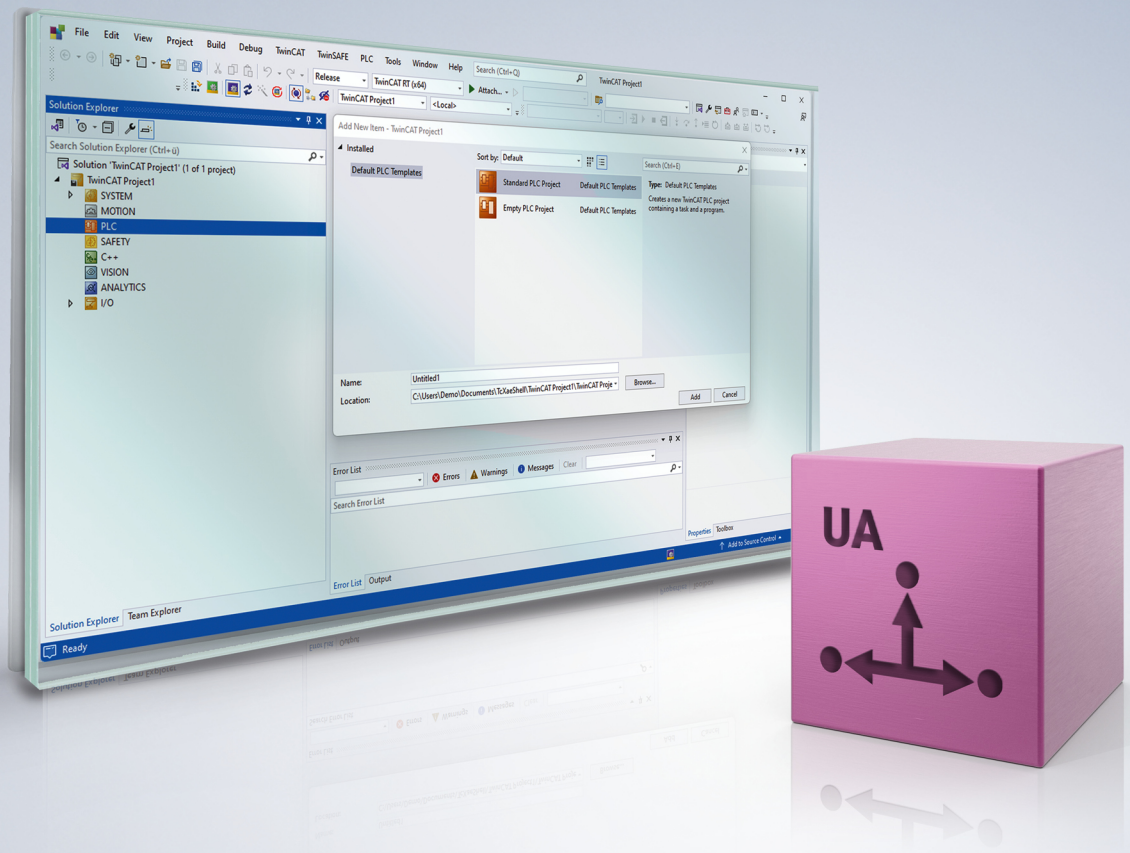
BECKHOFF New Automation Technology

Manual | EN

TF6100

TwinCAT 3 | OPC UA Client



2026-04-08 | Version: 1.3.0

Table of contents

1 Foreword 5
1.1 Notes on the documentation 5
1.2 For your safety 6
1.3 Notes on information security 7
1.4 Documentation issue status 7
2 Overview 8
3 Installation 10
3.1 System Requirements 10
3.2 Installation 11
3.3 Licensing 14
4 Technical introduction 18
4.1 Quick start 18
4.2 Quick Start (TwinCAT 2) 21
4.3 Version overview 25
4.4 Software architecture 26
4.5 Data types 28
4.6 Application directories 29
4.7 Reading variables 30
4.8 Writing variables 31
4.9 Method calls 33
4.10 Timestamp and StatusCode 36
4.11 Structures 36
4.12 Code generation 39
4.13 PLCopen function blocks 43
4.14 Security 55
4.14.1 Overview 55
4.14.2 Certificate exchange 55
4.14.3 Authentication 56
4.15 Logging 58
4.16 ApplicationURI 60
5 PLC API 62
5.1 Tc2_OpcUa 62
5.1.1 Data types 62
5.1.2 Function blocks 63
5.2 Tc3_PLCopen_OpcUa 65
5.2.1 Instructions for use 65
5.2.2 Data types 67
5.2.3 Function blocks 81
5.2.4 Parameter list 100
6 Samples 101
7 Tutorials 102
8 Appendix 103

8.1 Error diagnosis 103
8.2 Status codes 103
8.2.1 ADS Return Codes 103
8.2.2 Client I/O 107
8.2.3 Client PLCopen 108
8.3 Support and Service 109

1 Foreword

1.1 Notes on the documentation

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

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

Disclaimer

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

Trademarks

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



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

Copyright

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

Third-party trademarks

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

1.2 For your safety

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Signal words

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

Personal injury warnings

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

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

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

Warning of damage to property or environment

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

Information on handling the product



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

1.3 Notes on information security

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

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

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

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

1.4 Documentation issue status

Version	Change
1.3.x	**New:** Installation - TwinCAT 3 Usermode Runtime [▶ 12] Version overview [▶ 25] Data types [▶ 28] ApplicationURI [▶ 60] Instructions for use [▶ 65]
1.2.x	**New:** TS6100_Installation [▶ 11] Licensing under TwinCAT 2 [▶ 14] Technical introduction/logging [▶ 58] Parameter list [▶ 100]
1.1.x	**New:** Supported functions Quick Start (TwinCAT 2) [▶ 21] Security [▶ 55] Tutorials [▶ 102]

2 Overview

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

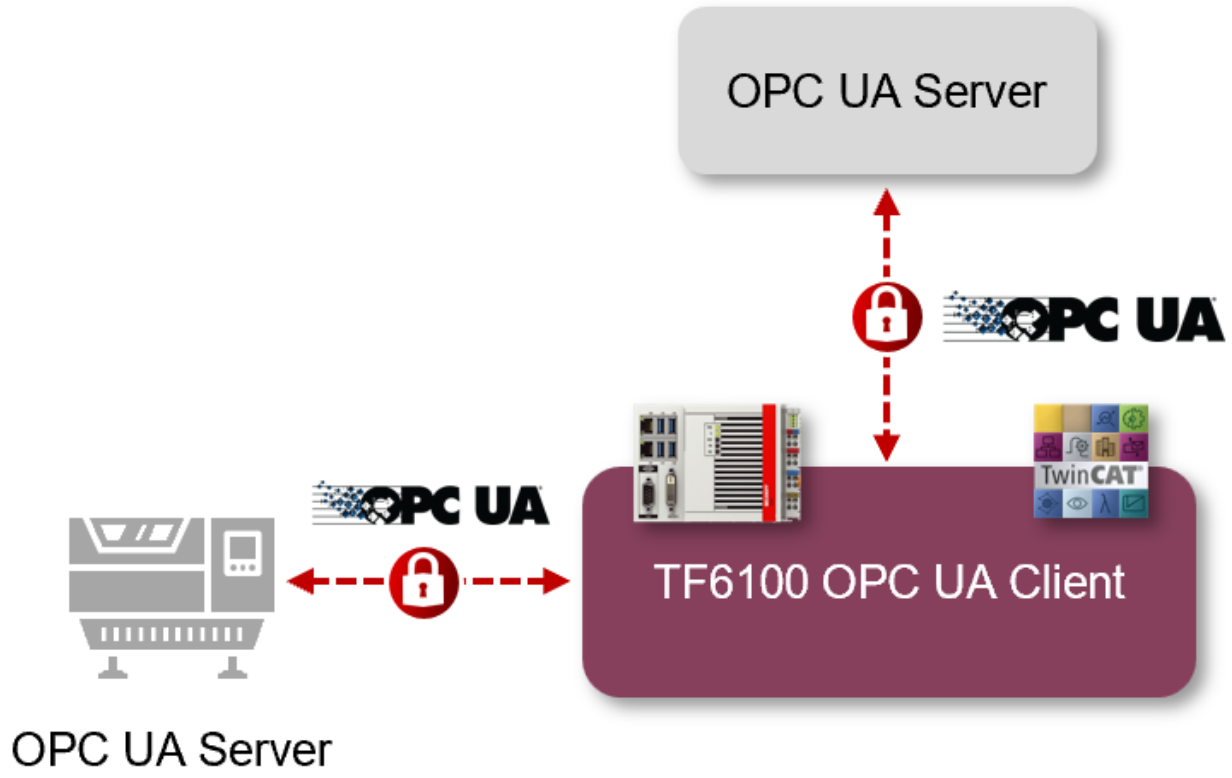


Detailed information on OPC UA can be found on the web pages of the OPC Foundation.

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

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

This documentation describes the TwinCAT 3 OPC UA Client, which is a software component that provides an OPC UA Client interface for the TwinCAT Runtime environment. The TwinCAT 3 OPC UA Client can therefore be used to initiate connections with OPC UA servers in order to exchange data with them.



The technical use cases range from TCP-based (and therefore non-real-time capable) machine-to-machine communication to machine-to-cloud communication if the OPC UA server to be connected is located in the cloud.

The TwinCAT OPC UA Client is technically available in two different variants:

1. As a TwinCAT I/O device
2. As PLC function blocks

Further information

- For an overview of any functional differences, we recommend our article Supported functions.
- Please note the System Requirements [▶ 10] for this product.
- For a quick introduction to the product, we recommend our articles Installation [▶ 11] and Quick start [▶ 18].

3 Installation

3.1 System Requirements

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

Client v2.x

This client version is part of setup version 2.x for TwinCAT 3.1 Build 4026.

Technical data	Description
Operating system	Windows 10 Windows 11 Windows Server 2022 TwinCAT/BSD TwinCAT/Beckhoff RT Linux®
Target platforms	PC architecture (x86, x64, Arm®)
.NET Framework	---
Minimum TwinCAT version	TwinCAT 3.1 Build 4026.19
Minimum TwinCAT installation level	TwinCAT 3 XAE, XAR
Required TwinCAT license	TF6100 TC3 OPC UA

Client v1.x

This client version is part of setup version 4.4.x for TwinCAT 3.1 Build 4024 and workload version 1.x for TwinCAT 3.1 Build 4026.

Technical data	Description
Operating system	Windows 10 Windows 11 Windows CE 6/7 Windows Server 2022
Target platforms	PC architecture (x86, x64, Arm®)
.NET Framework	---
Minimum TwinCAT version	TwinCAT 3.1
Minimum TwinCAT installation level	TwinCAT 3 XAE, XAR
Required TwinCAT license	TF6100 TC3 OPC UA

SampleServer

The SampleServer is part of all setups and workloads.

Technical data	Description
Operating system	Windows 10 Windows Server 2022
Target platforms	PC architecture (x86, x64)
.NET Framework	4.7.2
Minimum TwinCAT version	---
Minimum TwinCAT installation level	---
Required TwinCAT license	---

Version compatibility between engineering and runtime

The following table provides an overview of the supported version combinations between the engineering and runtime system when using the client.

Engineering	Runtime	Supported
Client v1	Client v1	Yes
Client v1	Client v2	Yes
Client v2	Client v1	No
Client v2	Client v2	Yes

For a complete overview of the version differences, we recommend the following articles: Version overview [▶ 25], Software architecture [▶ 26].

3.2 Installation

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

NOTICE

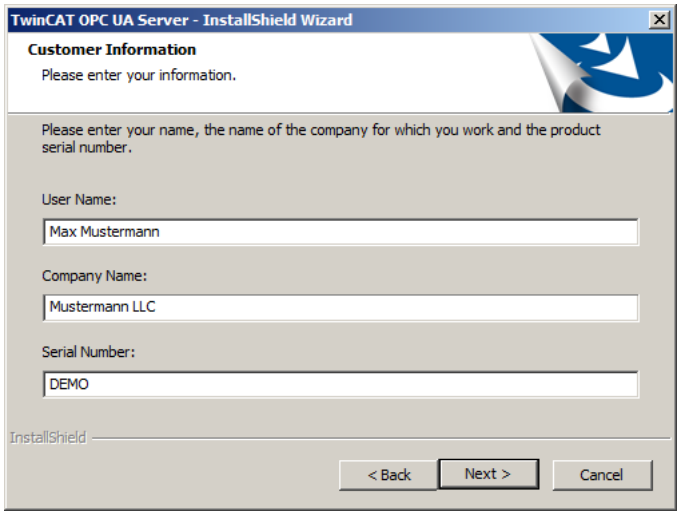
Update installation

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

TwinCAT 2 Setup

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

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



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

TwinCAT 3 Package Manager

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

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

NOTICE

Unprepared TwinCAT restart can cause data loss

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

Command line program TcPkg

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

```
tcpkg list -t workload
tcpkg install TF610x.OpcUaClientPubSub.XAE
tcpkg install TF610x.OpcUaClientPubSub.XAR
```

TwinCAT Package Manager UI

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

TwinCAT 3 Usermode Runtime

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

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

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

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

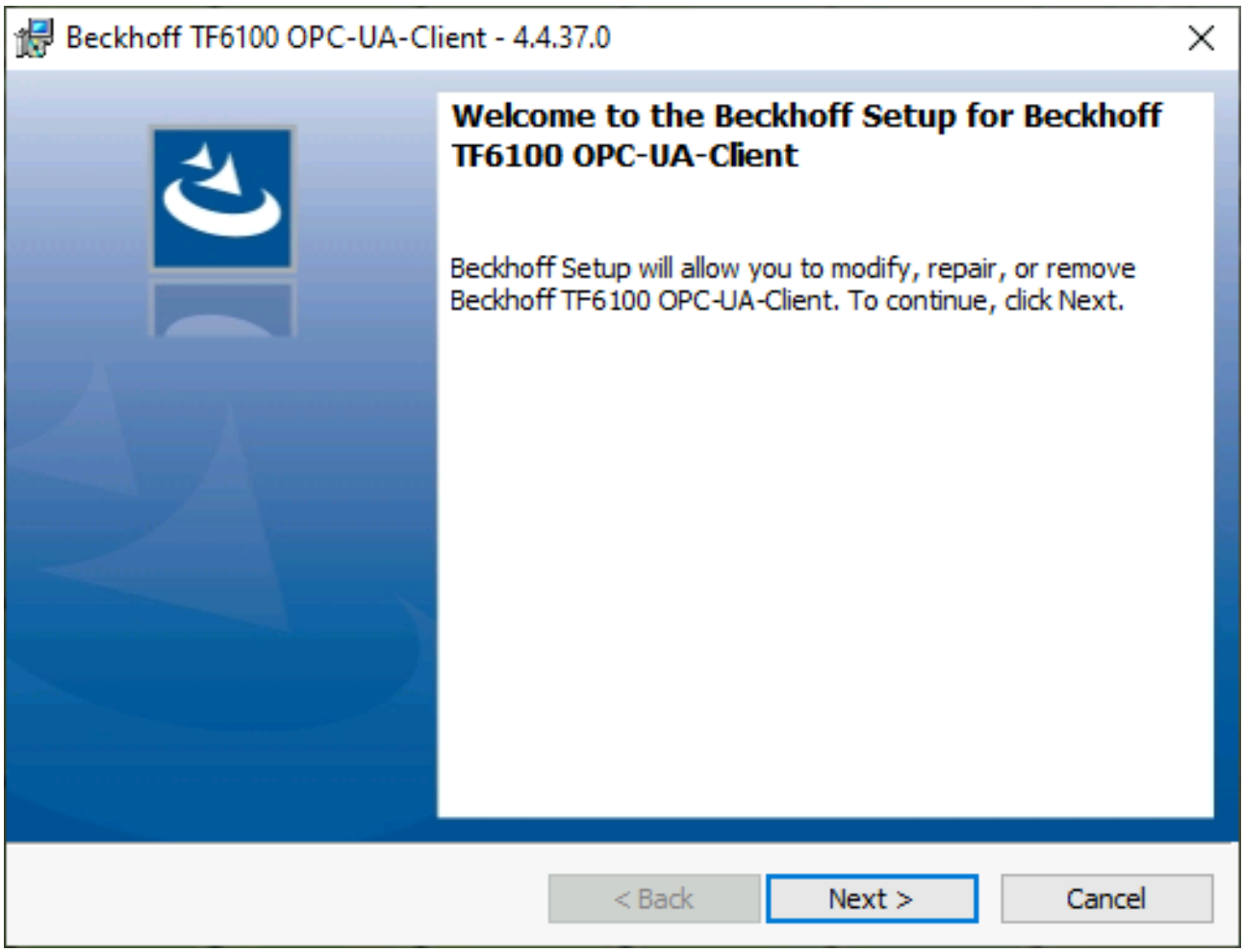
- The installation path of the executable to be launched
- Any parameters required to start the TwinCAT 3 Function

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

TwinCAT 3 Setup

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

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



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

NOTICE

Unprepared TwinCAT restart can cause data loss

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

Beckhoff RT Linux®

1. Use the CLI directly on your device. Alternatively, you can also connect via SSH.
2. Log in as an administrator.
3. Update the packages in the Package Manager.

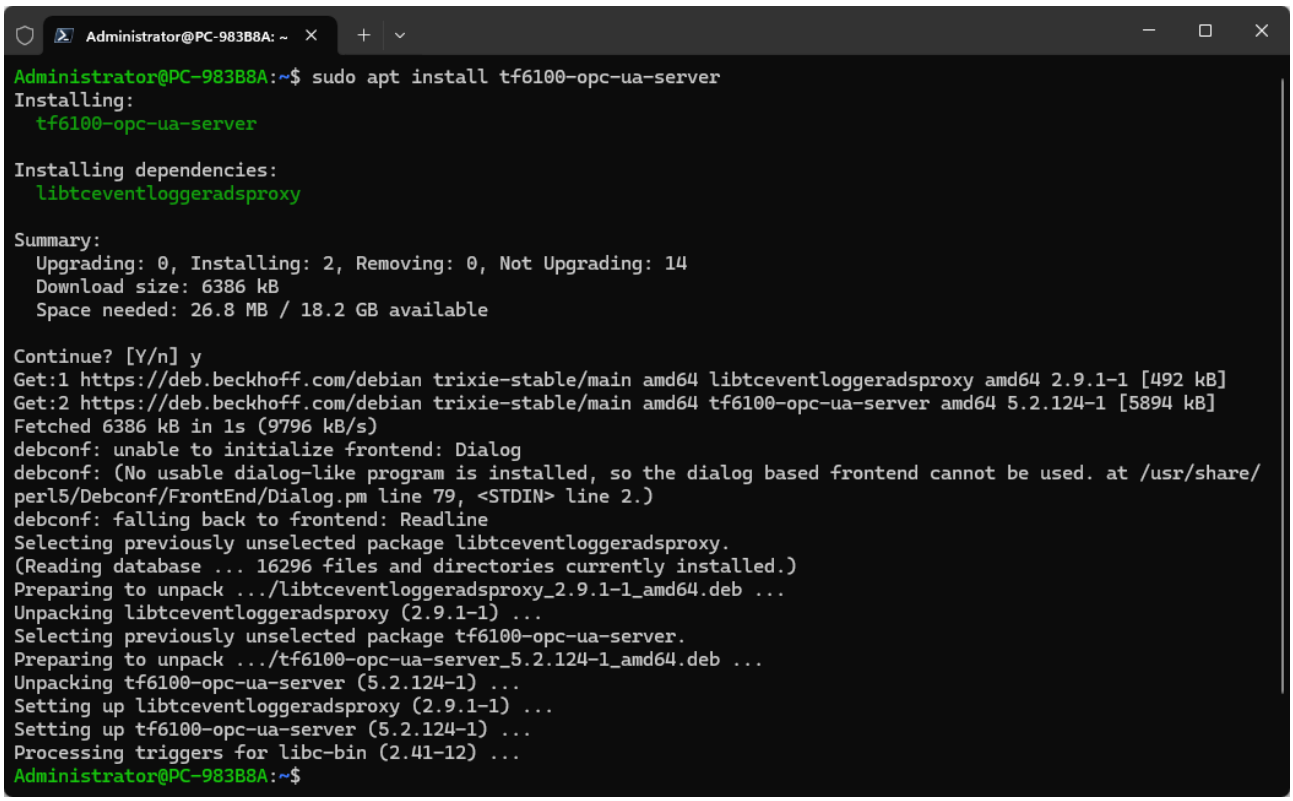
```
sudo apt update
```

1. Search for the existing TF6100 packages.

```
sudo apt search tf610x
```

1. Install TF6100 OPC UA server via the Package Manager.

```
sudo apt install tf610x-opc-ua-client-pubsub
```



Windows CE

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

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

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

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

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

Device restart

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

3.3 Licensing

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

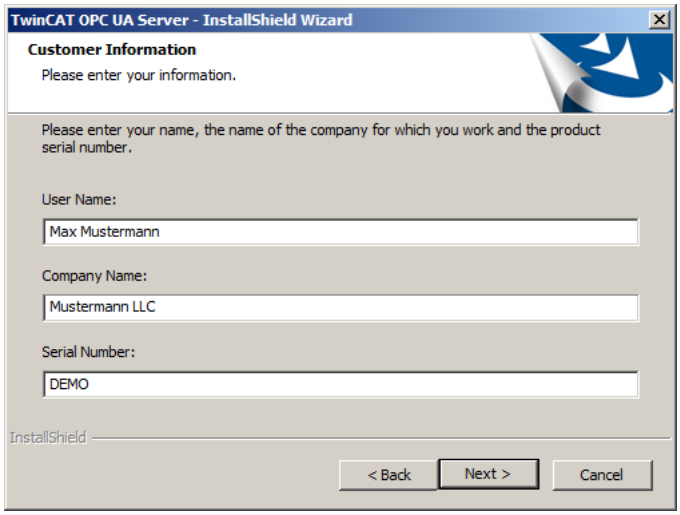
For more information about licensing the product, see the following sections:

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

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

Licensing under TwinCAT 2

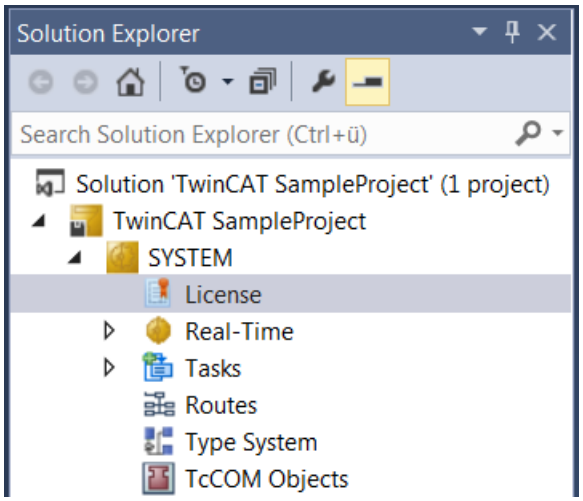
The product is licensed under TwinCAT 2 during installation; a corresponding text field allows you to enter a license key. To evaluate this product, you can activate it for a period of 30 days. To do this, enter "Demo" as the license key.



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

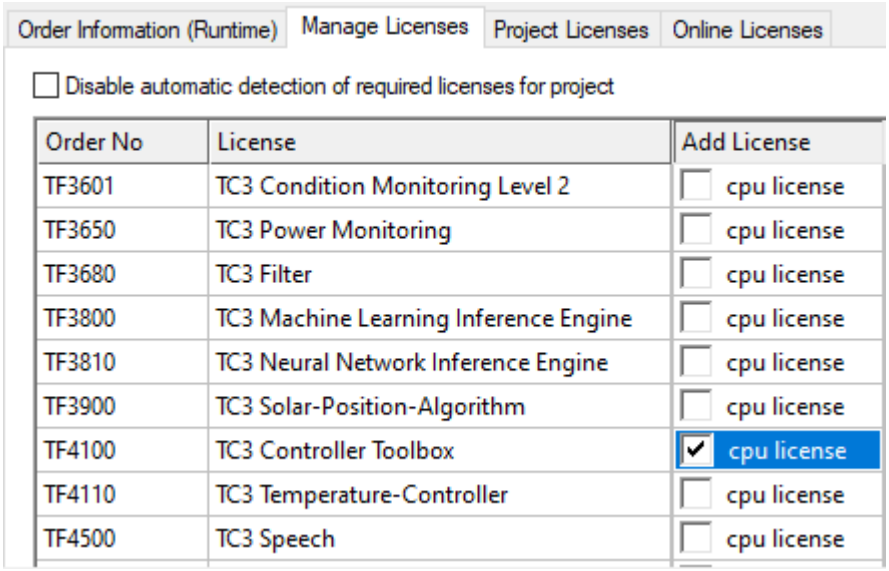
A 7-day test version cannot be enabled for a TwinCAT 3 license dongle.

