

Handbuch | DE

TS6100-0030

TwinCAT 2 | OPC UA Server CE



Inhaltsverzeichnis

1	Vorwort.....	5
1.1	Hinweise zur Dokumentation	5
1.2	Zu Ihrer Sicherheit.....	6
1.3	Hinweise zur Informationssicherheit	7
2	Übersicht.....	8
2.1	Szenarien	8
2.2	Anwendungsbeispiele	11
2.2.1	Nachbearbeitung in der Cloud	11
3	Installation	17
3.1	Setupübersicht	17
3.2	Systemvoraussetzungen.....	17
3.3	Installation (TC3).....	19
3.4	Installation (TC2).....	22
3.5	Installation Windows CE (TC3)	25
3.6	Installation Windows CE (TC2)	27
3.7	Lizenzierung (TC3).....	32
3.8	Lizenzierung (TC2).....	34
4	Technische Einführung	35
4.1	Server.....	35
4.1.1	Übersicht.....	35
4.1.2	Schnelleinstieg	35
4.1.3	Initialisierung	37
4.1.4	Empfohlene Schritte.....	42
4.1.5	Optimierungen.....	45
4.1.6	Data Access	49
4.1.7	Historical Access.....	76
4.1.8	Alarms and Conditions	82
4.1.9	Method Call	87
4.1.10	File Transfer	93
4.1.11	Global Discovery Service	95
4.1.12	TwinCAT Eventlogger	99
4.1.13	Security	104
4.1.14	Verschiedenes	110
4.2	Konfigurator.....	115
4.2.1	Visual Studio	116
4.2.2	Standalone	145
4.3	Client I/O	159
4.3.1	Übersicht.....	159
4.3.2	Schnelleinstieg	161
4.3.3	Unterstützte Datentypen	164
4.3.4	Knoten vom Server hinzufügen.....	165
4.3.5	Knotenattribute.....	166
4.3.6	Methodenaufruf.....	167

4.3.7	StructuredTypes.....	170
4.3.8	Datenaufnahme.....	172
4.3.9	Schreiben von Variablen.....	174
4.3.10	Sicherheit.....	176
4.4	Client PLCopen.....	178
4.4.1	Übersicht.....	178
4.4.2	Unterstützte Datentypen.....	178
4.4.3	Best Practice.....	179
4.4.4	Sicherheit.....	194
4.5	Gateway.....	196
4.5.1	Übersicht.....	196
4.5.2	Schnelleinstieg.....	196
4.5.3	Lizenzierung.....	198
4.5.4	Szenarien.....	198
4.5.5	Konfigurator.....	200
4.5.6	Migration von Tx6120.....	205
4.5.7	Sicherheit.....	207
4.6	Sample Client.....	209
4.6.1	Übersicht.....	209
4.6.2	Sichere Verbindung mit OPC UA Server herstellen.....	210
4.6.3	UA-Namensraum durchsuchen.....	213
4.6.4	Watchliste verwenden.....	214
5	SPS API.....	216
5.1	Tc2_OpcUa.....	216
5.1.1	Datentypen.....	216
5.1.2	Funktionsbausteine.....	217
5.2	Tc3_PLCopen_OpcUa.....	219
5.2.1	Datentypen.....	219
5.2.2	Funktionsbausteine.....	233
6	Beispiele.....	255
7	Anhang.....	256
7.1	Fehlerdiagnose.....	256
7.1.1	Server.....	257
7.1.2	Client I/O.....	259
7.1.3	Client PLCopen.....	259
7.1.4	Gateway.....	259
7.2	Statuscodes.....	260
7.2.1	ADS Return Codes.....	260
7.2.2	Client I/O.....	264
7.2.3	Client PLCopen.....	266
7.3	Support und Service.....	268

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.

Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.

Das Fachpersonal ist verpflichtet, stets die aktuell gültige Dokumentation zu verwenden.

Das Fachpersonal hat sicherzustellen, dass die Anwendung bzw. der Einsatz der beschriebenen Produkte alle Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiterentwickelt.

Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.

Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® und XPlanar® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.

Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Bezeichnungen führen.

Patente

Die EtherCAT-Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:

EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702

mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.



EtherCAT® ist eine eingetragene Marke und patentierte Technologie lizenziert durch die Beckhoff Automation GmbH, Deutschland

Copyright

© Beckhoff Automation GmbH & Co. KG, Deutschland.

Weitergabe sowie Vervielfältigung dieses Dokuments, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet.

Zu widerhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksmustereintragung vorbehalten.

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

Lesen Sie die folgenden Erklärungen zu Ihrer Sicherheit.
Beachten und befolgen Sie stets produktspezifische Sicherheitshinweise, die Sie gegebenenfalls an den entsprechenden Stellen in diesem Dokument vorfinden.

Haftungsausschluss

Die gesamten Komponenten werden je nach Anwendungsbestimmungen in bestimmten Hard- und Software-Konfigurationen ausgeliefert. Änderungen der Hard- oder Software-Konfiguration, die über die dokumentierten Möglichkeiten hinausgehen, sind unzulässig und bewirken den Haftungsausschluss der Beckhoff Automation GmbH & Co. KG.

Qualifikation des Personals

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs-, Automatisierungs- und Antriebstechnik, das mit den geltenden Normen vertraut ist.

Signalwörter

Im Folgenden werden die Signalwörter eingeordnet, die in der Dokumentation verwendet werden. Um Personen- und Sachschäden zu vermeiden, lesen und befolgen Sie die Sicherheits- und Warnhinweise.

Warnungen vor Personenschäden

GEFAHR

Es besteht eine Gefährdung mit hohem Risikograd, die den Tod oder eine schwere Verletzung zur Folge hat.

WARNUNG

Es besteht eine Gefährdung mit mittlerem Risikograd, die den Tod oder eine schwere Verletzung zur Folge haben kann.

VORSICHT

Es besteht eine Gefährdung mit geringem Risikograd, die eine mittelschwere oder leichte Verletzung zur Folge haben kann.

Warnung vor Umwelt- oder Sachschäden

HINWEIS

Es besteht eine mögliche Schädigung für Umwelt, Geräte oder Daten.

Information zum Umgang mit dem Produkt



Diese Information beinhaltet z. B.:
Handlungsempfehlungen, Hilfestellungen oder weiterführende Informationen zum Produkt.

1.3 Hinweise zur Informationssicherheit

Die Produkte der Beckhoff Automation GmbH & Co. KG (Beckhoff) sind, sofern sie online zu erreichen sind, mit Security-Funktionen ausgestattet, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen. Trotz der Security-Funktionen sind die Erstellung, Implementierung und ständige Aktualisierung eines ganzheitlichen Security-Konzepts für den Betrieb notwendig, um die jeweilige Anlage, das System, die Maschine und die Netzwerke gegen Cyber-Bedrohungen zu schützen. Die von Beckhoff verkauften Produkte bilden dabei nur einen Teil des gesamtheitlichen Security-Konzepts. Der Kunde ist dafür verantwortlich, dass unbefugte Zugriffe durch Dritte auf seine Anlagen, Systeme, Maschinen und Netzwerke verhindert werden. Letztere sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn entsprechende Schutzmaßnahmen eingerichtet wurden.

Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem <https://www.beckhoff.de/secguide>.

Die Produkte und Lösungen von Beckhoff werden ständig weiterentwickelt. Dies betrifft auch die Security-Funktionen. Aufgrund der stetigen Weiterentwicklung empfiehlt Beckhoff ausdrücklich, die Produkte ständig auf dem aktuellen Stand zu halten und nach Bereitstellung von Updates diese auf die Produkte aufzuspielen. Die Verwendung veralteter oder nicht mehr unterstützter Produktversionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Hinweise zur Informationssicherheit zu Produkten von Beckhoff informiert zu sein, abonnieren Sie den RSS Feed unter <https://www.beckhoff.de/secinfo>.

2 Übersicht

OPC Unified Architecture (OPC UA) ist die nächste Generation des klassischen OPC-Standards. Es handelt es sich hierbei um ein weltweit standardisiertes Kommunikationsprotokoll, über das Maschinendaten hersteller- und plattformunabhängig ausgetauscht werden können. OPC UA integriert gängige Sicherheitsstandards bereits direkt im Protokoll. Ein weiterer großer Vorteil von OPC UA gegenüber dem klassischen OPC-Standard ist die Unabhängigkeit vom COM/DCOM-System.

Detaillierte Informationen zu OPC UA finden Sie auf den Webseiten der [OPC Foundation](#).

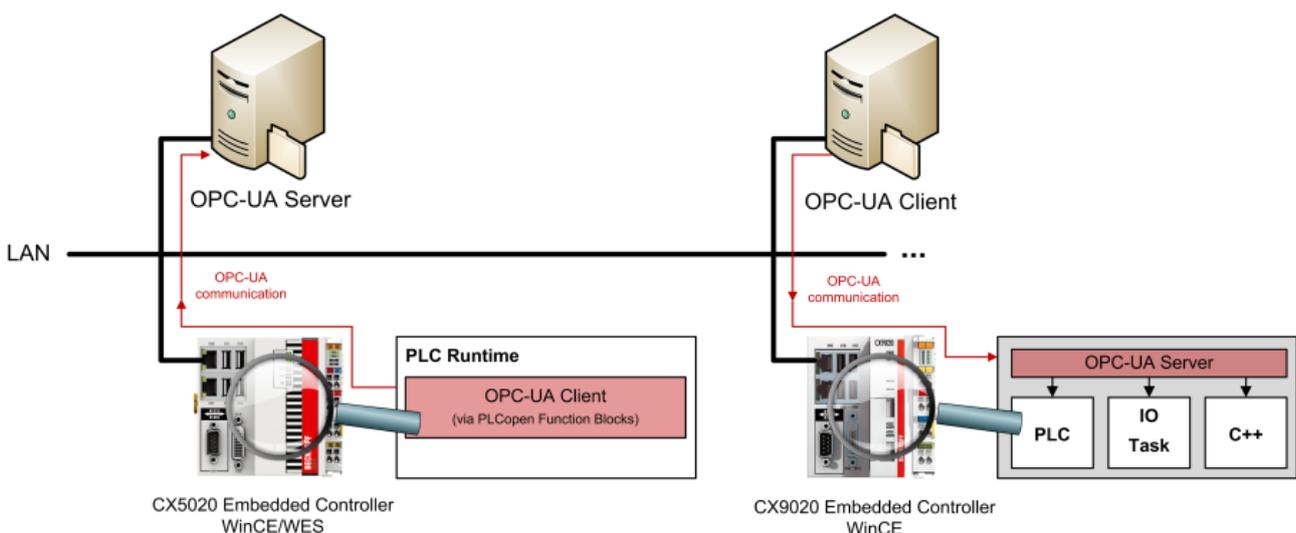
Komponenten

Folgende Softwarekomponenten wurden für Win32/64- und Windows-CE-basierte Systeme eingebunden:

Software-Komponente	Beschreibung
OPC UA Server [▶ 35]	Stellt eine OPC-UA-Server-Schnittstelle zur Verfügung, damit UA-Clients auf die TwinCAT-Laufzeit zugreifen können.
OPC UA Client [▶ 178]	Stellt eine OPC-UA-Client-Funktionalität zur Verfügung, damit die Kommunikation mit anderen OPC UA Servern auf der Grundlage von PLCopen-normten Funktionsbausteinen sowie einem einfach zu konfigurierenden I/O-Gerät möglich ist.

Folgende Softwarekomponenten wurden für Win32/64-basierte Systeme eingebunden:

Software-Komponente (nur Windows)	Beschreibung
OPC UA Configurator	Grafische Benutzerschnittstelle für die Konfiguration des TwinCAT OPC UA Server
OPC UA Sample Client [▶ 209]	Grafische Beispielimplementierung eines OPC UA Clients um einen ersten Verbindungstest mit dem TwinCAT OPC UA Server durchführen zu können.
OPC UA Gateway [▶ 196]	Wrapper-Technologie, die sowohl eine OPC-COM-DA-Server-Schnittstelle als auch OPC-UA-Server-Aggregationsfähigkeiten zur Verfügung stellt.



2.1 Szenarien

Im Folgenden werden einige Szenarien erläutert, gemäß derer die Komponenten je nach Anwendungsfall und Infrastruktur implementiert und verwendet werden können:

- OPC UA Server: [Integriert in Industrie-PC oder Embedded-PC](#) [▶ 9]

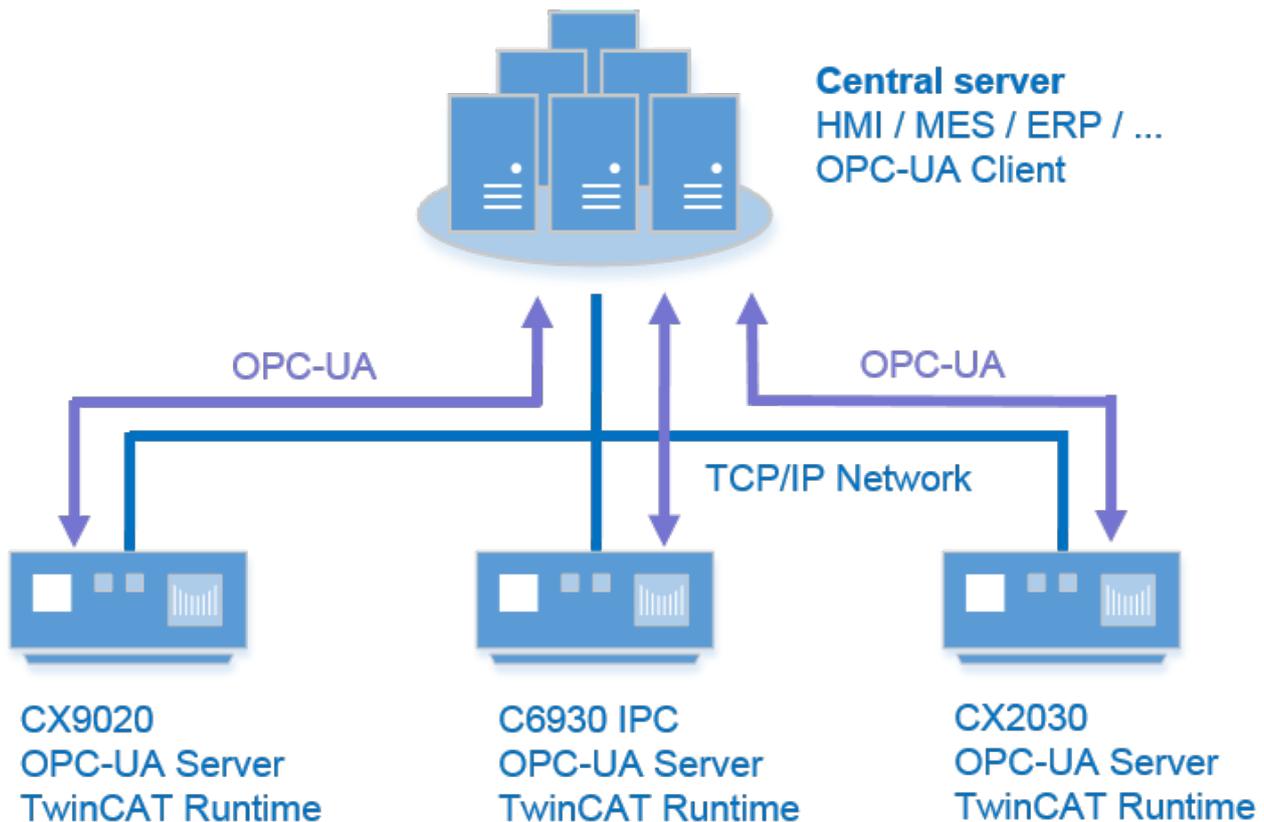
- OPC UA Server: Läuft auf einem Zentralrechner mit Verbindung zu ferner(n) TwinCAT-Laufzeit(en).
[▶ 10]
- OPC UA Server: Zugriff auf BC Controller [▶ 10]

OPC UA Server: Integriert in Industrie-PC oder Embedded-PC

Empfohlenes Szenario

i Dieses Szenario beschreibt, wie der TwinCAT OPC UA Server unter normalen Umständen eingesetzt werden sollte.

Einer der größten Vorteile des TwinCAT OPC UA Servers besteht darin, dass er selbst in die kleinste Embedded-Plattform, z. B. die CX8000-Baureihe, integriert werden kann. Dank dieser Integration ist das allgemeine Handling sehr einfach und komfortabel. OPC UA Clients, z. B. HMI- oder MES/ERP-Systeme, können eine Verbindung mit dem OPC UA Server herstellen und aus der TwinCAT-Laufzeit stammende Symbolinformationen lesen oder schreiben.



Auf dem zentralen Server laufen folgende Softwarekomponenten und Konfigurationen:

- Dritter OPC UA Client, der z. B. ein HMI-, MES-, oder ERP-System sein kann.

Auf dem Industrie-PC oder Embedded-PC laufen folgende Softwarekomponenten und Konfigurationen:

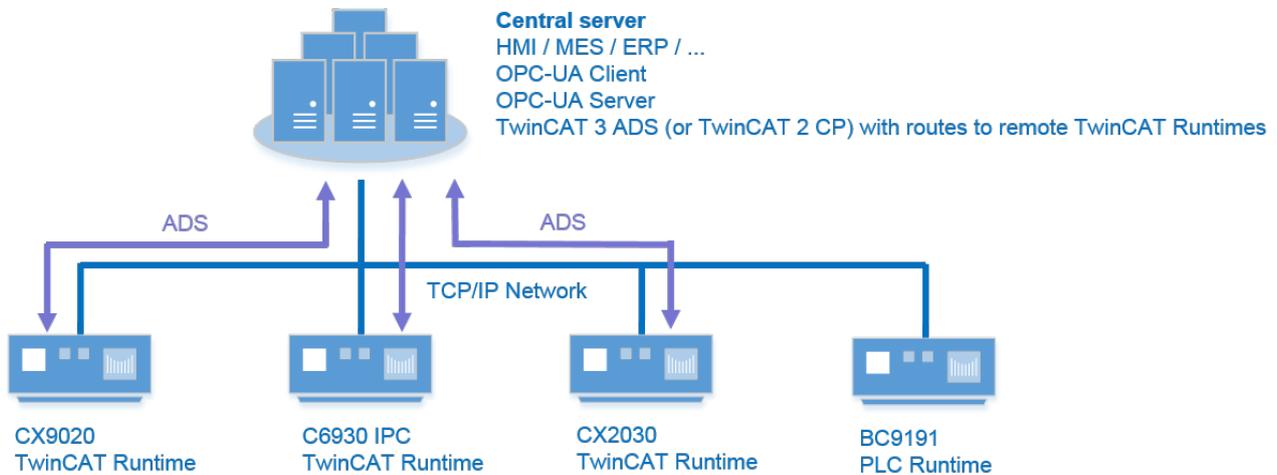
- Der OPC UA Server stellt automatisch eine Verbindung mit der ersten lokalen TwinCAT-SPS-Laufzeit her.
- TwinCAT-Laufzeit

Dieses Szenario hat folgende Vorteile:

- Die Netzwerknutzung ist optimiert, weil sie auf OPC-UA-Kommunikationstechniken aufbaut, z. B. OPC-UA-Registrierungen.
- Die Speichernutzung ist dezentral. Jedes Gerät ist ausschließlich für den eigenen Speicherbedarf zuständig.
- OPC UA verfügt über direkt ins Protokoll integrierte Sicherheitsmechanismen. Dies ist z. B. sehr praktisch, wenn einer der Industrie-PCs oder Embedded-PC übers Internet verbunden ist.

