STRING(MAX_STRING_LENGTH)	OPC UA Server URL, i.e. 'opc.tcp://172.16.3.207:4840' or 'opc.tcp://CX_0193BF:4840'.
SessionConnectInfo	ST_UASessionConnectInfo	Connection information (see ST_UASessionConnectInfo [▶ 226])
Timeout	TIME	Time until the function is aborted. DEFAULT_ADS_TIMEOUT is a global constant, set to 5 seconds. The value must be set to match the ST_UASessionConnectInfo.tConnectionTimeout. The rule of thumb is: ADS Timeout > 2 * ConnectionTimeout.

Outputs

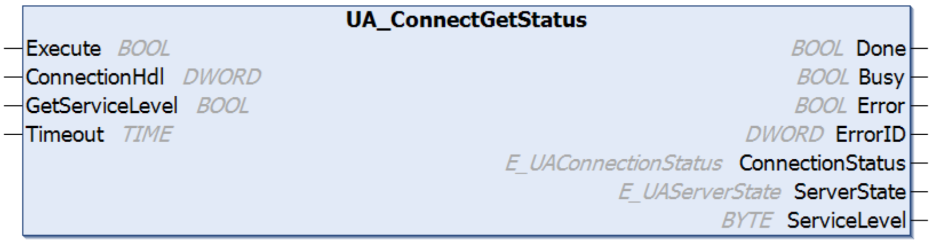
```
VAR_OUTPUT
    ConnectionHdl : DWORD;
    Done          : BOOL;
    Busy          : BOOL;
    Error         : BOOL;
    ErrorID       : DWORD;
END_VAR
```

Name	Type	Description
ConnectionHdl	DWORD	OPC UA connection handle.
Done	BOOL	Switches to TRUE if the function block was executed successfully.
Busy	BOOL	TRUE until the function block has executed a command, at the most for the duration of the "Timeout" at the input. The inputs accept no new command as long as Busy = TRUE. It is not the connection time that is monitored but the reception time.
Error	BOOL	Switches to TRUE if an error occurs while executing a command. The command-specific error code is included in ErrorID.
ErrorID	DWORD	Contains the command-specific error code of the most recently executed command.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.2.3 UA_ConnectGetStatus



This function block checks the connection status of an existing connection to another OPC UA Server. The connection is referenced via the respective connection handle. The status is then returned as E_UAConnectionStatus [▶ 220]. The connection status is determined based on the internal session info or the OPC UA heartbeat, no additional communication (read or similar) is performed.

The service level of the OPC UA Server can be read out via the additional input parameter GetServiceLevel. For this purpose, a read command is sent to the server in the background to determine this information.

Inputs

```
VAR_INPUT
    Execute            : BOOL;
    ConnectionHdl      : DWORD;
    GetServiceLevel    : BOOL;
    Timeout            : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
Execute	BOOL	The command is triggered by a rising edge at this input.
ConnectionHdl	DWORD	Connection handle of an existing communication link.
GetServiceLevel	BOOL	Reads out the ServiceLevel of the OPC UA Server.
Timeout	TIME	Time until the function is aborted. DEFAULT_ADS_TIMEOUT is a global constant, set to 5 seconds. The value must be set to match the ST_UASessionConnectInfo.tConnectionTimeout. The rule of thumb is: ADS Timeout > 2 * ConnectionTimeout.

Outputs

```
VAR_OUTPUT
    Done              : BOOL;
    Busy              : BOOL;
    Error             : BOOL;
    ErrorID           : DWORD;
    ConnectionStatus  : E_UAConnectionStatus;
    ServerState       : E_UAServerState;
    ServiceLevel      : BYTE;
END_VAR
```

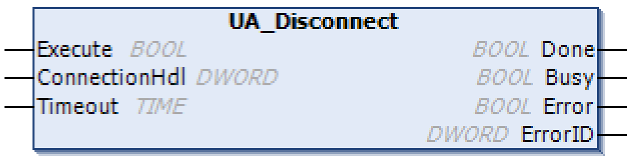
Name	Type	Description
Done	BOOL	Switches to TRUE if the function block was executed successfully.
Busy	BOOL	TRUE until the function block has executed a command, at the most for the duration of the "Timeout" at the input. The inputs accept no new command as long as Busy = TRUE. It is not the connection time that is monitored but the reception time.
Error	BOOL	Switches to TRUE if an error occurs while executing a command. The command-specific error code is included in ErrorID.
ErrorID	DWORD	Contains the command-specific error code of the most recently executed command.
ConnectionStatus	E_UAConnectionStatus	Connection status (see E_UAConnectionStatus [▸ 220]).
ServerState	E_UAServerState	Server state (see E_UAServerState [▸ 224]).
ServerState	BYTE	ServiceLevel of the OPC UA Server.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

PLC library	Required version
Tc3_PLCopen_OpcUa	>= 3.2.11.0

5.2.2.4 UA_Disconnect



This function block closes an OPC UA Remote connection to another OPC UA Server. The connection is specified via its connection handle.

Disconnect all connections

If the UA-Disconnect method is called and a connection handle of 0 is passed, the OPC UA client disconnects all existing connections. This also applies to connections established via an OPC UA I/O client configuration.

Inputs

```
VAR_INPUT
    Execute             : BOOL;
    ConnectionHdl       : DWORD;
    Timeout             : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
Execute	BOOL	The command is triggered by a rising edge at this input.
ConnectionHdl	DWORD	Connection handle previously output by the function block UA_Connect.
Timeout	TIME	Time until the function is aborted. DEFAULT_ADS_TIMEOUT is a global constant, set to 5 seconds.

Outputs

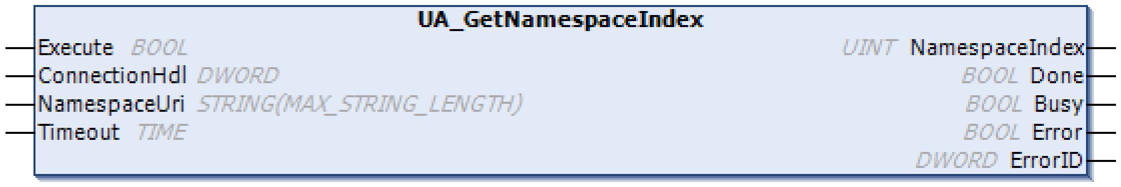
```
VAR_OUTPUT
    Done        : BOOL;
    Busy        : BOOL;
    Error       : BOOL;
    ErrorID     : DWORD;
END_VAR
```

Name	Type	Description
Done	BOOL	Switches to TRUE if the function block was executed successfully.
Busy	BOOL	TRUE until the function block has executed a command, at the most for the duration of the "Timeout" at the input. The inputs accept no new command as long as Busy = TRUE. It is not the connection time that is monitored but the reception time.
Error	BOOL	Switches to TRUE if an error occurs while executing a command. The command-specific error code is contained in nErrID.
ErrorID	DWORD	Contains the command-specific error code of the most recently executed command.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.2.5 UA_GetNamespaceIndex



This function block collects the namespace index for a namespace URI. The namespace index is required for identifying symbols, for example, if the function blocks UA_Read [▸ 249]or UA_Write [▸ 252]are used.