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

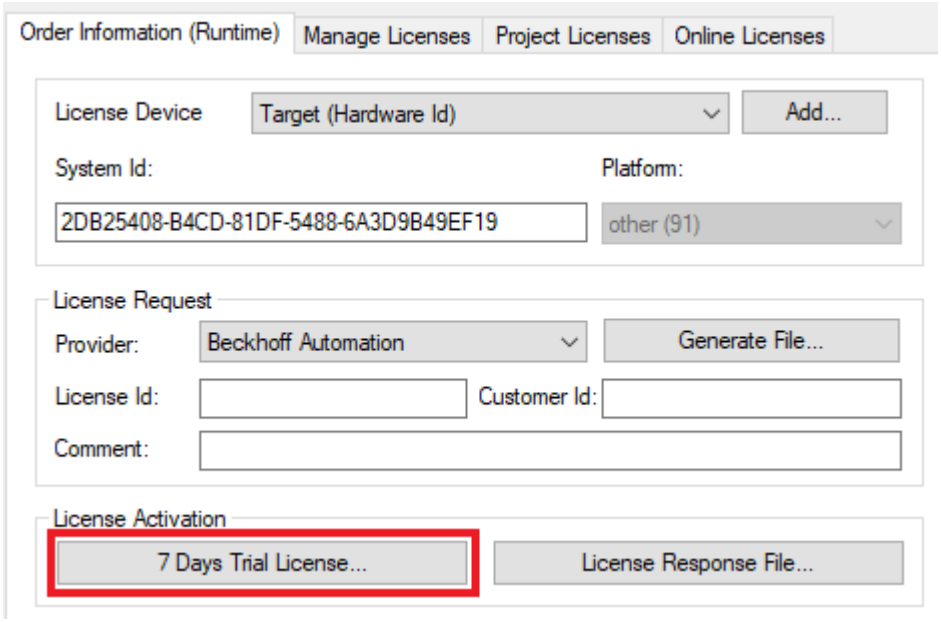


⇨ The TwinCAT 3 license manager opens.

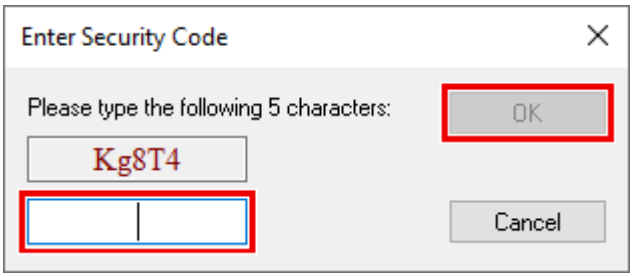
5. Open the **Manage Licenses** tab. In the **Add License** column, check the check box for the license you want to add to your project (e.g. "TF4100 TC3 Controller Toolbox").



6. Open the **Order Information (Runtime)** tab.
   ⇨ In the tabular overview of licenses, the previously selected license is displayed with the status “missing”**.**
7. Click **7-Day Trial License...** to activate the 7-day trial license.



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



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

10. Restart the TwinCAT system.

⇨ The 7-day trial version is enabled.

Licensing the full version of a TwinCAT 3 Function

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

4 Technical introduction

4.1 Quick start

The following chapter provides a quick start to the TwinCAT OPC UA I/O Client. In these instructions, the connection to a Sample OPC UA Server is set up, which offers some variables in its namespace. These variables are added to the process image of the TwinCAT OPC UA I/O Client and then linked to PLC variables.

Sample OPC UA Server

The Sample OPC UA Server is delivered together with the TF6100 OPC UA Client setup and is located in the installation directory [▶ 29] of the client. Alternatively, you can also use the TwinCAT OPC UA Server instead of the Sample OPC UA Server to provide a few variables via OPC UA.

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

- Starting the Sample OPC UA Server
- Creating a TwinCAT project
- Reading the OPC UA variables
- Starting the code generation

Starting the Sample OPC UA Server

1. In Windows Explorer, navigate to the installation directory of the TF6100 FunctionSample and then to the subdirectory "SampleServer".
2. Start the file *TcOpcUaSampleServer.exe* as administrator.

⇨ The server is started in a console window and can then be accessed at the following OPC UA URL:

```
opc.tcp://localhost:48030
```



You can confirm the message regarding the limited runtime.

Creating a TwinCAT project

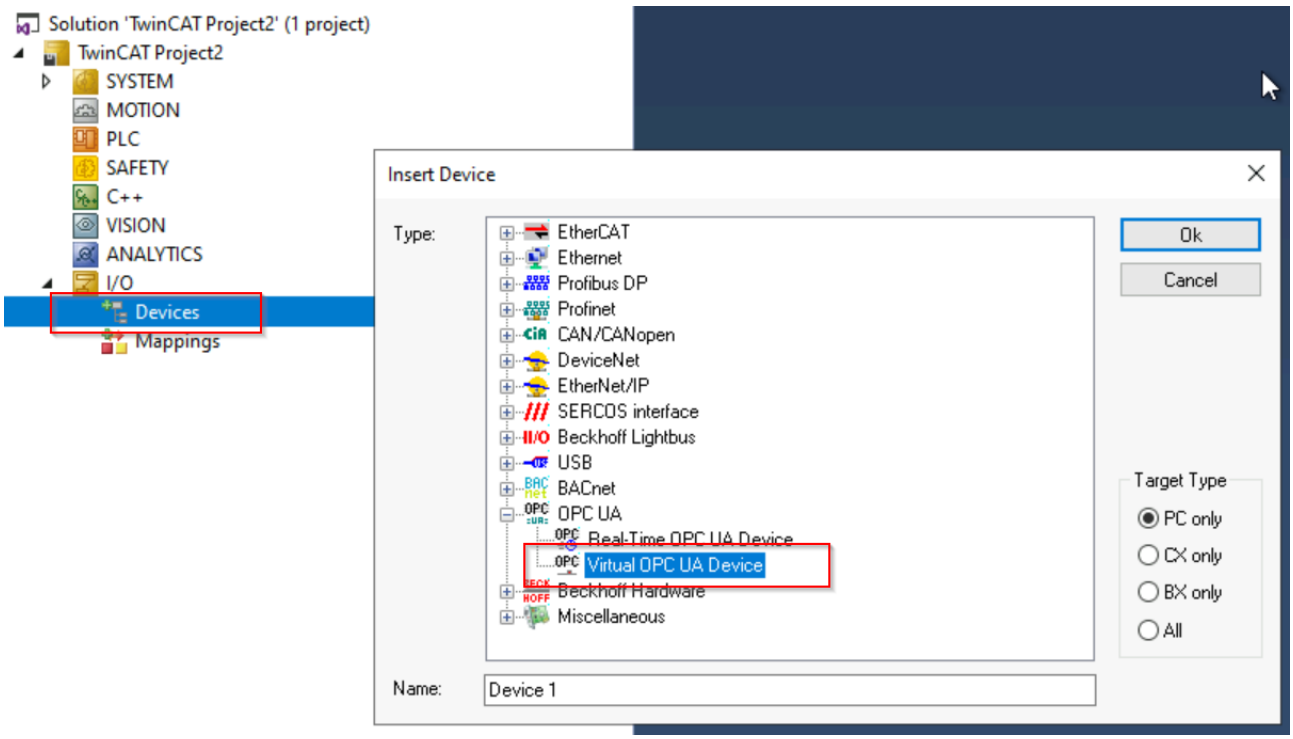
1. Open the TwinCAT XAE Shell.
2. In the **File** menu, select the command **New > Project**.
3. Add a PLC project to the project.

⇨ A new TwinCAT project including PLC project was created.

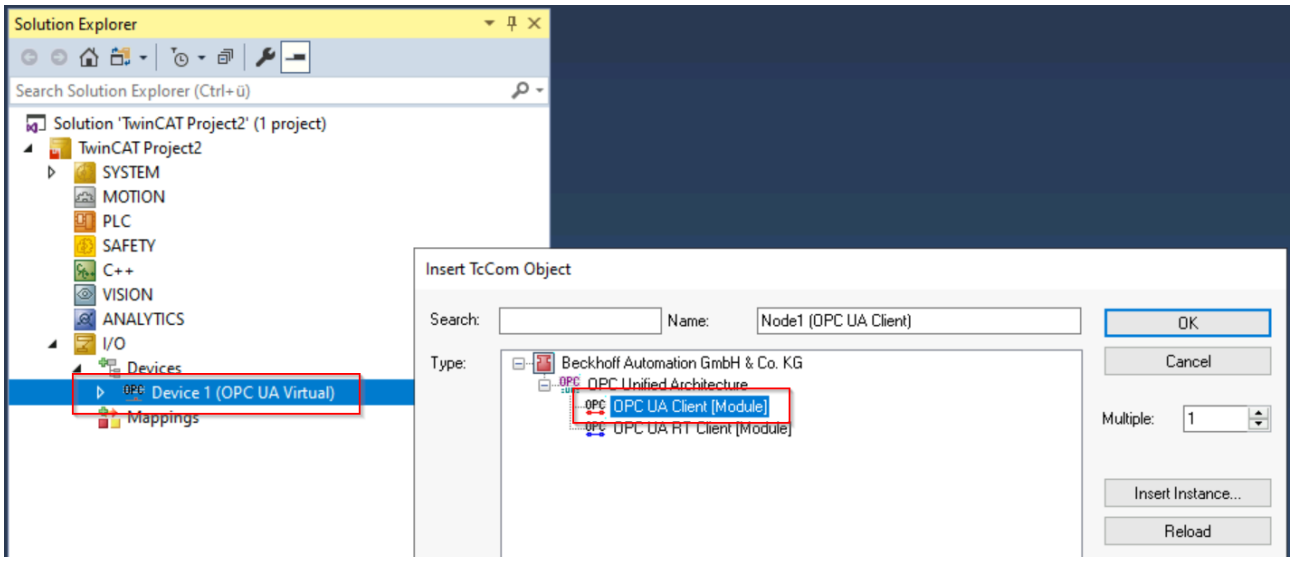
Reading the OPC UA variables

✓ In this step, the TwinCAT OPC UA Client is used to establish a connection to the server and read in the variables available there.

1. Add a new I/O device to the TwinCAT project.
   Use "OPC UA Virtual Device" as the device type

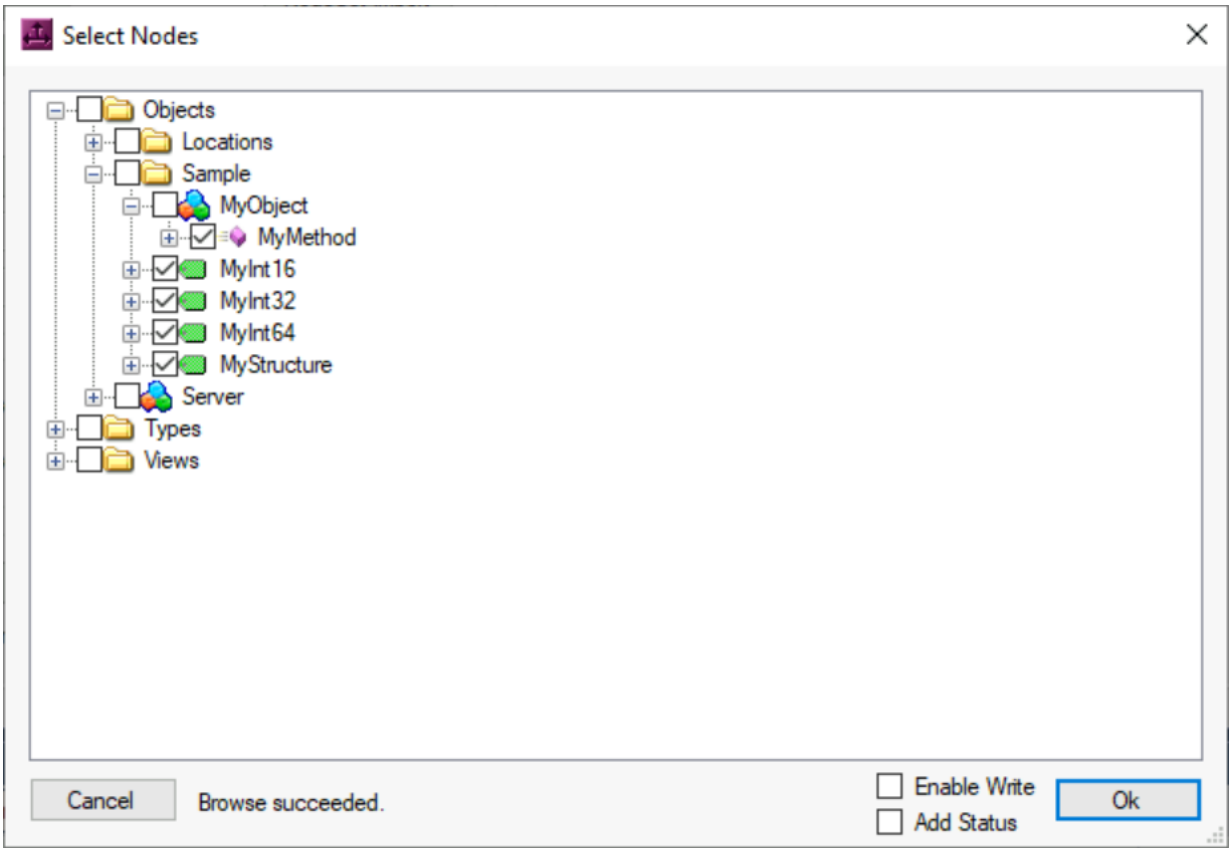


2. Add an OPC UA Client to the device



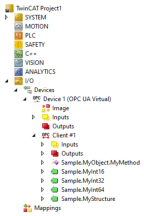
3. Open the settings of the OPC UA Client by double-clicking on the client.
4. Navigate to the **Settings** tab. Enter the server URL of the OPC UA Server. In this sample this is "opc.tcp://localhost:48030".

5. Click **Add Nodes**. A connection to the server is established and the address space of the server is displayed in a separate dialog.



6. Select the nodes shown above and click the **Ok** button.

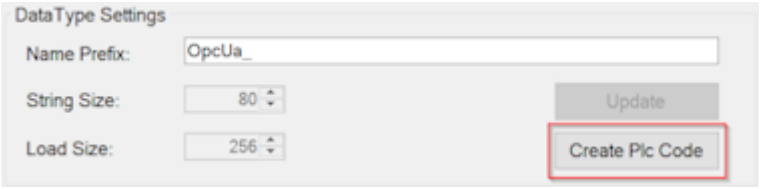
⇨ The variables and the method for the process image of the client have been added.



Starting the code generation

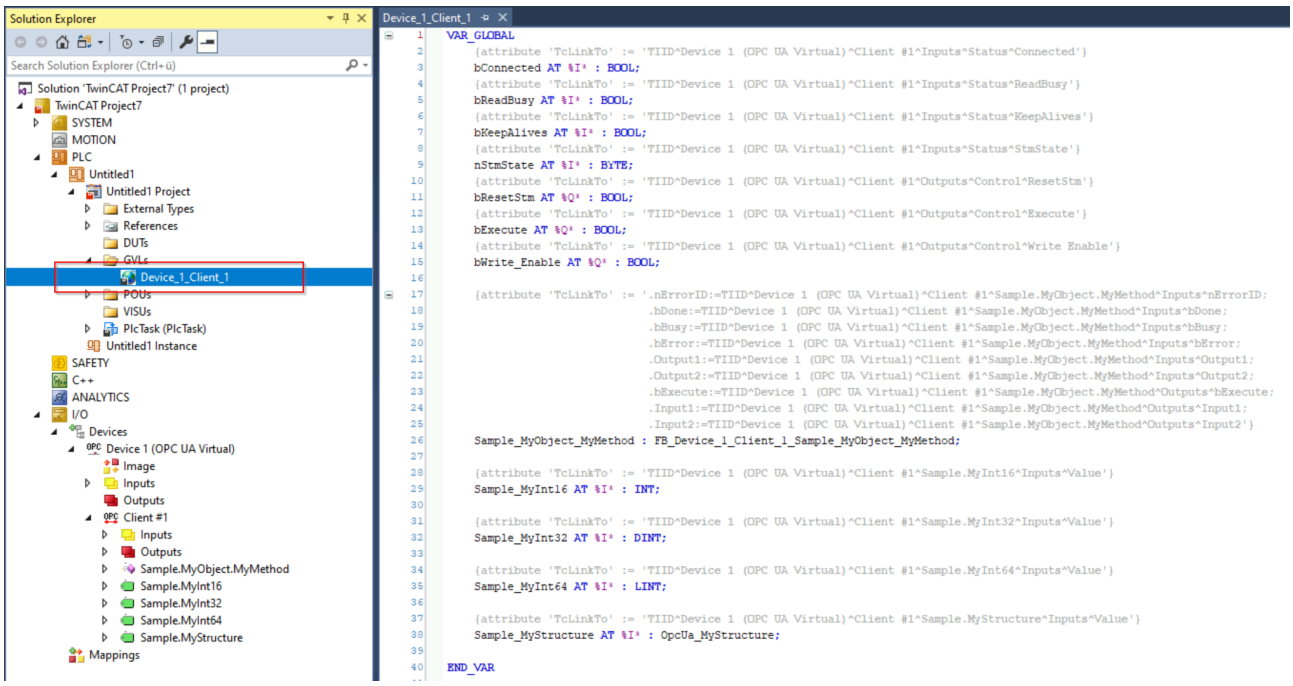
✓ Automatic code generation should be used to generate PLC variables to match the added OPC UA nodes. The generated PLC variables are automatically linked to the nodes. Alternatively, you can perform the mapping manually.

1. Double-click the OPC UA client.
2. In the **Settings** tab, select the **DataType Settings** **section.**



3. Click the **Create Plc Code** button, which starts the code generation.

⇨ The code generator creates a GVL in the existing PLC project whose name is derived from the OPC UA Client device name. PLC variables have now been automatically created within the GVL and linked to the corresponding nodes in the process image of the I/O device via the "TcLinkTo" pragma.



4. Activate the configuration.

⇨ The values of the OPC UA nodes are read from the server and written to the PLC variables via the mapping.

> **Further information on calling the method**
>
> Further PLC logic is required to call the method. For this purpose, a suitable function block has already been created by the code generation and provided with the corresponding input/output parameters, see chapter Method calls [▶ 33].

4.2 Quick Start (TwinCAT 2)

The following chapter provides a quick start to the TwinCAT OPC UA Client. In these instructions, the PLC [▶ 101] sample is used to establish a connection to an OPC UA server and read out a variable. Any server can be used as an OPC UA server, e.g. the TwinCAT OPC UA Sample Server [▶ 18], which can also be started as an executable file on a TwinCAT 2 system. Alternatively, you can also use the OPC Foundation's .NET Standard Sample Server, which can be obtained from the OPC Foundation's GitHub repository.

The following sample is based on the TwinCAT OPC UA Sample Server. The server does not necessarily have to be started in order to follow the individual steps. The aim of this quick start is to familiarize you with the PLC sample and learn which parts of the sample you need to adapt to your operating environment.

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

- Address space of the Sample OPC UA server
- Identify the OPC UA node to be read
- Adjusting the PLC sample for the TwinCAT OPC UA Client

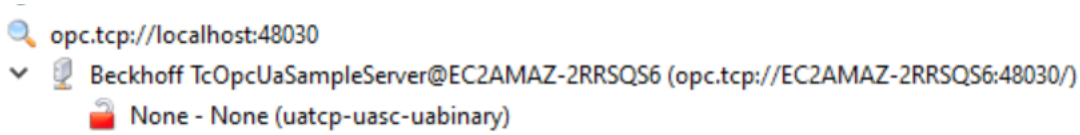
> **TwinCAT OPC UA I/O Client**
>
> The TwinCAT OPC UA I/O Client is only available under TwinCAT 3. Under TwinCAT 2, we enable communication exclusively via the PLC library Tc2_PLCopen_OpcUa. This article has been written accordingly for the use of this library.

Address space of the Sample OPC UA server

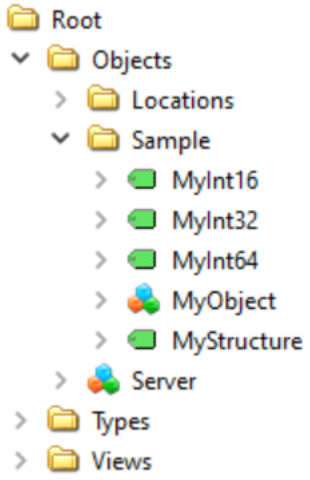
After starting the server, its address space contains a number of nodes that can be read by a client. An OPC UA client requires the following information to establish a connection:

- Server URL
- Endpoint
- Information on the IdentityToken used

In this sample, the server can be reached under the following server URL and the endpoint "None/None". Anonymous" is used as the IdentityToken. The following screenshots were created based on the "UA Expert" software.