OPC UA Server: Läuft auf einem Zentralrechner mit Verbindung zu ferner(n) TwinCAT-Laufzeit(en)

Dieses Szenario beschreibt die klassische Implementierung von OPC-Server. Server, die die alte OPC-DA-Technologie verwenden, waren häufig auf einem zentralen Server implementiert, statt auf dem Industrie-PC, auf dem die TwinCAT-Laufzeit ausgeführt wird, um die DCOM-Konfigurationen zu vermeiden. (Zur Erinnerung: Im Gegensatz zu OPC UA basiert OPC DA auf COM/DCOM-Technologien)



Auf dem zentralen Server laufen folgende Softwarekomponenten und Konfigurationen:

- TwinCAT 3 ADS (oder TwinCAT 2 CP) für die erforderliche ADS-Konnektivität
- OPC UA Server mit Datenzugriffsgeräten für ferne TwinCAT-Laufzeiten konfiguriert
- ADS-Routen zu fernen TwinCAT-Laufzeiten
- Symboldateien von jeder fernen TwinCAT-Laufzeit
- OPC UA Client, der z. B. ein HMI-, MES- oder ERP-System sein kann.

Auf dem Industrie-PC oder Embedded-PC laufen folgende Softwarekomponenten und Konfigurationen:

- TwinCAT-Laufzeit
- ADS-Route zum zentralen Server

Je mehr ferne TwinCAT-Laufzeiten mit dem zentralen OPC UA Server verbunden werden, desto höher ist die Netzwerknutzung.

Der OPC UA Server verwendet fortschrittliche ADS-Aufzeichnungstechniken, um die Symbolwerte von der TwinCAT-Laufzeit abzufragen. Je mehr Symbole vorhanden sind, desto mehr zyklische Abfragen sind unterwegs. Demzufolge findet die optimierte OPC-UA-Kommunikation nur lokal auf dem zentralen Server statt (zwischen OPC UA Client und OPC UA Server).

Dieses Szenario weist im Vergleich zu Szenario 1 folgende Nachteile auf:

- Die Netzwerknutzung kann abhängig von der Anzahl vorhandener Geräte und Symbole sehr hoch sein.
- Der Speicherbedarf auf dem zentralen Server ist sehr hoch, weil der OPC UA Server Symbolinformationen von jeder TwinCAT-Laufzeit importieren muss.
- Keine Sicherheit zwischen zentralem Server und der TwinCAT-Laufzeit dank Datenverschlüsselung – ADS wurde auf hohe Leistung ausgelegt.
- Die Notwendigkeit, bei jeder SPS-Programmänderung Symboldateien zwischen TwinCAT-Laufzeit und zentralem Server auszutauschen.

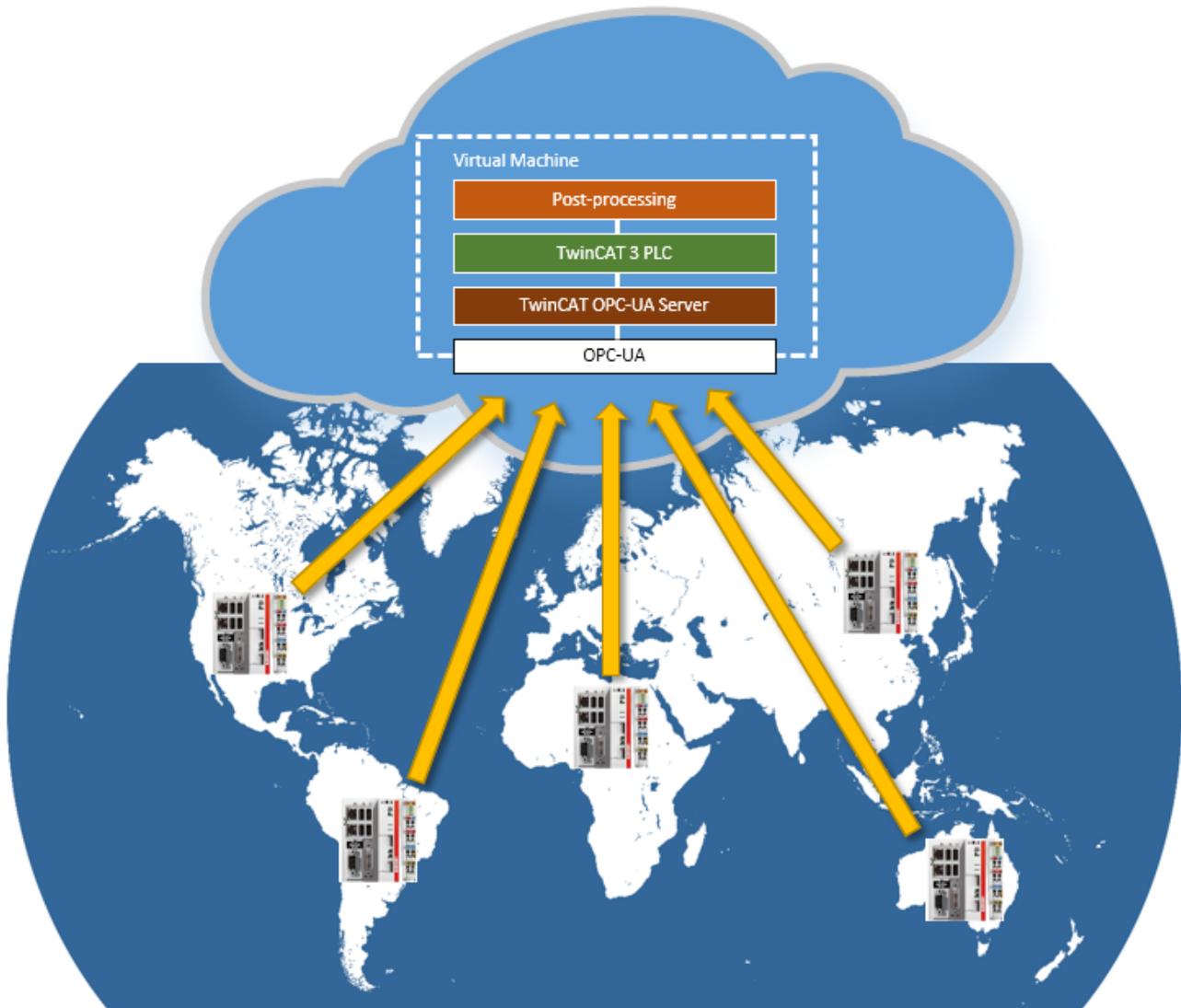
OPC UA Server: Zugriff auf BC Controller

Auch wenn kleine BC Controller nicht in der Lage sind, einen eigenen OPC UA Server auszuführen, können Sie trotzdem Zugriff auf die auf einem fernen BC Controller, z. B. ein BC9191, ausgeführte SPS-Laufzeit haben und deren Symbolwerte über OPC UA veröffentlichen. Bei diesem Szenario muss der OPC UA Server zusammen mit einer TwinCAT 3 ADS (oder TwinCAT 2 CP) auf einem anderen Rechner ausführt und das Szenario auf die gleiche Weise wie Szenario 2 konfiguriert werden.

2.2 Anwendungsbeispiele

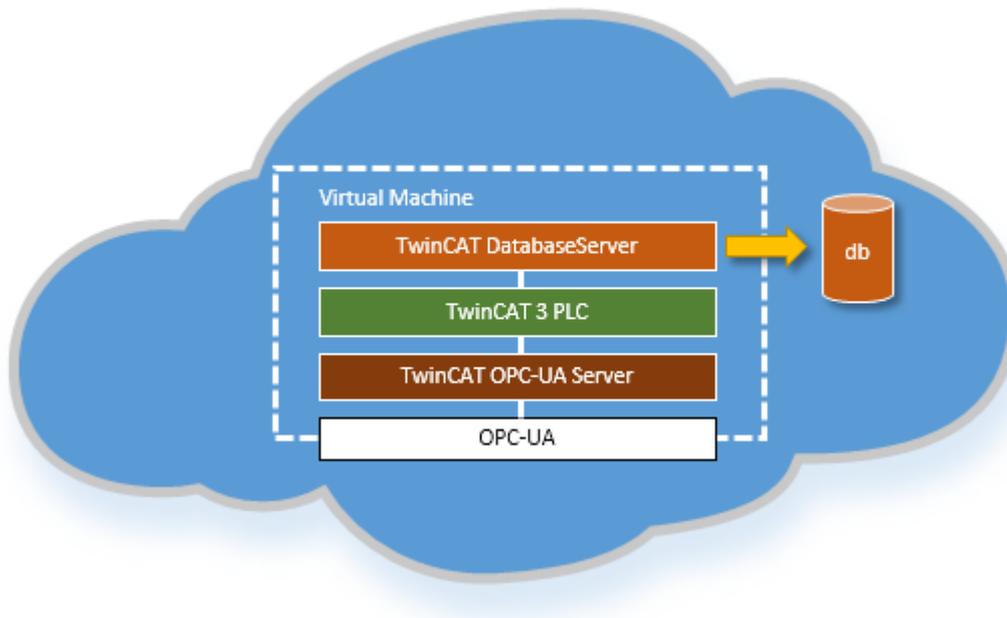
2.2.1 Nachbearbeitung in der Cloud

Das Gesamtkonzept zur Bereitstellung einer generischen, standardisierten und aufrufbaren Schnittstelle in der Cloud, die eingehende Daten zur weiteren Analyse akzeptiert, basiert neben den dezentralisierten Clients, die diesen Service aufrufen, auf einem Endpunkt in der Cloud, der entweder eine Windows-basierte virtuelle Maschine oder eine echte Hardware ist, auf der ein Windows Betriebssystem läuft. In dieser Dokumentation wird dieses Gerät im Allgemeinen als „OPC UA Server“ (in der Cloud) bezeichnet.



Daten-Logger

Die Dokumentation zeigt, wie auf Rechenressourcen in der Cloud zugegriffen werden kann und dezentrale TwinCAT-SPS-Geräte angeschlossen werden können, um diese Ressourcen zu nutzen. Als erstes Anwendungsbeispiel kann dieser Anwendungsfall auf ein typisches Datenspeicherungsszenario angewandt werden, in dem dezentrale TwinCAT-SPS-Geräte Daten an die Cloud senden, in der die eingehenden Daten in einer SQL-Datenbank über den TwinCAT Database Server gespeichert werden. Obwohl das allgemeine Szenario ebenfalls das Hin- und Zurücksenden von Daten umfasst (d. h. Senden von Parametern an die Cloud, Ausführung der Berechnungen in der Cloud und dann Rücksendung der berechneten Ergebnisse), erfordert dieses spezifische Szenario nicht die Rückgabe von Ergebnissen aus der Cloud.



Systemanforderungen: OPC UA Server („Die Cloud“)

Die Cloud muss in der Lage sein, eine der folgenden Betriebssystemvarianten zu hosten. Beachten Sie dabei, dass die Verwendung von Virtualisierungstechnologie eine kostenintensivere Lösung für 64-Bit-Betriebssysteme erfordert.

Gerät ist eine virtuelle Maschine

- 32-Bit-Windows-Betriebssystem
- 64-Bit-Windows-Betriebssystem mit CPU Core Isolation (mindestens 2 Kerne erforderlich)

Gerät ist eine dedizierte Hardware

- 32-Bit-Windows-Betriebssystem
- 64-Bit-Windows-Betriebssystem

Zusätzlich müssen die folgenden Softwarekomponenten auf dem Gerät installiert werden:

- TwinCAT 3 XAR (nur Runtime) oder XAE (Runtime und Engineering)
- TwinCAT 3 Function TF6100 OPC UA

Beachten Sie, dass bei Installation eines kompletten TwinCAT 3 XAE weitere Konfigurationseinstellungen sinnvoll sein können, wie z. B. das Handling von Lizenzen oder die Möglichkeit zum Debuggen des Geräts direkt in der Cloud. Zum Speichern von Daten in einer zentralen Datenbank die von einem Client eingehen, müssen die folgenden Softwarekomponenten installiert sein:

- TwinCAT 3 Function TF6420 Database

Systemanforderungen: Dezentralisierter OPC UA Client

Der dezentralisierte OPC UA Client basiert auf einer TwinCAT-3-SPS-Laufzeit und kann daher auf jeder Hardwarekonfiguration gehostet werden, die den Betrieb einer TwinCAT-3-SPS unterstützt. Zudem muss die TwinCAT 3 Function TF6100 OPC UA auf dem Client-Gerät installiert werden, um den TwinCAT OPC UA Client nutzen zu können, um UA-Methoden auf dem Server in der Cloud auszuführen.

Technische Beschreibung

Die Realisierung eines OPC UA Servers in der Cloud ist sehr anwendungsspezifisch. Beckhoff stellt eine allgemeine Architekturbeschreibung zur Verfügung, unabhängig von dem Anwendungsfall im Einzelnen. Als Anwendungsbeispiel beschreibt dieser Artikel häufig ein Datenaggregationsszenario, bei dem Prozessdaten von dezentralisierten Clients über OPC UA zu einem zentralen Server übertragen werden, auf welchem die Daten in einer zentralen Datenbank angesammelt werden. Der Vorteil von OPC UA besteht in diesem Fall

nicht nur in der standardisierten Schnittstelle (und daher zur Nutzung dieses Konzepts für jede Art von OPC UA Client), sondern auch in der Möglichkeit, den Kommunikationskanal zu sichern, indem der integrierte Sicherheitsmechanismus von OPC UA eingesetzt wird.

Der TwinCAT OPC UA Server wird verwendet, um die in der IEC61131-3 SPS-definierten Methoden dem OPC-UA-Namensraum offenzulegen. Diese Methoden können von jedem TwinCAT OPC UA Client (auf der Grundlage der PLCopen-Funktionsbausteine) oder von 3rd-Party OPC UA Clients aufgerufen werden.

Dezentralisierter OPC UA Client

Der Hauptzweck des OPC UA Clients besteht darin, Methoden auf dem Remote-Server aufzurufen. Der Client ist mit großer Wahrscheinlichkeit ein TwinCAT-3-Embedded-Gerät, das den dezentralen Teil der Anwendung erfüllt. Dies würde für das Datenaggregations-Beispiel bedeuten, Prozessdaten abzufragen und sie durch Aufruf einer Methode an den Server zu senden.

Schnittstellenbeschreibung (zur Cloud)

Als Schnittstelle zu einem zentralisierten Cloud-System gibt es nur zwei vom Server bereitgestellte Methoden:

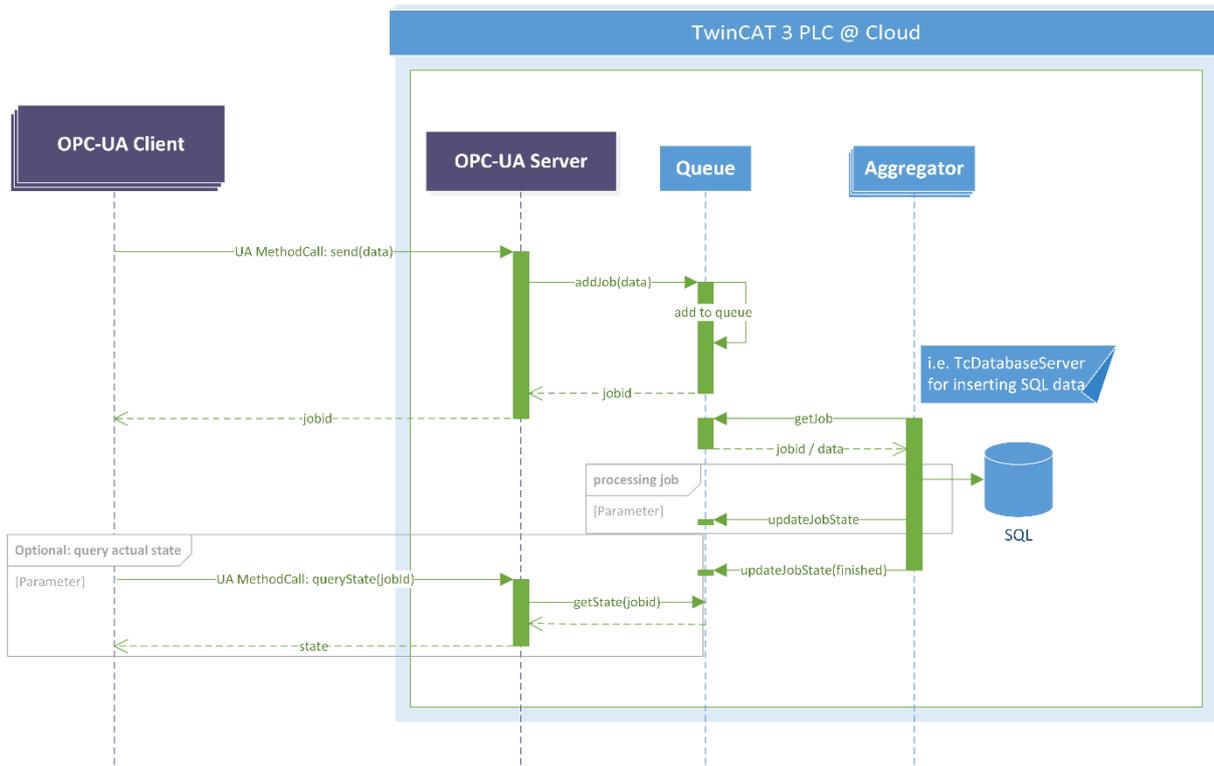
- `int Send(data)`: Diese Methode überträgt die Daten über einen OPC-UA-Methodenaufruf vor und gibt eine JobID zurück
- `int QueryState(jobID)`: Unter Verwendung der zuvor abgerufenen JobID kann der Client zyklisch den aktuellen Status des Jobs abfragen. Dies könnte ausgelassen werden, wenn der Server in der Cloud alle Situationen bearbeiten kann oder der Client bei Problemen ohnehin nicht aushelfen kann.

Die Kommunikation zwischen Clients und der Cloud kann durch die eingebauten Sicherheitskonzepte von OPC UA gesichert werden, z. B. durch die Verwendung von Client- und Server-Zertifikaten zur Verschlüsselung der Datenkommunikation.

Die Cloud

Der OPC UA Server in der Cloud basiert auf TwinCAT-3-SPS-Methoden, die dem OPC-UA-Namensraum offengelegt werden. Das SPS-Projekt enthält die folgenden drei Komponenten:

- **MethodCall Provider**: Diese Komponente bietet die zuvor erwähnte Send-Methode, bei der Daten gesammelt, eine JobID erstellt und die Daten als verschiedene Jobs in die Queue eingestellt werden. Darüber hinaus könnte es eine QueryState-Methode geben, um es OPC UA Clients zu ermöglichen, den aktuellen Jobstatus abzufragen.
- **Queue**: Die Warteschlange speichert eine Liste der Jobs, die ausgeführt werden sollen. Jeder Job in der Warteschlange enthält die folgenden Informationen: {ID, Status, Data}. Neue Elemente (Jobs) werden durch den MethodCall Provider hinzugefügt und können durch diesen auch gelöscht werden. Die gesamte Warteschlange kann innerhalb des SPS-Programms auf einer Hash-Tabelle basieren.
- **Aggregator**: Die Anwendung, die die Jobs verarbeitet. Sie schaut zyklisch in der Queue, ob es zu bearbeitende Jobs gibt. Wenn dies der Fall ist, übernimmt sie den Job (stellt den Status auf „verarbeitet“) und beginnt mit der Verarbeitung des Jobs, z. B. der Verbindung zu einer Datenbank über die Funktionsbausteine des TwinCAT Database Server. Beachten Sie, dass es mehr als einen Aggregator geben kann, um eine parallele Verarbeitung der Queue zu ermöglichen.