Inputs

```
VAR_INPUT
    Execute       : BOOL;
    ConnectionHdl : DWORD;
    NamespaceUri  : STRING(MAX_STRING_LENGTH);
    Timeout       : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
Execute	BOOL	The command is triggered by a rising edge at this input.
ConnectionHdl	DWORD	Connection handle previously output by the function block UA_Connect.
NamespaceUri	STRING	Namespace URI to be resolved. For the TwinCAT OPC UA Server, this is "urn:BeckhoffAutomation:Ua:PLC1“" for the first PLC runtime.
Timeout	TIME	Time until the function is aborted. DEFAULT_ADS_TIMEOUT is a global constant, set to 5 seconds.

Outputs

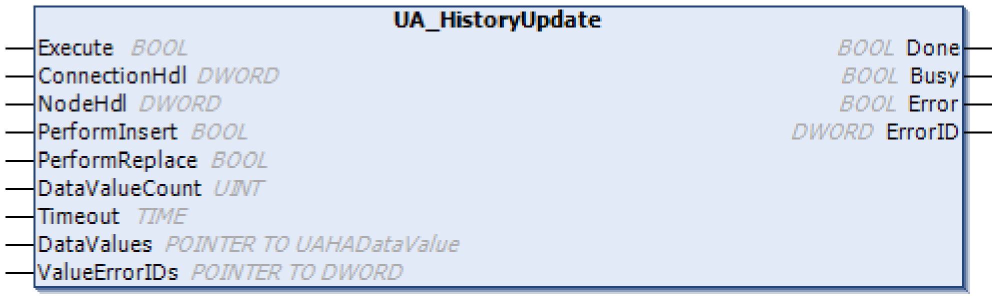
```
VAR_OUTPUT
    NamespaceIndex : UINT;
    Done           : BOOL;
    Busy           : BOOL;
    Error          : BOOL;
    ErrorID        : DWORD;
END_VAR
```

Name	Type	Description
NamespaceIndex	UINT	Namespace Index of the given namespace URI. This can be used in other function blocks, e.g. UA_NodeGetHandle or UA_MethodGetHandle.
Done	BOOL	Switches to TRUE if the function block was executed successfully.
Busy	BOOL	TRUE until the function block has executed a command, at the most for the duration of the "Timeout" at the input. The inputs accept no new command as long as Busy = TRUE. It is not the connection time that is monitored but the reception time.
Error	BOOL	Switches to TRUE if an error occurs while executing a command. The command-specific error code is included in ErrorID.
ErrorID	DWORD	Contains the command-specific error code of the most recently executed command.

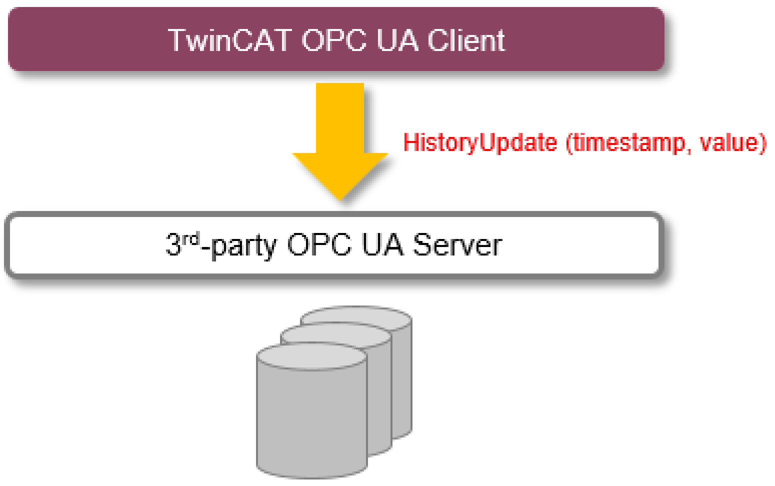
Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.2.6 UA_HistoryUpdate



This function block sends historical data via OPC UA to a server that supports the OPC UA HistoryUpdate function, e.g. the TwinCAT OPC UA Server. With one call you can transfer a large number of values including time stamps to the server for a node handle. The server ensures that the values transmitted are saved in a data memory and are available via Historical Access.



The function block can be instanced several times if values of several node handles (different variables) are to be transmitted.

Operation with TwinCAT OPC UA Server

The function block is well suited if you use Historical Access in the TwinCAT OPC UA Server and want to make data available from a certain time interval in which, for example, a special machine state prevailed. Values for the desired period can be purposefully transmitted.

If on the other hand values are sent cyclically and are to be made available in the server via Historical Access, then the Historical Access function on the server side is better suited, as in this case you only have to configure the recording node in the configurator and set the desired sampling rate.

See also: Program sample TF6100_OPCUA_HASample [▶ 254]

Inputs

```
VAR_INPUT
    Execute         : BOOL;
    ConnectionHdl   : DWORD;
    NodeHdl         : DWORD;
    PerformInsert   : BOOL;
    PerformReplace  : BOOL;
    DataValueCount  : UINT;
    Timeout         : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
Execute	BOOL	The command is triggered by a rising edge at this input.
ConnectionHdl	DWORD	Connection handle previously output by the function block UA_Connect.
NodeHdl	DWORD	Node handle that was previously output by the function block UA_NodeGetHandle.
PerformInsert	BOOL	The default is TRUE.
PerformReplace	BOOL	The default is FALSE. If a value for the given timestamp already exists in the history, it should be replaced if the PerformReplace option is set (= TRUE). Currently this option can only be selected for SQL adapters. Other adapters do not support the option.
DataValueCount	UINT	Defines the number of values transferred. A maximum number of 1000 values is supported.
Timeout	TIME	Time until the function is aborted. DEFAULT_ADS_TIMEOUT is a global constant, set to 5 seconds.

Inputs/outputs

```
VAR_IN_OUT
    DataValues        : ARRAY[*] OF UAHADataValue;
    ValueErrorIDs     : ARRAY[*] OF DWORD;
END_VAR
```

Name	Type	Description
DataValues (read-only)	ARRAY	All collected values are transferred in the form of a field of the type UAHADataValue. The length of the field is not prescribed, but it must correspond at least to the specification of DataValueCount. Internally the values are accessed only for reading.
ValueErrorIDs (write-only)	ARRAY	After execution of the command this field contains an error code for each value. The length of the field must correspond at least to the specification of DataValueCount. If one or more values report an error, it is also signaled via the outputs Error and ErrorID of the function block. With the help of this field you can then determine which error has occurred for which value. The error code 16#80000000, for example, signalizes a failed operation, meaning that the value could not be written.