After the connection has been established, the address space of the server used here is as follows:

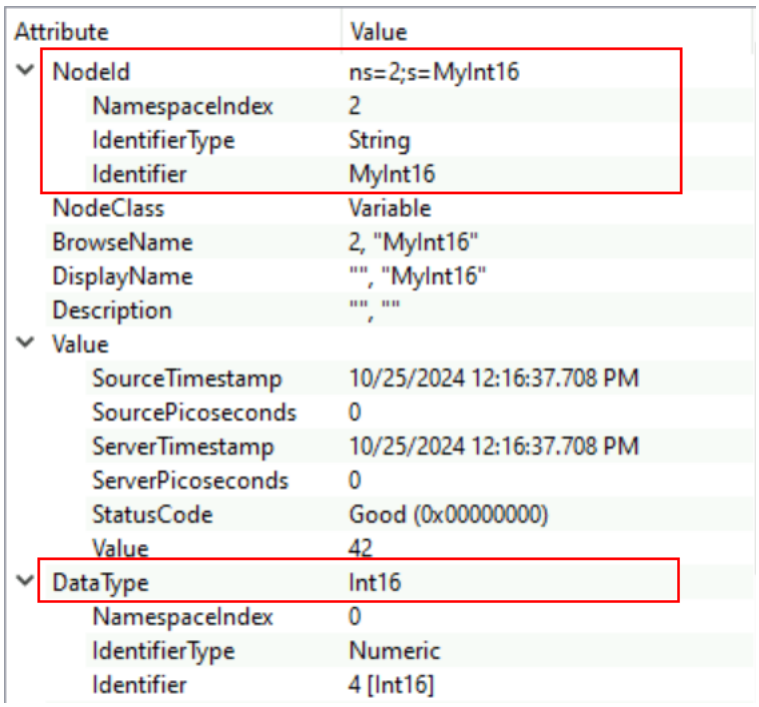


Identify the OPC UA node to be read

To read out a node, you need the specific address information, which you will find in the form of the so-called "attributes". The following attributes are required for reading a node:

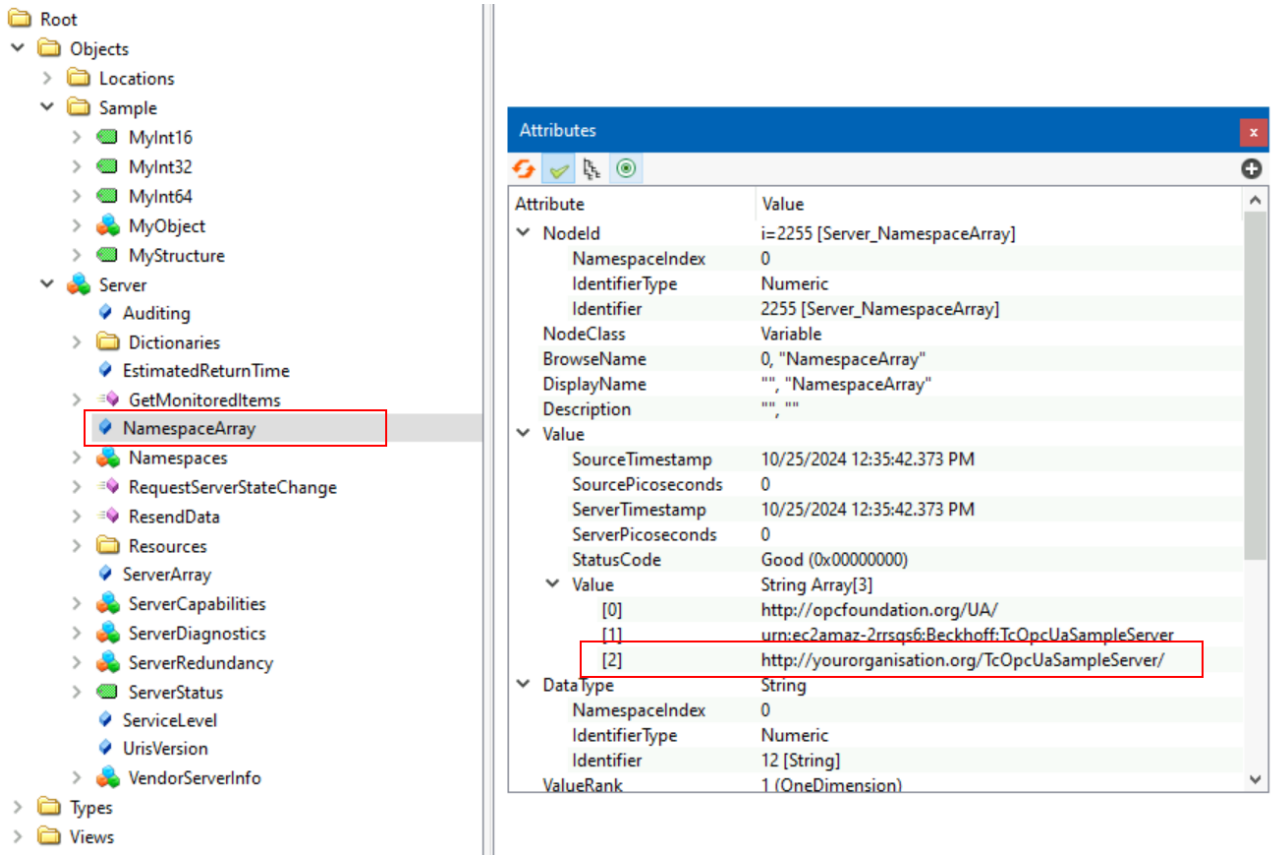
- NodeID (consisting of NamespaceIndex or NamespaceName, as well as IdentifierType and Identifier)
- DataType of the node

In this sample, we want to read the "MyInt16" node. The attributes mentioned above are as follows:



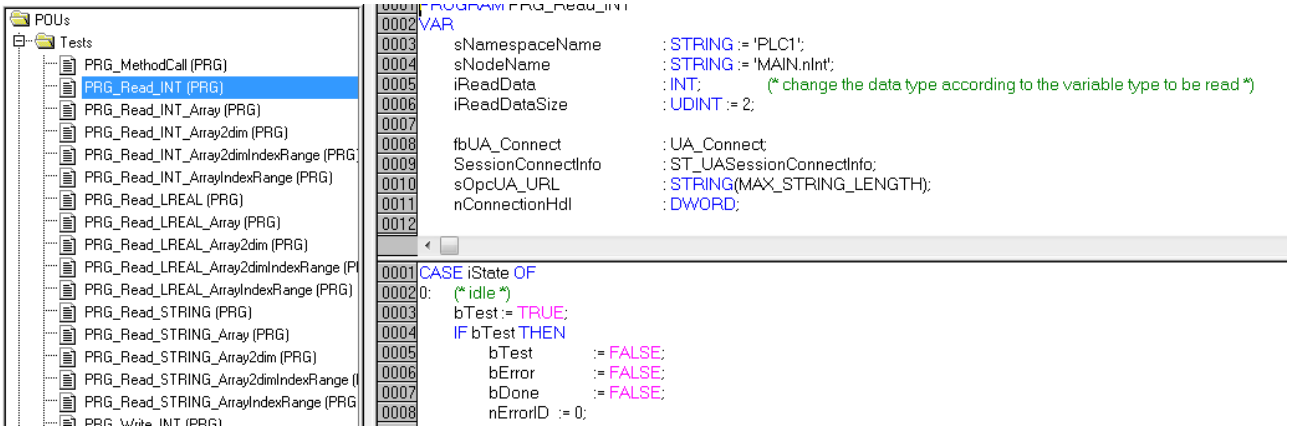
The PLCopen mapping specifies that an OPC UA node must be mapped from the data type "Int16" to the PLC data type "INT". We must take this into account in the following step.

You can use the NamespaceIndex for communication. However, best practice is to use the associated NamespaceName, as the NamespaceIndex can change dynamically. The NamespaceName, on the other hand, is static and enables a resolution in the NamespaceIndex at runtime. You will find the NamespaceName below the "Server" object and there in the so-called "NamespaceArray". The NamespaceName is then the xth entry in the array, i.e. the second entry here.



Adjusting the PLC sample for the TwinCAT OPC UA Client

Open the PLC sample in TwinCAT PLC Control and open the PRG_Read_INT program. This program already contains a ready-made State Machine that uses the function blocks from the Tc2_PLCopen_OpcUa library to connect to an OPC UA server and read a node.



Establishing the connection

The following variable values may now need to be adjusted to establish the connection:

- sOpcUA_URL: Set the variable value to the ServerURL of the server, in this case 'opc.tcp://localhost:48030'
- SessionConnectInfo.eSecurityMode: In this case, the server used by us uses the SecurityMode 'None'. The variable value must therefore be set to 'eUASecurityMsgMode_None'.
- SessionConnectInfo.eSecurityPolicyUri: In this case, the server used by us uses the SecurityPolicy 'None'. The variable value must therefore be set to 'eUASecurityPolicy_None'.
- As the server used uses 'Anonymous' as the IdentityToken, you do not need to specify any values for the SessionConnectInfo.stUserIdentTokenType variable.

This part of the PLC program therefore looks as follows:

```
(* Parameterize *)
sOpcUA_URL                          := 'opc.tcp://localhost:48030';     (* OpcUA Url of the device with ip address *)

SessionConnectInfo.tConnectTimeout  := T#1S;
SessionConnectInfo.tSessionTimeout  := T#1S;
SessionConnectInfo.sApplicationName := '';
SessionConnectInfo.eSecurityMode    := eUASecurityMsgMode_None;
SessionConnectInfo.eSecurityPolicyUri := eUASecurityPolicy_None;
SessionConnectInfo.eTransportProfileUri := eUATransportProfileUri_UATcp;
```

Resolve the NamespaceName

In the next step, we have to resolve the NamespaceName into the corresponding NamespaceIndex. We use the function block UA_GetNamespaceIndex for this. This function block receives the NamespaceName determined by us as input. In the PLC sample, set the value of the sNamespaceName variable to 'http://yourorganisation.org/TcOpcUaSampleServer/'.

Adjusting the address information

Now we can adjust the other address information (IdentifierType and Identifier) of the node in the PLC sample. To do this, we set the variables sNodeName and NodeID.eIdentifierType to the previously determined values. The declaration part of the program therefore looks as follows:

```
PROGRAM PRG_Read_INT
VAR
    sNamespaceName      : STRING := 'http://yourorganisation.org/TcOpcUaSampleServer/';
    sNodeName           : STRING := 'MyInt16';
```

And the implementation part:

```
(* Get Node Handle *)
NodeID.eIdentifierType := eUAIdentifierType_String;
NodeID.nNamespaceIndex := nNamespaceIndex;
NodeID.sIdentifier := sNodeName;
fbUA_NodeGetHandle(
    Execute          := TRUE,
    ConnectionHdl    := nConnectionHdl,
    NodeID           := NodeID,
    NodeHdl          => nNodeHdl
);
```

After these adjustments, you can activate the program. The variable value of the OPC UA node is then read by the server and stored in the PLC variable iReadData.

4.3 Version overview

The following table provides an overview of the functionality of the TwinCAT OPC UA Client in the various major versions. In particular, a distinction is made between the two available usage variants (PLCopen function blocks and I/O client).

Feature	PLCopen function blocks	I/O client v1.x.x	I/O client v2.x.x
Polling	x	x	x
Subscriptions	-	x	x
Method calls	x	x	x
Basic data types according to IEC61131	x	x	x
Arrays of basic data types according to IEC61131	x	x	x
Structures	-	x	x
Arrays of structures	-	x	x
Arrays with fixed length	x	x	x
Arrays with dynamic length	-	-	x
Security at transport layer with X.509 certificates (self-signed + CA)	x	x	x
Security at application layer with user name/password	x	x	x
Security at application layer with X.509 certificates	x	x	x
Communication with None/None endpoint	x	x	x
Communication with Basic128 endpoint (Sign&Encrypt)	x	x	x
Communication with Basic128Rsa15 endpoint (Sign&Encrypt)	x	x	x
Communication with Basic256 endpoint (Sign&Encrypt)	x	x	x
Communication with Basic256Sha256 endpoint (Sign&Encrypt)	x	x	x
Code generation	-	x	x
Support of TwinCAT 3 Usermode Runtime	-	x	x
Support for Microsoft Windows	x	x	x
Support for TwinCAT/BSD	x	-	x
Support for Beckhoff RT Linux®	x	-	x

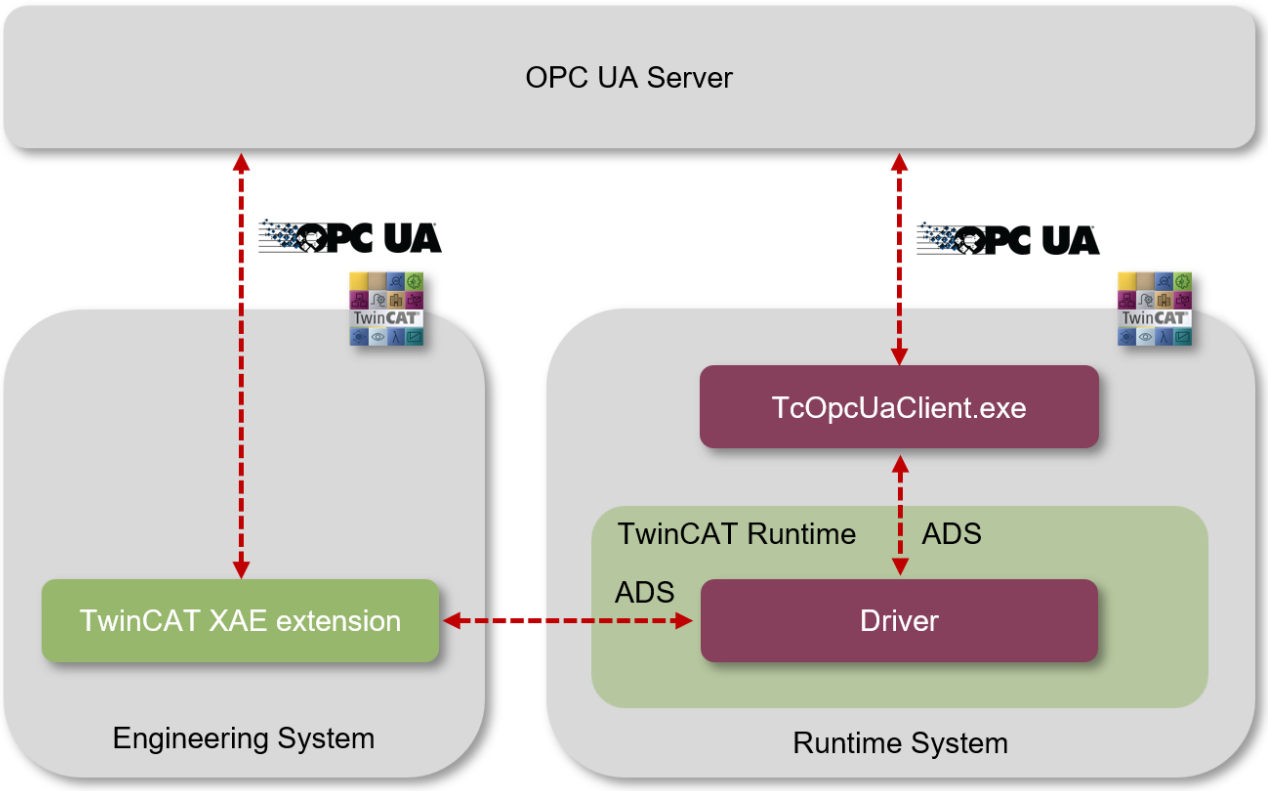
Feature	PLCopen function blocks	I/O client v1.x.x	I/O client v2.x.x
Tunneling the engineering connection through the target system (see Software architecture [▶ 26])	-	-	x

4.4 Software architecture

Previous versions of the TwinCAT OPC UA Client (versions 1.x.x) consisted of several components that were closely interlinked. These were:

- A process in the operating system (TcOpcUaClient.exe)
- A communication driver in real-time (TcIoOpcUa.sys)
- An extension for the TwinCAT XAE

The interaction of these components is illustrated in the following diagram:



Process in the operating system

The process in the operating system (TcOpcUaClient.exe) takes care of the OPC UA protocol functions and makes them available via an ADS server so that the TwinCAT real-time components (e.g. the driver or the PLC library) can access them.

Communication driver in real-time

The driver in real-time is responsible for the communication of the I/O device with the process in the operating system. It converts the configured process data into ADS telegrams, which are exchanged with the process in order to be converted into OPC UA commands. There is an engineering component in TwinCAT XAE for the configuration of the communication connection and the choice of data points.

TwinCAT XAE Extension

The engineering component in TwinCAT XAE provides a graphical configuration interface for the I/O device. This means that variables can be read from OPC UA servers and added to the process image so that they can be processed later at runtime.

PLC library

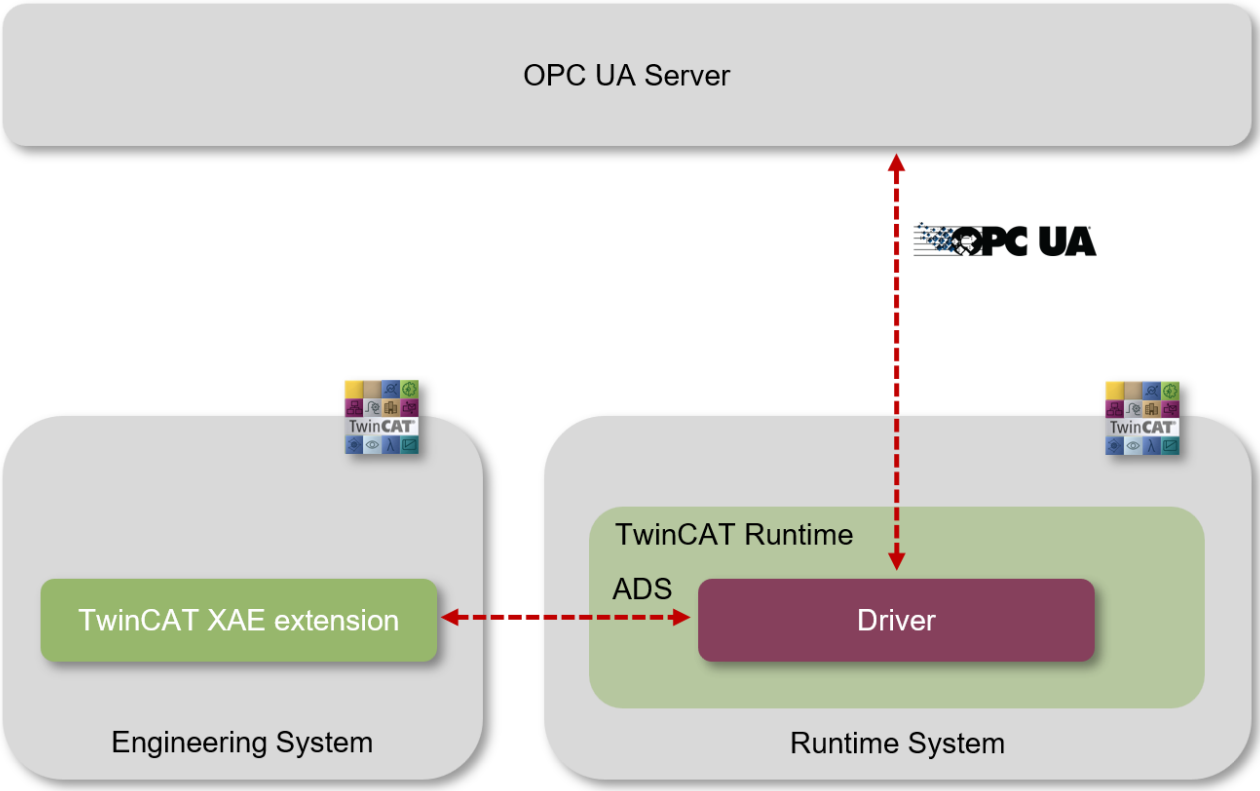
The PLC library provides the OPC UA functions of the process in the operating system for the PLC logic. Similar to the driver, the PLC library communicates with the process via ADS in order to access the OPC UA functions.

TwinCAT OPC UA Client as of version 2.x.x

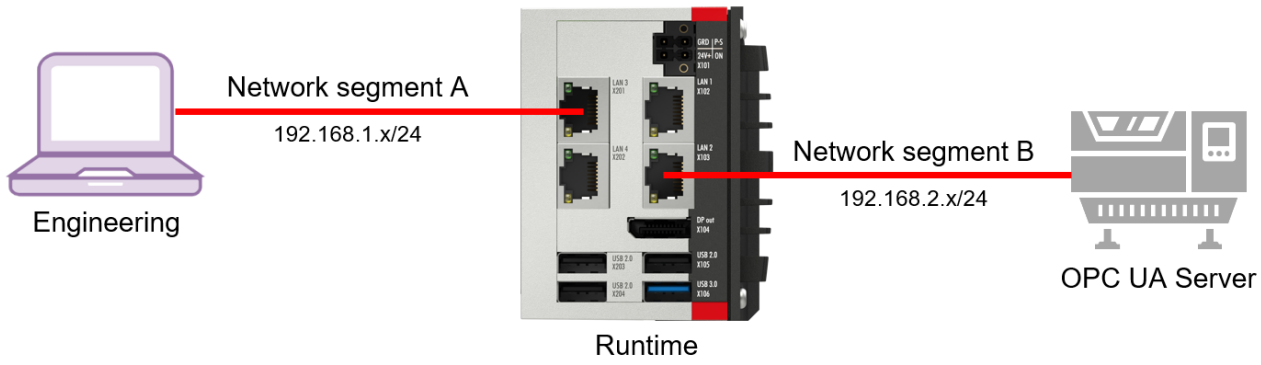
Newer versions of the TwinCAT OPC UA Client only consist of an engineering component and a runtime component (the so-called "driver"). The process in the operating system (TcOpcUaClient.exe) is omitted.

The driver provides the complete OPC UA communication stack, which is then used by both the engineering and TwinCAT runtime. This not only reduces complexity, but also enables the use of a standardized and centralized communication connection with the server. The engineering no longer establishes its own communication connection with the OPC UA server, but instead uses the driver of the selected target device. (Of course, the target device can also be the local system.)

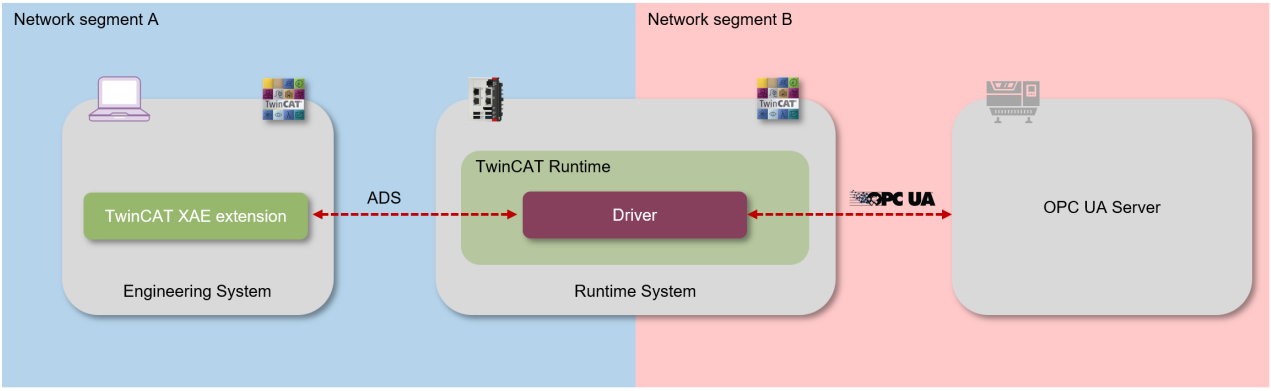
The following diagram illustrates this relationship once again.



This procedure reduces the complexity of certificate management, as there is only one certificate store to be managed - that of the driver. In addition, network environments are also supported in which the OPC UA server can be reached from the runtime system but not from the engineering system. This is typically the case in operating environments in which the controller splits different network segments onto dedicated network interface cards, e.g. a "Machine" segment and a "Maintenance" segment - the latter for connecting an engineering laptop in the event of maintenance. This relationship is illustrated once again in the following diagram.



The following diagram illustrates this relationship once again in a slightly different form.



4.5 Data types

Basic data types according to IEC61131

For the reading and writing of data, the data type of the OPC UA Node must be assigned to the TwinCAT data type (mapping). The assignment of basic data types is described in the standardized information model "PLCopen OPC UA Information Model for IEC 61131-3" and is listed below. You can apply this mapping both to the PLCopen function blocks and to the TwinCAT OPC UA I/O Client.

PLC data type	OPC UA data type
BOOL	Boolean
SINT	SByte
USINT	Byte
INT	Int16
DINT	Int32
STRING	String
BYTE	USint
REAL	Float
LREAL	Double
UINT	UInt16
UDINT	UInt32
LINT	Int64
ULINT	UInt64
DT	DateTime
TIME	Int64
LTIME	Int64

4.6 Application directories

This application uses various directories to store relevant information, such as configuration or certificate files.

Client version 1.x.x

Installation directory

The base installation directory of the application is relative to the TwinCAT installation directory.

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

The application is then installed in the following directory below this directory:

```
%TcInstallDir%\Functions\TF6100-OPC-UA\Win32\Client
```

The Engineering Extension for the TwinCAT OPC UA I/O Client is installed in the following directory:

```
%TcInstallDir%\Functions\TF6100-OPC-UA\Build_4026
```

The driver for the TwinCAT OPC UA I/O Client is installed in the following directory:

```
%TcInstallDir%\3.1\Driver\Autoload
```

Certificate directory

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

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

Log files

Log files [▶ 58] are stored in the following directory:

```
%ProgramData%\Beckhoff\TF6100-OPC-UA\TcOpcUaClient\Logs
```

Client version 2.x.x

The software architecture [▶ 26] of client version 2.x. is fundamentally different from that of the previous version. This results in different application directories.

Installation directory

The base installation directory of the application is relative to the TwinCAT installation directory.

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

The Engineering Extension for the TwinCAT OPC UA I/O Client is installed in the following directory:

```
%TcInstallDir%\Functions\TF6100-OPC-UA\Build_4026
```

The driver for the TwinCAT OPC UA I/O Client is installed in the following directory:

```
%TcInstallDir%\3.1\Driver\Autoload
```

Certificate directory

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

```
%TcInstallDir%\3.1\Target\OpcUa\Client\PKI
```

Log files

The handling of log files [▶ 58] is explained in a separate article.

4.7 Reading variables

Variable values can be read from an OPC UA server via the TwinCAT OPC UA I/O Client. The values can be sampled using various mechanisms, which are described in more detail below.

The various settings in this context are made in the configuration pages of the I/O device. The **Process Data Configuration** area shows various parameters for setting the different modes of data collection from a server.

The TwinCAT OPC UA I/O Client offers three different modes for data collection:

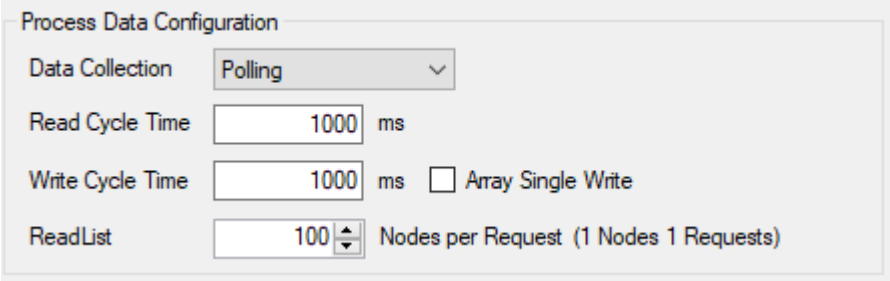
- Polling (cyclic reading/writing)
- Subscriptions
- OnTrigger

Polling (cyclic reading/writing)

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.