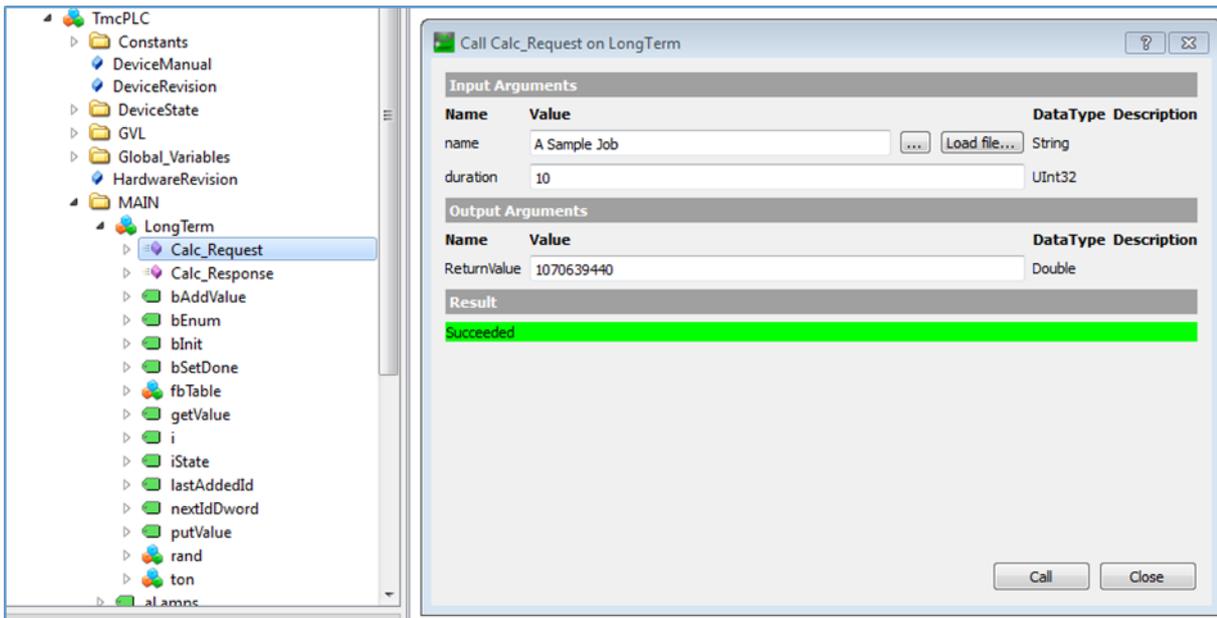
Beispiel

Das folgende Beispiel illustriert das Szenario durch die Ausführung einfacher Jobs: Jobs können dem Server von einem OPC UA Client vorgelegt werden. In diesem Beispiel bestehen die Jobdaten aus einem definierbaren Zeitintervall, das vom Aggregator aufgenommen wird, um den Job fertigzustellen. Das Beispiel besteht aus den folgenden SPS-Komponenten auf dem OPC UA Server:

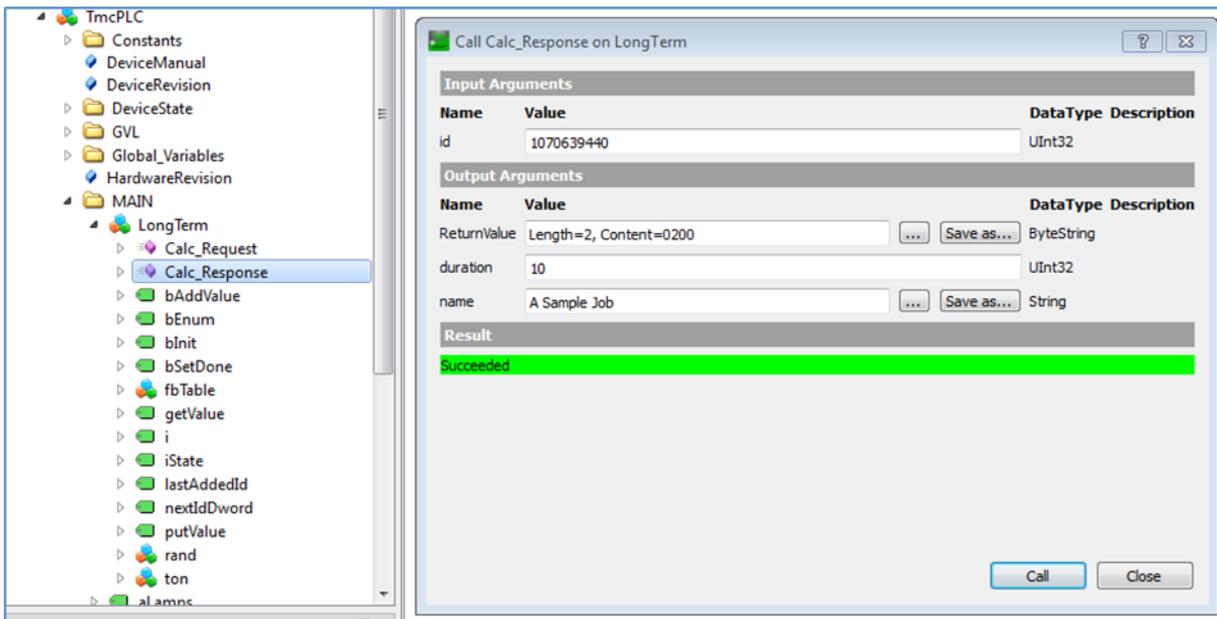
- **ST_JobEntry**: Repräsentiert einen Job in der Queue. Der Job besteht bei den Daten lediglich aus einem Namen und einer Zeitdauer. Eine Dauer definiert die Zeitspanne, die ein Job benötigt.
- **E_JobState**: Repräsentiert den Status eines Jobs. Die Beispielimplementierung definiert die folgenden Werte: QUEUED, PROCESSING, READY, FAILED und INVALID.
- **LongTerm**: Repräsentiert den MethodCall Server (besteht aus den Methoden Calc_Request und Calc_Response) sowie den Aggregator, der den Job verarbeitet (der im Funktionsbaustein implementiert ist)
- **FB_SpecialHashTableCtrl**: Repräsentiert die Queue in Form einer Hash-Tabelle, wie dies durch SPS-Beispiele vom Beckhoff Informationssystem wiedergegeben ist. Hier werden verschiedene Methoden zum Umgang mit der Warteschlange gegeben (Hinzufügen, Zählen, GetFirst, GetNext, Lookup, Entfernen, Zurücksetzen).

Verwendung des Beispiels

Der UA-Expert könnte als Beispiel-Client verwendet werden, um die Methoden aufzurufen, die vom OPC UA Server in der Cloud bereitgestellt werden. Durch Aufruf der Methode Calc_Request erhält der Client eine JobID (oder 0 zur Anzeige eines Fehlers):



Durch Aufruf der Methode Calc_Response mit der JobID kann der Client den JobState abfragen. Zudem werden die vorher eingestellte Dauer und der Jobname als spätere Referenz zurückgegeben.



Installation

Das folgende Kapitel beschreibt die Installation und Konfiguration jeder Softwarekomponente, die auf dem Server in der Cloud erforderlich ist.

Nur Runtime

Wenn der Server nicht die Engineering-Umgebung von TwinCAT 3 hostet, muss die TwinCAT-3-XAR-Installation verwendet werden. Beachten Sie, dass in diesem Fall das gesamte Handling zur ordnungsgemäßen Funktion mehrere Schritte involviert und die zukünftige Wartung ggf. schwieriger ist, da der XAR-Installation die Programmierungs- und Debugging-Umgebung fehlt. In diesem Szenario muss der Benutzer sicherstellen, dass die Pflege der ADS-Routen zwischen dem tatsächlichen Engineering-Computer (auf dem das SPS-Programm für den Server entwickelt wird) und dem Gerät, das den Server in der Cloud hostet, nicht nur den Download und das Debugging des SPS-Programms erlauben muss, sondern auch die Aktivierung von Lizenzen für die TwinCAT-3-Laufzeit. Beachten Sie, dass Sie die Firewall-Ports öffnen müssen, um ADS-Verkehr zu ermöglichen. In der ADS-Dokumentation erhalten Sie weitere Informationen. Zur einfacheren Handhabung der TwinCAT-3-Laufzeitlizenzen wurde das Tool

TC3 License Request Generator entwickelt. Dieses kann über den Beckhoff FTP-Server heruntergeladen werden und ermöglicht die Erstellung von License Request Files und den Import von License Response Files:

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

Runtime und Engineering

Dies ist das empfohlene Setup-Szenario. Der Server in der Cloud hostet eine TwinCAT 3 Runtime und die entsprechenden Engineering-Komponenten, um das Debuggen und eine einfachere Handhabung des Laufzeitsystems zu ermöglichen. In diesem Fall ist es erforderlich, TwinCAT 3 XAE auf dem System zu installieren.

Zusätzliche Software

Nach erfolgreicher Installation von TwinCAT XAE oder XAR müssen die folgenden Softwarekomponenten auf dem Gerät installiert und konfiguriert werden:

TF6100 OPC UA

Die Function TF6100 installiert einen OPC UA Server (und Client) auf dem Gerät. Der Server ist erforderlich, um den OPC UA Clients die SPS-Methoden zur Verfügung zu stellen. Wenn das SPS-Programm in die Laufzeit heruntergeladen wird, importiert der OPC UA Server automatisch die erste SPS-Laufzeit in seinen Namensraum. Sämtliche Variablen und Methoden der SPS, die als verfügbar über OPC UA gekennzeichnet sind, werden in den OPC-UA-Namensraum importiert und werden für Clients verfügbar. In der TF6100-Dokumentation erhalten Sie weitere Informationen.

TF6420 Database

Wenn der Server die von den Clients erhaltenen Daten in einer Datenbank speichern soll, muss die TF6420-Function verwendet werden. Sie bietet generische Funktionsblöcke für die TwinCAT-3-SPS für den Zugriff auf die Datenbank, z. B. um Werte in eine Datenbanktabelle einzufügen. Installieren Sie das TF6420-Setup und konsultieren Sie die TF6420-Dokumentation für weitere Informationen zur Nutzung der entsprechenden Funktionsbausteine.

Lizenzierung

Alle TwinCAT-3-Softwareprodukte erfordern eine Lizenz, um auf dem Server verfügbar zu sein. Die Lizenz kann entweder aus einer 7-Tage-Testlizenz oder aus einer uneingeschränkten Lizenz bestehen. Lizenzen können unter Verwendung des TwinCAT 3 XAE Lizenzmanagers aktiviert werden. Wenn eine TwinCAT-3-XAR-Installation auf dem Server verwendet wird, verwenden Sie den TC3 License Generator.

3 Installation

3.1 Setupübersicht

Für eine bessere Komponentisierung und schnellere Releasezyklen wird das Produkt in mehreren Setups ausgeliefert. Hierbei hat jede Produktkomponente ein eigenes Setup. Die folgende Tabelle gibt hierzu einen Überblick.

Setup-Name	Beschreibung
TF6100-OPC-UA-Server	Beinhaltet den TwinCAT OPC UA Server und SampleClient.
TF6100-OPC-UA-Client	Beinhaltet den TwinCAT OPC UA Client.
TF6100-OPC-UA-Configurator	Beinhaltet den TwinCAT OPC UA Configurator als Visual Studio Extension und Standalone Applikation.
TF6100-OPC-UA-Gateway	Beinhaltet das TwinCAT OPC UA Gateway.

● Updateinstallation

i Bei einer Updateinstallation wird immer die vorherige Installation deinstalliert. Bitte stellen Sie sicher, dass Sie vorher ein Backup Ihrer Konfigurationsdateien erstellt haben.

● CE-ARMV4I-LF

i Beim TwinCAT OPC UA Server wird eine CAB-Installationsdatei für Windows CE ausgeliefert, welche mit dem Suffix CE-ARMV4I-LF versehen ist. Hierbei handelt es sich um eine eingeschränkte Servervariante für Hardwareplattformen mit wenig Arbeitsspeicher. In dieser Variante sind bestimmte OPC UA Features (z.B. Historical Access und Alarms & Conditions) deaktiviert um die Arbeitsspeicherauslastung so gering wie möglich zu halten.

3.2 Systemvoraussetzungen

OPC UA Server

Technische Daten	Beschreibung
Betriebssystem	Windows 7, 10 Windows CE 6/7 Windows Embedded 7 Windows Server
Zielplattformen	PC-Architektur (x86, x64, ARM)
.NET Framework	4.6.1 (nur benötigt für das SysTray-Icon)
TwinCAT-Version	TwinCAT 2, TwinCAT 3
TwinCAT-Installationslevel	TwinCAT 2 CP, PLC, NC-PTP TwinCAT 3 XAE, XAR, ADS
Benötigte TwinCAT-Lizenz	TS6100 TwinCAT OPC UA (für TwinCAT 2) TF6100 TC3 OPC UA (für TwinCAT 3)

OPC UA Client

Technische Daten	Beschreibung
Betriebssystem	Windows 7, 10 Windows CE 6/7 Windows Embedded 7 Windows Server
Zielplattformen	PC-Architektur (x86, x64, ARM)
.NET Framework	---
Minimale TwinCAT-Version	TwinCAT 2, TwinCAT 3
Minimales TwinCAT-Installationslevel	TwinCAT 2 PLC, NC-PTP TwinCAT 3 XAE, XAR
Benötigte TwinCAT-Lizenz	TS6100 TwinCAT OPC UA (für TwinCAT 2) TF6100 TC3 OPC UA (für TwinCAT 3)

OPC UA I/O Client

Technische Daten	Beschreibung
Betriebssystem	Windows 7, 10 Windows CE 7 Windows Embedded 7 Windows Server
Zielplattformen	PC-Architektur (x86, x64, ARM)
.NET Framework	4.6 (only for namespace browser)
Minimale TwinCAT-Version	TwinCAT 3.1 Build 4022.4
Minimales TwinCAT-Installationslevel	TwinCAT 3 XAE, XAR
Benötigte TwinCAT-Lizenz	TF6100 TC3 OPC UA

OPC UA Gateway

Technische Daten	Beschreibung
Betriebssystem	Windows 10 Windows Embedded 7 Windows Server
Zielplattformen	PC-Architektur (x86, x64, ARM)
.NET Framework	---
Minimale TwinCAT-Version	---
Minimales TwinCAT-Installationslevel	---
Benötigte TwinCAT-Lizenz	---

OPC UA Configurator Visual Studio

OPC UA Configurator	Beschreibung
Betriebssystem	Windows 10 Windows Embedded 7 Windows Server
Zielplattformen	PC-Architektur (x86, x64)
Visual Studio Abhängigkeit	Installation des .NET-Targeting Package 4 über den Visual Studio Installer
Unterstützte Visual Studio Versionen	2017, 2019
.NET Framework	4.6.1
Minimales TwinCAT-Installationslevel	TwinCAT 3 XAE
Benötigte TwinCAT-Lizenz	---

OPC UA Configurator Standalone

OPC UA Configurator	Beschreibung
Betriebssystem	Windows 10 Windows Embedded 7 Windows Server
Zielplattformen	PC-Architektur (x86, x64)
.NET Framework	4.6.1
Minimales TwinCAT-Installationslevel	TwinCAT 2 CP, PLC, NC-PTP TwinCAT 3 XAE, XAR, ADS
Benötigte TwinCAT-Lizenz	---

OPC UA Sample Client

OPC UA Sample Client	Beschreibung
Betriebssystem	Windows 10 Windows Embedded 7 Windows Server
Zielplattformen	PC-Architektur (x86, x64)
.NET Framework	4.6.1
Minimale TwinCAT-Version	---
Minimales TwinCAT-Installationslevel	---
Benötigte TwinCAT-Lizenz	---

3.3 Installation (TC3)

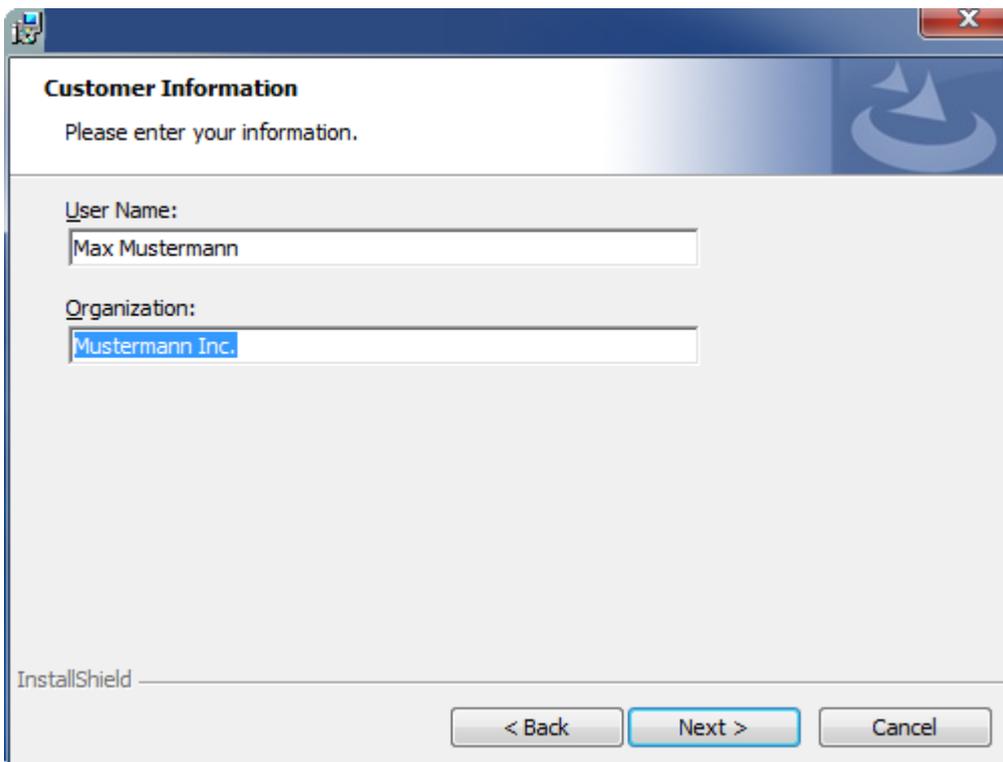
Nachfolgend wird beschrieben, wie die TwinCAT 3 Function TF6100 OPC UA für Windows-basierte Betriebssysteme installiert wird.

- ✓ Die Setup-Datei der TwinCAT 3 Function wurde von der Beckhoff-Homepage heruntergeladen.
- 1. Führen Sie die Setup-Datei als Administrator aus. Wählen Sie dazu im Kontextmenü der Datei den Befehl **Als Administrator ausführen**.
 - ⇒ Der Installationsdialog öffnet sich.

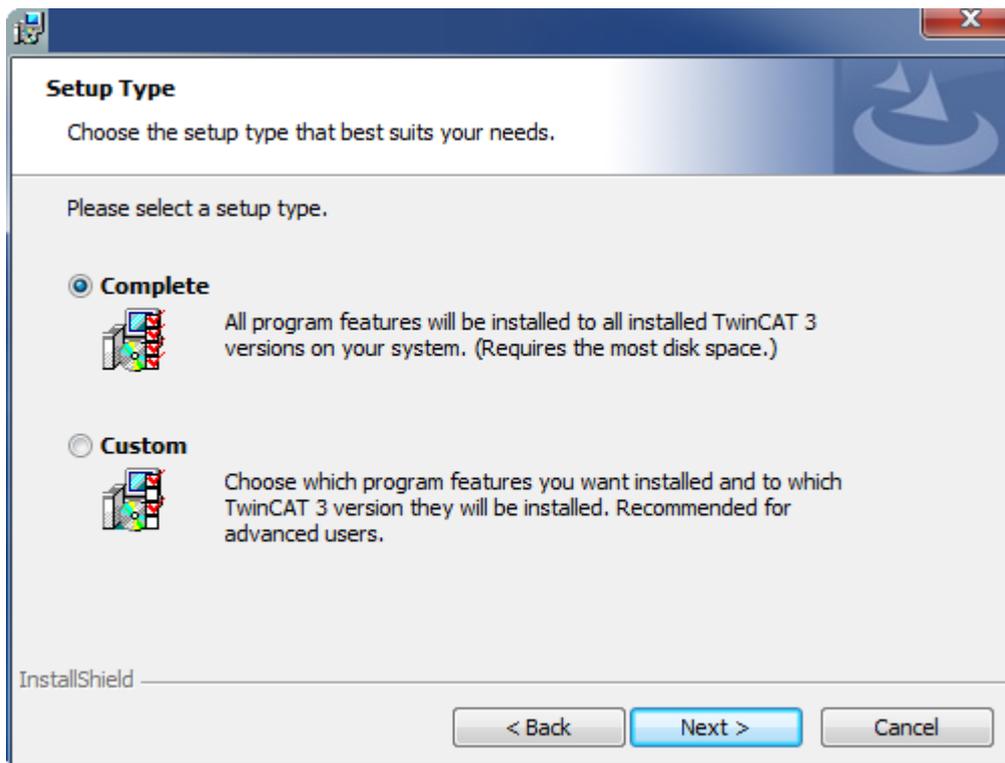
2. Akzeptieren Sie die Endbenutzerbedingungen und klicken Sie auf **Next**.