Outputs

```
VAR_OUTPUT
    Done      : BOOL;
    Busy      : BOOL;
    Error     : BOOL;
    ErrorID   : DWORD;
END_VAR
```

Name	Type	Description
Done	BOOL	Switches to TRUE if the function block was executed successfully.
Busy	BOOL	TRUE until the function block has executed a command, at the most for the duration of the "Timeout" at the input. The inputs accept no new command as long as Busy = TRUE. It is not the connection time that is monitored but the reception time.
Error	BOOL	Switches to TRUE if an error occurs while executing a command. The command-specific error code is included in ErrorID.
ErrorID	DWORD	Contains the command-specific ADS error code of the most recently executed command.

Number of values transferred

The larger the number, the greater the required computing effort and thus the longer the PLC execution time when executing the command.

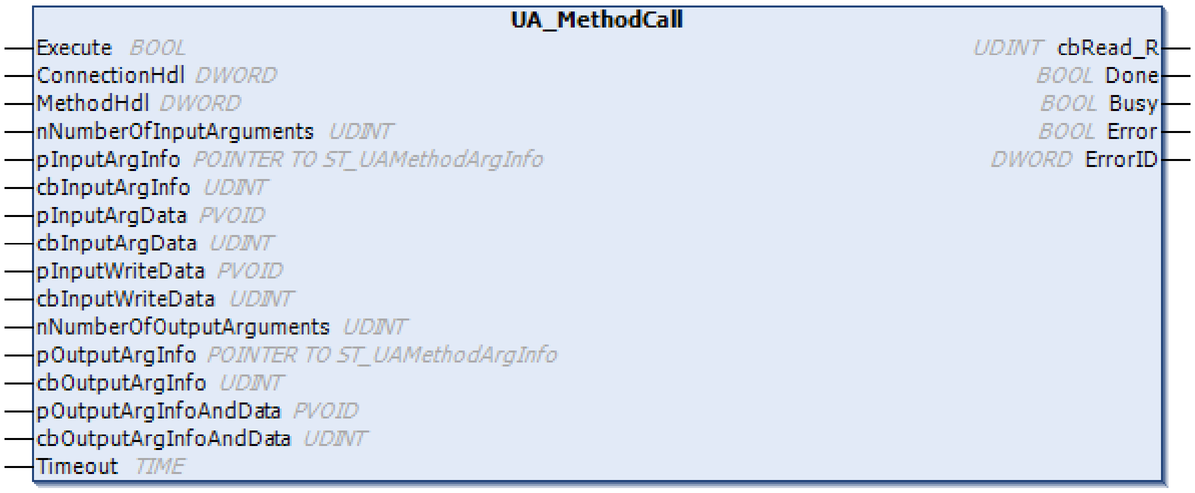
Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1 >= 4024.1	Win32, Win64, WinCE-x86	Tc3_PLCopen_OpcUa >= v3.1.9.0

Also see about this

- UA_Connect [▸ 234]
- UA_NodeGetHandle [▸ 245]
- UAHADataValue [▸ 230]

5.2.2.7 UA_MethodCall



This function block calls a method on a remote UA Server. The method is determined by a connection and a method handle. The former can be queried by UA_Connect [▸ 234], the latter by UA_MethodGetHandle [▸ 243].

Inputs

```
VAR_INPUT
    Execute                     : BOOL;
    ConnectionHdl               : DWORD;
    MethodHdl                   : DWORD;
    nNumberOfInputArguments     : UDINT;
    pInputArgInfo               : POINTER TO ST_UAMethodArgInfo;
    cbInputArgInfo              : UDINT;
    pInputArgData               : PVOID;
    cbInputArgData              : UDINT;
    pInputWriteData             : PVOID;
    cbInputWriteData            : UDINT;
    nNumberOfOutputArguments    : UDINT;
    pOutputArgInfo              : POINTER TO ST_UAMethodArgInfo;
    cbOutputArgInfo             : UDINT;
    pOutputArgInfoAndData       : PVOID;
    cbOutputArgInfoAndData      : UDINT;
    Timeout                     : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
Execute	BOOL	The command is triggered by a rising edge at this input.
ConnectionHdl	DWORD	Connection handle previously output by the function block UA_Connect.
MethodHdl	DWORD	Method handle, previously output by the function block UA_MethodGetHandle.
nNumberOfInputArguments	UDINT	Number of input parameters.
pInputArgInfo	POINTER TO ST_UAMethodArgInfo	Points to the buffer address where input parameter information is stored in the form of an array ST_UAMethodArgInfo.
cbInputArgInfo	UDINT	Size of the buffer where the input parameter information is stored.
pInputArgData	PVOID	Points to the buffer address where input parameters (constant length) are stored.
cbInputArgData	UDINT	Size of the input buffer where input parameters (with constant length) are stored.
pInputWriteData	PVOID	Pointer to buffer address where input parameters (dynamic length) are stored.
cbInputWriteData	UDINT	Size of the input buffer where input parameters (with dynamic length) are stored.
nNumberOfOutputArguments	UDINT	Number of output parameters.
pOutputArgInfo	POINTER TO ST_UAMethodArgInfo	Points to the buffer address where output parameter information is stored as array ST_UAMethodArgInfo. nLenData is required to determine the target memory of the individual output parameters. The other elements can be set in such a way that a type check of the returned parameters takes place or remains undefined.
cbOutputArgInfo	UDINT	Size of the buffer where the output parameter information is stored.
pOutputArgInfoAndData	PVOID	Points to the buffer address where the output parameters are to be saved as a BYTE array. The BYTE array contains the number of output parameters as DINT, four reserved bytes and parameter information as ARRAY OF ST_UAMethodArgInfo [▶ 228] (with the length of the output parameters), followed by pure data. Note that the data is packed as 1-byte alignment.
cbOutputArgInfoAndData	UDINT	Size of the buffer in which the output parameters are to be saved as a BYTE array.
Timeout	TIME	Time until the function is aborted. DEFAULT_ADS_TIMEOUT is a global constant, set to 5 seconds.

Outputs

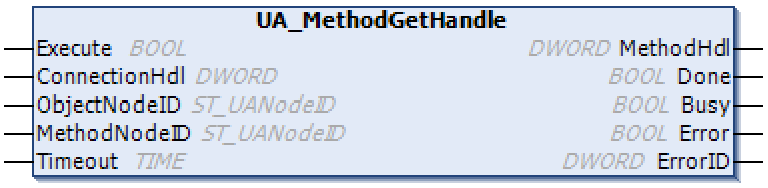
```
VAR_OUTPUT
    cbRead_R      : UDINT;
    Done          : BOOL;
    Busy          : BOOL;
    Error         : BOOL;
    ErrorID       : UDINT;
END_VAR
```

Name	Type	Description
cbRead_R	UDINT	Counts all the bytes received.
Done	BOOL	Switches to TRUE if the function block was executed successfully.
Busy	BOOL	TRUE until the function block has executed a command, at the most for the duration of the "Timeout" at the input. The inputs accept no new command as long as Busy = TRUE. It is not the connection time that is monitored but the reception time.
Error	BOOL	Switches to TRUE if an error occurs while executing a command. The command-specific error code is contained in nErrID.
ErrorID	UDINT	Contains the command-specific error code of the most recently executed command.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.2.8 UA_MethodGetHandle



This function block collects a handle for a UA method, which can then be used to call a method using UA_MethodCall [▸ 240].

Inputs