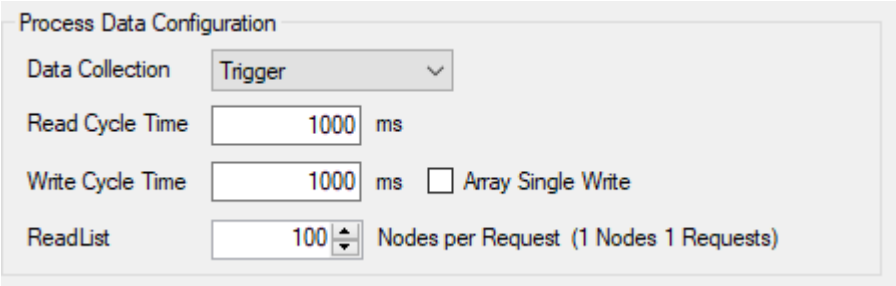


Parameter	Description
Read Cycle Time	Specifies how fast variables are cyclically read.
Write Cycle Time	Defines how often a write command is triggered on the OPC UA channel. If a variable value changes several times within a specific cycle time, only the last value is written to the OPC UA channel. 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.

OnTrigger

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.

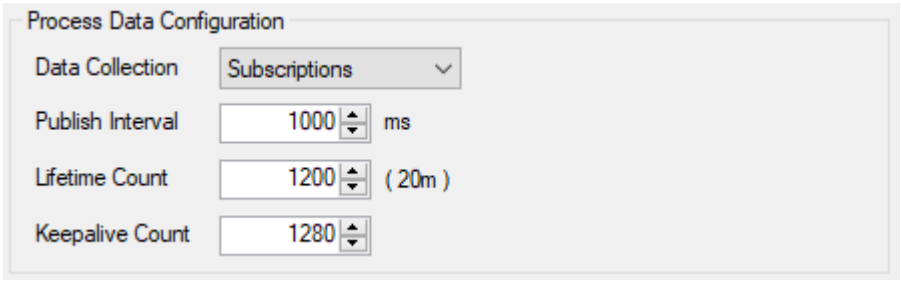
When writing, on the other hand, a value is written in each cycle if the trigger variable is set. No change in value is considered here.



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).



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.

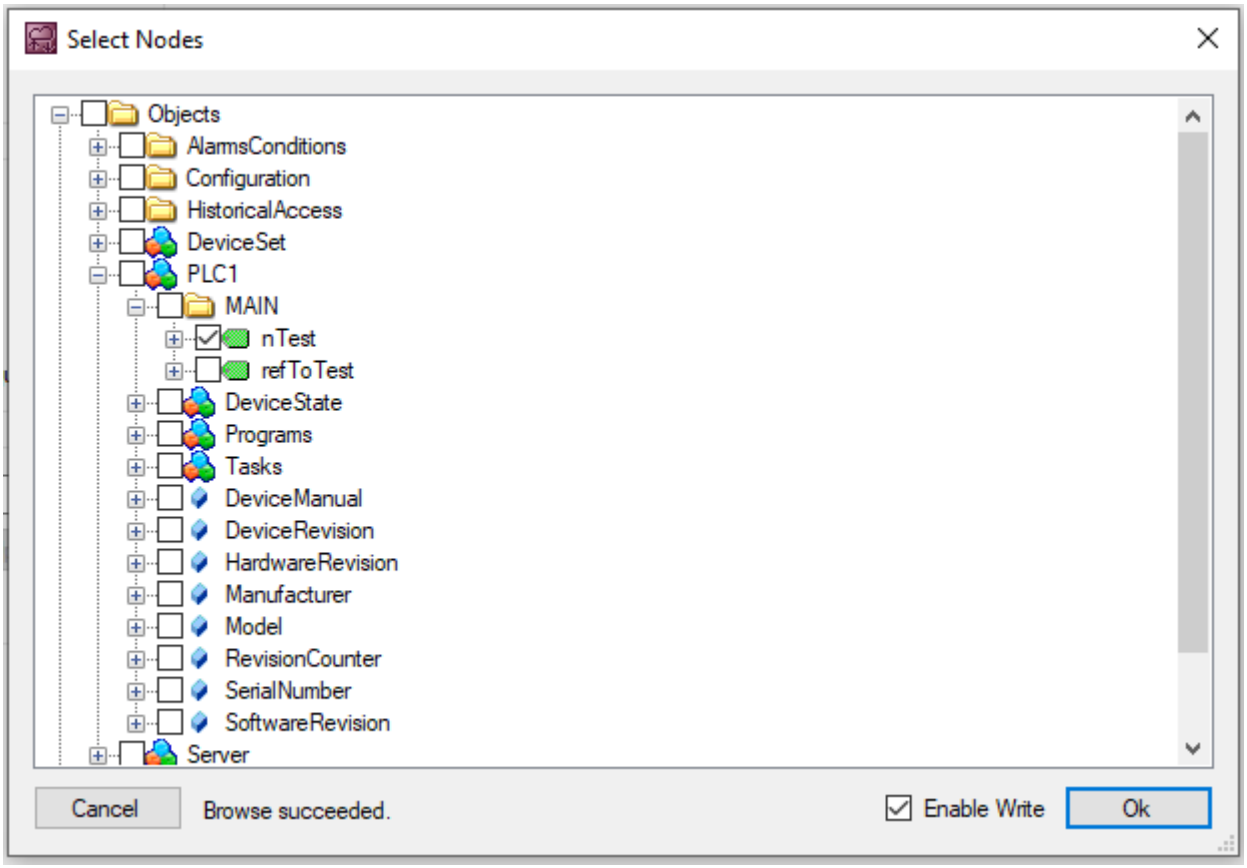
4.8 Writing variables

To enable the writing of variables, several conditions must be met:

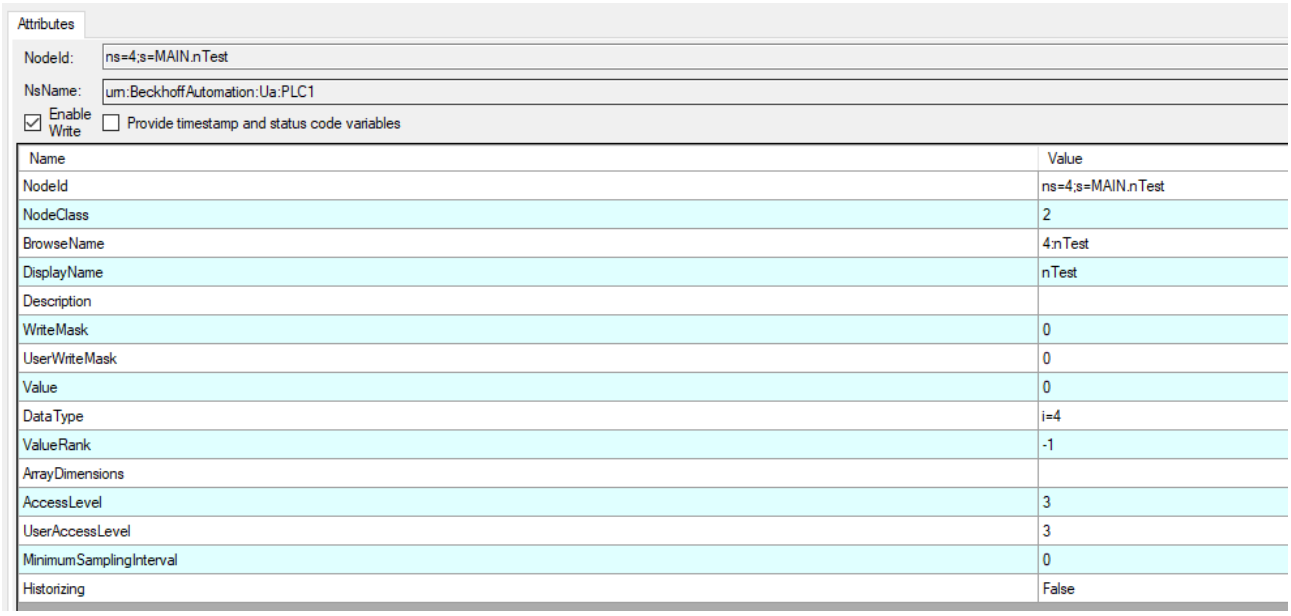
1. The flag "Enable Write" must be set for the variable. This can be done either during the adding process via the button **Add Nodes** or afterwards in the parameter settings of the variable.
2. Before a write command, the "Write Enable" output for the I/O client must be enabled globally. Only then 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) element 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:



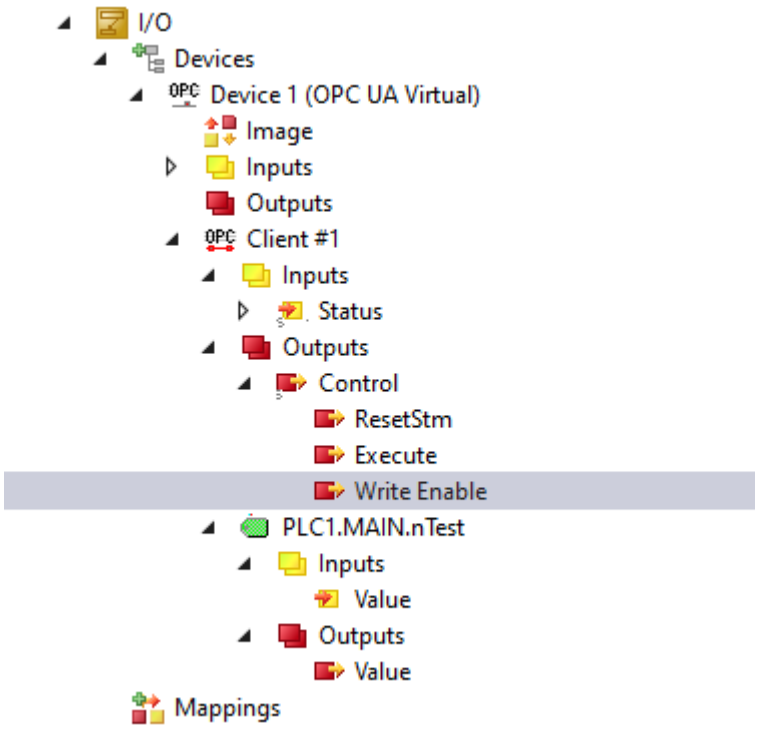
Alternatively, this setting can be enabled/disabled at a later stage via the configuration parameters of the variables in the process image.



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

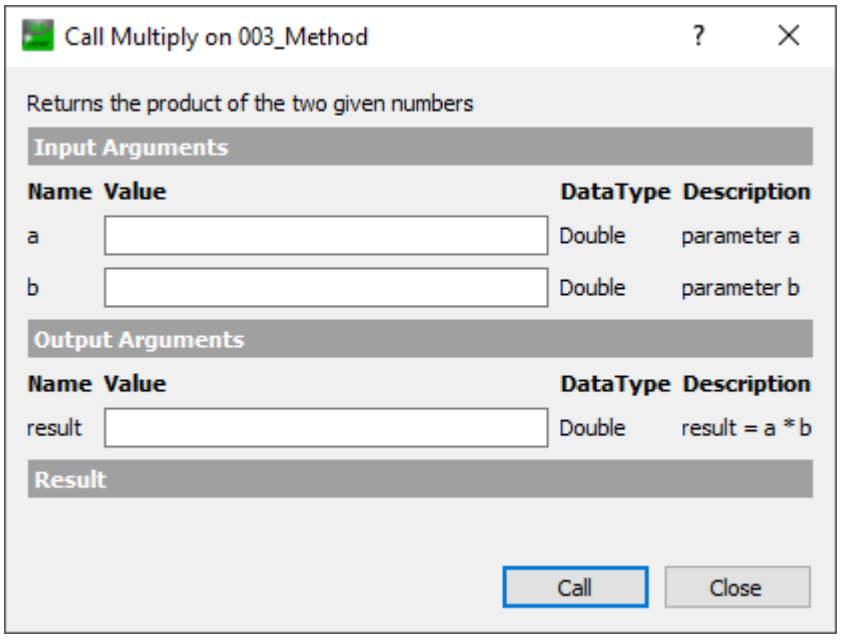
Before write commands can be sent, they must be enabled globally. This is done by setting the output variable "Write Enable" for the I/O client:



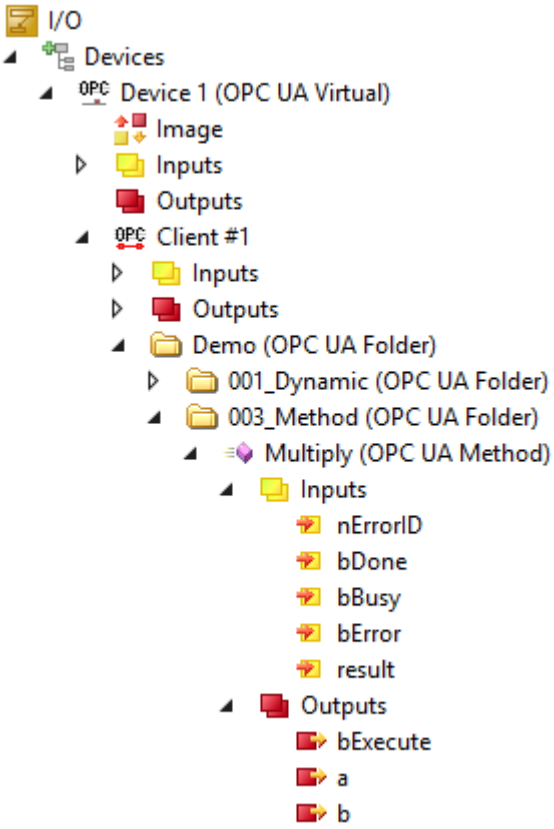
4.9 Method calls

The TwinCAT OPC UA I/O Client supports the calling 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.

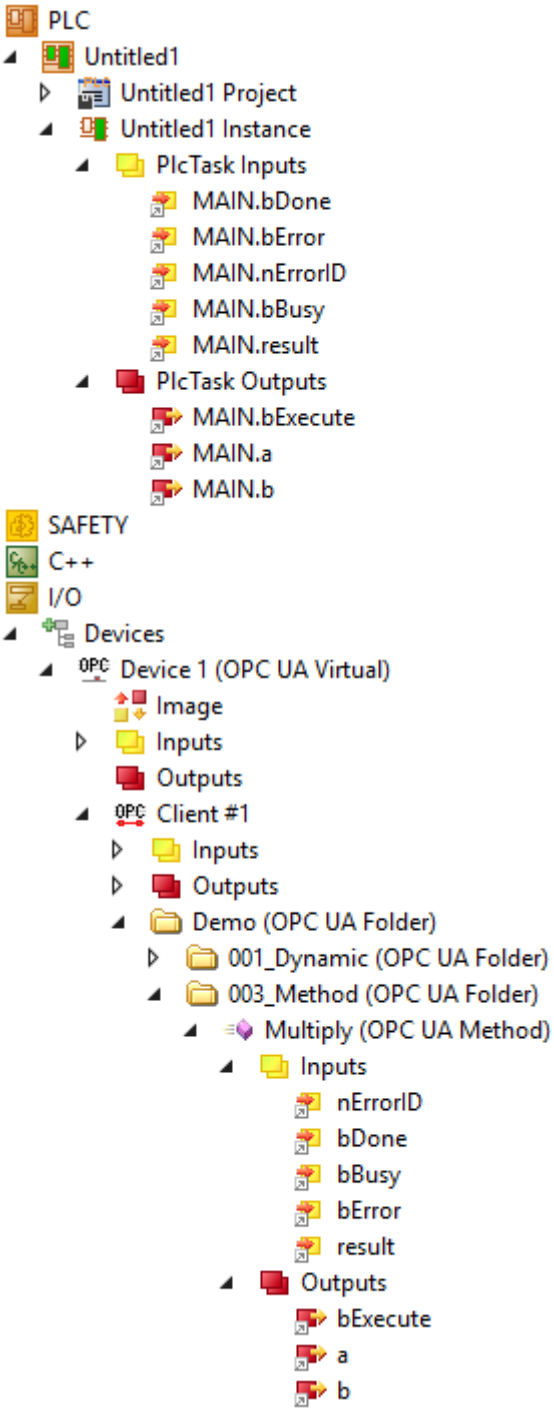
Example: Method on server



Example: Method after addition to the process image



You can then create a mapping between the input/output variables and the PLC variables.



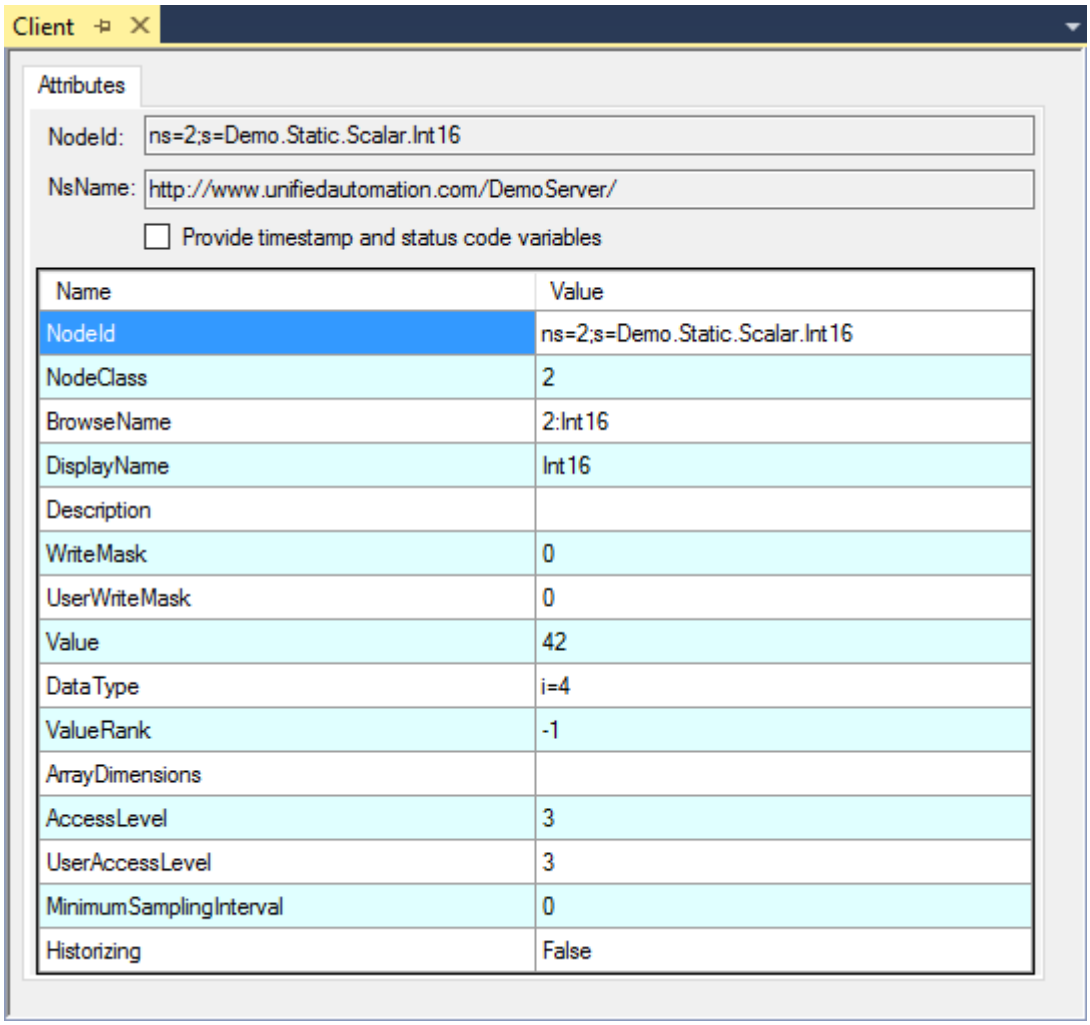
Calling of a method

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.

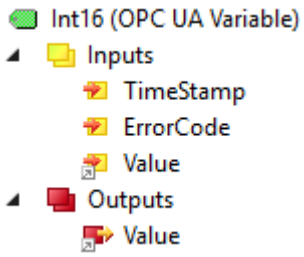
a	LREAL	3
b	LREAL	42
result	LREAL	126
bExecute	BOOL	TRUE
nErrorID	DINT	0
bDone	BOOL	TRUE
bError	BOOL	FALSE
bBusy	BOOL	FALSE

4.10 Timestamp and StatusCode

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.



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**.



4.11 Structures

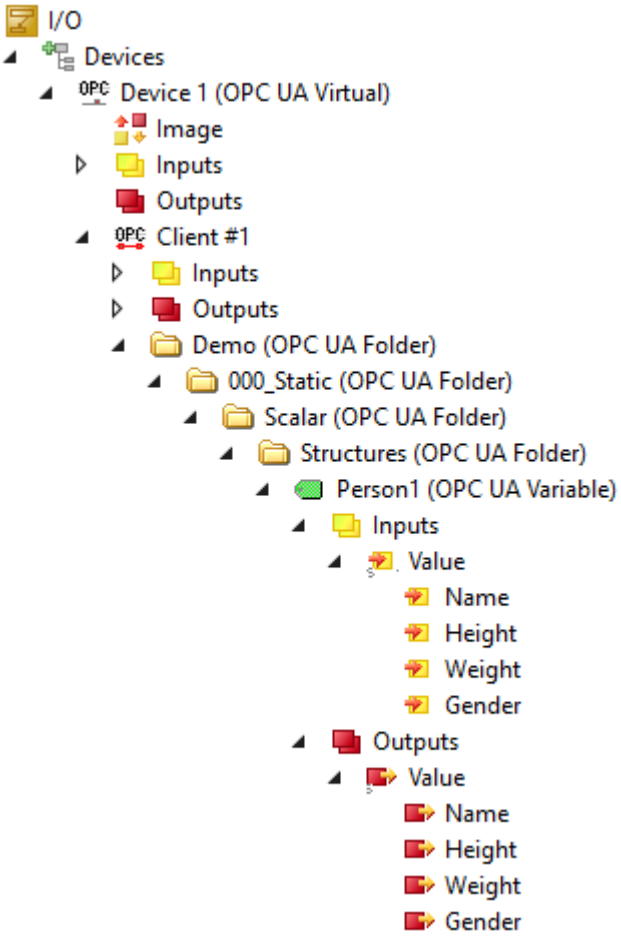
The OPC UA I/O Client supports read/write procedures with structured data types (StructuredTypes). You can also add StructuredTypes to the process image like any other variable. When adding a StructuredType to the process image, the type to be parsed is added to the TwinCAT type system so that, for example, 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).

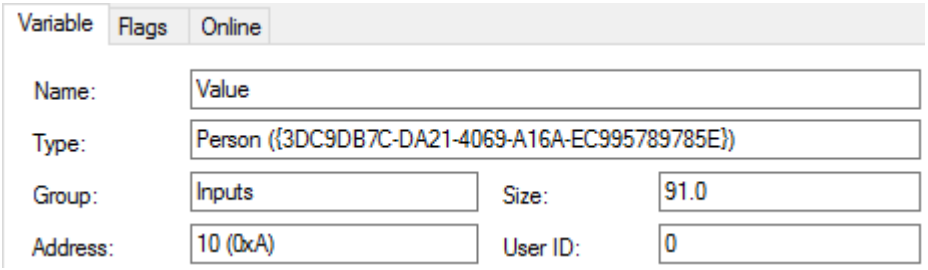
Example: StructuredTypes in the process image



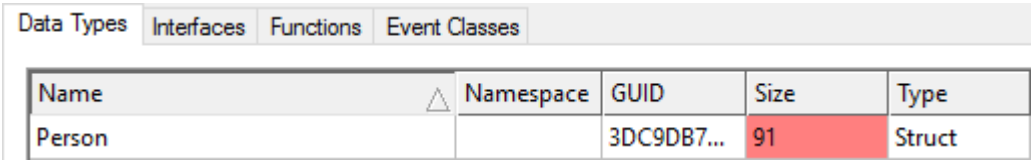
After you have added a node to the process image, the process image contains the node and also the structural information of the type, e.g. whether individual member variables of the node should be read or written.