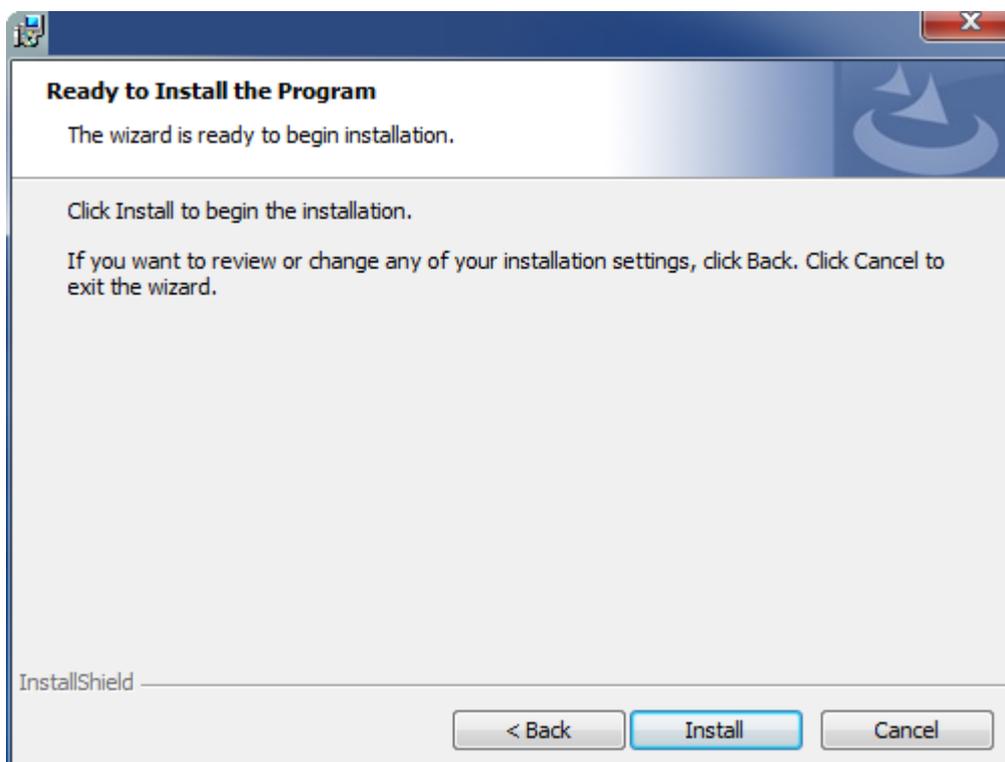
3. Geben Sie Ihre Benutzerdaten ein.



4. Wenn Sie die TwinCAT 3 Function vollständig installieren möchten, wählen Sie **Complete** als Installationstyp. Wenn Sie die Komponenten der TwinCAT 3 Function separat installieren möchten, wählen Sie **Custom**.

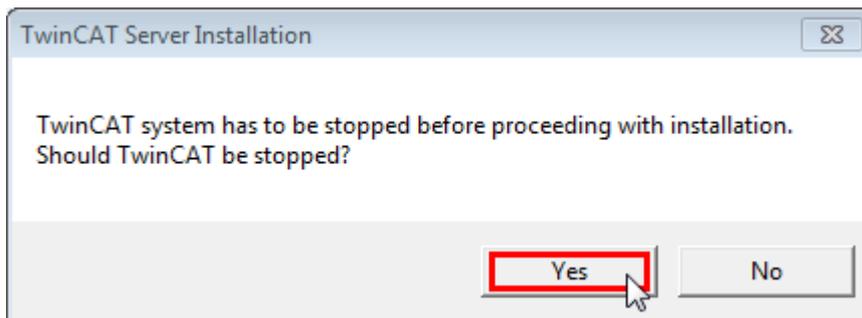


5. Wählen Sie **Next** und anschließend **Install**, um die Installation zu beginnen.

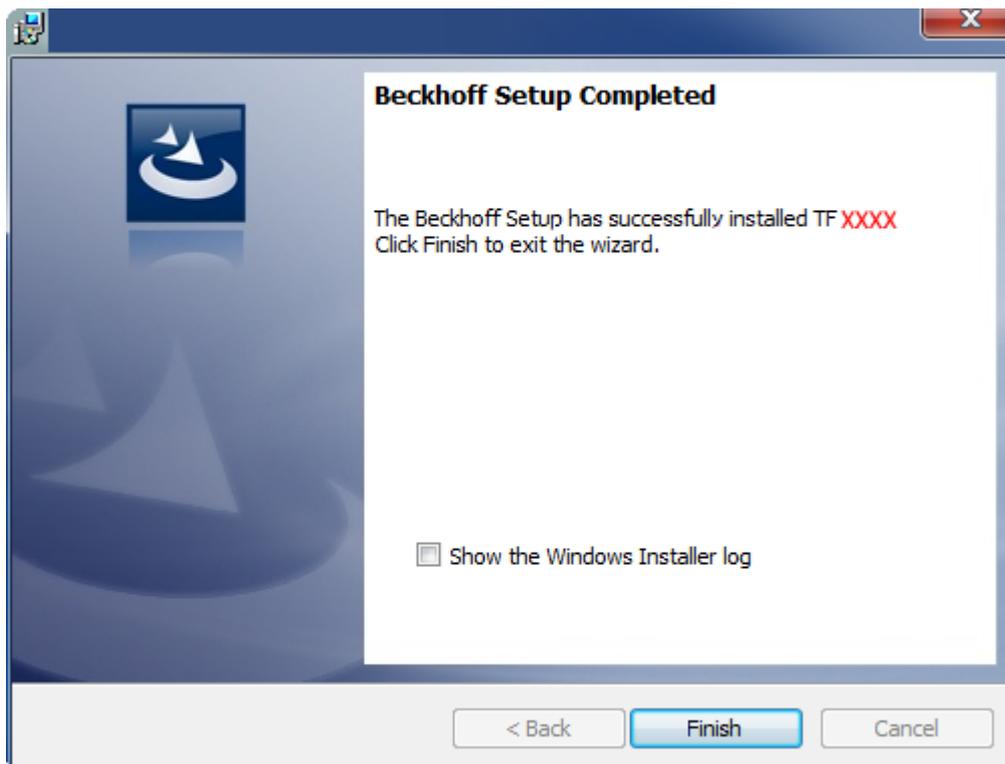


- ⇒ Ein Dialog weist Sie darauf hin, dass das TwinCAT-System für die weitere Installation gestoppt werden muss.

6. Bestätigen Sie den Dialog mit **Yes**.



7. Wählen Sie **Finish**, um das Setup zu beenden.



⇒ Die TwinCAT 3 Function wurde erfolgreich installiert und kann lizenziert werden (siehe [Lizenzierung \(TC3\)](#) [▶ 32]).

3.4 Installation (TC2)

Nachfolgend wird beschrieben, wie das TwinCAT 2 Supplement TwinCAT OPC UA für Windows-basierte Betriebssysteme installiert wird.

- [Download der Setup-Datei und Installation](#) [▶ 22]
- [Nach der Installation](#) [▶ 25]

Download der Setup-Datei und Installation

Genauso wie viele andere TwinCAT-Ergänzungsprodukte steht OPC UA als 30-Tage-Demoversion oder als Vollversion zum Download auf der Beckhoff Homepage www.beckhoff.com zur Verfügung.

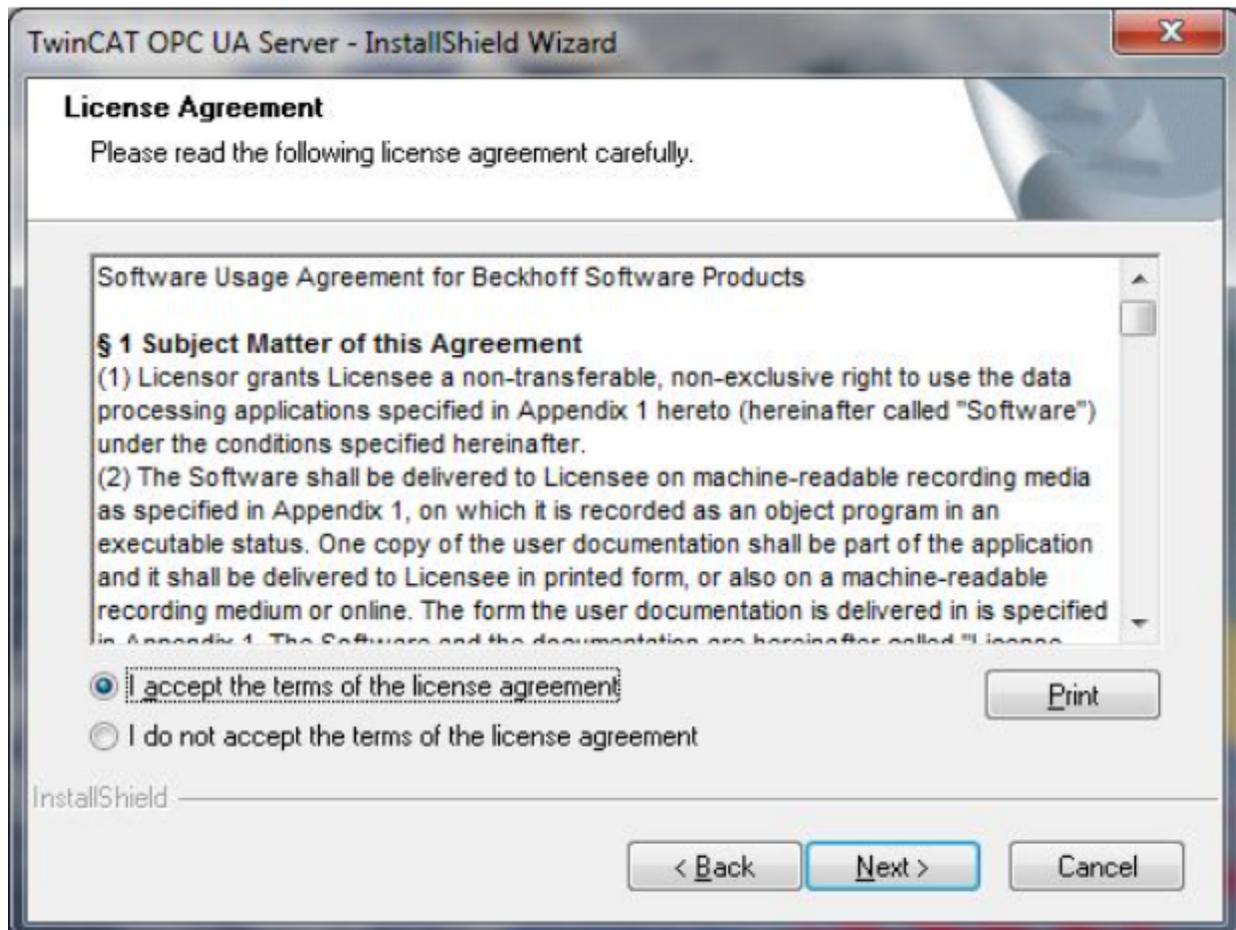
✓ Die Setup-Datei des TwinCAT 2 Supplement wurde von der Beckhoff-Homepage heruntergeladen.

1. Führen Sie die heruntergeladene Setup-Datei *TcOpcUaSvr.exe* als Administrator aus. Wählen Sie dazu im Kontextmenü der Datei den Befehl **Als Administrator ausführen**.

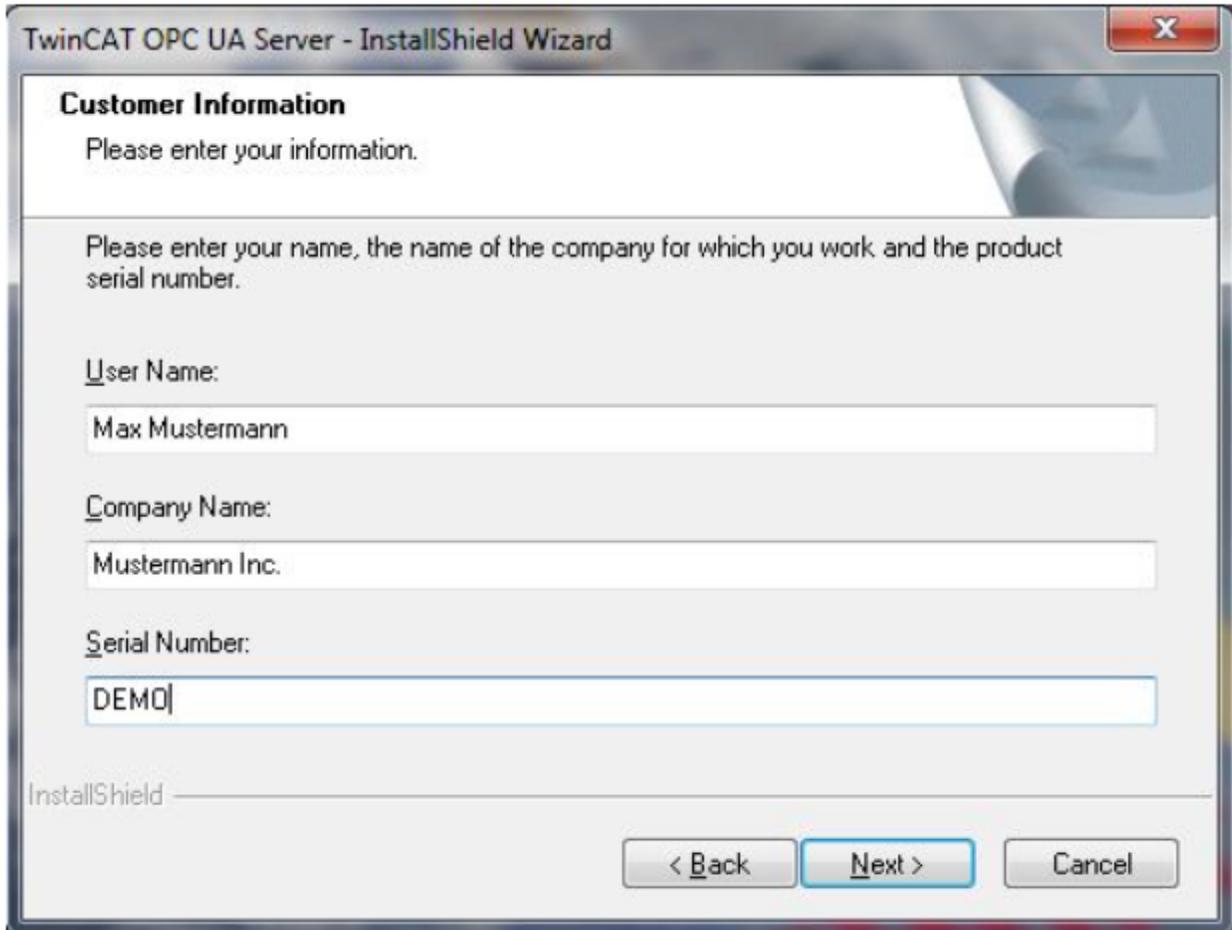
⇒ Der Installationsdialog öffnet sich.

2. Wählen Sie eine Installationssprache.

3. Klicken Sie auf **Weiter** und stimmen Sie der Lizenzvereinbarung zu.



4. Geben Sie Ihre Benutzerinformationen ein. Alle Felder müssen ausgefüllt werden. Wenn Sie eine 30-Tage-Demo installieren möchten, geben Sie bitte „DEMO“ als Lizenzschlüssel ein.



TwinCAT OPC UA Server - InstallShield Wizard

Customer Information
Please enter your information.

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

User Name:
Max Mustermann

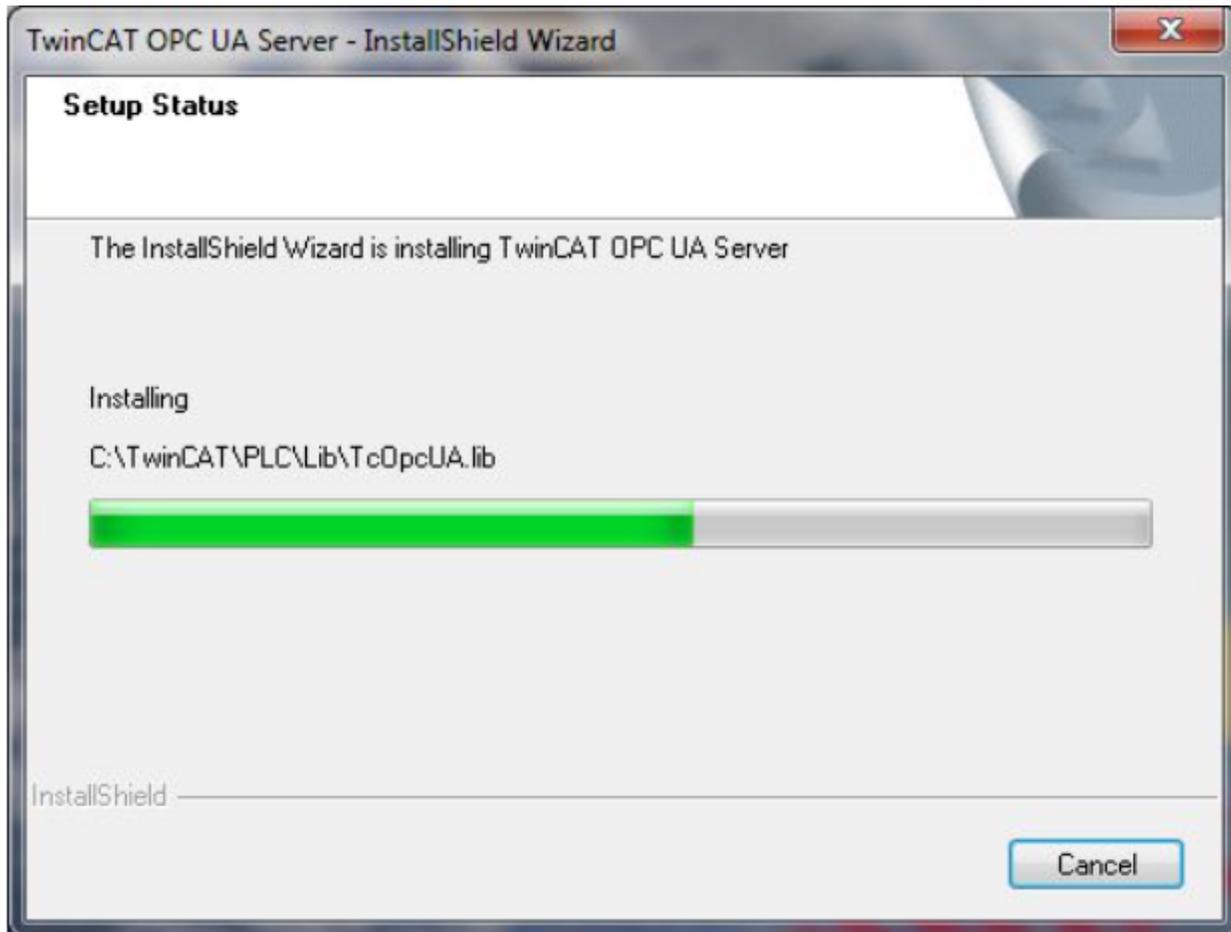
Company Name:
Mustermann Inc.