```
VAR_INPUT
    Execute           : BOOL;
    ConnectionHdl     : DWORD;
    ObjectNodeID      : ST_UANodeID;
    MethodNodeID      : ST_UANodeID;
    Timeout           : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
Execute	BOOL	The command is triggered by a rising edge at this input.
ConnectionHdl	DWORD	Connection handle previously output by the function block UA_Connect.
ObjectNodeID	ST_UANodeID	Object node ID of the method to be called. (Type: ST_UANodeID [▸ 228]).
MethodNodeID	ST_UANodeID	Method node ID of the method to be called. Corresponds to the ID attribute in the UA namespace. (Type: UA_Connect [▸ 234]).
Timeout	TIME	Time until the function is aborted. DEFAULT_ADS_TIMEOUT is a global constant, set to 5 seconds.

Outputs

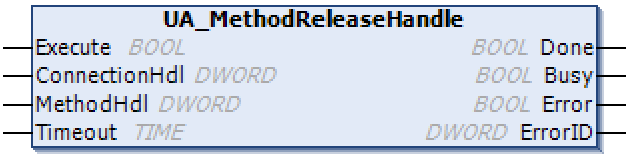
```
VAR_OUTPUT
    MethodHdl    : DWORD;
    Done         : BOOL;
    Busy         : BOOL;
    Error        : BOOL;
    ErrorID      : UDINT;
END_VAR
```

Name	Type	Description
MethodHdl	DWORD	Returns a method handle that can be used to call a method via UA_MethodCall [▸ 240].
Done	BOOL	Switches to TRUE if the function block was executed successfully.
Busy	BOOL	TRUE until the function block has executed a command, at the most for the duration of the "Timeout" at the input. The inputs accept no new command as long as Busy = TRUE. It is not the connection time that is monitored but the reception time.
Error	BOOL	Switches to TRUE if an error occurs while executing a command. The command-specific error code is contained in nErrID.
ErrorID	UDINT	Contains the command-specific error code of the most recently executed command.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.2.9 UA_MethodReleaseHandle



This function block releases the specified method handle.

Inputs

```
VAR_INPUT
    Execute            : BOOL;
    ConnectionHdl      : DWORD;
    MethodHdl          : DWORD;
    Timeout            : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
Execute	BOOL	The command is triggered by a rising edge at this input.
ConnectionHdl	DWORD	Connection handle previously output by the function block UA_Connect.
MethodHdl	DWORD	Method handle previously output by the function block UA_MethodGetHandle.
Timeout	TIME	Time until the function is aborted. DEFAULT_ADS_TIMEOUT is a global constant, set to 5 seconds.

Outputs

```
VAR_OUTPUT
    Done      : BOOL;
    Busy      : BOOL;
    Error     : BOOL;
    ErrorID   : UDINT;
END_VAR
```

Name	Type	Description
Done	BOOL	Switches to TRUE if the function block was executed successfully.
Busy	BOOL	TRUE until the function block has executed a command, at the most for the duration of the "Timeout" at the input. The inputs accept no new command as long as Busy = TRUE. It is not the connection time that is monitored but the reception time.
Error	BOOL	Switches to TRUE if an error occurs while executing a command. The command-specific error code is contained in nErrID.
ErrorID	UDINT	Contains the command-specific ADS error code of the most recently executed command.

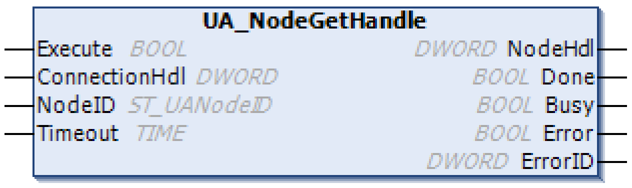
Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

Also see about this

- UA_Connect [▸ 234]
- UA_MethodGetHandle [▸ 243]

5.2.2.10 UA_NodeGetHandle



This function block queries a node handle for a given symbol in the UA namespace. The symbol is specified by a connection handle and its node ID.

Inputs

```
VAR_INPUT
    Execute       : BOOL;
    ConnectionHdl : DWORD;
    NodeID        : ST_UANodeID;
    Timeout       : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
Execute	BOOL	The command is triggered by a rising edge at this input.
ConnectionHdl	DWORD	Connection handle previously output by the function block UA_Connect.
Node ID	ST_UANodeID	Unique addressing of the UA node, consisting of Identifier, IdentifierType and NamespaceIndex, which are resolved from a NamespaceName, e.g. by means of the method UA_GetNamespaceIndex [▸ 237].
Timeout	TIME	Time until the function is aborted. DEFAULT_ADS_TIMEOUT is a global constant, set to 5 seconds.

Outputs

```
VAR_OUTPUT
    NodeHdl     : DWORD;
    Done        : BOOL;
    Busy         : BOOL;
    Error        : BOOL;
    ErrorID      : DWORD;
END_VAR
```

Name	Type	Description
NodeHdl	DWORD	Node handle that can be used for other function blocks, such as UA_Read or UA_Write.
Done	BOOL	Switches to TRUE if the function block was executed successfully.
Busy	BOOL	TRUE until the function block has executed a command, at the most for the duration of the "Timeout" at the input. The inputs do not accept new commands as long as Busy is TRUE. It is not the connection time that is monitored but the reception time.
Error	BOOL	Switches to TRUE if an error occurs while executing a command. The command-specific error code is included in ErrorID.
ErrorID	DWORD	Contains the command-specific ADS error code of the most recently executed command.