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.



You can also view the data type in the TwinCAT type system under **SYSTEM > Type System**.

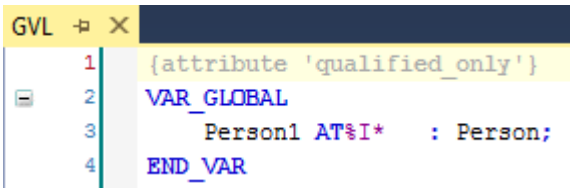


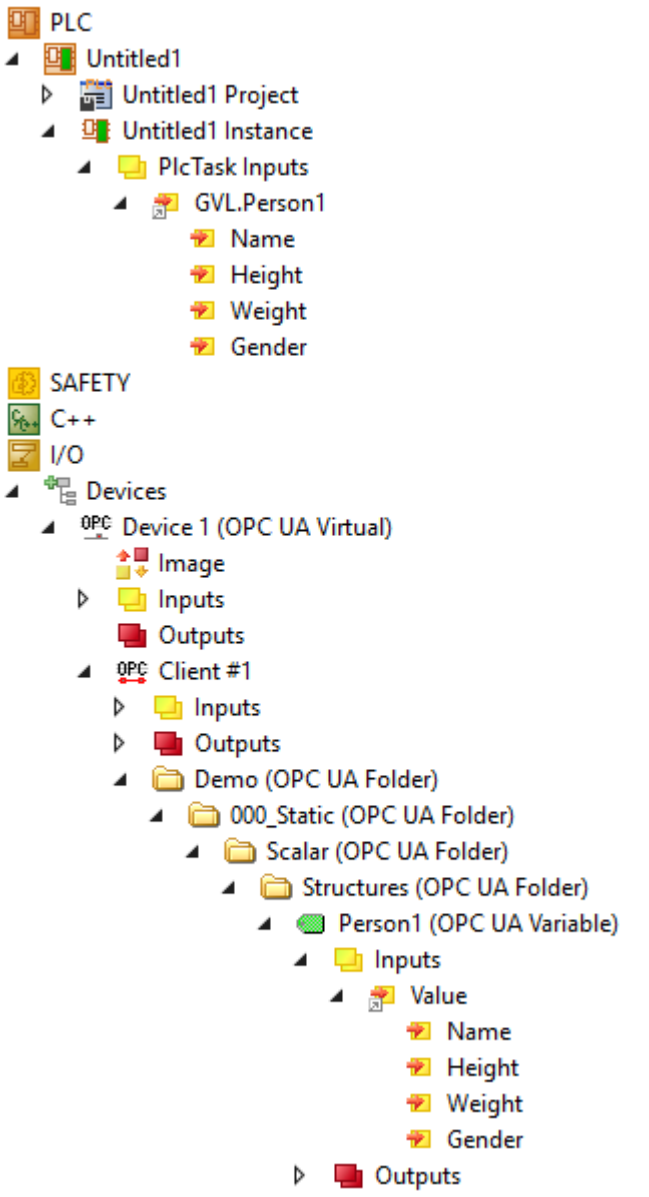
To distinguish the data type from other data types you can add a prefix in the settings of the OPC UA Client.



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.





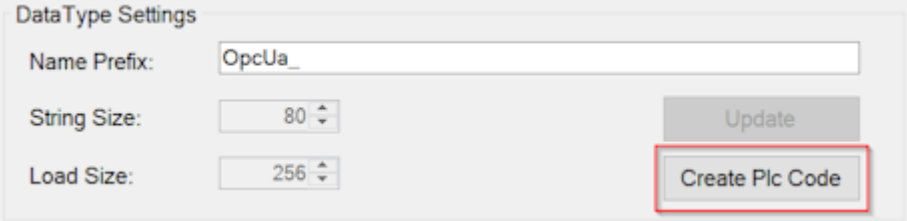
MAIN [Online]

TwinCAT_Project5.Untitled1.MAIN

Expression	Type	Value
Person1	OpcUa_Person	
Name	STRING	'John Wayne'
Height	UINT	193
Weight	REAL	77
Gender	OPCUA_GENDER	Male

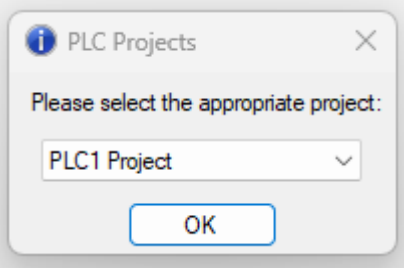
4.12 Code generation

With the help of the automatic code generation, PLC variables for the OPC UA nodes configured in the I/O process image can be generated quickly and easily and linked to them automatically. This function is available in the configuration dialog of the I/O client via the button **Create Plc Code**. This function requires an existing PLC project in the current solution.

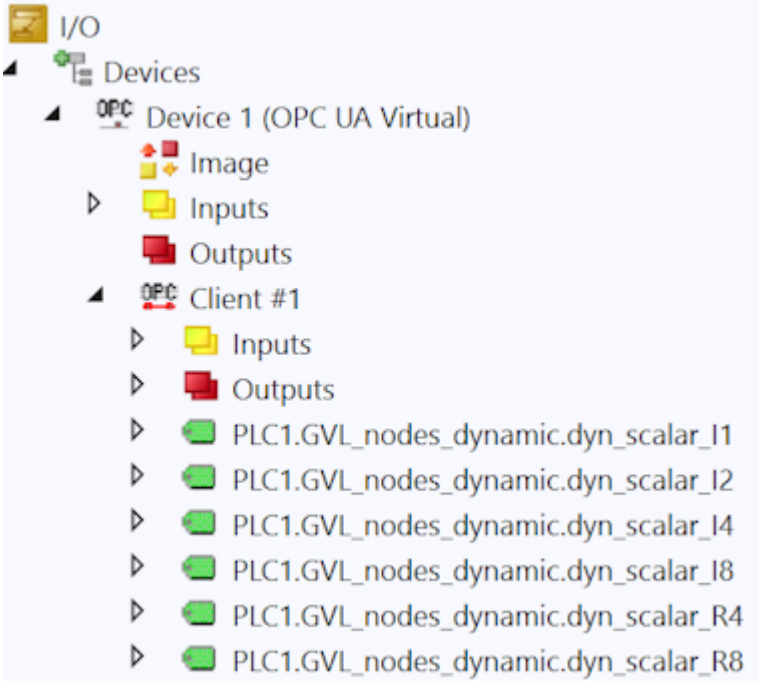


After calling this function, a new **G**lobal **V**ariable **L**ist (GVL) with the name of the I/O client is created in the PLC project. Subsequently, all OPC UA nodes are read and corresponding variables are created in the GVL. Each variable receives the TcLinkTo attribute for an automatic linking with the corresponding variable from the I/O process image.

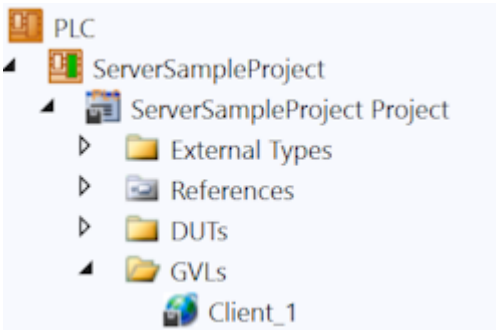
If several PLC projects are available, a selection dialog appears. In this selection dialog, you can specify which PLC project the GVL should be integrated into.



Example: In the I/O part of TwinCAT XAE a TwinCAT OPC UA I/O client with the name "Client 1" was created, to which various OPC UA nodes from a server were added.



After calling the code generation, a new GVL with the name "Client_1" was now created in the (already existing) PLC project. This then contains corresponding PLC variables for the individual nodes, which were then automatically linked via the TcLinkTo attribute.




```
VAR_GLOBAL

    {attribute 'TcLinkTo' := 'TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_nodes_dynamic.dyn_scalar_I1^Inputs^Value'}
    PLC1_GVL_nodes_dynamic_dyn_scalar_I1_Client_1_Device_1 AT %I* : SINT;

    {attribute 'TcLinkTo' := 'TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_nodes_dynamic.dyn_scalar_I2^Inputs^Value'}
    PLC1_GVL_nodes_dynamic_dyn_scalar_I2_Client_1_Device_1 AT %I* : INT;

    {attribute 'TcLinkTo' := 'TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_nodes_dynamic.dyn_scalar_I4^Inputs^Value'}
    PLC1_GVL_nodes_dynamic_dyn_scalar_I4_Client_1_Device_1 AT %I* : DINT;

    {attribute 'TcLinkTo' := 'TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_nodes_dynamic.dyn_scalar_I8^Inputs^Value'}
    PLC1_GVL_nodes_dynamic_dyn_scalar_I8_Client_1_Device_1 AT %I* : LINT;

    {attribute 'TcLinkTo' := 'TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_nodes_dynamic.dyn_scalar_R4^Inputs^Value'}
    PLC1_GVL_nodes_dynamic_dyn_scalar_R4_Client_1_Device_1 AT %I* : REAL;

    {attribute 'TcLinkTo' := 'TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_nodes_dynamic.dyn_scalar_R8^Inputs^Value'}
    PLC1_GVL_nodes_dynamic_dyn_scalar_R8_Client_1_Device_1 AT %I* : LREAL;

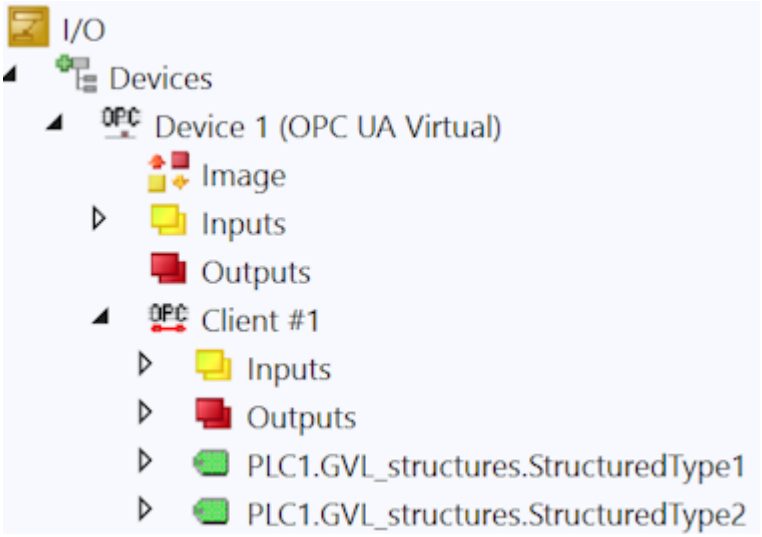
END_VAR
```

In addition, the control variables from the respective I/O client are also created and linked as variables in the GVL (not shown in the screenshot above).

Code generation for structures

An OPC UA node, which represents a so-called StructuredDataType, is also considered by the code generation and a corresponding variable is created in the GVL. Since the StructuredDataType was created as a native data type in the TwinCAT type system, it can be handled like a normal structure.

Example: Two StructuredDataTypes from a server were added to the process image of the I/O client. The data types of the StructuredDataTypes on the server are ST_Complex1 and ST_Complex2 (not visible in the screenshot below).



The code generation has now created a corresponding GVL with two variables from the respective automatically generated TwinCAT data type, which corresponds to the respective StructuredDataType.

```
VAR_GLOBAL

    {attribute 'TcLinkTo' := 'TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_structures.StructuredType1^Inputs^Value'}
    PLC1_GVL_structures_StructuredType1_Client_1_Device_1 AT %I* : OpcUa_ST_Complex_1;

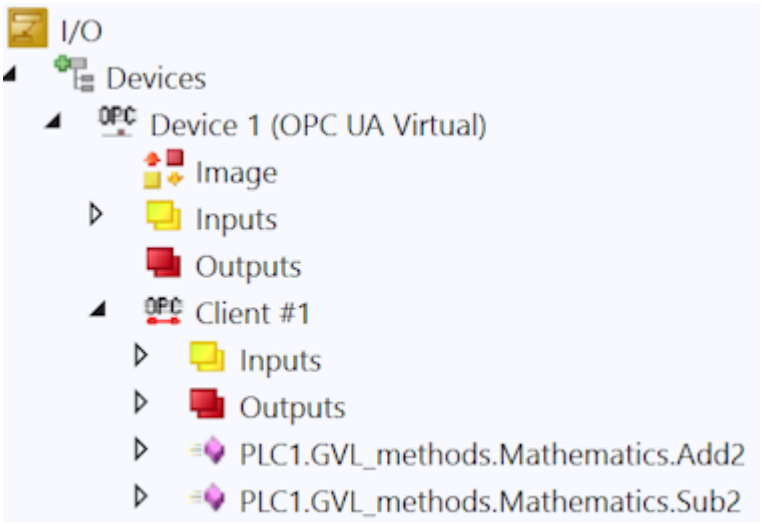
    {attribute 'TcLinkTo' := 'TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_structures.StructuredType2^Inputs^Value'}
    PLC1_GVL_structures_StructuredType2_Client_1_Device_1 AT %I* : OpcUa_ST_Complex_2;

END_VAR
```

Code generation for methods

An OPC UA method has input and output parameters which are passed to or returned from the method accordingly. Furthermore, a method is a self-contained call; it must be explicitly started by the client. This methodology is mapped accordingly in the process image of the I/O client at the method and is also taken into account during code generation. For a method, unlike normal variables or structures, a separate function block is created, which is then referenced in the GVL.

Example: Two methods from a server have been added to the process image of the I/O client.



The code generation has now created a corresponding GVL, as well as two function blocks, which represent the input/output and control variables of the respective method.

```
VAR_GLOBAL

    {attribute 'TcLinkTo' := '.nErrorID:=TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_methods.Mathematics.Add2^Inputs^nErrorID;
                              .bDone:=TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_methods.Mathematics.Add2^Inputs^bDone;
                              .bBusy:=TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_methods.Mathematics.Add2^Inputs^bBusy;
                              .bError:=TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_methods.Mathematics.Add2^Inputs^bError;
                              .ReturnValue:=TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_methods.Mathematics.Add2^Inputs^ReturnValue;
                              .bExecute:=TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_methods.Mathematics.Add2^Outputs^bExecute;
                              .a:=TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_methods.Mathematics.Add2^Outputs^a;
                              .b:=TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_methods.Mathematics.Add2^Outputs^b'}
    PLC1_GVL_methods_Mathematics_Add2_Client_1_Device_1 : FB_PLC1_GVL_methods_Mathematics_Add2_Client_1_Device_1;

    {attribute 'TcLinkTo' := '.nErrorID:=TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_methods.Mathematics.Sub2^Inputs^nErrorID;
                              .bDone:=TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_methods.Mathematics.Sub2^Inputs^bDone;
                              .bBusy:=TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_methods.Mathematics.Sub2^Inputs^bBusy;
                              .bError:=TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_methods.Mathematics.Sub2^Inputs^bError;
                              .ReturnValue:=TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_methods.Mathematics.Sub2^Inputs^ReturnValue;
                              .bExecute:=TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_methods.Mathematics.Sub2^Outputs^bExecute;
                              .a:=TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_methods.Mathematics.Sub2^Outputs^a;
                              .b:=TIID^Device 1 (OPC UA Virtual)^Client #1^PLC1.GVL_methods.Mathematics.Sub2^Outputs^b'}
    PLC1_GVL_methods_Mathematics_Sub2_Client_1_Device_1 : FB_PLC1_GVL_methods_Mathematics_Sub2_Client_1_Device_1;

END_VAR
```

```
FUNCTION_BLOCK FB_PLC1_GVL_methods_Mathematics_Add2_Client_1_Device_1
VAR_INPUT
    bExecute AT %Q* : BOOL;
    a AT %Q* : LREAL;
    b AT %Q* : LREAL;
END_VAR
VAR_OUTPUT
    nErrorID AT %I* : DINT;
    bDone AT %I* : BOOL;
    bBusy AT %I* : BOOL;
    bError AT %I* : BOOL;
    ReturnValue AT %I* : LREAL;
END_VAR
```

```
FUNCTION_BLOCK FB_PLC1_GVL_methods_Mathematics_Sub2_Client_1_Device_1
VAR_INPUT
    bExecute AT %Q* : BOOL;
    a AT %Q* : LREAL;
    b AT %Q* : LREAL;
END_VAR
VAR_OUTPUT
    nErrorID AT %I* : DINT;
    bDone AT %I* : BOOL;
    bBusy AT %I* : BOOL;
    bError AT %I* : BOOL;
    ReturnValue AT %I* : LREAL;
END_VAR
```

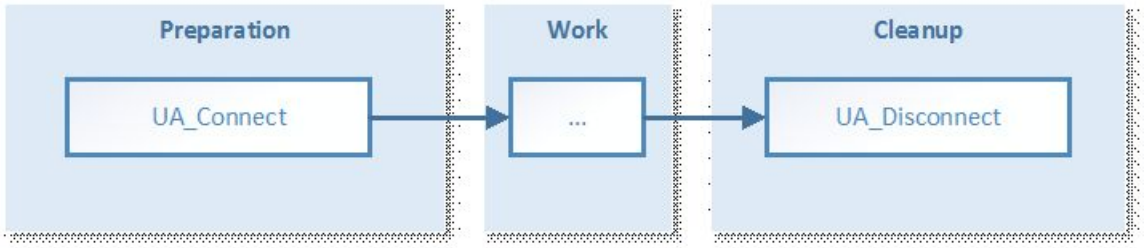
4.13 PLCopen function blocks

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, there is a TwinCAT I/O device, which offers a simple, mapping-based interface. On the other hand, PLCopen provides standardized function blocks that can be used to initiate a connection with an OPC UA server directly from the PLC logic. The handling of these function blocks is described in more detail below. This article consists of the following sections:

- Workflow
- Determination of the communication parameters
- Establishing a connection
- Reading variables
- Writing variables
- Calling methods

Workflow

The general workflow when using the PLCopen function blocks can be schematically represented as follows:



In the preparation phase, the communication parameters are set up and a connection to the server is established. The desired function is then executed (read, write, method calls), followed by disconnection of the communication connection.

Determination of the 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.:

- NodeID
- NamespaceIndex and corresponding NamespaceURI
- DataType
- MethodNodeID and ObjectNodeID

The following documentation uses the generic OPC UA Client UA Expert as an example. This client can be purchased via the Unified Automation web pages: www.unified-automation.com.

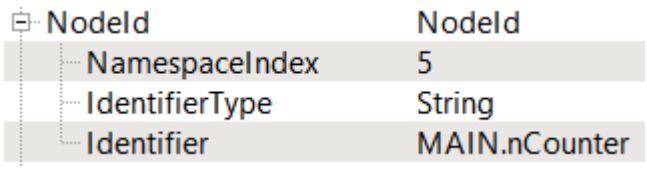
Nodes are characterized by the following three attributes, which form the so-called NodeID:

- 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

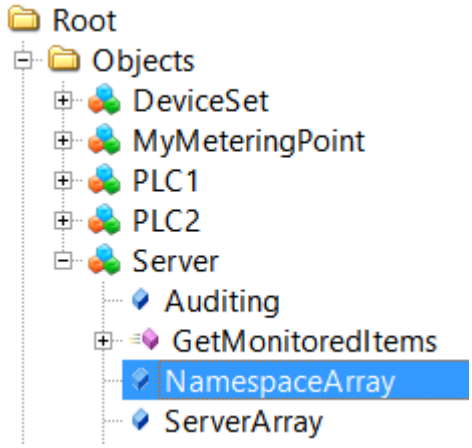
These attributes represent the so-called *NodeID* - the representation of a node on an OPC UA server - and are required by many subsequent function blocks.

With the help of the UA Expert software 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, e.g.

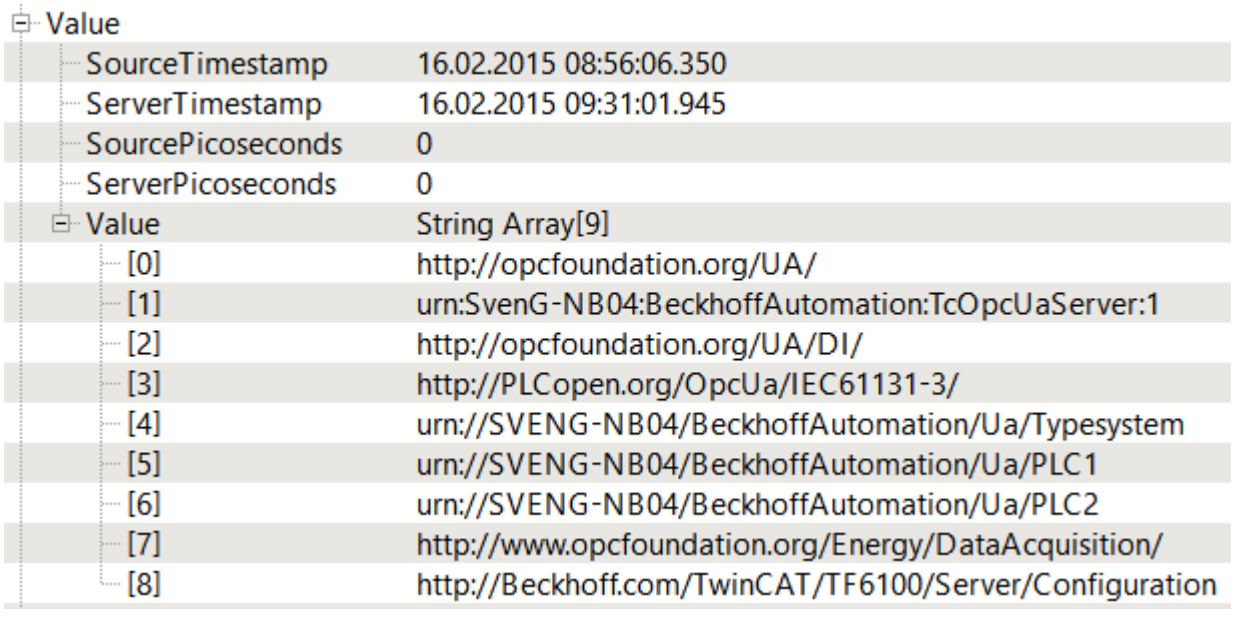


According to the OPC UA specification, the NamespaceIndex can be a dynamically generated value. Therefore, OPC UA Clients must always use the corresponding namespace URI to resolve the NamespaceIndex before a node handle is detected.

Use the function block UA_GetNamespaceIndex [▶ 85] 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 namespace URIs are visible in the Attributes panel, for example:



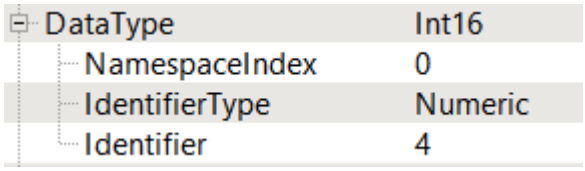
The section above shows an example of a NodeID in which the namespace index is 5. According to the NamespaceArray shown in the figure, the corresponding NamespaceURI is urn*://SVENG-NB04/BeckhoffAutomation/Ua/PLC1*. This URI can now be used for the function block UA_GetNamespaceIndex. The OPC UA Server ensures that the URI always remains the same, even after a restart.

Observe the correct NamespaceIndex

As the NamespaceIndex shown can change, the NamespaceURI should always be used in combination with the function block UA_GetNamespaceIndex for later use with other function blocks, e.g. UA_Read [▶ 96], UA_Write [▶ 99], to resolve the correct NamespaceIndex.

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, for example:



In this case the data type (DataType) is "Int16". This must be assigned to an equivalent data type in the PLC, e.g. "INT".