Serial Number:
DEMO

InstallShield

< Back Next > Cancel

5. Klicken Sie auf **Installieren**, um die Installation zu starten.



6. Starten Sie den Computer neu, um die Installation abzuschließen.

Nach der Installation

Die TwinCAT-Ergänzung OPC UA wird automatisch während der Installation konfiguriert und es sind keine weiteren Einstellungen für die Nutzung dieses Produkts erforderlich. Zu den weiteren Schritten gehören gegebenenfalls:

- Stellen Sie eine erste Verbindung mit dem installierten UA-Server her und testen Sie dessen Konfiguration mit dem OPC UA Sample Client. (Siehe [Sample Client](#) [▶ 209])
- Personalisieren Sie das UA-Server-Setup mithilfe des OPC-UA-Konfigurators. (Siehe [Konfigurator](#) [▶ 115])
- Vergewissern Sie sich zudem, dass Ihre Firewall den TCP Port 4840 öffnet, weil der OPC UC Server diesen Port benötigt.

3.5 Installation Windows CE (TC3)

Nachfolgend wird beschrieben, wie die TwinCAT 3 Function TF6100 OPC UA auf einem Beckhoff Embedded-PC mit Windows CE installiert wird.

- [Download der Setup-Datei und Installation](#) [▶ 26]
- [CAB-Datei auf das Windows-CE-Gerät übertragen](#) [▶ 26]
- [CAB-Datei auf dem Windows-CE-Gerät ausführen](#) [▶ 26]

Wenn schon eine ältere TF6100-Version auf dem Windows-CE-Gerät installiert ist, kann diese aktualisiert werden:

- [Upgrade der Software](#) [▶ 27]

Download der Setup-Datei und Installation

Die für Windows CE ausführbare Datei ist Teil des TF6100-OPC-UA-Setups. Dieses wird Ihnen auf der Beckhoff Homepage www.beckhoff.com zur Verfügung gestellt und enthält automatisch alle Versionen für Windows XP, Windows 7 und Windows CE (x86 und ARM).

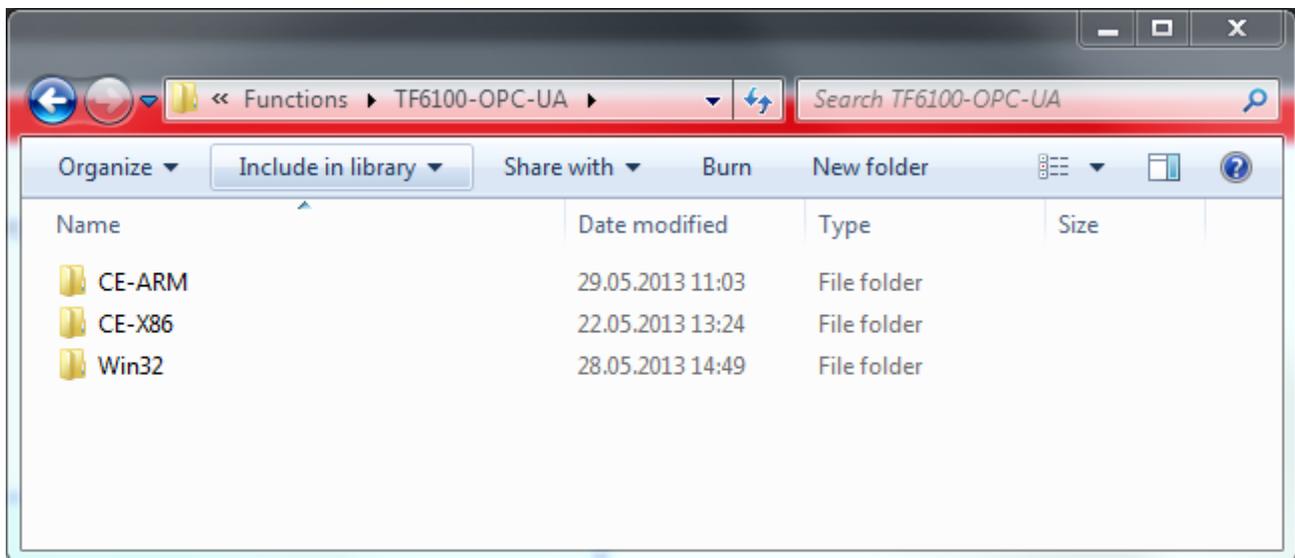
Laden Sie das Setup herunter und installieren Sie die Function wie es im Abschnitt Installation beschrieben wird.

Nach der Installation enthält der Installationsordner drei Verzeichnisse (pro Hardware-Plattform ein Verzeichnis)

- **CE-ARM:** ARM-basierte Embedded-PC, die unter Windows CE laufen, z. B. CX8090, CX9020
- **CE-X86:** X86-basierte Embedded-PC, die unter Windows CE laufen, z. B. CX50xx, CX20x0
- **Win32:** Embedded-PC, die unter Windows XP, Windows 7 oder Windows Embedded Standard laufen

Die Verzeichnisse CE-ARM und CE-X86 enthalten die CAB-Dateien der TwinCAT 3 Function für Windows CE in Bezug auf die jeweilige Hardware-Plattform des Windows-CE-Gerätes.

Beispiel: Installationsordner „TF6310“



CAB-Datei auf das Windows-CE-Gerät übertragen

Übertragen Sie die entsprechende ausführbare Datei auf das Windows-CE-Gerät.

Für die Übertragung der ausführbaren Datei stehen Ihnen verschiedene Möglichkeiten zur Verfügung:

- über Netzwerkfreigaben
- über den integrierten FTP-Server
- über ActiveSync
- über CF/SD-Karten

Weitere Informationen finden Sie im Beckhoff Information System in der Dokumentation „Betriebssysteme“ (Embedded-PC> Betriebssysteme > [CE](#)).

CAB-Datei auf dem Windows-CE-Gerät ausführen

Nachdem Sie die CAB-Datei auf das Windows-CE-Gerät übertragen haben, führen Sie die Datei dort mit einem Doppelklick aus. Bestätigen Sie den Installationsdialog mit **OK**. Starten Sie das Windows-CE-Gerät anschließend neu.

Nach dem Neustart des Gerätes werden die Dateien (Client und Server) automatisch im Hintergrund geladen und sind verfügbar.

Die Software wird in dem folgenden Verzeichnis auf dem Windows-CE-Gerät installiert:
`\Hard Disk\TwinCAT\Functions\TF6310-TCP-IP`

Upgrade der Software

Wenn auf dem Windows-CE-Gerät bereits eine ältere TF6100-Version installiert ist, führen Sie die folgenden Schritte auf dem Windows-CE-Gerät durch, um ein Upgrade auf eine neue Version durchzuführen:

1. Öffnen Sie den CE Explorer, indem Sie auf **Start > Run** klicken und „Explorer“ eingeben.
 2. Navigieren Sie nach `\Hard Disk\TwinCAT\Functions\Tf6100-OPC-UA\Server` bzw. (in einem zweiten Schritt) `\Hard Disk\TwinCAT\Functions\Tf6100-OPC-UA\Client`.
 3. Benennen Sie die Datei `TcOpcUaServer.exe` bzw. `TcOpcUaClient.exe` um.
 4. Starten Sie das Windows-CE-Gerät neu.
 5. Übertragen Sie die neue CAB-Datei auf das Windows-CE-Gerät.
 6. Führen Sie die CAB-Datei auf dem Windows-CE-Gerät aus und installieren Sie die neue Version.
 7. Löschen Sie die alten (umbenannten) Dateien.
 8. Starten Sie das Windows-CE-Gerät neu.
- ⇒ Nach dem Neustart ist die neue Version aktiv.

3.6 Installation Windows CE (TC2)

Nachfolgend wird beschrieben, wie das TwinCAT 2 Supplement auf einem Beckhoff Embedded-PC mit Windows CE, z. B. CX1000, CX1020, CX9000, CX9001, CX9010, CX8090, CP62xx, CP69xx, installiert wird.

- [Download der Setup-Datei und Installation](#) [▶ 27]
- [Setup-Datei auf ein Windows-CE-Gerät übertragen](#) [▶ 30]
- [Setup-Datei auf dem Windows-CE-Gerät ausführen](#) [▶ 30]

● Installation auf kleinen Embedded-Plattformen (CX9001, CX9010)

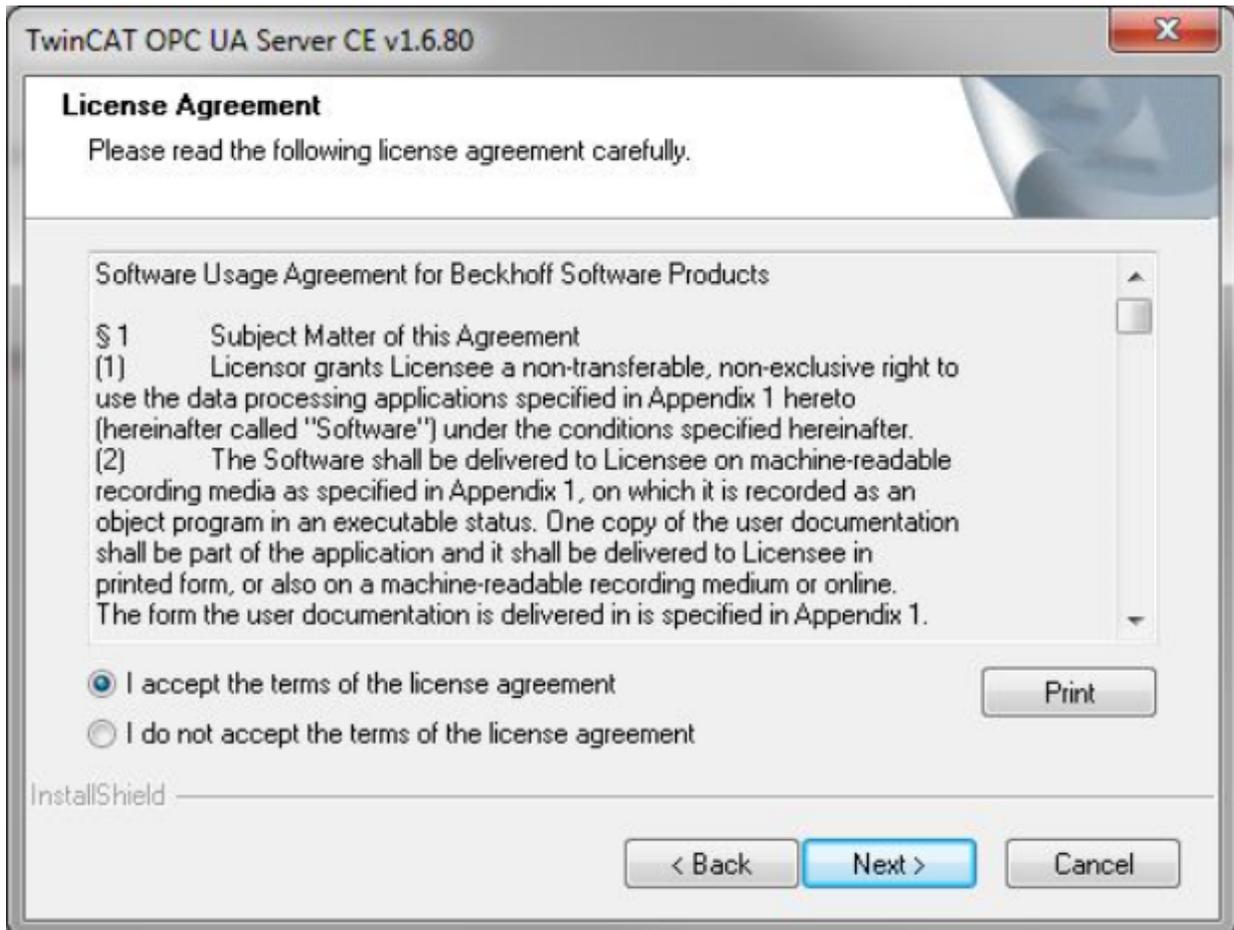
I Sehr kleine Embedded-Geräte erfordern möglicherweise einige zusätzliche manuelle Schritte, damit die CAB-Datei installiert werden kann. Zu diesen Schritten kann z. B. das Löschen von unnötigen Dateien im Speicher der Geräte gehören, damit ausreichend Platz bereitsteht, um alle Dateien der Anwendung zu installieren.

Download der Setup-Datei und Installation

Genauso wie viele andere TwinCAT-Ergänzungsprodukte steht OPC UA für CE als Download auf dem Beckhoff Homepage zur Verfügung. Damit Sie die Installationsdateien für Windows CE erhalten können, müssen Sie zunächst die heruntergeladene Setup-Datei auf einem Host-Computer installieren. Dieser kann ein beliebiges Windows-CE-basiertes System sein kann.

1. Doppelklicken Sie auf die heruntergeladene Datei `TcOpcUaSvrCE.exe`.
2. Wählen Sie eine Installationssprache.

3. Klicken Sie auf **Weiter** und stimmen Sie der Lizenzvereinbarung zu.



4. Geben Sie ihre Benutzerinformationen ein. Alle Felder müssen ausgefüllt werden. Beachten Sie, dass OPC UA für CE derzeit nicht als Demo-Version verfügbar ist. Demzufolge benötigen Sie einen gültigen Lizenzschlüssel, um mit der Installierung fortzufahren.

TwinCAT OPC UA Server CE v1.6.80

Customer Information
Please enter your information.

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

User Name:
Max Mustermann

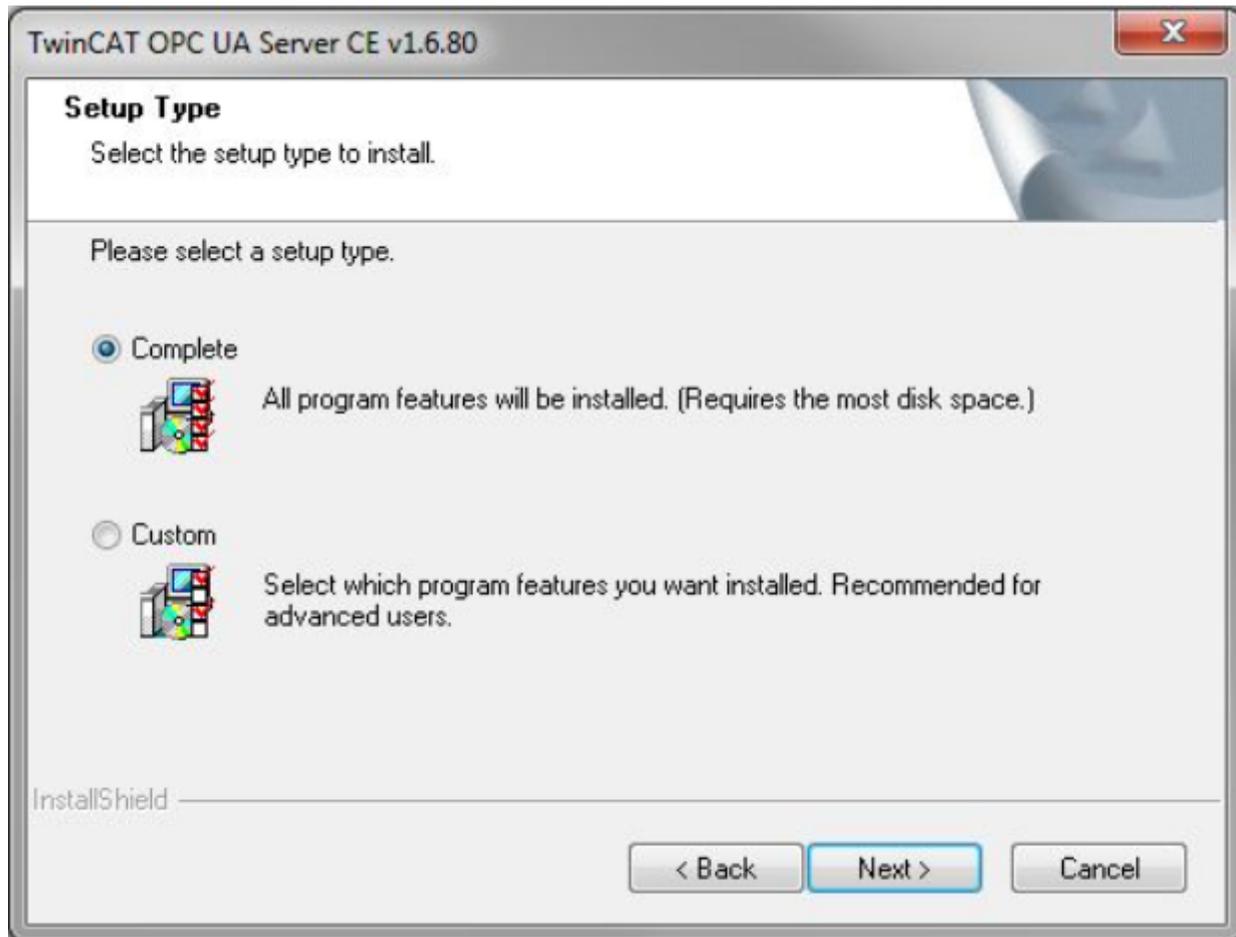
Company Name:
Mustermann Inc.

Serial Number:
xxx-xxx-xxx-xxx

InstallShield

< Back Next > Cancel

5. Wählen Sie als Installationsart **Vollständig** und klicken Sie auf **Weiter**



6. Klicken Sie auf **Installieren**, um die Installation zu starten.

⇒ Nach der Installierung finden Sie die Setup-Dateien für Windows CE im Verzeichnis `...\\TwinCAT\\CE`. Dieses Verzeichnis beinhaltet Setup-Dateien für die folgenden CE-Plattformen:

- TwinCAT OPC UA Client CE\\I586: OPC UA Client (SPS-Bibliothek) für x86 basierte CPUs (wie CX10xx, CP62xx, C69xx, ...)
- TwinCAT OPC UA Client CE\\ARMV4I: OPC UA Client (SPS-Bibliothek) für ARM-basierte CPUs (wie CX9001, CX9010, CP6608, ...)
- TwinCAT OPC UA Server CE\\I586: OPC UA Server für x86-basierte CPUs (wie CX10xx, CP62xx, C69xx, ...)
- TwinCAT OPC UA Server CE\\ARMV4I: OPC UA Server für ARM-basierte CPUs (wie CX9001, CX9010, CP6608, ...)