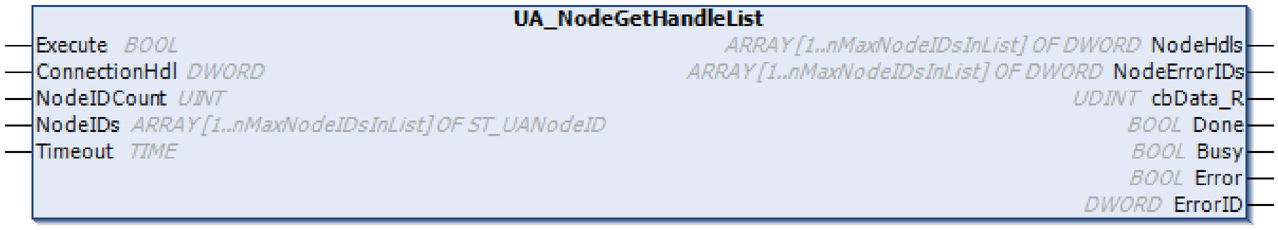
Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

Also see about this

UA_Connect [▸ 234]

5.2.2.11 UA_NodeGetHandleList


UA_NodeGetHandleList
Execute BOOL
ConnectionHdl DWORD
NodeIDCount UINT
NodeIDs ARRAY [1..nMaxNodeIDsInList] OF ST_UANodeID
Timeout TIME
ARRAY [1..nMaxNodeIDsInList] OF DWORD NodeHdls
ARRAY [1..nMaxNodeIDsInList] OF DWORD NodeErrorIDs
UDINT cbData_R
BOOL Done
BOOL Busy
BOOL Error
DWORD ErrorID

This function block queries node handles for nodes in the UA namespace.

Inputs

```
VAR_INPUT
    Execute           : BOOL;
    ConnectionHdl     : DWORD;
    NodeIDCount       : UINT;
    NodeIDs           : ARRAY[1..nMaxNodeIDsInList] OF ST_UANodeID;
    Timeout           : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
Execute	BOOL	The command is triggered by a rising edge at this input.
ConnectionHdl	DWORD	Connection handle previously output by the function block UA_Connect.
NodeIDCount	UINT	Number of nodes for which a node handle is required.
NodeIDs	ARRAY	Array of NodeIDs created with struct ST_UANodeID [▸ 228].
Timeout	TIME	Time until the function is aborted. DEFAULT_ADS_TIMEOUT is a global constant, set to 5 seconds.

Outputs

```
VAR_OUTPUT
    NodeHdls     : ARRAY[1..nMaxNodeIDsInList] OF DWORD;
    NodeErrorIDs : ARRAY[1..nMaxNodeIDsInList] OF DWORD;
    cbData_R     : UDINT;
    Done         : BOOL;
    Busy         : BOOL;
    Error        : BOOL;
    ErrorID      : DWORD;
END_VAR
```

Name	Type	Description
NodeHdls	ARRAY	Array of requested node handles.
NodeErrorIDs	ARRAY	Array of error IDs if no node handles are available.
cbData_R	UDINT	Size of the data read.
Done	BOOL	Switches to TRUE if the function block was executed successfully.
Busy	BOOL	TRUE until the function block has executed a command, at the most for the duration of the "Timeout" at the input. The inputs accept no new command as long as Busy = TRUE. It is not the connection time that is monitored but the reception time.
Error	BOOL	Switches to TRUE if an error occurs while executing a command. The command-specific error code is in nErrID.
ErrorID	DWORD	Contains the error ID if an error occurs.

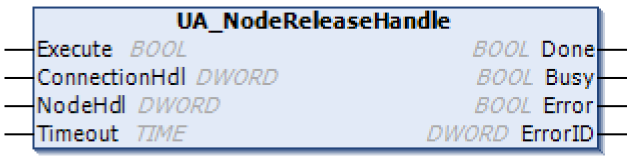
Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

Also see about this

- UA_Connect [▸ 234]

5.2.2.12 UA_NodeReleaseHandle



This function block releases a node handle.

Inputs

```
VAR_INPUT
    Execute       : BOOL;
    ConnectionHdl : DWORD;
    NodeHdl       : DWORD;
    Timeout       : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
Execute	BOOL	The command is triggered by a rising edge at this input.
ConnectionHdl	DWORD	Connection handle previously output by the function block UA_Connect.
NodeHdl	DWORD	Node handle to be released.
Timeout	TIME	Time until the function is aborted. DEFAULT_ADS_TIMEOUT is a global constant, set to 5 seconds.

Outputs

```
VAR_OUTPUT
    Done        : BOOL;
    Busy        : BOOL;
    Error       : BOOL;
    ErrorID     : DWORD;
END_VAR
```

Name	Type	Description
Done	BOOL	Switches to TRUE if the function block was executed successfully.
Busy	BOOL	TRUE until the function block has executed a command, at the most for the duration of the "Timeout" at the input. The inputs accept no new command as long as Busy = TRUE. It is not the connection time that is monitored but the reception time.
Error	BOOL	Switches to TRUE if an error occurs while executing a command. The command-specific error code is included in ErrorID.
ErrorID	DWORD	Contains the command-specific ADS error code of the most recently executed command.

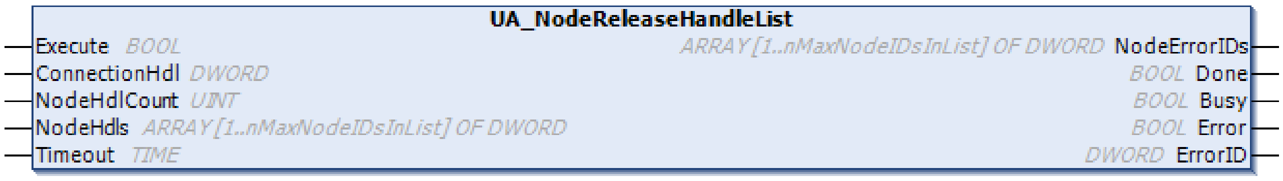
Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

Also see about this

UA_Connect [▸ 234]

5.2.2.13 UA_NodeReleaseHandleList



This function block releases several node handles.

Inputs

```
VAR_INPUT
    Execute             : BOOL;
    ConnectionHdl       : DWORD;
    NodeHdlCount        : UINT;
    NodeHdls            : ARRAY[1..nMaxNodeIDsInList] OF DWORD;
    Timeout             : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
Execute	BOOL	The command is triggered by a rising edge at this input.
ConnectionHdl	DWORD	Connection handle previously output by the function block UA_Connect.
NodeHdlCount	UINT	Number of node handles.
NodeHdls	ARRAY	Array of node handles to be released.
Timeout	TIME	Time until the function is aborted. DEFAULT_ADS_TIMEOUT is a global constant, set to 5 seconds.

Outputs