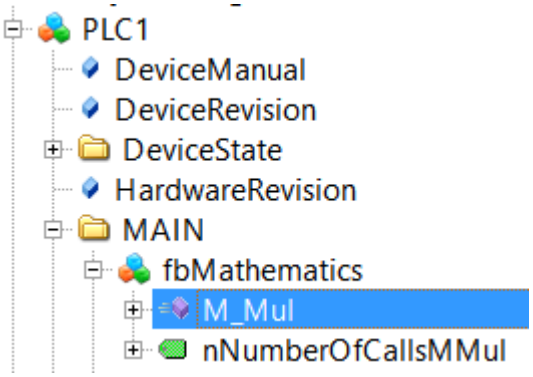
MethodNodeID and ObjectNodeID

When calling methods from the OPC UA namespace, two identifiers are required if the method handle is get using the function block UA_MethodGetHandle [▶ 91]:

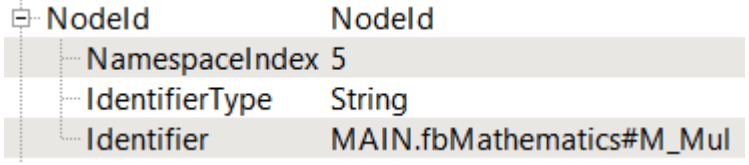
- ObjectNodeID: Identifies the UA object that contains the method.
- MethodNodeID: Identifies the method itself.

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.

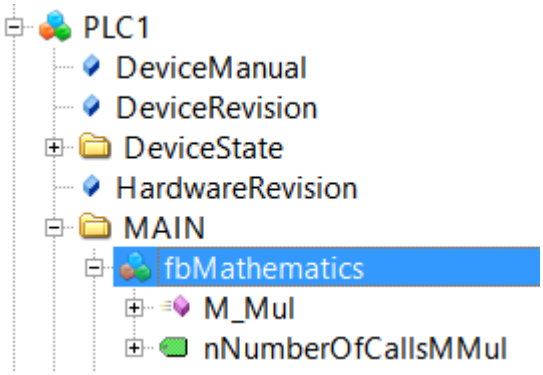
Sample Method M_Mul:



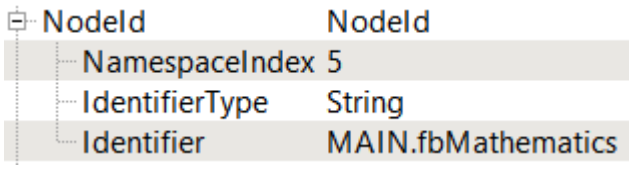
The method identifier is then visible in the Attributes panel.



Sample Object fbMathematics:



The object identifier is then visible in the Attributes panel.



Establishing a connection

The following section describes how you use the function block TcX_PLCopen_OpcUa 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.

The following function blocks are required to establish a connection to an OPC UA server and subsequently interrupt the session: UA_Connect [▶ 82], UA_Disconnect [▶ 84].



First read the section How to determine communication parameters to better understand certain UA functionalities (e.g. how to determine NodeIdentifiers).

The function block UA_Connect 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 host name 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". The Server URL can then look like this: opc.tcp://CX-12345:4840.

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

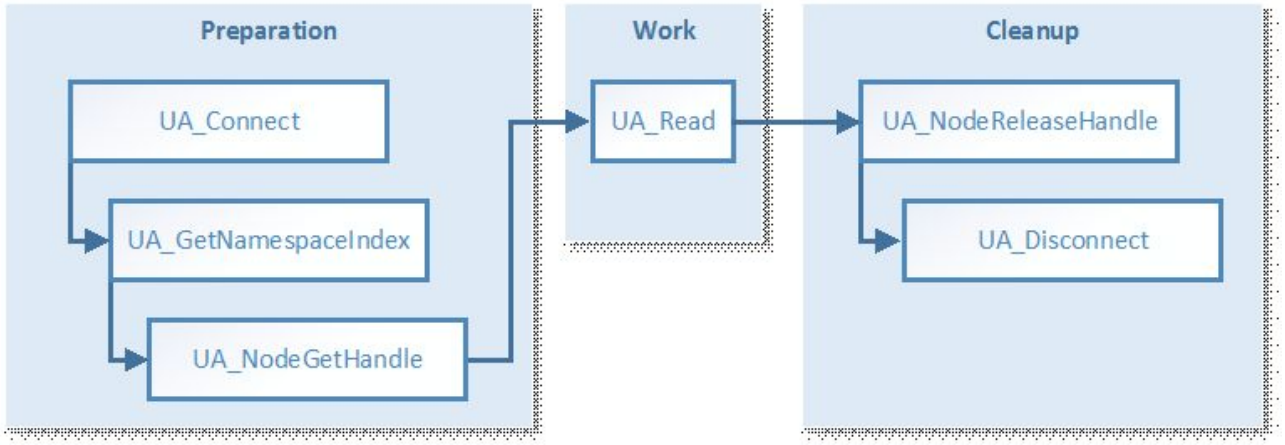
END_CASE
```

Reading variables

The following section describes how to use the function blocks TcX_PLCopen_OpcUa to read an OPC UA node from a local or remote OPC UA server. The following function blocks are required to establish a connection to an OPC UA server, read UA nodes and later interrupt the session: UA_Connect [▶ 82], UA_GetNamespaceIndex [▶ 85], UA_NodeGetHandle [▶ 93], UA_Read [▶ 96], UA_NodeReleaseHandle [▶ 95], UA_Disconnect [▶ 84].

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:



- The function block UA_Connect requires the following information to establish a connection to a local or remote OPC UA Server (see also How to establish a connection):
  - Server URL
  - Session Connect Information
- The function block UA_GetNamespaceIndex requires a Connection Handle (from UA_Connect) and a NamespaceURI for resolution to a NamespaceIndex, which is later used by UA_NodeGetHandle to capture a node handle (see also How to determine communication parameters).
- The function block UA_NodeGetHandle requires a Connection Handle (from UA_Connect) and NodeID (from ST_UANodeID) to capture a node handle (see also How to determine communication parameters).
- The function block UA_Read 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 also How to determine communication parameters).
- The function block UA_NodeReleaseHandle requires a connection handle (from UA_Connect) and a node handle (from UA_NodeGetHandle).

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:
    [...]

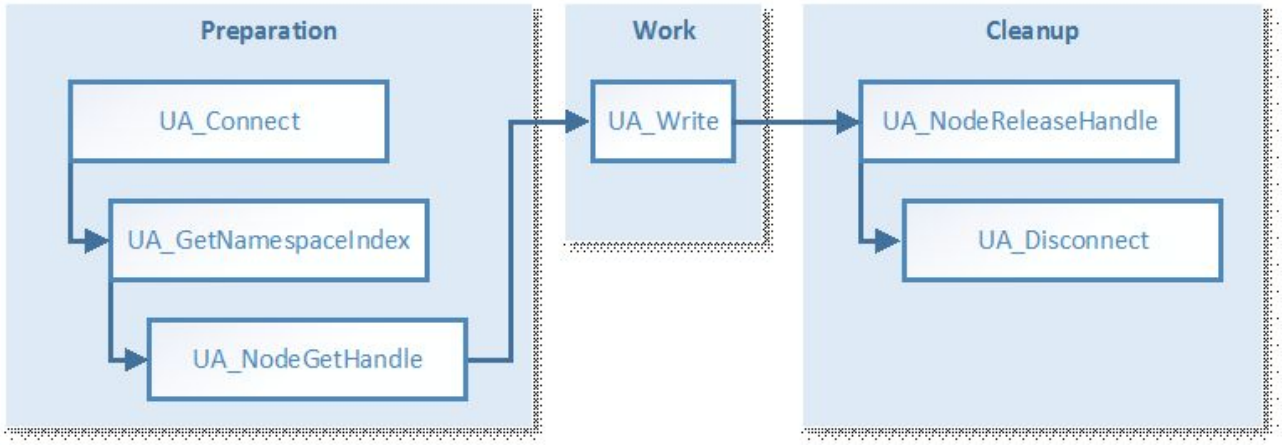
END_CASE
```

Writing variables

The following section describes how you use the function block TcX_PLCopen_OpcUa to write values in an OPC UA node from a local or remote OPC UA server. 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 [▶ 82], UA_GetNamespaceIndex [▶ 85], UA_NodeGetHandle [▶ 93], UA_Write [▶ 99], UA_NodeReleaseHandle [▶ 95], UA_Disconnect [▶ 84].

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:



- The function block UA_Connect requires the following information in order to be able to establish a connection to a local or remote OPC UA server (see also How to establish a connection):
  - Server URL
  - Session Connect Information
- The function block UA_GetNamespaceIndex requires a Connection Handle (from UA_Connect) and a NamespaceURI for resolution to a NamespaceIndex, which is later used by UA_NodeGetHandle to capture a node handle (see also How to determine communication parameters).
- The function block UA_NodeGetHandle requires a Connection Handle (from UA_Connect) and NodeID (from ST_UANodeID) to capture a node handle (see also How to determine communication parameters).
- The function block UA_Write 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 also How to determine communication parameters).
- The function block UA_NodeReleaseHandle requires a connection handle (from UA_Connect) and a node handle (from UA_NodeGetHandle).

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:
    [...]

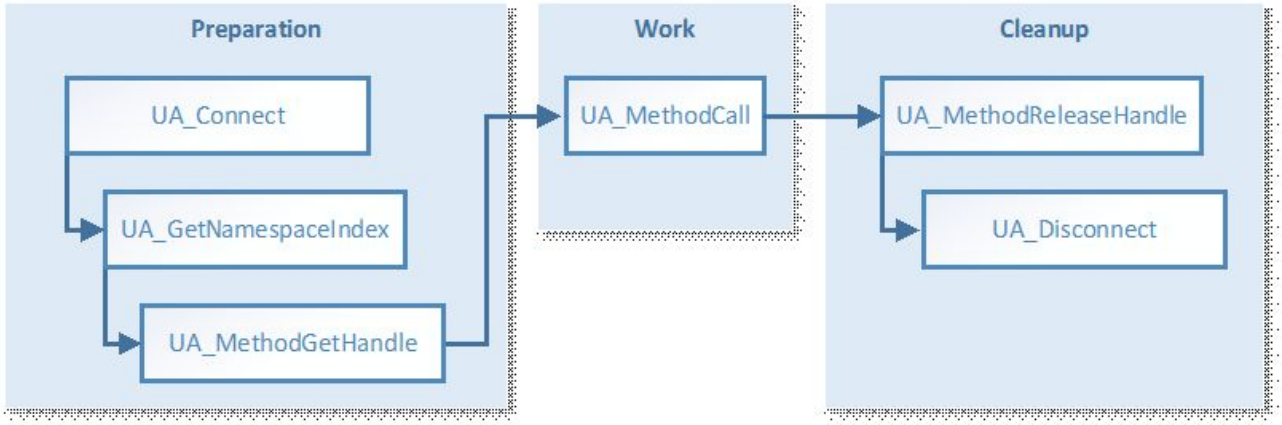
END_CASE
```

Calling methods

The following section describes how you use the function block TcX_PLCopen_OpcUa to call methods on a local or remote OPC UA server. The following function blocks are required to connect to an OPC UA server, call UA methods, and subsequently interrupt the session: UA_Connect [▶ 82], UA_GetNamespaceIndex [▶ 85], UA_MethodGetHandle [▶ 91], UA_MethodCall [▶ 89], UA_MethodReleaseHandle [▶ 92], UA_Disconnect [▶ 84].

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:



- The function block UA_Connect requires the following information in order to be able to establish a connection to a local or remote OPC UA server (see also How to establish a connection):
  - Server URL
  - Session Connect Information
- The function block UA_GetNamespaceIndex requires a Connection Handle (from UA_Connect) and a NamespaceURI for resolution to a NamespaceIndex, which is later used by UA_NodeGetHandle to capture a node handle (see also How to determine communication parameters).
- The function block UA_MethodGetHandle requires a connection handle (from UA_Connect), an ObjectNodeID and a MethodNodeID to capture a method handle (see also How to determine communication parameters).
- The function block UA_MethodCall 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. The following code snippet calculates and creates the pInputArgInfo and pInputArgData parameters in the M_Init method.
- The function block UA_NodeReleaseHandle requires a connection handle (from UA_Connect) and a method handle (from UA_MethodGetHandle).

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
```

4.14 Security

4.14.1 Overview

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

Endpoints

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

Transport layer

A communication connection based on OPC UA can be secured at the transport layer. This is done through the use of client/server certificates and a mutual trust relationship between client and server application. Here, the client must trust the server certificate and vice versa in order for a communication connection to be established. This requires a mutual certificate exchange [▶ 55].

Application layer

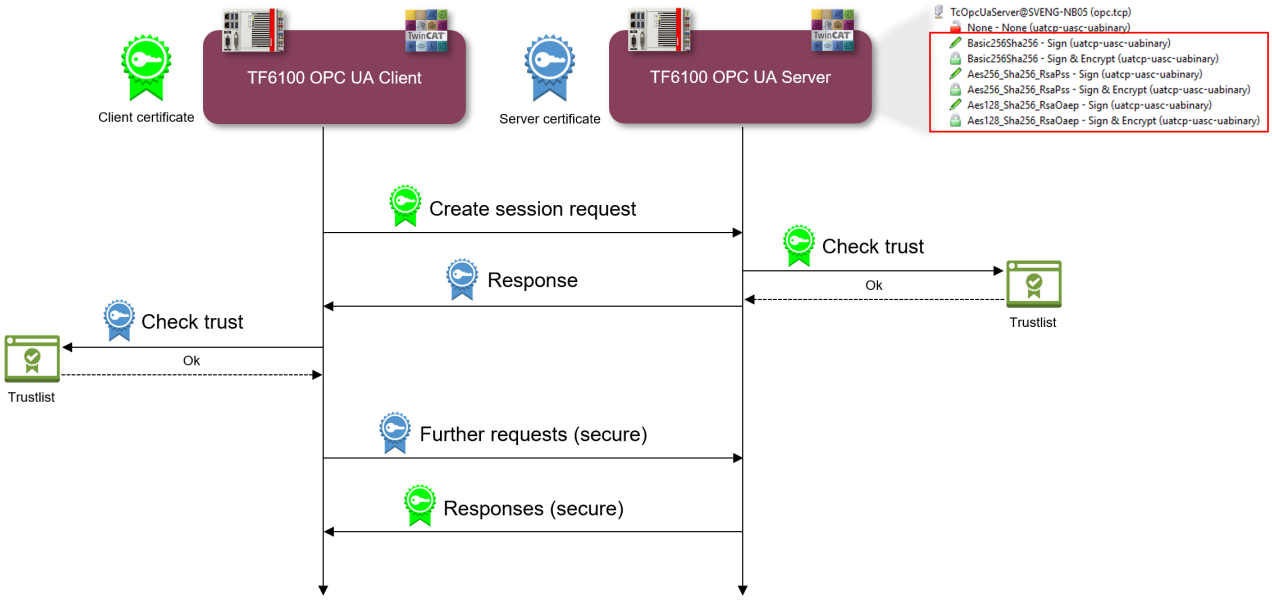
In addition to the transport layer, a communication connection can also be secured at the application layer. For this purpose, various authentication mechanisms [▶ 56] are available, which are offered by the server endpoint.

4.14.2 Certificate exchange

To secure the communication connection at transport layer via a secure endpoint, 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 key pair consisting of a public and a private key when they are started for the first time. However, you can also use any certificate authority or technology for integration into your IT infrastructure, e.g. Active Directory or OpenSSL. For simple administration and secure access to certificates, it makes sense to set up a Global Discovery Server.

To establish a trust relationship between an OPC UA server and the TwinCAT OPC UA Client, you need the public key of the server certificate. The client must trust this accordingly. The client manages the trust settings for server certificates in a subdirectory of the application directory [▶ 29].

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



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

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

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

Configure trust relationship via file system

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

4.14.3 Authentication

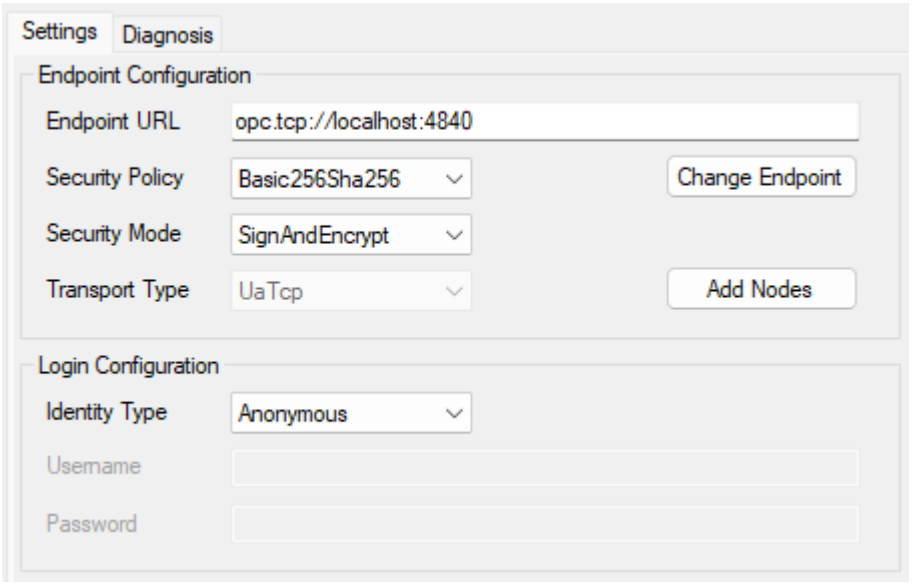
The TwinCAT OPC UA Client supports the use of different IdentityTokens to authenticate to an OPC UA Server.

- Anonymous
- Username/Password
- User certificate

The following screenshots are taken with the TwinCAT OPC UA I/O Client. All of the functions described are also supported when using the PLCOpen function blocks [▶ 43].

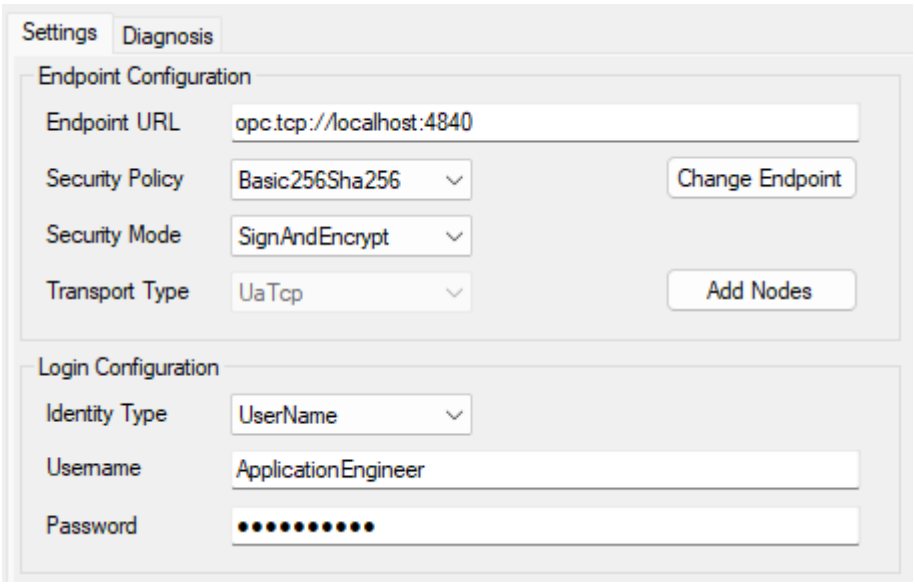
Anonymous

This type of authentication enables the TwinCAT OPC UA Client to establish a connection to the server application without specifying a user identity. Below you will find an example screenshot from the TwinCAT OPC UA I/O Client.



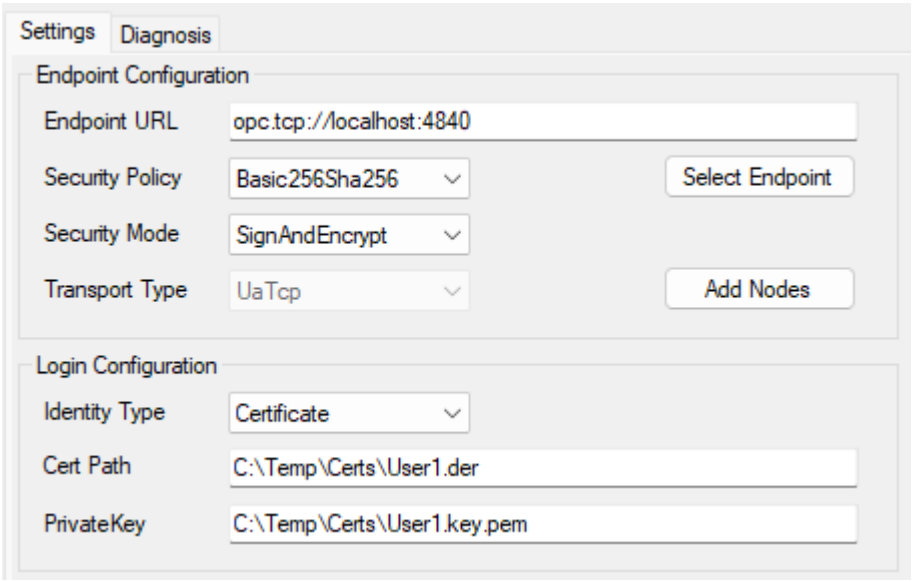
Username/Password

This type of authentication uses a username/password combination to authenticate the TwinCAT OPC UA Client to the server application. Below you will find an example screenshot from the TwinCAT OPC UA I/O Client.



User certificate

With this type of authentication, the TwinCAT OPC UA Client uses a certificate to authenticate itself 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. Below you will find an example screenshot from the TwinCAT OPC UA I/O Client.



NOTICE

Authentication and server certificate

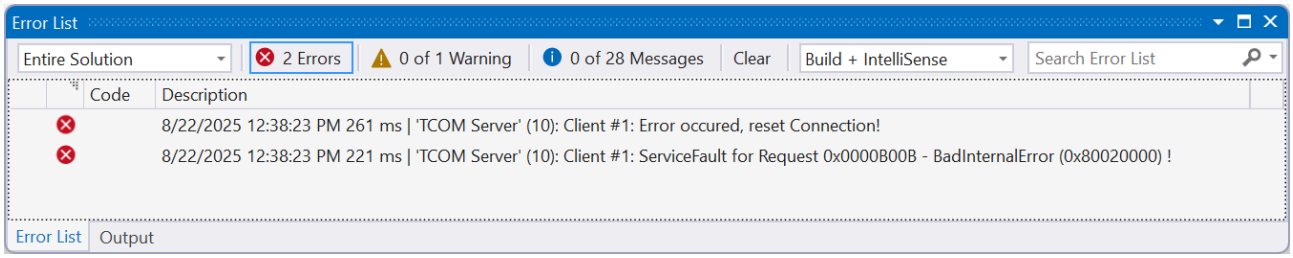
When using the unencrypted endpoint in combination with authentication, the TwinCAT OPC UA Client still requires the public key from the OPC UA Server certificate in order to encrypt the password during transmission. To this end the certificate must be trusted in the TwinCAT OPC UA Client (see certificate exchange [▶ 55]).

4.15 Logging

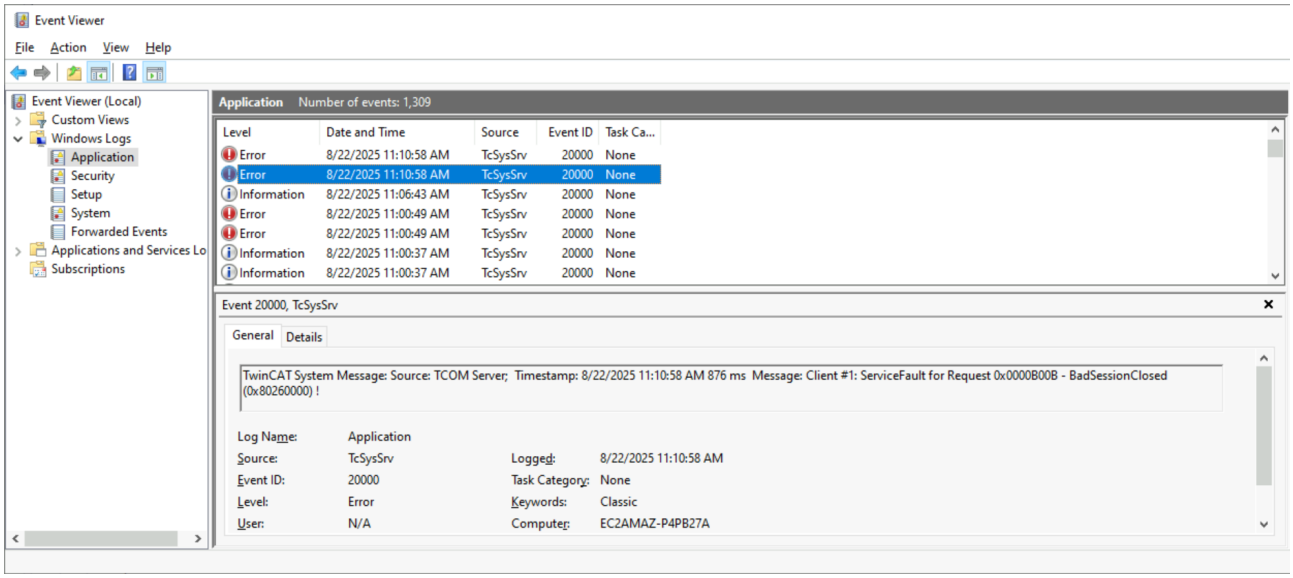
The TwinCAT OPC UA Client offers various options for carrying out error diagnostics.