Setup-Datei auf das Windows-CE-Gerät übertragen

Übertragen Sie die entsprechende ausführbare Datei auf Ihr Windows-CE-Gerät. Für die Übertragung der Setup-Datei stehen Ihnen verschiedene Möglichkeiten zur Verfügung:

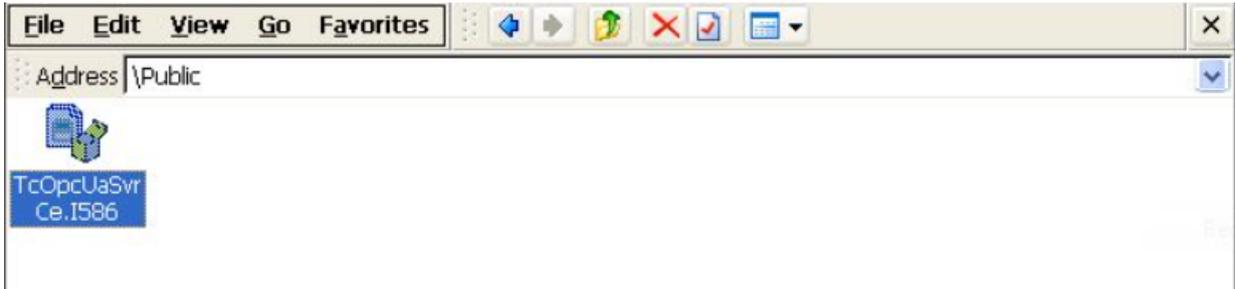
- über einen freigegebenen Ordner
- über den integrierten FTP-Server
- über ActiveSync
- über eine CF-Karte

Weitere Informationen finden Sie im Beckhoff Information System in der Dokumentation „Betriebssysteme“ (Embedded-PC > Betriebssysteme > CE)

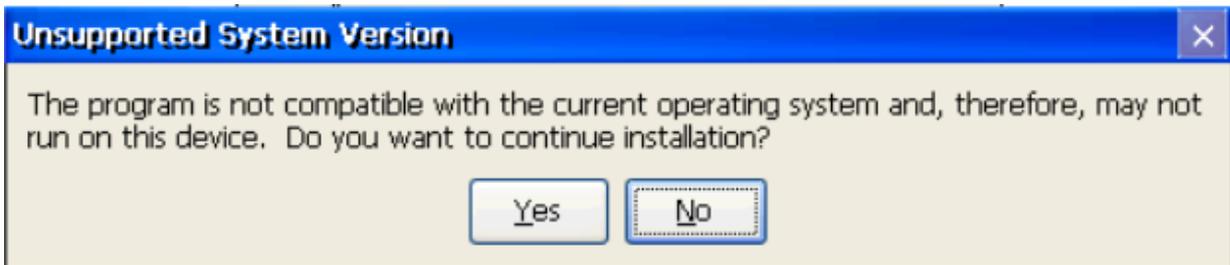
Setup-Datei auf dem Windows-CE-Gerät ausführen

Führen Sie die übertragene Setup-Datei `TcOpcUaSvrCe.xxxx.CAB` auf dem CE-Gerät aus:

1. Navigieren Sie zu dem Verzeichnis, in das Sie die Setup-Datei übertragen haben.



2. Doppelklicken Sie auf die CAB-Datei. Wird ein Meldungsdialog eingeblendet, dass dieses Programm nicht mit dem aktuellen Betriebssystem kompatibel ist, vergewissern Sie sich, dass Sie die richtige CAB-Datei (ARM, I586) für Ihren IPC/Embedded-PC verwenden.
3. Wenn Sie sicher sind, dass die CAB-Datei zum Embedded-PC/IPC passt, bestätigen Sie diesen Meldungsdialog mit **Ja**.



4. Wählen Sie \\Hard Disk\System\ als Zielverzeichnis



5. Klicken Sie auf **OK**, um die Installation zu starten.



⇒ Nach der Installierung wird die Setup-Datei automatisch gelöscht.

Wenn Sie den OPC UA Server installiert haben, wird diese Komponente nach einem Neustart Ihres Windows-CE-Geräts verfügbar.

3.7 Lizenzierung (TC3)

Die TwinCAT 3 Function ist als Vollversion oder als 7-Tage-Testversion freischaltbar. Beide Lizenztypen sind über die TwinCAT-3-Entwicklungsumgebung (XAE) aktivierbar.

Nachfolgend wird die Lizenzierung einer TwinCAT 3 Function beschrieben. Die Beschreibung gliedert sich dabei in die folgenden Abschnitte:

- [Lizenzierung einer 7-Tage Testversion \[► 32\]](#)
- [Lizenzierung einer Vollversion \[► 34\]](#)

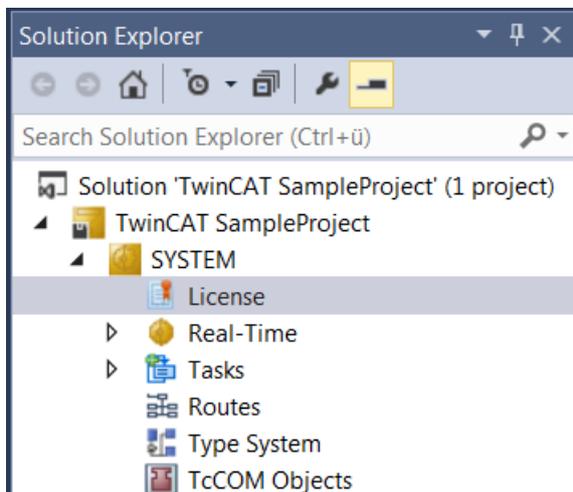
Weitere Informationen zur TwinCAT-3-Lizenzierung finden Sie im Beckhoff Information System in der Dokumentation „Lizenzierung“ (TwinCAT 3 > [Lizenzierung](#)).

Lizenzierung der 7-Tage-Testversion einer TwinCAT 3 Function



Eine 7-Tage-Testversion kann nicht für einen [TwinCAT-3-Lizenz-Dongle](#) freigeschaltet werden.

1. Starten Sie die TwinCAT-3-Entwicklungsumgebung (XAE).
2. Öffnen Sie ein bestehendes TwinCAT-3-Projekt oder legen Sie ein neues Projekt an.
3. Wenn Sie die Lizenz für ein Remote-Gerät aktivieren wollen, stellen Sie das gewünschte Zielsystem ein. Wählen Sie dazu in der Symbolleiste in der Drop-down-Liste **Choose Target System** das Zielsystem aus.
 - ⇒ Die Lizenzierungseinstellungen beziehen sich immer auf das eingestellte Zielsystem. Mit der Aktivierung des Projekts auf dem Zielsystem werden automatisch auch die zugehörigen TwinCAT-3-Lizenzen auf dieses System kopiert.
4. Klicken Sie im **Solution Explorer** im Teilbaum **SYSTEM** doppelt auf **License**.



⇒ Der TwinCAT-3-Lizenzmanager öffnet sich.

- Öffnen Sie die Registerkarte **Manage Licenses**. Aktivieren Sie in der Spalte **Add License** das Auswahlkästchen für die Lizenz, die Sie Ihrem Projekt hinzufügen möchten (z. B. „TF4100 TC3 Controller Toolbox“).

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

- Öffnen Sie die Registerkarte **Order Information (Runtime)**.
 ⇒ In der tabellarischen Übersicht der Lizenzen wird die zuvor ausgewählte Lizenz mit dem Status „missing“ angezeigt.
- Klicken Sie auf **7 Days Trial License...**, um die 7-Tage-Testlizenz zu aktivieren.

- Es öffnet sich ein Dialog, der Sie auffordert, den im Dialog angezeigten Sicherheitscode einzugeben.

- Geben Sie den Code genauso ein, wie er angezeigt wird, und bestätigen Sie ihn.
- Bestätigen Sie den nachfolgenden Dialog, der Sie auf die erfolgreiche Aktivierung hinweist.

⇒ In der tabellarischen Übersicht der Lizenzen gibt der Lizenzstatus nun das Ablaufdatum der Lizenz an.

10. Starten Sie das TwinCAT-System neu.

⇒ Die 7-Tage-Testversion ist freigeschaltet.

Lizenzierung der Vollversion einer TwinCAT 3 Function

Die Beschreibung der Lizenzierung einer Vollversion finden Sie im Beckhoff Information System in der Dokumentation „[TwinCAT 3 Lizenzierung](#)“.

3.8 Lizenzierung (TC2)

Die Lizenzierung der TwinCAT OPC UA für TwinCAT 2 erfolgt im Rahmen der [Installation](#) [► 22] über die Eingabe eines Lizenzschlüssels.

4 Technische Einführung

4.1 Server

4.1.1 Übersicht

Der TwinCAT OPC UA Server stellt eine standardisierte Kommunikationsschnittstelle für den Zugriff auf Symbolwerte aus der TwinCAT Laufzeit bereit. Dies ermöglicht eine einfache Anbindung von Fremdsoftware zum Auslesen oder Schreiben von Variablenwerten. Dieser Teil der Dokumentation beschreibt die verschiedenen Konfigurationsmöglichkeiten des TwinCAT OPC UA Servers.

Für einen Schnelleinstieg empfehlen wir unseren [QuickStart \[► 35\]](#) Artikel. Bitte beachten Sie anschließend auch unseren Artikel [Empfohlene Schritte \[► 42\]](#) und [Optimierungen \[► 45\]](#).

4.1.2 Schnelleinstieg

Nach der erfolgreichen Installation führen Sie die folgenden Schritte aus, um SPS-Variablen über den TwinCAT OPC UA Server zur Verfügung zu stellen.

- Schritt 1: OPC UA Server initialisieren
- Schritt 2: SPS-Variablen für den OPC-UA-Zugriff konfigurieren
- Schritt 3: Herunterladen der Symboldatei konfigurieren
- Schritt 4: Lizenz für den Server aktivieren
- Schritt 5: Projekt aktivieren
- Schritt 6: Verbindung zum OPC UA Server herstellen



Auslieferungszustand

Dieser Schnelleinstieg setzt voraus, dass sich der TwinCAT OPC UA Server im Auslieferungszustand befindet. Hierbei wird der Server automatisch für den Zugriff auf die (lokale) erste SPS-Laufzeit konfiguriert.

Schritt 1: OPC UA Server initialisieren

Initialisieren Sie den TwinCAT OPC UA Server wie im entsprechenden [Dokumentationsartikel zur Initialisierung \[► 37\]](#) beschrieben.

Schritt 2: SPS-Variablen für den OPC-UA-Zugriff konfigurieren

Öffnen Sie ein bestehendes SPS-Projekt und fügen Sie den folgenden Kommentar vor den ausgewählten Variablen ein. Alternativ können Sie auch ein neues SPS-Projekt erstellen.

TwinCAT 3 (TMC-Import):

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

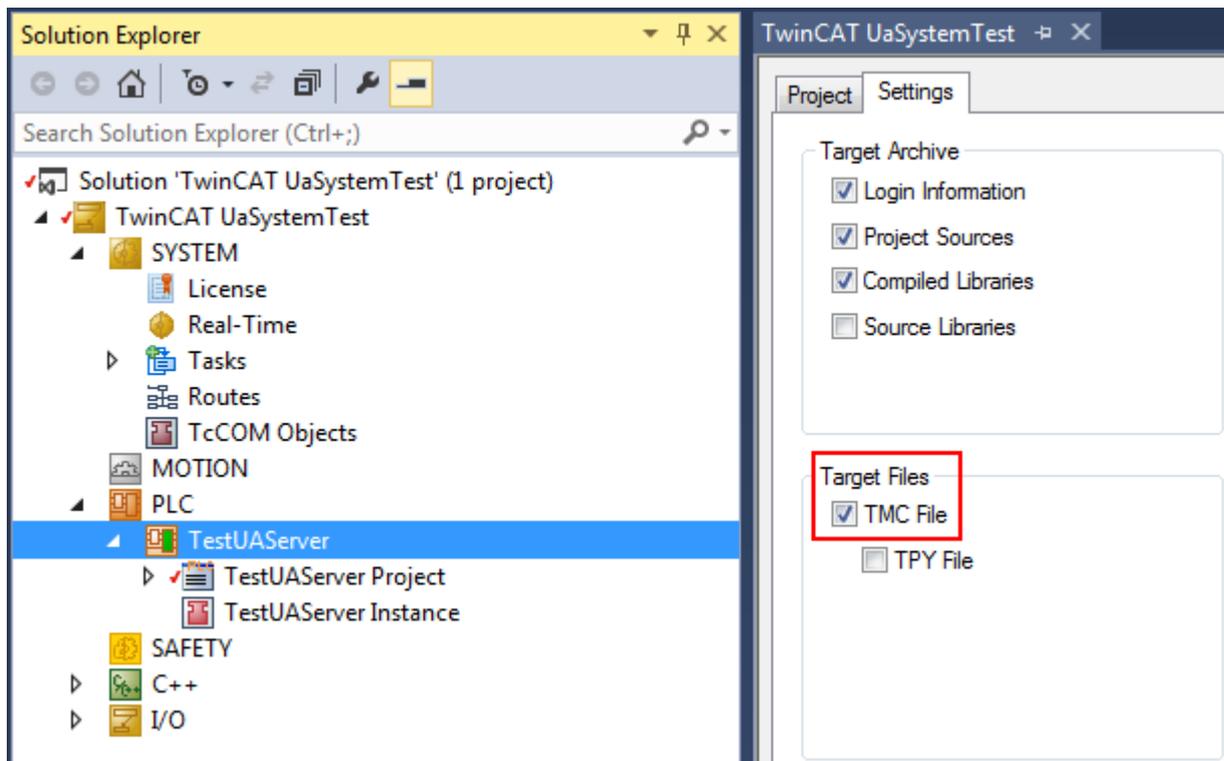
TwinCAT 2 (TPY-Import):

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

Schritt 3: Herunterladen der Symboldatei konfigurieren

Der TwinCAT OPC UA Server stellt standardmäßig eine Verbindung mit der ersten SPS-Laufzeit auf dem lokalen System her und verwendet die entsprechende Symboldatei für den Aufbau des Namensraums.

Damit die Symboldatei bereitgestellt wird, müssen Sie den Download der Symboldatei in den Einstellungen des SPS-Projekts aktivieren.



Schritt 4: Lizenz für den Server aktivieren

Prüfen Sie, ob eine Lizenz vorhanden ist. In TwinCAT 3 können Sie eine TF6100-Lizenz in der TwinCAT-Lizenzverwaltung eintragen (siehe [Lizenzierung \(TC3\)](#) [▶ 32]). Eine 7-Tage Testlizenz ist für den Schnelleinstieg ausreichend.

Schritt 5: Projekt aktivieren

Aktivieren Sie das TwinCAT Projekt und starten Sie TwinCAT neu. Hierdurch wird der TwinCAT OPC UA Server ebenfalls neu gestartet (bitte beachten Sie den Hinweis unten). Loggen Sie sich anschließend in die SPS-Laufzeit ein und starten Sie das SPS-Programm.

HINWEIS

Hinweis zur Windows CE Plattform

Bitte beachten Sie, dass der TwinCAT OPC UA Server unter Windows CE nicht mit einem TwinCAT Neustart ebenfalls neu gestartet wird. Hier müssen Sie entweder das CE-Gerät neu starten oder die `Restart()` Methode aus dem [Konfigurations-Namensraum](#) [▶ 111] aufrufen um den Server neu zu starten, sodass dieser die TMC-Datei einliest und die SPS Variablen zur Verfügung stellt.

Schritt 6: Verbindung zum OPC UA Server herstellen

Um eine Verbindung von einem OPC UA Client herzustellen, muss der Client eine Verbindung mit der URL des OPC UA Servers herstellen, z.B. `opc.tcp://CX-12345:4840` oder `opc.tcp://192.168.1.1:4840`.

● Standard-Port und Endpunkte

i Bitte beachten Sie an dieser Stelle auch unsere Hinweise zum verwendeten [Standard-Port und Endpunkte](#) [▶ 105] des Servers.

Zur Verbindung mit dem TwinCAT OPC UA Server können Sie den mitgelieferten OPC UA Sample Client verwenden, welcher über das Windows Startmenü aufrufbar ist.

Für weitergehende Tests empfehlen wir jedoch die Verwendung der kostenfrei erhältlichen Software UA Expert von Unified Automation.

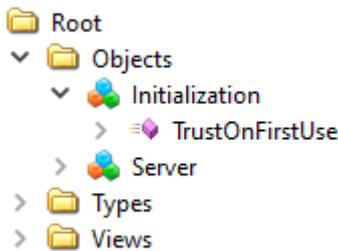
Nach erfolgreichem Verbindungsaufbau mit dem Server finden Sie die freigegebenen SPS-Variablen unterhalb des Objekts PLC1.

4.1.3 Initialisierung

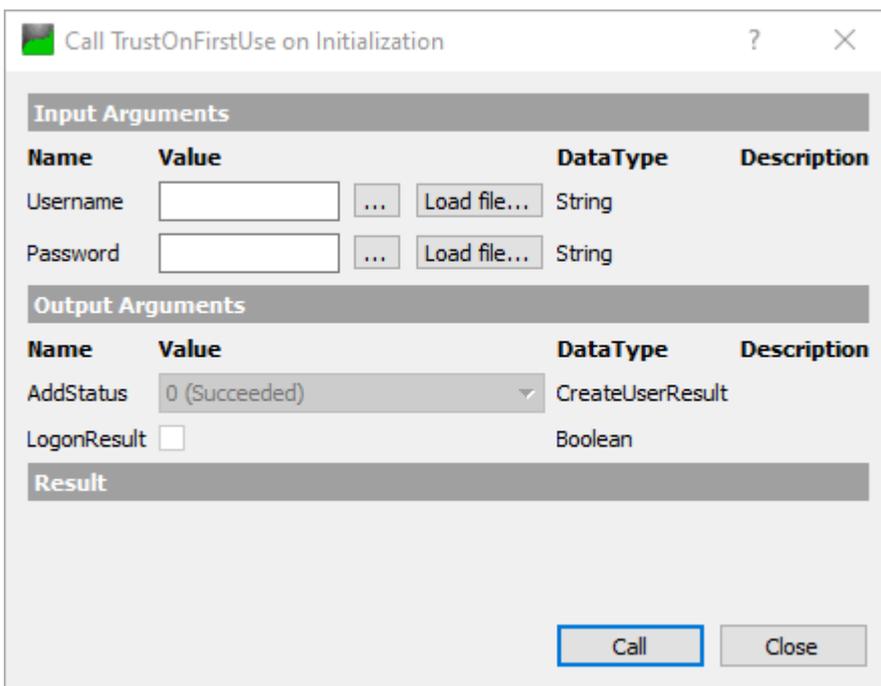
Beginnend mit Setupversion 4.4.0 benötigt der TwinCAT OPC UA Server die Durchführung einer Initialisierungsphase, welche sich an dem TOFU-Prinzip (Trust On First Use) orientiert. Das bedeutet, dass der Server aktiv durch den Anwender initialisiert werden muss, damit er für seine verschiedenen Funktionen (Data Access, Historical Access, usw.) verwendet werden kann.

Im Auslieferungszustand ermöglicht es der Server den Clients eine unauthentifizierte Verbindung („Anonymous“) aufzubauen. Die einmalig durchzuführende TOFU-Initialisierung erfordert nun die Konfiguration eines Betriebssystem-Benutzers, den ein OPC UA Client anschliessend verwenden muss, um sich erfolgreich am Server anzumelden.

Hierfür stellt der Server im uninitialisierten Zustand ausschließlich einen speziellen Initialisierungs-Namensraum zur Verfügung. Dieser Namensraum beinhaltet ein Objekt „Initialization“ mit einer Methode „TrustOnFirstUse“.



Die Methode definiert die folgenden Ein-/Ausgabeparameter:



Parameter	Description
[in] Username	Benutzername für den anzulegenden Betriebssystem-Benutzer. Existiert der Benutzer bereits, so versucht der Server einen Test-Login mit dem angegebenen Passwort durchzuführen und übernimmt, falls erfolgreich, den bereits vorhandenen Benutzer in seine Security-Konfiguration.
[in] Password	Passwort für den Betriebssystem-Benutzer. Das Passwort wird nicht in der Serverkonfiguration gespeichert, sondern ist ausschließlich in der Benutzerdatenbank des Betriebssystems vorhanden. Beachten Sie, dass die Art des Passworts abhängig von etwaigen Sicherheitseinstellungen des Betriebssystems sein kann (Stichwort „komplexe Passwörter“).
[out] AddStatus	Gibt an, ob das Anlegen des Betriebssystem-Benutzers erfolgreich war oder ob der Benutzer ggf. schon existiert.
[out] LogonResult	Gibt an, ob sich der Server mit der angegebenen Benutzername/Passwort-Kombination am Betriebssystem anmelden konnte. Hiermit lässt sich gut überprüfen, ob man eventuell das falsche Passwort angegeben hat, falls der Benutzer schon existiert.
[out] OPC UA Statuscode	Der reguläre OPC UA Statuscode beim Methodenaufruf. Ist die Methode auf OPC UA Ebene erfolgreich aufgerufen worden, gibt dieser Statuscode GOOD zurück, anderenfalls BAD.