```
VAR_OUTPUT
    NodeErrorIDs : ARRAY[1..nMaxNodeIDsInList] OF DWORD;
    Done         : BOOL;
    Busy         : BOOL;
    Error        : BOOL;
    ErrorID      : DWORD;
END_VAR
```

Name	Type	Description
NodeErrorIDs	ARRAY	Array of error IDs if a node handle could not be released.
Done	BOOL	Switches to TRUE if the function block was executed successfully.
Busy	BOOL	TRUE until the function block has executed a command, at the most for the duration of the "Timeout" at the input. The inputs accept no new command as long as Busy = TRUE. It is not the connection time that is monitored but the reception time.
Error	BOOL	Switches to TRUE if an error occurs while executing a command. The command-specific error code is in nErrID.
ErrorID	DWORD	Contains the error ID if an error occurs.

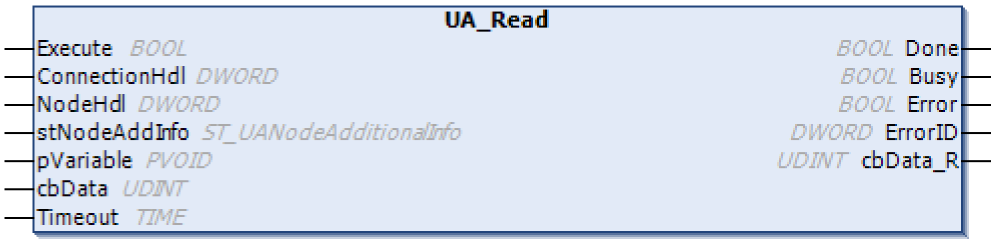
Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

Also see about this

UA_Connect [▸ 234]

5.2.2.14 UA_Read



This function block reads values from a given node and connection handle.

Inputs

```
VAR_INPUT
    Execute             : BOOL;
    ConnectionHdl       : DWORD;
    NodeHdl             : DWORD;
    stNodeAddInfo       : ST_UANodeAdditionalInfo;
    pVariable           : PVOID;
    cbData              : UDINT;
    Timeout             : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
Execute	BOOL	The command is triggered by a rising edge at this input.
Connection Hdl	DWORD	Connection handle previously output by the function block UA_Connect.
NodeHdl	DWORD	Node handle that was previously output by the function block UA_NodeGetHandle.
stNodeAddl nfo	ST_UANodeAdditionalInfo	Defines additional information, such as which attribute is read from the UA namespace (default: 'Value' attribute) or which IndexRange is to be used. Specified by STRUCT ST_UANodeAdditionalInfo [▶ 229].
pVariable	PVOID	Pointer to data memory where the read data is to be stored.
cbData	UDINT	Determines the size of the data to be read.
Timeout	TIME	Time until the function is aborted. DEFAULT_ADS_TIMEOUT is a global constant, set to 5 seconds.

Outputs

```
VAR_OUTPUT
    Done         : BOOL;
    Busy         : BOOL;
    Error        : BOOL;
    ErrorID      : UDINT;
    cbData_R     : UDINT;
END_VAR
```

Name	Type	Description
Done	BOOL	Switches to TRUE if the function block was executed successfully.
Busy	BOOL	TRUE until the function block has executed a command, at the most for the duration of the "Timeout" at the input. The inputs accept no new command as long as Busy = TRUE. It is not the connection time that is monitored but the reception time.
Error	BOOL	Switches to TRUE if an error occurs while executing a command. The command-specific error code is contained in nErrID.
ErrorID	UDINT	Contains the command-specific ADS error code of the most recently executed command.
cbData_R	UDINT	Number of bytes to be read.

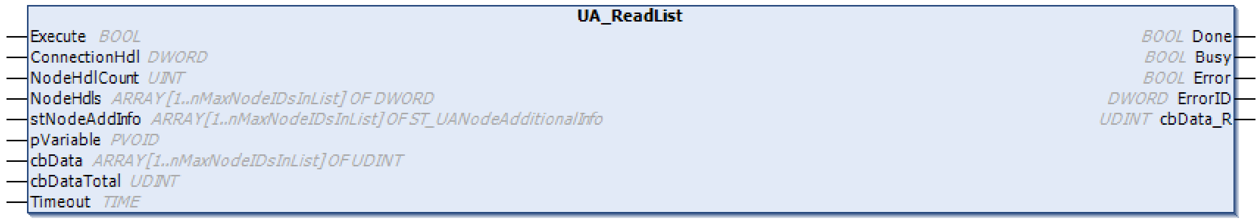
Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

Also see about this

- UA_Connect [▶ 234]
- UA_NodeGetHandle [▶ 245]

5.2.2.15 UA_ReadList



This function block reads values from several given node and connection handles.

Inputs

```
VAR_INPUT
    Execute           : BOOL;
    ConnectionHdl     : DWORD;
    NodeHdlCount      : UINT;
    NodeHdls          : ARRAY[1..nMaxNodeIDsInList] OF DWORD;
    stNodeAddInfo     : ARRAY[1..nMaxNodeIDsInList] OF ST_UANodeAdditionalInfo;
    pVariable         : PVOID;
    cbData            : ARRAY[1..nMaxNodeIDsInList] UDINT;
    cbDataTotal       : UDINT;
    Timeout           : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
Execute	BOOL	The command is triggered by a rising edge at this input.
ConnectionHdl	DWORD	Connection handle previously output by the function block UA_Connect [▶ 234].
NodeHdlCount	UINT	Number of node handles stored in the input variable NodeHdls.
NodeHdls	ARRAY	Array of node handles previously received by the function block UA_NodeGetHandle [▶ 245] or UA_NodeGetHandleList [▶ 246].
stNodeAddInfo	ARRAY	Defines additional information, such as which attribute is read from the UA namespace (default: 'Value' attribute) or which IndexRange is to be used. Specified by STRUCT ST_UANodeAdditionalInfo [▶ 229].
pVariable	PVOID	Pointer to data memory where the read data is to be stored.
cbData	ARRAY	Determines the size of the data to be read.
cbDataTotal	UDINT	Total size of the data to be received.
Timeout	TIME	Time until the function is aborted. DEFAULT_ADS_TIMEOUT is a global constant, set to 5 seconds.

Outputs

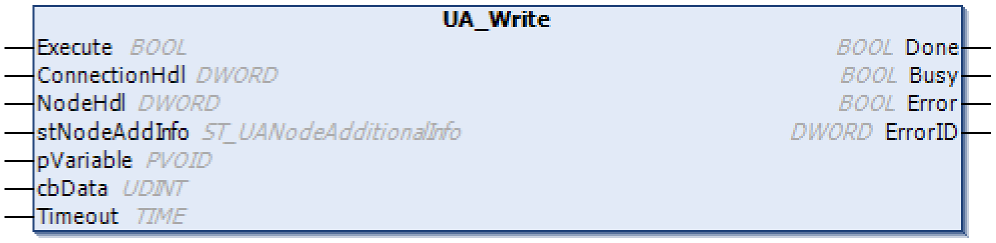
```
VAR_OUTPUT
    Done      : BOOL;
    Busy      : BOOL;
    Error     : BOOL;
    ErrorID   : UDINT;
    cbData_R  : UDINT;
END_VAR
```

Name	Type	Description
Done	BOOL	Switches to TRUE if the function block was executed successfully.
Busy	BOOL	TRUE until the function block has executed a command, at the most for the duration of the "Timeout" at the input. The inputs accept no new command as long as Busy = TRUE. It is not the connection time that is monitored but the reception time.
Error	BOOL	Switches to TRUE if an error occurs while executing a command. The command-specific error code is in nErrID.
ErrorID	UDINT	Contains the command-specific ADS error code of the most recently executed command.
cbData_R	UDINT	Number of bytes read.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.2.16 UA_Write



This function block writes values to a given node and connection handle.

Inputs