From client version 2.x.x

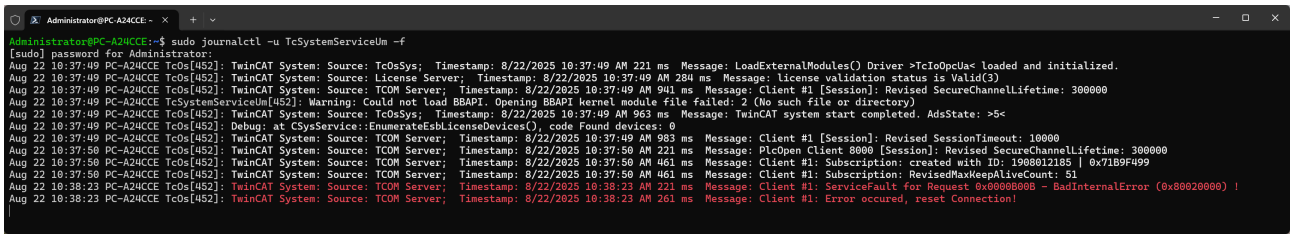
The TwinCAT XAE ErrorLog, which is enabled by default and can be viewed in the TwinCAT XAE Engineering environment, is used to log error states:



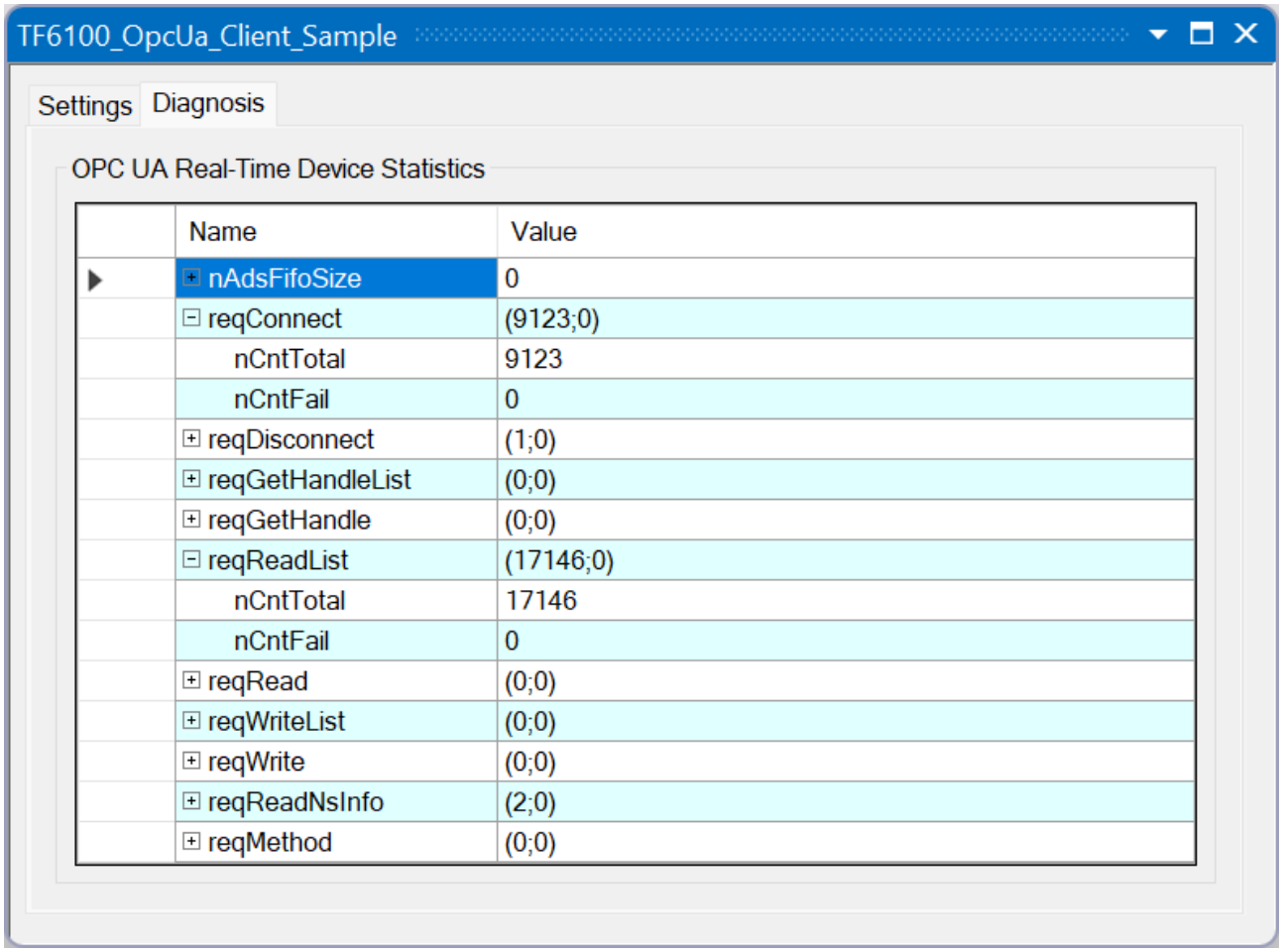
Depending on the operating system used, the ErrorLog can be read out via other mechanisms. Under Microsoft Windows, the Windows Event Viewer is available for this purpose, in which the messages are saved within the "Application Log".



Under Linux®, these messages can be viewed using the "journalctl" command.



In addition to these functions of the TwinCAT XAE ErrorLog, extended diagnostic options are available on the I/O device.



Previous client versions

Both a so-called “StackTrace” and an “AppTrace” can be enabled. The StackTrace provides diagnostic information on the OPC UA stack, whereas the AppTrace is limited to the pure client application and is also sufficient in most cases. All settings listed in the following table are created under the following registry key:

4026: HKLM\SOFTWARE\WOW6432Node\Beckhoff\TwinCAT3 Functions\TF6100 OPC UA Client

4024: HKLM\SOFTWARE\WOW6432Node\Beckhoff\TwinCAT3 Functions\TF6100 OPC UA\Configuration

Type	Name	Data type	Data
AppTrace	logfile	REG_SZ	Name of the logfile
AppTrace	logmode	REG_DWORD	0x00 (0) = disabled 0x02 (2) = enabled 0x12 (18) = enabled logging with additional debug outputs
StackTrace	tracelogfile	REG_SZ	Name of the logfile
StackTrace	tracelogmode	REG_DWORD	0x00 (0) = disabled 0x01 (1) = enabled

After a change, at least one TwinCAT context change must occur (Windows CE: full device restart). This ensures that the changes to logging are enabled.

The log is stored in the following folder: *C:\ProgramData\Beckhoff\TF6100-OPC-UA\TcOpcUaClient\logs.* The file name can be changed via the registry.

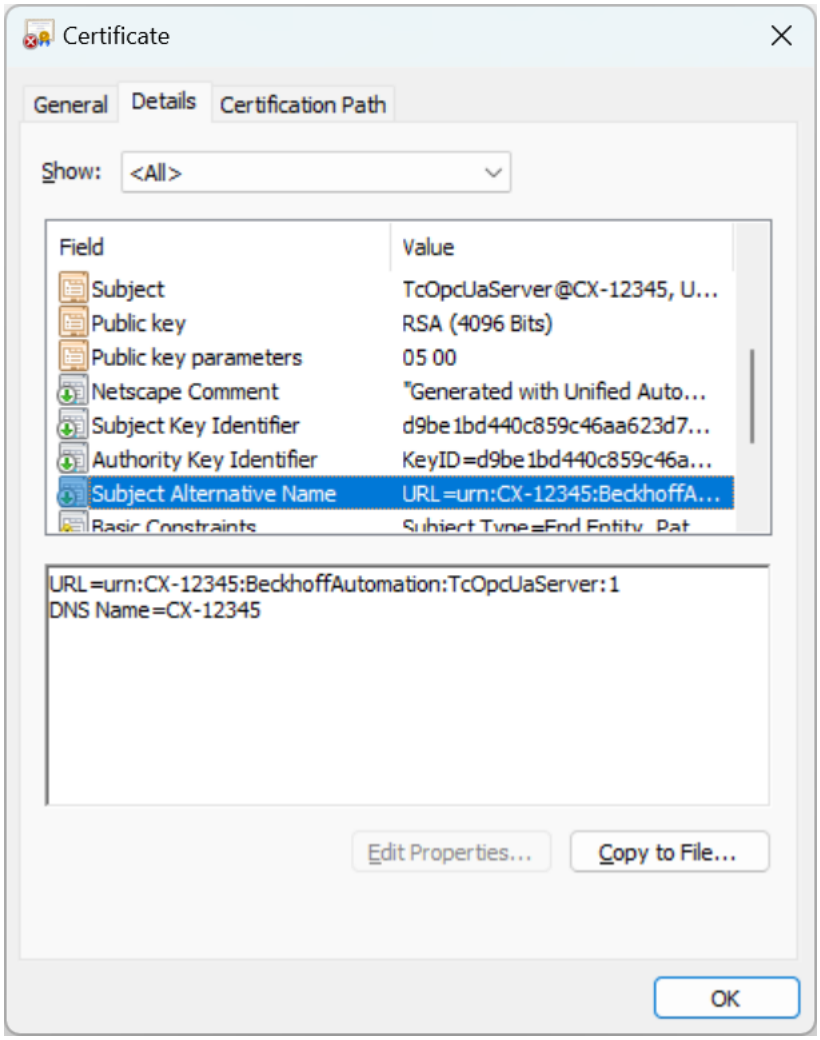
4.16 ApplicationURI

In OPC UA (Open Platform Communications Unified Architecture), the ApplicationURI is a globally unique identifier string that represents one specific OPC UA application instance (client, server, or both). You can think of it as the application’s digital identity inside the OPC UA ecosystem.

The ApplicationURI plays several critical roles: security & certificates, application identification and during discovery. It is usually formatted as a URI, commonly using the “urn:” scheme, for example:

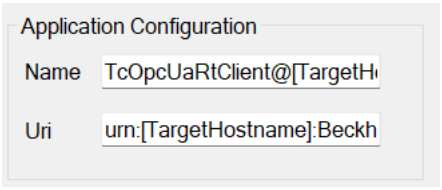
urn:CX-12345:BeckhoffAutomation:TcOpcUaServer:1

The ApplicationURI is added to the SubjectAlternativeName of an OPC UA application certificate, for example:



Setting ApplicationURI for TwinCAT OPC UA Client

When configuring the TwinCAT OPC UA Client, its ApplicationURI can be set on the configuration pages of the client I/O device.



We recommend to use the default settings for the ApplicationURI but if a special use cases requires you to change it, you can do so on this configuration page. After the setting has been changed and the configuration is activated, the driver will determine if an existing application certificate that matches the ApplicationURI can be found on the target device. If a matching application certificate is found, it will be used. If no matching application certificate is found, the driver will create a new (self-signed) application certificate and uses that certificate for communication with the server. This also means that you can use different application certificates for multiple TwinCAT OPC UA Client I/O devices in your project. The default configuration is that one application certificate is used for all devices.

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 [▶ 62].

5.1.1.3 E_OpcUAServerStatus

E_OpcUAServerStatus represents the runtime status of a TwinCAT OPC UA Server.

Syntax

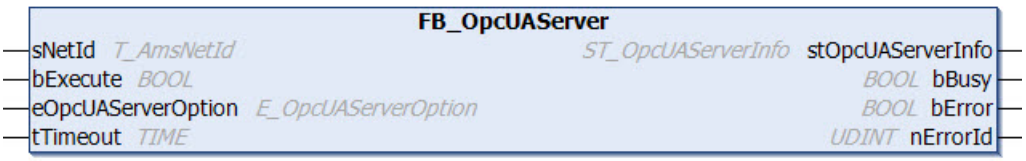
```
TYPE E_OpcUAServerStatus
(
    eOPCUAServerStatus_None,
    eOPCUAServerStatus_Alive,
    eOPCUAServerStatus_NotResponding
);
END_TYPE
```

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
```

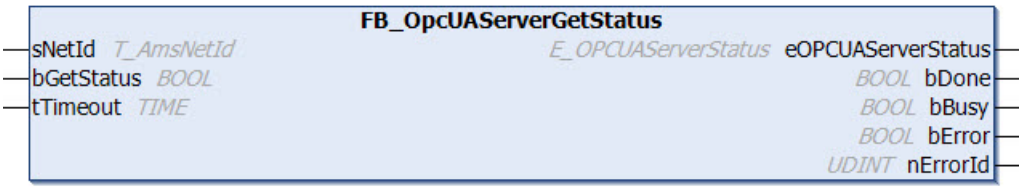
Inputs

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 [▶ 62]	Specifies the operation to be performed.
tTimeout	TIME	ADS Timeout

Outputs

Name	Type	Description
stOpcUAServerInfo	ST_OpcUAServerInfo [▶ 62]	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

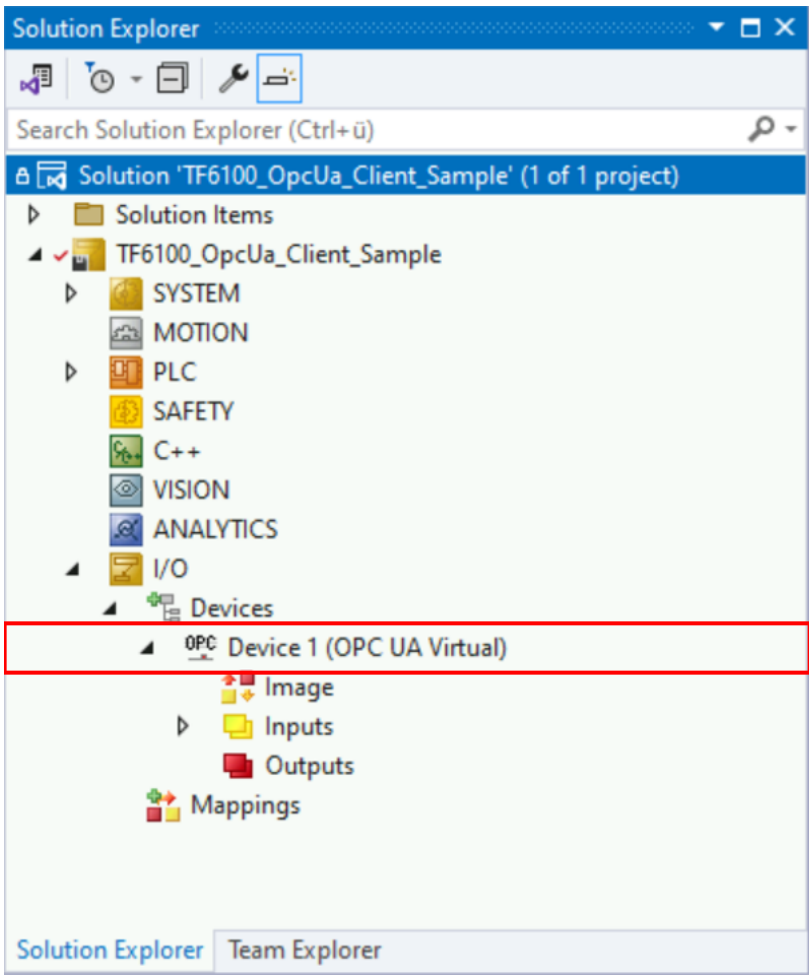
Outputs

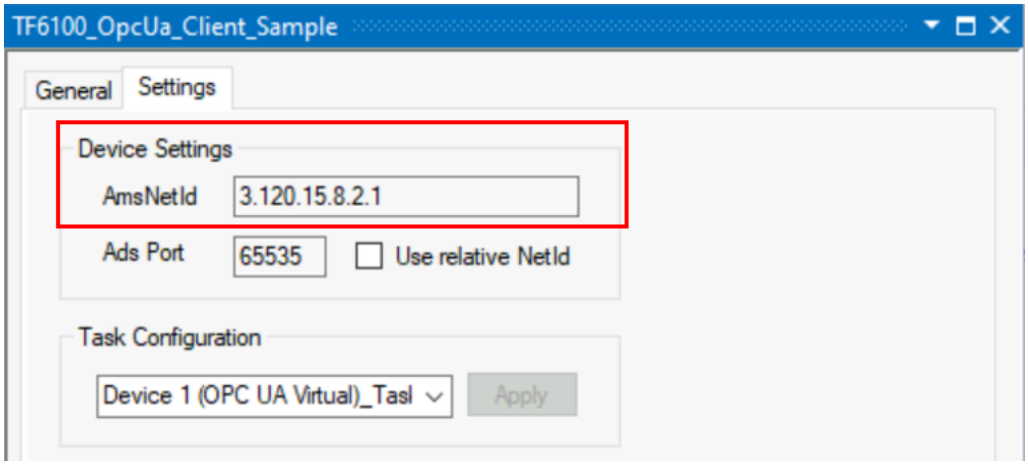
Name	Type	Description
eOPCUAServerStatus	E_OpcUAServerStatus [▶ 63]	Contains status information about the server.
bDone	BOOL	TRUE when processing of the function block is complete.
bBusy	BOOL	TRUE as long as processing of the function block is in progress.
bError	BOOL	Becomes TRUE as soon as an error situation occurs.
nErrorId	UDINT	Contains the error code when an error (bError) occurs.

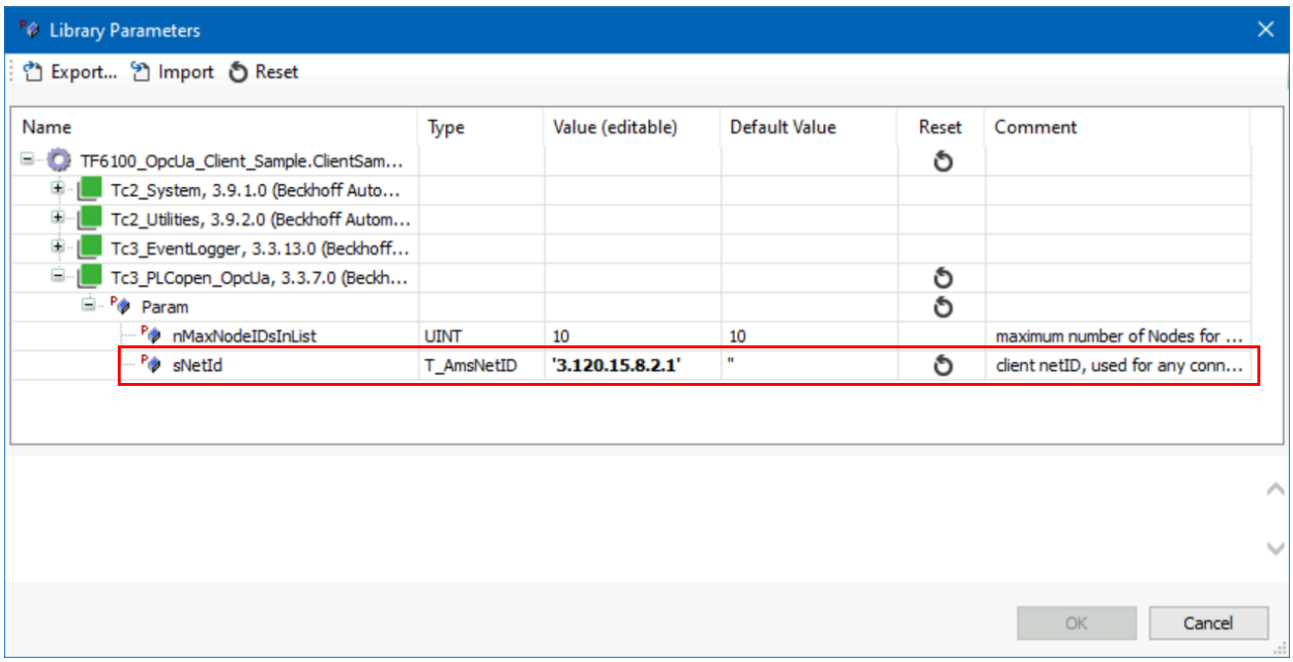
5.2 Tc3_PLCopen_OpcUa

5.2.1 Instructions for use

From TwinCAT OPC UA Client version 2.x.x, an "OPC UA Virtual Device" must be created in the I/O part of TwinCAT to use the Tc3_PLCopen_OpcUa library and the AMS Net ID of the device must be stored in the parameter list of the library for the sAmsNetId parameter. This only needs to be done once.


Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TF6100_OpcUa_Client_Sample' (1 of 1 project)
Solution Items
TF6100_OpcUa_Client_Sample
SYSTEM
MOTION
PLC
SAFETY
C++
VISION
ANALYTICS
I/O
Devices
Device 1 (OPC UA Virtual)
Image
Inputs
Outputs
Mappings
Solution Explorer
Team Explorer


TF6100_OpcUa_Client_Sample
General
Settings
Device Settings
AmsNetId
3.120.15.8.2.1
Ads Port
65535
Use relative NetId
Task Configuration
Device 1 (OPC UA Virtual)_Tasl
Apply



5.2.2 Data types

5.2.2.1 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

Name	Description
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

5.2.2.2 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

5.2.2.3 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

Name	Description
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

5.2.2.4 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

5.2.2.5 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

5.2.2.6 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

5.2.2.7 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

5.2.2.8 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

5.2.2.9 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

5.2.2.10 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

5.2.2.11 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

5.2.2.12 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

5.2.2.13 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

5.2.2.14 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 [▶ 77])

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-Arm®	Tc3_PLCopen_OpcUa

5.2.2.15 ST_UASessionConnectInfo

Syntax

```
TYPE ST_UASessionConnectInfo:
STRUCT
    sApplicationName     : STRING(MAX_STRING_LENGTH);

    eSecurityMode        : E_UASecurityMsgMode;
    eSecurityPolicyUri   : E_UASecurityPolicy;
    eTransportProfileUri : E_UATransportProfile;
    stUserIdentTokenType : ST_UAUserIdentityTokenType;

    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 this will automatically be 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 [▶ 72].
eSecurityPolicyUri	Security policy Uri. For available security policy Uri see E_UASecurityPolicy [▶ 72].
eTransportProfileUri	Transport profile Uri. For available transport profile Uri see E_UATransportProfile [▶ 73];
stUserIdentTokenType	Structure with authentication data for logging on to the OPC UA Server. Full description under ST_UAUserIdentityTokenType [▶ 78].
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

5.2.2.16 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

5.2.2.17 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

5.2.2.18 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 [▶ 69])
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

5.2.2.19 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 [▶ 85].
nReserved	Placeholder
sIdentifier	Identifier as shown in the UA namespace (attribute 'Identifier').
eIdentifierType	Variable type, described by E_UAIdentifierType [▶ 70].

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-Arm®	Tc3_PLCopen_OpcUa

5.2.2.20 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 [▶ 67]).
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 [▶ 76]).

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-Arm®	Tc3_PLCopen_OpcUa

5.2.2.21 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 [▶ 77].
bIsForward	Indicates whether the reference is a forward or backward reference.
stNodeId	NodeId as data type ST_UAExpandedNodeId [▶ 74].
stBrowseName	BrowseName of the reference.
stDisplayName	DisplayName of the reference (ST_UALocalizedText [▶ 76]).
eNodeClass	NodeClass of the reference (E_UANodeClassMask [▶ 71]).
stTypeDefinition	Type definition (HasTypeDefinition) (ST_UAExpandedNodeId [▶ 74]).

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-Arm®	Tc3_PLCopen_OpcUa

5.2.2.22 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 [▶ 74]..
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

5.2.2.23 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 [▶ 86] 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 [▶ 80]	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

5.2.2.24 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 [▶ 79].