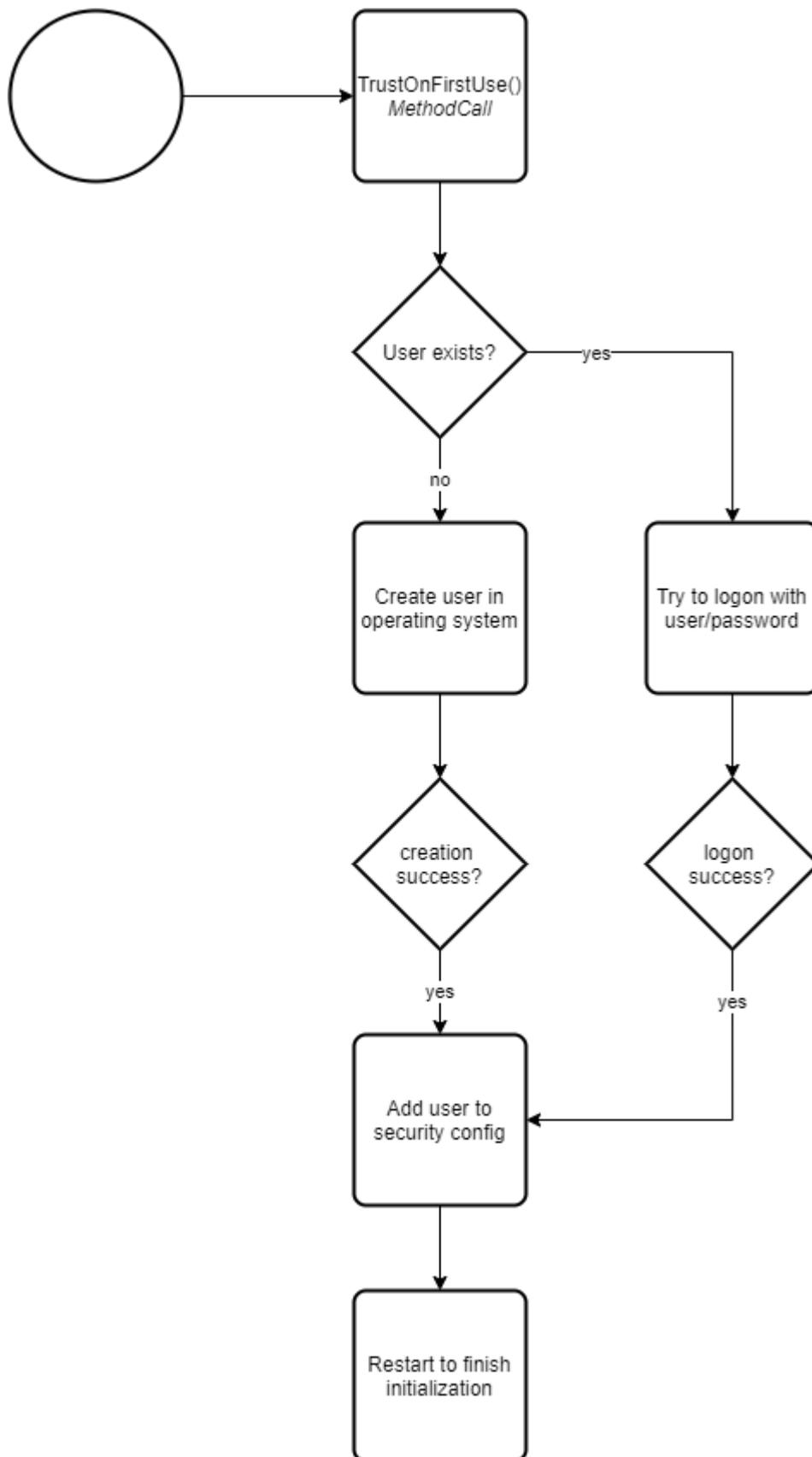
Durch den Aufruf dieser Methode wird der Server initialisiert. Die Methode versucht hierbei einen vom Anwender angegebenen Benutzer im unterlagerten Betriebssystem des Servers anzulegen. Ist dies erfolgreich, so wird der Benutzer automatisch der Security-Konfiguration (TcUaSecurityConfig.xml) des Servers hinzugefügt und als Serveradministrator definiert. Nach einem automatischen Neustart des Servers am Ende des Methodenaufrufs kann sich ein OPC UA Client dann mit diesem Benutzer am Server anmelden.

Existiert ein angegebener Benutzer schon im Betriebssystem, so wird dies über einen Ausgabeparameter (AddStatus) angezeigt. Der Server versucht in diesem Fall sich mit dem angegebenen Passwort am Betriebssystem anzumelden. Ist dieser Anmeldevorgang erfolgreich, so wird der Benutzer in der Security-Konfiguration des Servers eingetragen und die Initialisierung durch einen automatischen Neustart des Servers erfolgreich beendet. Schlägt die Anmeldung am Betriebssystem fehl (z. B., weil das falsche Passwort angegeben wurde), so wird dies über einen Ausgabeparameter (LogonResult) angezeigt und die Initialisierung wird nicht fortgesetzt. So wird verhindert, dass man versehentlich versucht den Server mit einer falschen Benutzername/Passwort-Kombination zu initialisieren und sich dadurch „auszusperren“.

● Ablaufen eines Benutzerpassworts

i Wenn der OPC UA-Server einen Betriebssystembenutzer anlegt, wird für diesen Benutzer **nicht** explizit aktiviert, dass das Passwort nicht abläuft. Hier werden die Einstellungen des Betriebssystems übernommen, wo das maximale Kennwortalter in den Kennwortrichtlinien festgelegt ist. Wenn das maximale Kennwortalter auf 0 steht, laufen Passwörter nicht ab, ansonsten nach der im Betriebssystem angegebenen Anzahl von Tagen.

Das folgende Diagramm veranschaulicht diesen Prozess noch einmal in stark vereinfachter Form:



Nach dem Neustart des Servers muss ein OPC UA Client beim Verbindungsaufbau den zur Initialisierung verwendeten Betriebssystembenutzer zur Authentifizierung verwenden.

Die folgenden Screenshots zeigen den gesamten Vorgang exemplarisch am Beispiel des OPC UA Clients „UA Expert“. In diesem Beispiel gehen wir davon aus, dass der Benutzer noch nicht im Betriebssystem existiert und somit durch den Server angelegt wird.

Schritt 1: OPC UA Client verbindet sich erstmalig mit dem Server

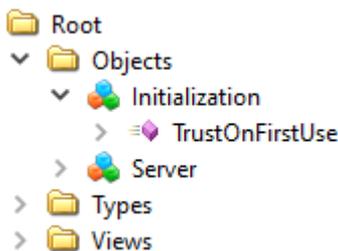
Der Server wurde installiert und der UA Expert baut zum ersten Mal eine Verbindung mit dem Server auf. Für diese Verbindung kann noch der Anonymous-Zugriff verwendet werden.

Server Information	
Endpoint Url	<input type="text" value="opc.tcp://DESKTOP-PDTN35I:4840"/>
Reverse Connect	<input type="checkbox"/>

Security Settings	
Security Policy	<input type="text" value="Basic256Sha256"/>
Message Security Mode	<input type="text" value="Sign & Encrypt"/>

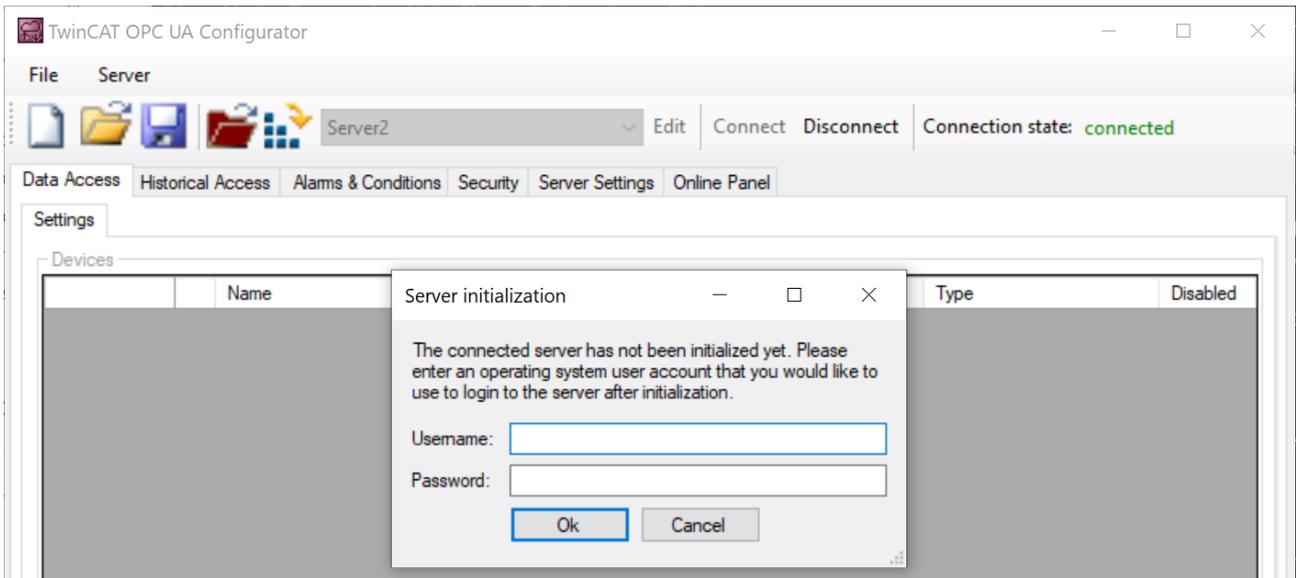
Authentication Settings	
<input checked="" type="radio"/> Anonymous	
<input type="radio"/> Username	<input type="text"/> <input type="checkbox"/> Store
<input type="radio"/> Password	<input type="text"/>
<input type="radio"/> Certificate	<input type="text"/> ...
<input type="radio"/> Private Key	<input type="text"/> ...

Nach dem Herstellen der Verbindung findet man im Adressraum des Servers das Initialization Objekt mitsamt der TrustOnFirstUse Methode.



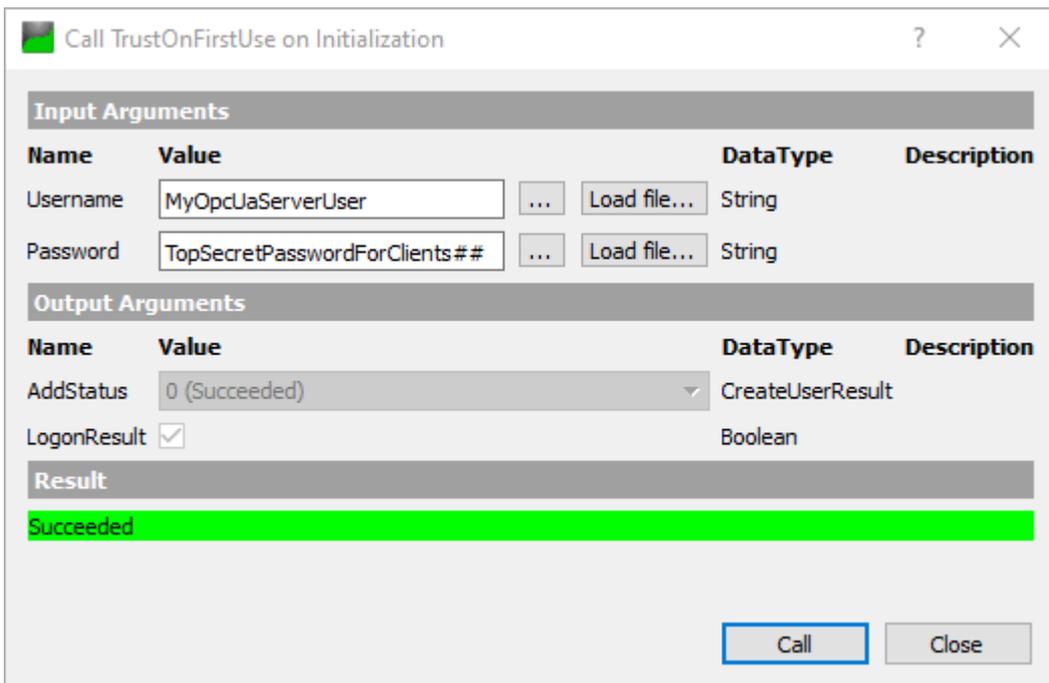
Schritt 2: OPC UA Client startet TrustOnFirstUse

Der Aufruf der TrustOnFirstUse Methode kann über einen beliebigen OPC UA Client erfolgen, z.B. den UA Expert. Aber auch die Beckhoff-eigenen Konfigurationstools erlauben die Verwendung dieser Initialisierungsschnittstelle. Der TwinCAT OPC UA Configurator (Standalone oder Visual Studio integriert) erkennt beim Herstellen einer Verbindung automatisch einen uninitialisierten Server und ermöglicht die Initialisierung über eine entsprechende Konfigurationsoberfläche:



Die folgenden Schritte zeigen denselben Prozess wie er z.B. in der Software UA Expert manuell durchgeführt werden kann:

Im UA Expert ruft man die TrustOnFirstUse Methode auf, um einen Benutzer zu erzeugen und den Server für diesen Benutzer zu konfigurieren. Als Benutzername wurde in diesem Beispiel „MyOpcUaServerUser“ verwendet. Das Passwort muss den Komplexitätsanforderungen des Betriebssystems entsprechen, anderenfalls schlägt die Initialisierung fehl. Der folgende Screenshot zeigt den erfolgreichen Aufruf der Methode.



Der AddStatus Parameter zeigt, dass das Anlegen des Benutzers in der Benutzerdatenbank des Betriebssystems erfolgreich war. Der LogonResult Parameter zeigt, dass eine initiale Test-Authentifizierung des Servers mit den angegebenen Benutzerinformationen erfolgreich war.

Der Server startet nach diesem erfolgreichen Methodenaufruf automatisch neu.

Schritt 3: OPC UA Client meldet sich am initialisierten Server an

Bitte beachten Sie, dass der UA Expert sich nach dem Methodenaufruf nicht automatisch neu mit dem Server verbinden kann, da der Anonymous-Zugriff deaktiviert wurde und von nun an die Anmeldung über den angegebenen Benutzernamen erfolgen muss.

Server Information	
Endpoint Url	<input type="text" value="opc.tcp://DESKTOP-PDTN35I:4840"/>
Reverse Connect	<input type="checkbox"/>
Security Settings	
Security Policy	<input type="text" value="Basic256Sha256"/>
Message Security Mode	<input type="text" value="Sign & Encrypt"/>
Authentication Settings	
<input type="radio"/> Anonymous	
<input checked="" type="radio"/> Username	<input type="text" value="MyOpcUaServerUser"/> <input checked="" type="checkbox"/> Store
Password	<input type="password" value="....."/>
<input type="radio"/> Certificate	<input type="text"/> ...
Private Key	<input type="text"/> ...

Nach dem Verbindungsaufbau findet man im Adressraum des Servers nun die regulären Namensräume und Objekte wieder und kann mit der Projektierung der Applikation beginnen.

- Root
 - Objects
 - AlarmsConditions
 - Configuration
 - DeviceSet
 - PLC1
 - Server
 - Types
 - Views

i Berechtigungen des TOFU-Benutzers

Der durch den TOFU-Mechanismus konfigurierte Benutzer hat Vollzugriff auf den Server, was unter Umständen nicht gewünscht ist. Wir empfehlen daher in einem nächsten Schritt die Erstellung eines expliziten Benutzers für den reinen Datenzugriff. Dies wird im Artikel [Empfohlene Schritte \[▶ 42\]](#) näher beschrieben.

4.1.4 Empfohlene Schritte

Nach der Erstinbetriebnahme empfehlen wir die Beachtung der folgenden Punkte, um den Server weiter zu konfigurieren und eine stabile und sichere Betriebsumgebung zu gewährleisten.

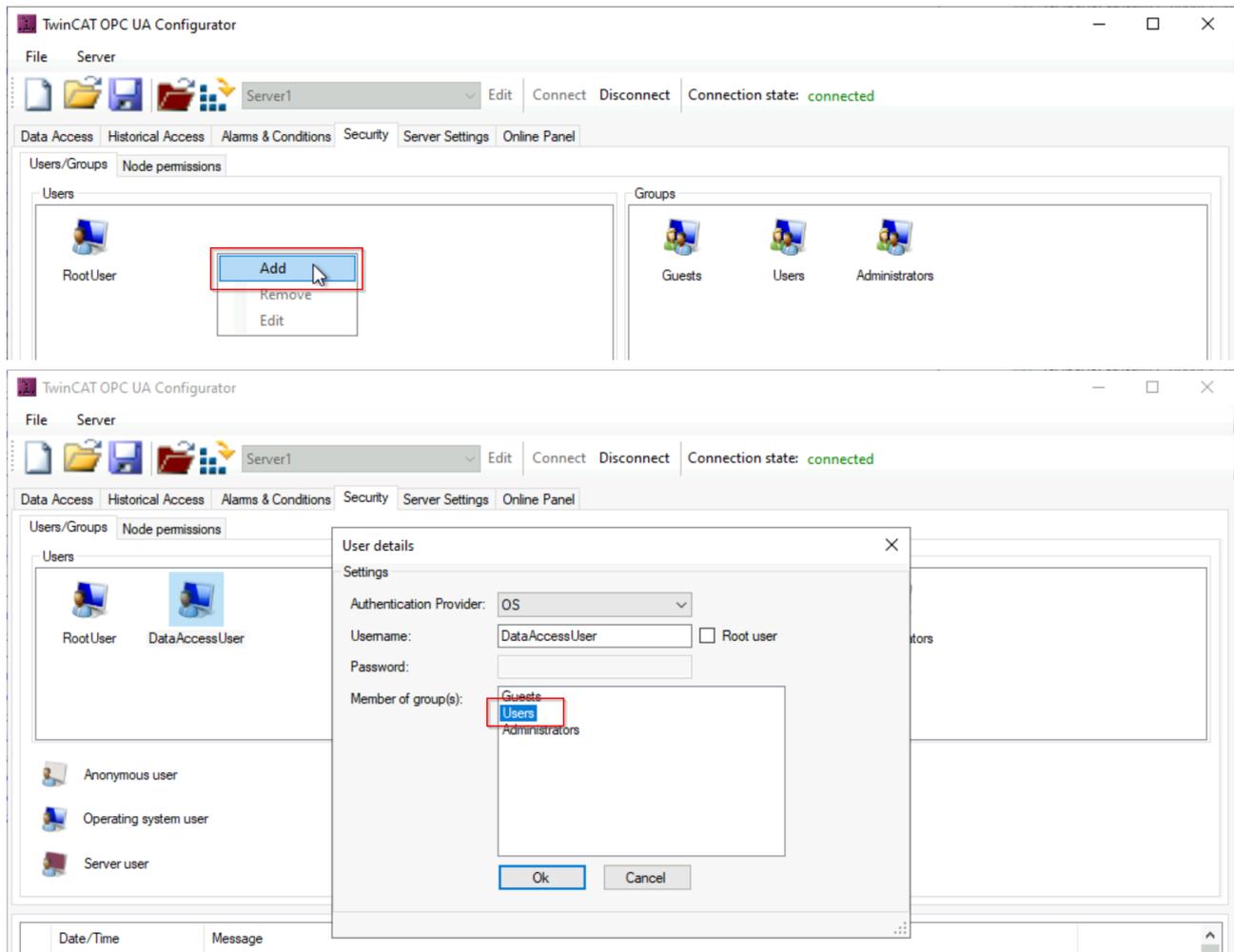
Sichere IdentityToken verwenden

Durch die einmalige [Initialisierung \[▶ 37\]](#) des Servers wird das „Anonymous“ IdentityToken deaktiviert. Aus Sicherheitsgründen empfehlen wir dieses deaktiviert zu lassen und den Zugriff auf den Server ausschließlich für authentifizierte Client-Applikationen zu erlauben, z.B. über die standardmäßig durch die Initialisierung konfigurierte Benutzername/Password Authentifizierung.