```
VAR_INPUT
    Execute          : BOOL;
    ConnectionHdl    : DWORD;
    NodeHdl          : DWORD;
    stNodeAddInfo    : ST_UANodeAdditionalInfo;
    pVariable        : PVOID;
    cbData           : UDINT;
    Timeout          : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
Execute	BOOL	The command is triggered by a rising edge at this input.
ConnectionHdl	DWORD	Connection handle previously output by the function block UA_Connect [▶ 234].
NodeHdl	DWORD	Node handle that was previously output by the function block UA_NodeGetHandle [▶ 245].
stNodeAddInfo	ST_UANodeAdditionalInfo	Defines additional information, e.g. which IndexRange or which attribute is to be written (by default, the' Value' attribute is used). Specified by STRUCT ST_UANodeAdditionalInfo [▶ 229].
pVariable	PVOID	Pointer to data to be written.
cbData	UDINT	Sets the size of the values to be written.
Timeout	TIME	Time until the function is aborted. DEFAULT_ADS_TIMEOUT is a global constant, set to 5 seconds.

Outputs

```
VAR_OUTPUT
    Done       : BOOL;
    Busy       : BOOL;
    Error      : BOOL;
    ErrorID    : DWORD;
END_VAR
```

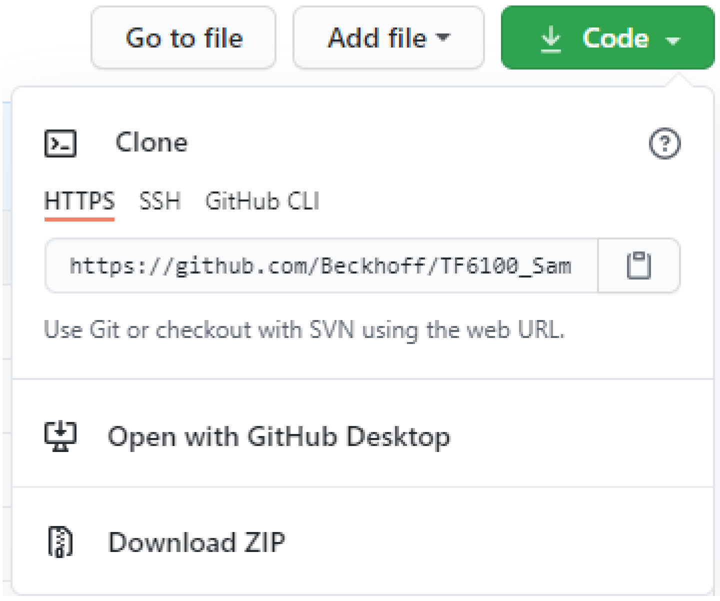
Name	Type	Description
Done	BOOL	Switches to TRUE if the function block was executed successfully.
Busy	BOOL	TRUE until the function block has executed a command, at the most for the duration of the "Timeout" at the input. The inputs accept no new command as long as Busy = TRUE. It is not the connection time that is monitored but the reception time.
Error	BOOL	Switches to TRUE if an error occurs while executing a command. The command-specific error code is included in ErrorID.
ErrorID	DWORD	Contains the command-specific ADS error code of the most recently executed command.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

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 option to clone the repository or download a ZIP file containing the sample.


Go to file
Add file
Code
Clone
HTTPS
SSH
GitHub CLI
https://github.com/Beckhoff/TF6100_Sam
Use Git or checkout with SVN using the web URL.
Open with GitHub Desktop
Download ZIP

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.

7 Appendix

7.1 Error diagnosis

In the following sections, possible errors for all components of the OPC UA setup are shown in the form of a table. In addition, helpful troubleshooting hints are provided for the respective errors.

7.1.1 Server

Behavior	Notes
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 released 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) appears.	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" appears	Check whether the server trusts the client certificate. The required configuration steps can be found in section Certificate exchange [▶ 108]. In this case, it must also be ensured that a signature hash algorithm of the SHA 2 group (SHA256, SHA364, SHA512) is used to sign the client certificate. If obsolete 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 variables, 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.

Behavior	Notes
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 section Arrays [▶ 55].

7.1.2 Client I/O

Behavior	Notes
When creating a new I/O client by specifying the server URL with the host name of the server, no connection can be established subsequently via the AddNodes dialog.	Please check if name resolution is operational in your network. Alternatively try again via the IP address of the server.
Some configuration items from the I/O client are not present, although they should be according to the documentation.	In this case, the system manager description file (TMC) was probably not updated after a TF6100 update. Please execute the command "Reload TMC" from the context menu of the I/O client to reload the description file.
Write commands to a variable are not executed or do not arrive at the server.	Please check whether the "Write Enable" output has been enabled on the I/O client.

7.1.3 Client PLCopen

Behavior	Notes
The attempt to read or write a Structured Data Type from a server fails.	Structured Data Types are not supported by the PLCopen-based client. Please use the I/O client for this purpose.

7.1.4 Gateway

Behavior	Notes
The gateway cannot connect to the server.	One of the possible causes is that an old configuration is being used. For example, if there is a new server certificate, the gateway only notices this when the configured endpoint is deleted and reinserted under a different name. With the same endpoint or a new endpoint with the same name, the gateway would use the connection information from a cache and as a result would no longer be able to connect to the server.

7.2 Status codes

7.2.1 ADS Return Codes

Grouping of error codes:
Global error codes: ADS Return Codes [▶ 259]... (0x9811_0000 ...)
Router error codes: ADS Return Codes [▶ 259]... (0x9811_0500 ...)
General ADS errors: ADS Return Codes [▶ 260]... (0x9811_0700 ...)
RTime error codes: ADS Return Codes [▶ 262]... (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 or is not reachable.
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.
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.
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.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Missing license.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	License expired.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	License exceeded.
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.

Hex	Dec	HRESULT	Name	Description
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 cancelled.

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.
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			

7.2.2 Client I/O

The OPC UA Client modules that belong to a virtual OPC UA device offer different status variables as well as control variables. These variables are explained below.

Reading the status codes

Please note that the status code of the state machine is listed here in hexadecimal notation. If the code is displayed as a decimal number in TwinCAT, it must be converted for interpretation.