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

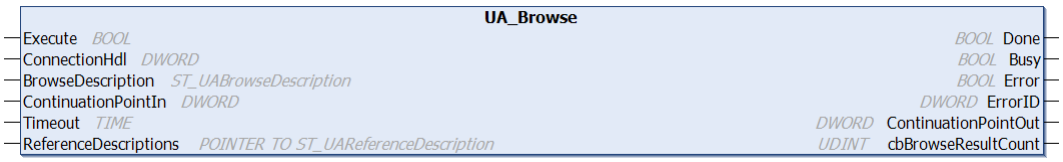
Name	Description
HistorianRaw	HistorianRaw
HistorianCalculated	HistorianCalculated
HistorianInterpolated	HistorianInterpolated
Reserved	Reserved
HistorianPartial	HistorianPartial
HistorianExtraData	HistorianExtraData
HistorianMultiValue	HistorianMultiValue

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

5.2.3 Function blocks

5.2.3.1 UA_Browse



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 [▶ 74]	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;
```

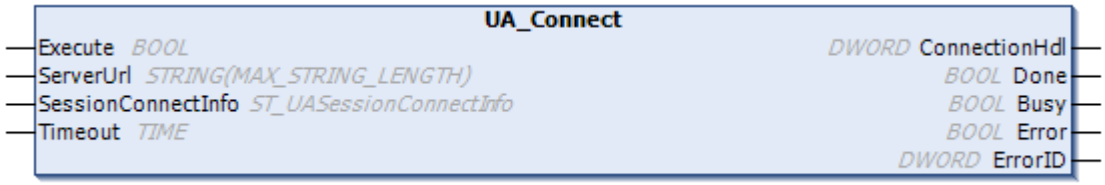
```
    ErrorID              : DWORD;
    ContinuationPointOut : DWORD;
    cbBrowseResultCnt    : UDINT;
END_VAR
```

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

5.2.3.2 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 [▶ 75])
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

Name	Type	Description
		ST_UASessionConnectInfo.tConnectionTimeout. The rule of thumb is: ADS Timeout > 2 * ConnectionTimeout.

Outputs

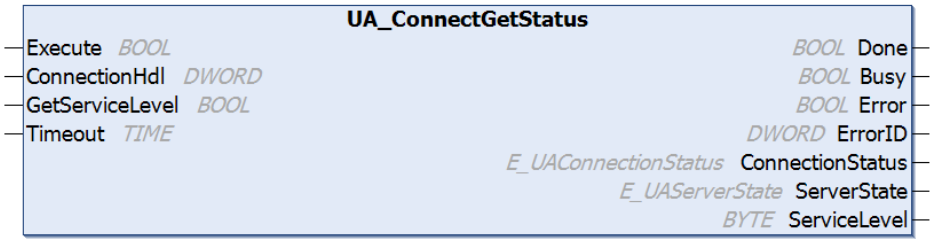
```
VAR_OUTPUT
    ConnectionHdl : DWORD;
    Done          : BOOL;
    Busy          : BOOL;
    Error         : BOOL;
    ErrorID       : DWORD;
END_VAR
```

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

5.2.3.3 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 [▶ 69]. 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.

Name	Type	Description
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
```

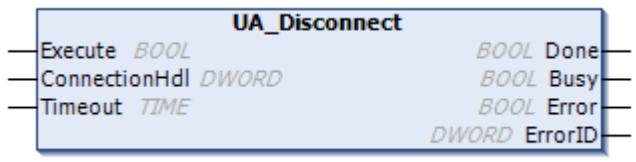
Name	Type	Description
Done	BOOL	Switches to TRUE if the function block was executed successfully.
Busy	BOOL	TRUE until the function block has executed a command, at the most for the duration of the "Timeout" at the input. The inputs accept no new command as long as Busy = TRUE. It is not the connection 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 [▶ 69]).
ServerState	E_UAServerState	Server state (see E_UAServerState [▶ 72]).
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

5.2.3.4 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

5.2.3.5 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 [▶ 96]or UA_Write [▶ 99]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

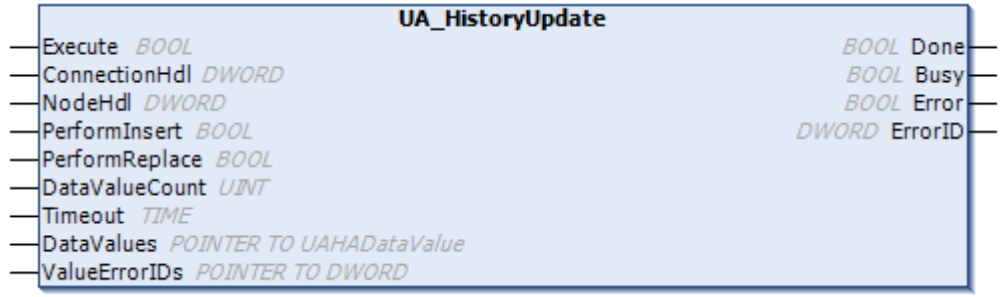
```
VAR_OUTPUT
    NamespaceIndex : UINT;
    Done           : BOOL;
    Busy           : BOOL;
    Error          : BOOL;
    ErrorID        : DWORD;
END_VAR
```

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.

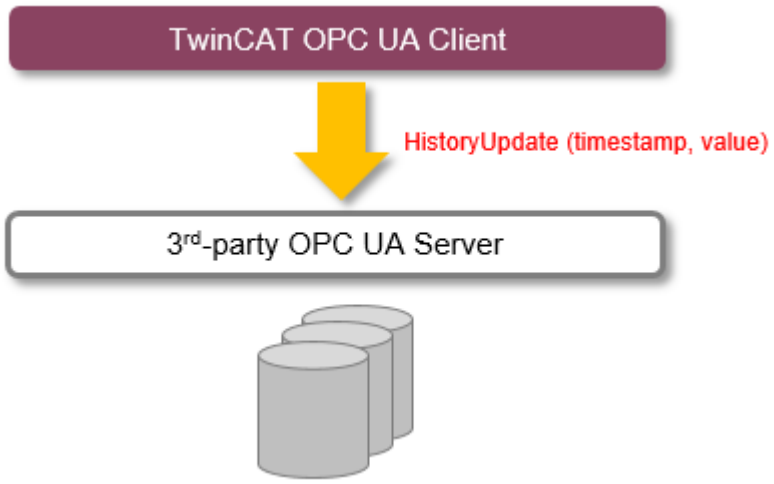
Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT 3.1	Win32, Win64, CE-X86, CE-Arm®	Tc3_PLCopen_OpcUa

5.2.3.6 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

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.

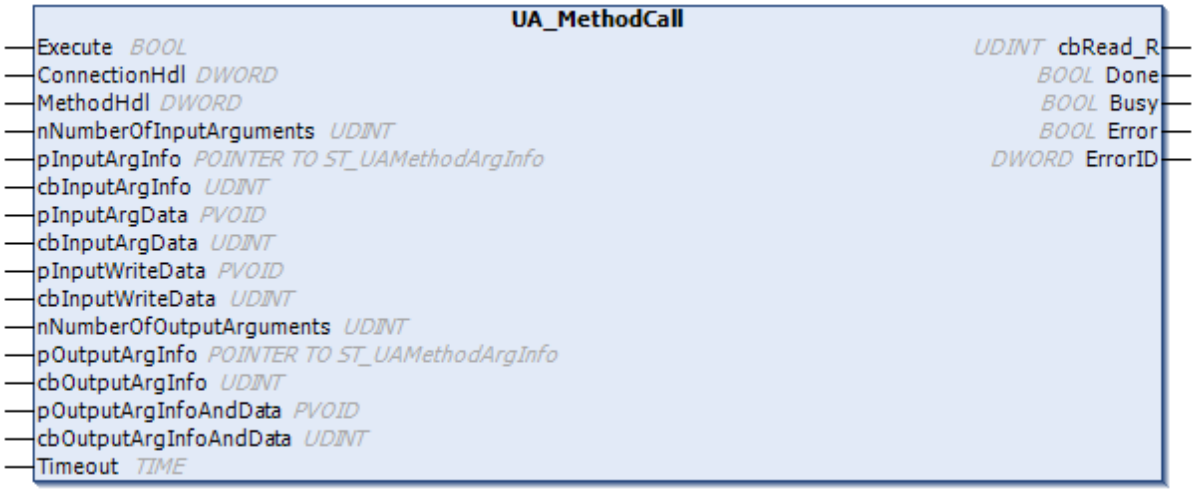
Number of values transferred

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

5.2.3.7 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 [▶ 82], the latter by UA_MethodGetHandle [▶ 91].

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.

Name	Type	Description
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 [▶ 76] (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

5.2.3.8 UA_MethodGetHandle



This function block collects a handle for a UA method, which can then be used to call a method using UA_MethodCall [▶ 89].

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 [▶ 77]).
MethodNodeID	ST_UANodeID	Method node ID of the method to be called. Corresponds to the ID attribute in the UA namespace. (Type: UA_Connect [▶ 82]).
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 [▶ 89].
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.

Name	Type	Description
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

5.2.3.9 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

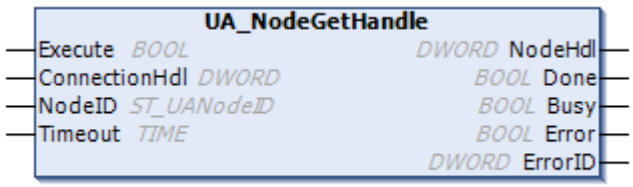
```
VAR_OUTPUT
    Done      : BOOL;
    Busy      : BOOL;
    Error     : BOOL;
    ErrorID   : UDINT;
END_VAR
```

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

5.2.3.10 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 [▶ 85].
Timeout	TIME	Time until the function is aborted. DEFAULT_ADS_TIMEOUT is a global constant, set to 5 seconds.

Outputs

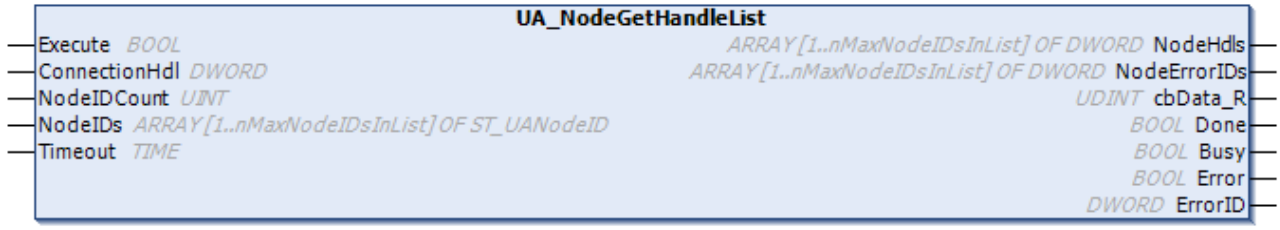
```
VAR_OUTPUT
    NodeHdl     : DWORD;
    Done        : BOOL;
    Busy        : BOOL;
    Error       : BOOL;
    ErrorID     : DWORD;
END_VAR
```

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

5.2.3.11 UA_NodeGetHandleList



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 [▶ 77].
Timeout	TIME	Time until the function is aborted. DEFAULT_ADS_TIMEOUT is a global constant, set to 5 seconds.

Outputs

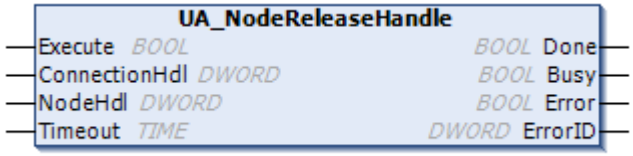
```
VAR_OUTPUT
    NodeHdls     : ARRAY[1..nMaxNodeIDsInList] OF DWORD;
    NodeErrorIDs : ARRAY[1..nMaxNodeIDsInList] OF DWORD;
    cbData_R     : UDINT;
    Done         : BOOL;
    Busy         : BOOL;
    Error        : BOOL;
    ErrorID      : DWORD;
END_VAR
```

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

5.2.3.12 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

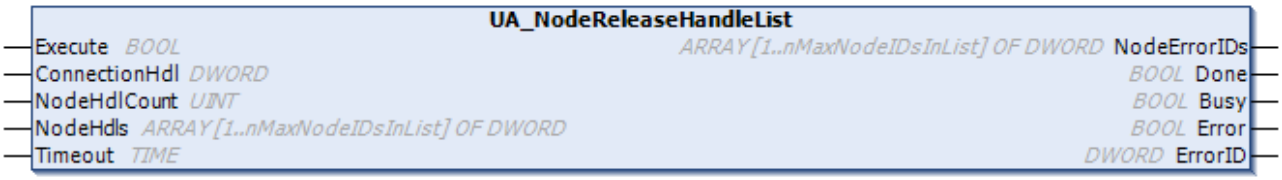
```
VAR_OUTPUT
    Done       : BOOL;
    Busy       : BOOL;
    Error      : BOOL;
    ErrorID    : DWORD;
END_VAR
```

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

5.2.3.13 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

5.2.3.14 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.
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.
stNodeAddInfo	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 [▶ 77].
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

5.2.3.15 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 [▶ 82].
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 [▶ 93] or UA_NodeGetHandleList [▶ 94].
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 [▶ 77].
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.

Name	Type	Description
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

5.2.3.16 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 [▶ 82].
NodeHdl	DWORD	Node handle that was previously output by the function block UA_NodeGetHandle [▶ 93].
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 [▶ 77].
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

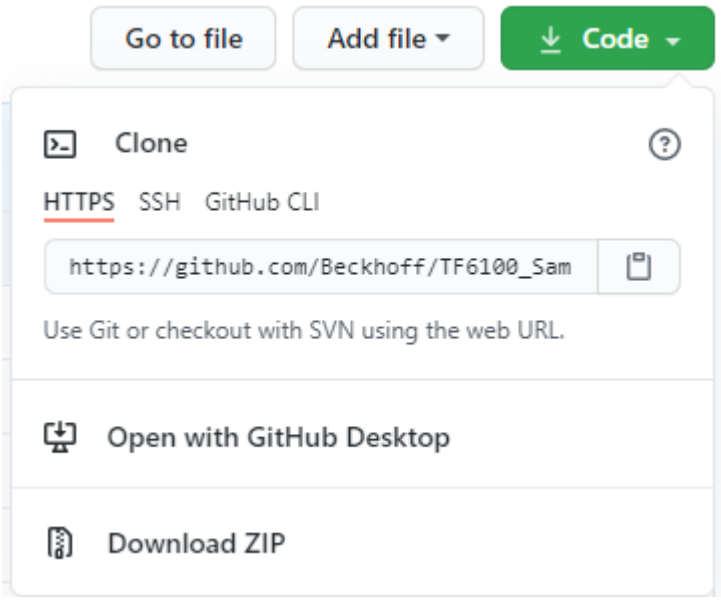
5.2.4 Parameter list

The following parameters from the Tc3_PLCopen_OpcUa library influence how the communication works.

Name	Type	Default value	Description
nMaxNodeIDsInList	UINT	10	Specifies the maximum number of nodes in a list, for example for the function blocks UA_NodeGetHandleList [▶ 94], UA_ReadList [▶ 98] and UA_NodeReleaseHandleList [▶ 95].
sNetId	T_AmsNetID	127.0.0.1.1.1	Specifies the AMS Net ID of the device on which the TwinCAT OPC UA Client is running. Normally, PLC and client run on the same system.

6 Samples

Sample code and configurations for this product can be obtained from the corresponding repository on GitHub: https://github.com/Beckhoff/TF6100_Samples. There you have the possibility to clone the repository or download a ZIP file with the sample.

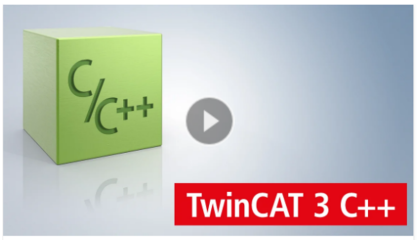


The following samples exist:

Name	TwinCAT Version	Description
TF6100_OpcUa_Client_Sample	TwinCAT 3	This sample contains sample code for various functions of the TwinCAT OPC UA Client (PLCOpen function blocks). These include Browse, Connect, HistoryUpdate, MethodCall, Read and Write. The server sample for access is also included.
TF6100_OpcUa_Server_Sample	TwinCAT 3	This sample contains a PLC with extensive provision of PLC data for the TwinCAT OPC UA Server (OPC UA Data Access).
TS6100_OpcUa_Client_Sample	TwinCAT 2	This sample contains sample code for various functions of the TwinCAT OPC UA Client (PLCOpen function blocks). These include MethodCall, Read and Write.
TS6100_OpcUa_Server_Sample	TwinCAT 2	This sample demonstrates the use of PLC comments to release variables via the TwinCAT OPC UA Server.

7 Tutorials

Video tutorials for this product can be found on our website at https://www.beckhoff.com/tutorials. The video tutorials offer a quick start to the product and the individual product facets.



14.03.2024 | Multimedia
Tutorial: TwinCAT 3 C++ overview
Get an overview of the C++ integration into TwinCAT.
Learn more →



12.02.2024 | Multimedia
Tutorial: Chart synchronization in TwinCAT Scope
Learn how to dock and synchronize charts in TwinCAT Scope.
Learn more →



12.02.2024 | Multimedia
Tutorial: Getting started with TF6100 TwinCAT 3 OPC UA Server
Learn how to configure the TwinCAT OPC UA Server.
Learn more →

8 Appendix

8.1 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.

Behavior	Remedy
The attempt to read or write a StructuredDataType from a server using the PLCopen function blocks fails.	Structured Data Types are not supported by the PLCopen-based client. Please use the I/O client for this purpose.
When the PLCopen function blocks are executed, the Busy output remains TRUE, but no error is output and the timeout is not triggered.	This is an indication of insufficient ADS router memory. Please increase the router memory in the corresponding settings of the TwinCAT ADS router.
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 working 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 by the I/O client or do not reach the server.	Please check whether the "Write Enable" output has been enabled on the I/O client.
After an upgrade from product version 1.x to 2.x, a PLCopen function block reports ADS error "6".	To use the PLCopen function blocks in product version 2.x, you must create a Virtual OPC UA Device in the I/O area of your TwinCAT project and store [▶ 65] its AMS Net ID (to be found in the **Settings** tab) in the "sNetId" variable of the parameter list of the Tc3_PLCopen_OpcUa library.

8.2 Status codes

8.2.1 ADS Return Codes

Grouping of error codes:
Global error codes: 0x0000 [▶ 103]... (0x9811_0000 ...)
Router error codes: 0x500 [▶ 104]... (0x9811_0500 ...)
General ADS errors: 0x700 [▶ 104]... (0x9811_0700 ...)
RTime error codes: 0x1000 [▶ 106]... (0x9811_1000 ...)

Global error codes

Hex	Dec	HRESULT	Name	Description
0x0	0	0x98110000	ERR_NOERROR	No error.
0x1	1	0x98110001	ERR_INTERNAL	Internal error.
0x2	2	0x98110002	ERR_NORTIME	No real time.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Allocation locked – memory error.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Mailbox full – the ADS message could not be sent. Reducing the number of ADS messages per cycle will help.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Wrong HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Target port not found – ADS server is not started, not reachable or not installed.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Target computer not found – AMS route was not found.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unknown command ID.

Hex	Dec	HRESULT	Name	Description
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. Several causes are possible. For example, an incorrect password was entered when creating routes.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parameter size not correct.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Invalid data values.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Device is not ready to operate.

Hex	Dec	HRESULT	Name	Description
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Device is busy.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Invalid operating system context. This can result from use of ADS blocks in different tasks. It may be possible to resolve this through multitasking synchronization in the PLC.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Insufficient memory.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Invalid parameter values.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Not found (files, ...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax error in file or command.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objects do not match.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Object already exists.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol not found.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Invalid symbol version. This can occur due to an online change. Create a new handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Device (server) is in invalid state.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode not supported.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification handle is invalid.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification client not registered.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	No further handle available.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Notification size too large.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Device not initialized.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Device has a timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface query failed.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Wrong interface requested.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class ID is invalid.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object ID is invalid.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Request pending.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Request is aborted.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal warning.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Invalid array index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol not active.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Access denied. Several causes are possible. For example, a unidirectional ADS route is used in the opposite direction.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Missing license.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	License expired.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	License exceeded.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Invalid license.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	License problem: System ID is invalid.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	License not limited in time.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Licensing problem: time in the future.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	License period too long.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception at system startup.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	License file read twice.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Invalid signature.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Invalid certificate.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public key not known from OEM.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	License not valid for this system ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo license prohibited.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Invalid function ID.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Outside the valid range.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Invalid alignment.
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Invalid platform level.
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Context – forward to passive level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Context – forward to dispatch level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Context – forward to real-time.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Client error.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Service contains an invalid parameter.

Hex	Dec	HRESULT	Name	Description
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling list is empty.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var connection already in use.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	The called ID is already in use.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout has occurred – the remote terminal is not responding in the specified ADS timeout. The route setting of the remote terminal may be configured incorrectly.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Error in Win32 subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Invalid client timeout value.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port not open.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	No AMS address.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Internal error in Ads sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Hash table overflow.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Key not found in the table.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	No symbols in the cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Invalid response received.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port is locked.
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	The request was canceled.

RTime error codes

Hex	Dec	HRESULT	Name	Description
0x1000	4096	0x98111000	RTERR_INTERNAL	Internal error in the real-time system.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer value is not valid.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task pointer has the invalid value 0 (zero).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack pointer has the invalid value 0 (zero).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	The request task priority is already assigned.
0x1005	4101	0x98111005	RTERR_NOMORETCB	No free TCB (Task Control Block) available. The maximum number of TCBs is 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	No free semaphores available. The maximum number of semaphores is 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	No free space available in the queue. The maximum number of positions in the queue is 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	An external synchronization interrupt is already applied.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	No external sync interrupt applied.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Application of the external synchronization interrupt has failed.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Call of a service function in the wrong context
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel® VT-x extension is not supported.
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.

Hex	Dec	Name	Description
0x274D	10061	WSAECONNREFUSED	Connection refused - no connection could be established because the target computer has explicitly rejected it. This error usually results from an attempt to connect to a service that is inactive on the external host, that is, a service for which no server application is running.
0x2751	10065	WSAEHOSTUNREACH	No route to host - a socket operation referred to an unavailable host.
More Winsock error codes: Win32 error codes			

8.2.2 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.

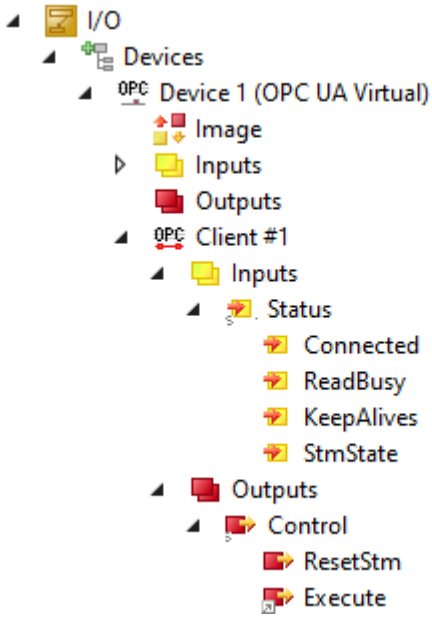


Fig. 1: OPCUAClientModulesStatusCodes

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		

Variable	Schema	0	1- State machine state	2- Keep alive count if using subscriptions	3- Connection state (&read busy)
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.**

8.2.3 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 [▶ 103]) 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...

Hex	Name	Description
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.
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.

8.3 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

Trademark statements

Beckhoff®, ATRO®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, MX-System®, Safety over EtherCAT®, TC/BSD®, TwinCAT®, TwinCAT/BSD®, TwinSAFE®, XFC®, XPlanar® and XTS® are registered and licensed trademarks of Beckhoff Automation GmbH.

Third-party trademark statements

Arm, Arm9 and Cortex are trademarks or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere.

DALI, DALI-2, D4i, DALI+, and DiiA are trademarks in various countries in the exclusive use of the Digital Illumination Interface Alliance.

Excel, IntelliSense, Microsoft, Microsoft Azure, Microsoft Edge, PowerShell, Visual Studio, Windows and Xbox are trademarks of the Microsoft group of companies.

Intel, the Intel logo, Intel Core, Xeon, Intel Atom, Celeron and Pentium are trademarks of Intel Corporation or its subsidiaries.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

More Information:
www.beckhoff.com/tf6100

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com