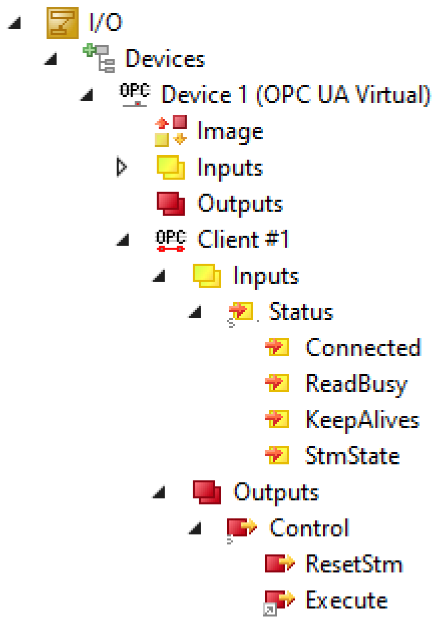


Fig. 1: OPCUAClientModulesStatusCodes

Variable	Schema	0	1- State machine state	2- Keep alive count if using subscriptions	3- Connection state (&read busy)
Status	0x0123	-	0 = Initialize (init)		0 = false(&off)
			1 = Connect		1 = true(&off)
			2 = Resolve namespace		2 = false(&on)
			3 = Get node handles		3 = true(&on)
			4 = Continuous read/write		
			5 = Triggered read/write		
			6 = Awaiting data change notifications (subscriptions))		
			7 = Disconnect		
			8 = Reset		
Control	0x0123	-	-	-	0 = Standard (default) 1 = Reset state machine 2 = Execute (in triggered read mode)

Variable	Data type	Description
Connected	BIT	1 TRUE | 0= FALSE.
ReadBusy	BIT	1 TRUE | 0= FALSE. This function is only active when reading and writing via trigger variables.
KeepAlives	BIT4	Shows the number of KeepAlive messages counted. Only active when reading and writing using subscriptions.
StmState	BYTE	Can be read in the table above.
ResetStm	BIT	The client is reset when this bit is set to 1.
Execute	BIT	1 TRUE | 0= FALSE. Reading/ writing takes place if this bit is set to 1 during reading and writing via trigger variables. **If this bit remains set, there is no difference to cyclic reading/ writing.**

7.2.3 Client PLCopen

The function blocks of the TwinCAT OPC UA Client have their own error codes, which indicate the occurrence of an error and use an ErrorID to display further information about the problem that has occurred. TwinCAT ADS error messages (ADS Return Codes [⏵ 258]) with the HighWord 0x0000 and custom error messages from the client or the PLC library with the HighWord 0xE4DD can occur.

Possible TwinCAT ADS errors include the following:

Hex	Name	Description
0x 0000 0705	DEVICE_INVALIDSIZE	Parameter size not correct
0x 0000 0706	DEVICE_INVALIDDATA	Invalid parameter values
0x 0000 070A	DEVICE_NOMEMORY	Not enough memory

This error code list shows the possible custom error values:

Hex	Name	Description
0x E4DD 0001	UAC_E_FAIL	UA service call failed
0x E4DD 0100	UAC_E_CONNECTED	Server already connected
0x E4DD 0101	UAC_E_CONNECT	General error when establishing a connection
0x E4DD 0102	UAC_E_UASECURITY	UA security could not be set up
0x E4DD 0103	UAC_E_ITEMEXISTS	Element ID already exists
0x E4DD 0104	UAC_E_ITEMNOTFOUND	Element does not exist
0x E4DD 0105	UAC_E_ITEMTYPE	Invalid or unsupported item type
0x E4DD 0106	UAC_E_CONVERSION	Variable types cannot be converted
0x E4DD 0107	UAC_E_SUSPENDED	Device hangs. Please try again later...
0x E4DD 0108	UAC_E_TYPE_NOT_SUPPORTED	Conversion variable type is not supported.
0x E4DD 0109	UAC_E_NSNAME_NOTFOUND	No namespace with the specified name found.
0x E4DD 0110	UAC_E_CONNECT_NOTFOUND	Connection failed: Target host could not be found.
0x E4DD 0111	UAC_E_TIMEOUT	Timeout: i.e. target host does not respond
0x E4DD 0112	UAC_E_INVALIDHDL	Session handle invalid
0x E4DD 0113	UAC_E_INVALIDNODEID	UA node ID unknown
0x E4DD 0114	UAC_E_INVAL_IDENTIFIER_TYPE	Identifier type of UaNodeId invalid
0x E4DD 0115	UAC_E_IDENTIFIER_NOTSUPP	Identifier type UaNodeId is not supported
0x E4DD 0116	UAC_E_INVAL_NODE_HDL	Invalid node handle
0x E4DD 0117	UAC_E_UAREADFAILED	UA read failed for unknown reasons
0x E4DD 0118	UAC_E_UAWRITEFAILED	UA write failed for unknown reasons
0x E4DD 0119	UAC_E_INVAL_NODEMETHOD_HDL	Invalid method handle
0x E4DD 011A	UAC_E_CALL_FAILED	Call failed, cause unknown
0x E4DD 011B	UAC_E_CALLDECODE_FAILED	Successful call, decoding return value failed
0x E4DD 011C	UAC_E_NOTMAPPEDTYPE	Unassigned data type in return value
0x E4DD 011D	UAC_E_CALL_FAILED_BADINTERNAL	Call failed with UA_BadInternal
0x E4DD 011E	UAC_E_METHODIDINVALID	Unknown MethodID (returned on call, even if provided by GetMethodHdl)
0x E4DD 011F	UAC_E_TOOMUCHDIM	Method call has returned parameters with more than 3 dimensions; not supported.
0x E4DD 0120	UAC_E_CALL_FAILED_INVALIDARG	Call failed with OpcUa_BadInvalidArgument
0x E4DD 0121	UAC_E_CALL_FAILED_TYPEMISMATCH	Call failed with UAC_E_CALL_FAILED_TYPEMISMATCH
0x E4DD 0122	UAC_E_CALL_FAILED_OUTOFRANGE	Call failed with UAC_E_CALL_FAILED_OUTOFRANGE
0x E4DD 0123	UAC_E_CALL_FAILED_BADSTRUCTURE	Call failed with OpcUa_BadStructureMissing
0x E4DD 0124	UAC_E_CALL_TYPEMISMATCH_OUTPARAM	Call successful, but type of output information provided does not match
0x E4DD 0125	UAC_E_NONVALIDTYPEINFO	Node has insufficient type information
0x E4DD 0126	UAC_E_INVALIDATTRIBID	Access to invalid node attribute
0x E4DD 0128	UAC_E_NOTSUPPORTED	The command is not supported by the connected UaServer, e.g. when calling UA_HistoryUpdate.
0x E4DE 0100	UAC_E_INVALID_ARRAY_LENGTH	An invalid array length not matching DataValueCount was assigned to UA_HistoryUpdate.
0x E4DE 0101	UAC_E_INVALID_DATASIZE	A data value with an invalid data type size was assigned to UA_HistoryUpdate. All assigned DataValues must be of the same data type.

Hex	Name	Description
0x E4DE 0102	UAC_E_SUBERROR	A lower-level error was output for at least one of the transferred data values. See ValueErrorIDs at UA_HistoryUpdate.

7.3 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

More Information:
www.beckhoff.com/ts6100

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com