Erstellung eines Benutzers für den reinen Datenzugriff

Durch die bereits genannte Initialisierung des Servers wird ein Benutzer für den Zugriff auf den Server konfiguriert und anschließend der Anonymous Zugriff auf den Server deaktiviert. Der konfigurierte Benutzer hat hierbei jedoch Vollzugriff auf alle Objekte im Namensraum des Servers. In den meisten Anwendungsszenarien ist dies nicht gewünscht und der Administrator-Benutzer soll vom Anwendungs-Benutzer separiert werden.

Wir empfehlen daher die Konfiguration eines zusätzlichen, dedizierten Benutzers, welcher die notwendigen Berechtigungen für den Zugriff auf Variablen eines Data Access Geräts bekommt, dabei jedoch nicht auf den Konfigurations-Namensraum [► 111] zugreifen darf. Diese Einstellung kann über den Konfigurator [► 115] durchgeführt werden, indem Sie einen neuen Benutzer hinzufügen, welcher der Benutzergruppe „Users“ zugewiesen wird.

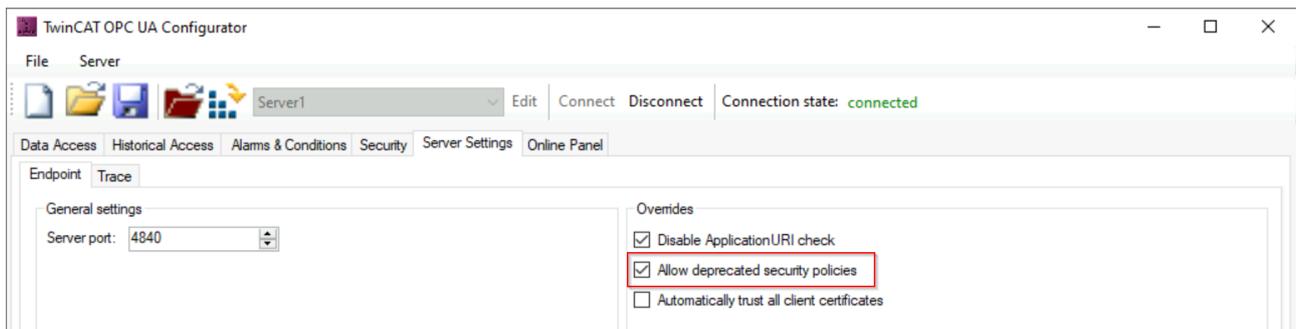


Der neu konfigurierte Benutzer hat dann alle notwendigen Berechtigungen auf TwinCAT Variablen zuzugreifen, das Typsystem auszulesen, jedoch nicht die Konfiguration des Servers zu beeinflussen.

Bitte beachten Sie, dass Sie bei Verwendung des Authentifizierungsproviders „OS“ den Benutzer ebenfalls im Betriebssystem anlegen.

Unsichere Endpunkte deaktiviert lassen

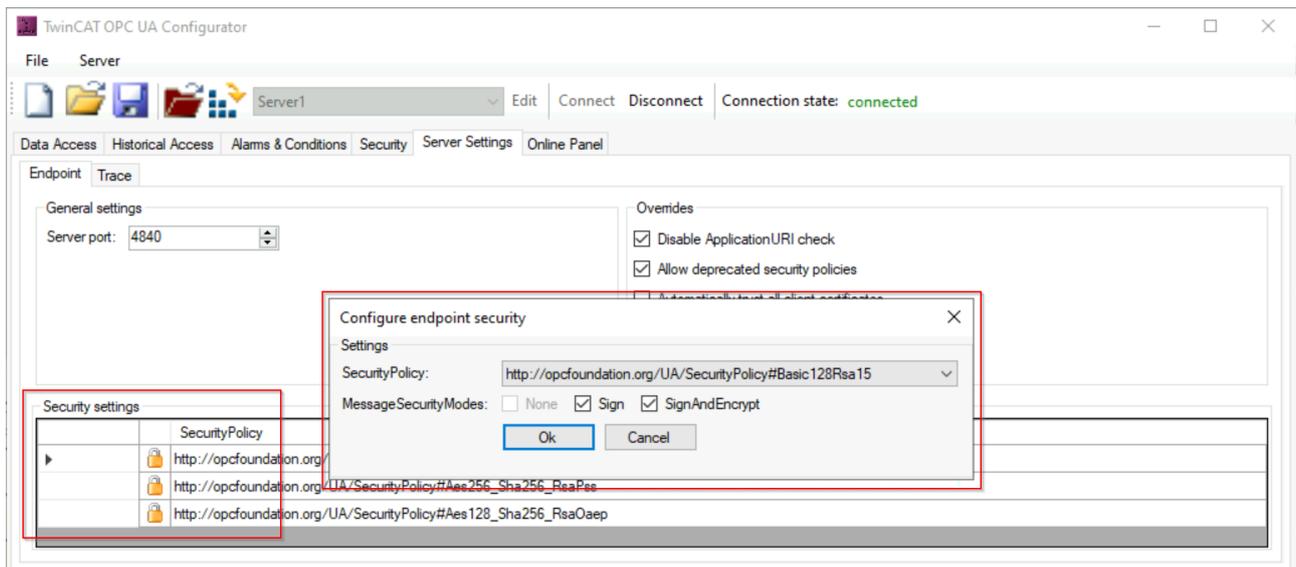
Als unsicher eingestufte Endpunkte [► 105] werden standardmäßig nicht vom TwinCAT OPC UA Server angeboten. Über einen Konfigurationsschalter in der TcUaServerConfig.xml lassen sich unsichere Endpunkte wieder im Server verfügbar machen – wir empfehlen dies jedoch ausdrücklich nicht.



Alternativ können Sie auch den entsprechenden Eintrag in der Konfigurationsdatei TcUaServerConfig.xml manuell anpassen:

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

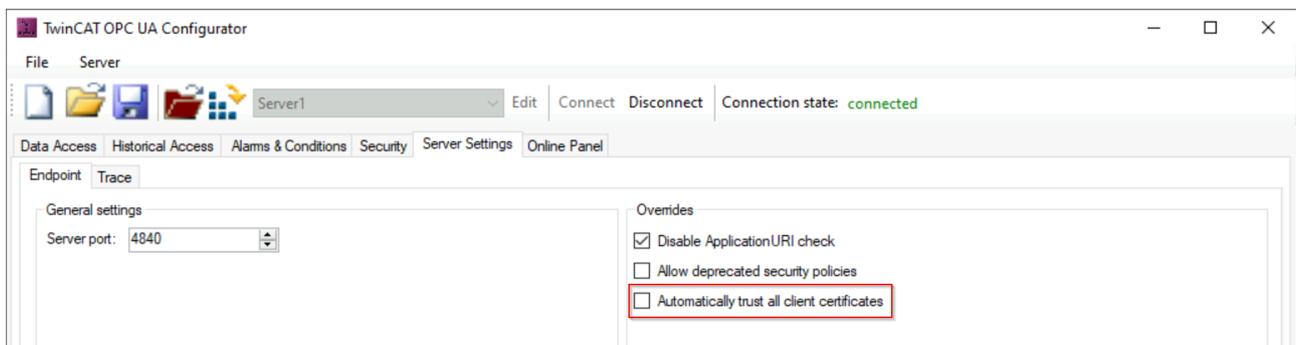
Anschließend können Sie die unsicheren Endpunkte wieder zur Konfiguration des Servers hinzufügen, zum Beispiel über das Kontextmenü im Konfigurator im Bereich „Security Settings“:



Desweiteren ist auch der None/None Endpunkt im Auslieferungszustand des Servers deaktiviert. Aus Sicherheitsgründen empfehlen wir diesen Endpunkt ebenfalls deaktiviert zu lassen und den Zugriff auf den Server nur über einen sicheren Endpunkt zu erlauben. Bei Bedarf kann der None/None Endpunkt jedoch auch über den oben genannten Weg wieder zur Konfiguration des Servers hinzugefügt werden.

'AutomaticallyTrustAllClientCertificates' deaktivieren

Im Auslieferungszustand wurde der Server so konfiguriert, dass dieser allen Client-Zertifikaten automatisch vertraut. Aus Sicherheitsgründen empfehlen wir die Deaktivierung dieser Einstellung. Diese Einstellung kann über den Konfigurator [► 115] vorgenommen werden:



Alternativ können Sie auch den entsprechenden Eintrag in der Konfigurationsdatei TcUaServerConfig.xml manuell anpassen:

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

Nach der Deaktivierung dieser Einstellung muss eine Vertrauensstellung [► 108] zwischen Client und Server hergestellt werden, indem die Applikationen gegenseitig ihren Zertifikaten vertrauen.

4.1.5 Optimierungen

Es gibt eine Vielzahl an verschiedenen Möglichkeiten die Kommunikationsverbindung zwischen OPC UA Client und Server bzw. SPS zu optimieren.

i Die hier dargestellten Screenshots und Performancewerte stellen Beispiele unter Laborbedingungen dar, welche auf unterschiedlichen Hardwaregeräten ausgeführt wurden. Sie lassen sich daher nicht 1:1 auf Kundenprojekte übertragen und dienen nur zur Veranschaulichung bestimmter Sachverhalte.

Der folgende Artikel stellt weiterführende Informationen zu den folgenden Themen bereit:

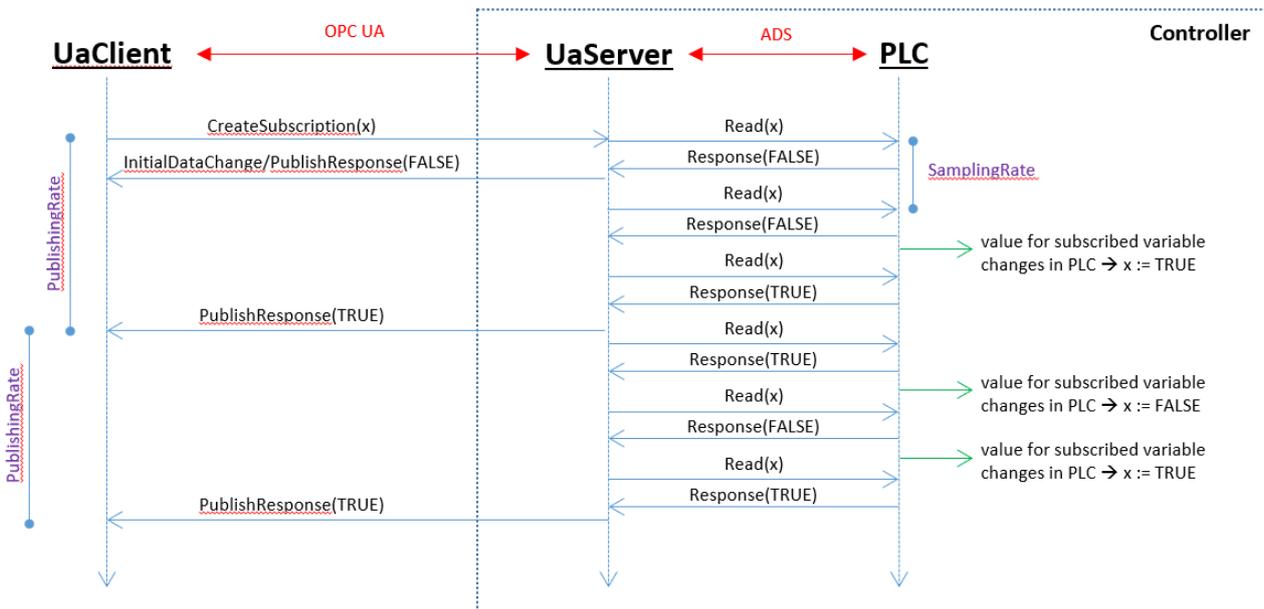
- SamplingInterval vs. PublishingInterval
- StructuredTypes
- StructuredTypes und deren Membervariablen

SamplingInterval vs. PublishingInterval

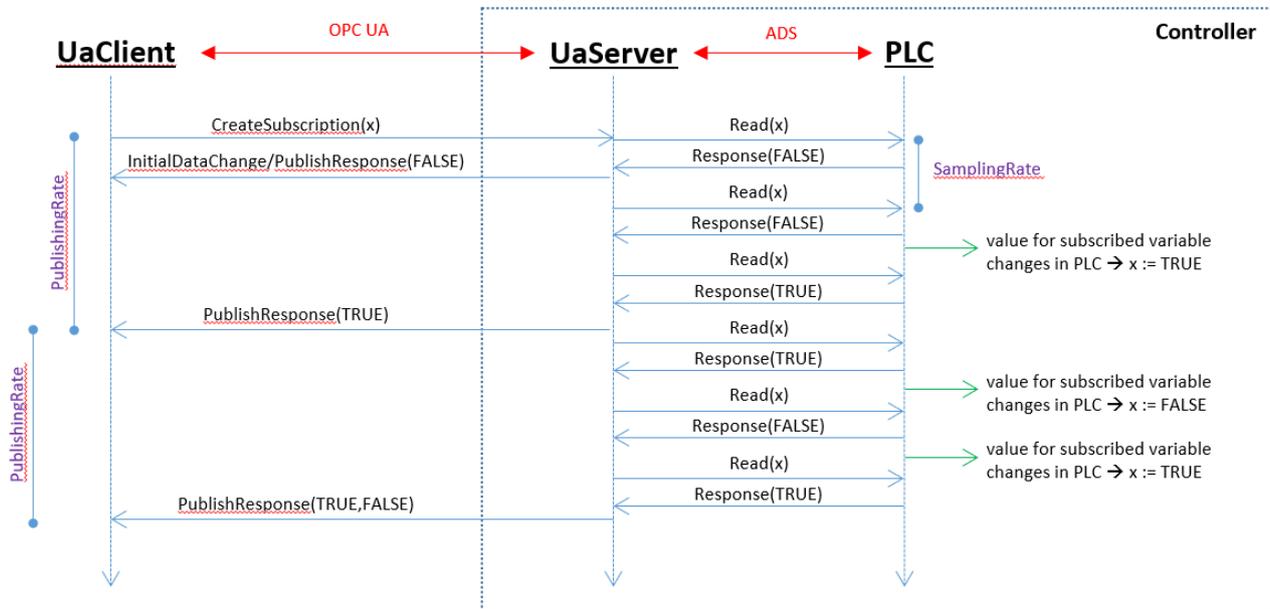
Beim Anlegen einer Subscription verwendet ein OPC UA Client verschiedene Parameter für die Subscription und die darin enthaltenen, sogenannten MonitoredItems, um Benachrichtigungen über Variablenänderungen zu erhalten. Die folgende Tabelle erklärt zwei dieser Parameter, welche anschließend näher beschrieben werden sollen.

Parameter	Beschreibung
PublishingInterval	Das PublishingInterval gibt die Rate an, mit der ein OPC UA Client vom Server über Wertänderungen informiert wird. Das PublishingInterval wird in Part 4 der OPC UA Spezifikation detailliert beschrieben.
SamplingInterval	Das SamplingInterval gibt die Rate an, mit der der OPC UA Server seine unterliegende Datenquelle nach Wertänderungen abtasten soll, im Falle des TwinCAT OPC UA Servers über die ADS Verbindung. Das SamplingInterval wird in Part 4 der OPC UA Spezifikation detailliert beschrieben.

Die folgende Grafik stellt den Zusammenhang zwischen diesen beiden Parametern noch einmal anschaulich dar. Es wird hierbei davon ausgegangen, dass der TwinCAT OPC UA Server auf der SPS Steuerung installiert wurde und der OPC UA Client von einem externen System aus auf den Server zugreift.



Wie in dem Schaubild erkennbar, kann hierbei durchaus die Situation entstehen, dass der OPC UA Client bestimmte Wertänderungen in der SPS nicht mitbekommt, z.B. wenn diese entweder „zu schnell“ in der SPS passieren bzw. die SamplingRate nicht hoch genug ist bzw. ein Variablenwert wieder auf den ursprünglichen Wert zurück geht (wie oben erkennbar). Über einen weiteren Parameter, die sogenannte QueueSize, lassen sich mehrere Wertänderungen zwischen den PublishingIntervallen erfassen und an den OPC UA Client übertragen. In obigem Beispiel wurde eine QueueSize von 1 gewählt, d.h. es wird immer der „Last Known Value“ an den Client übertragen. In dem folgenden Schaubild wurde hingegen eine QueueSize von 2 gewählt, d.h. es werden die letzten zwei bekannten Wertänderungen an den Client übertragen.

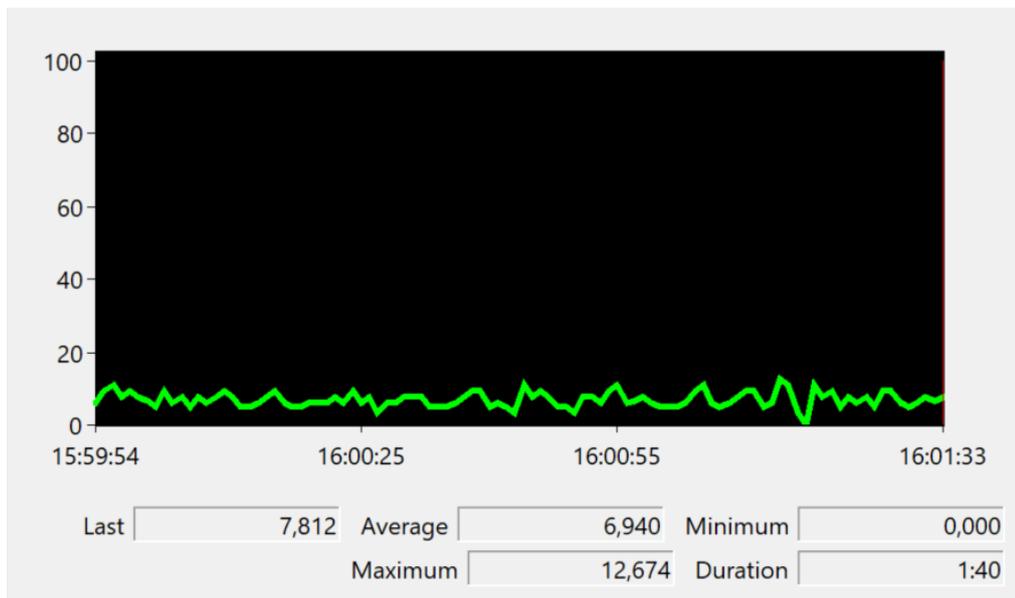


Bei dem ersten PublishResponse lässt sich erkennen, dass nur eine Wertänderung an den Client übermittelt wird, da auch nur eine Wertänderung in der SPS stattgefunden hat. Beim zweiten PublishResponse lässt sich erkennen, dass zwei Wertänderungen in der SPS stattgefunden haben und beide auch an den Client übertragen werden.

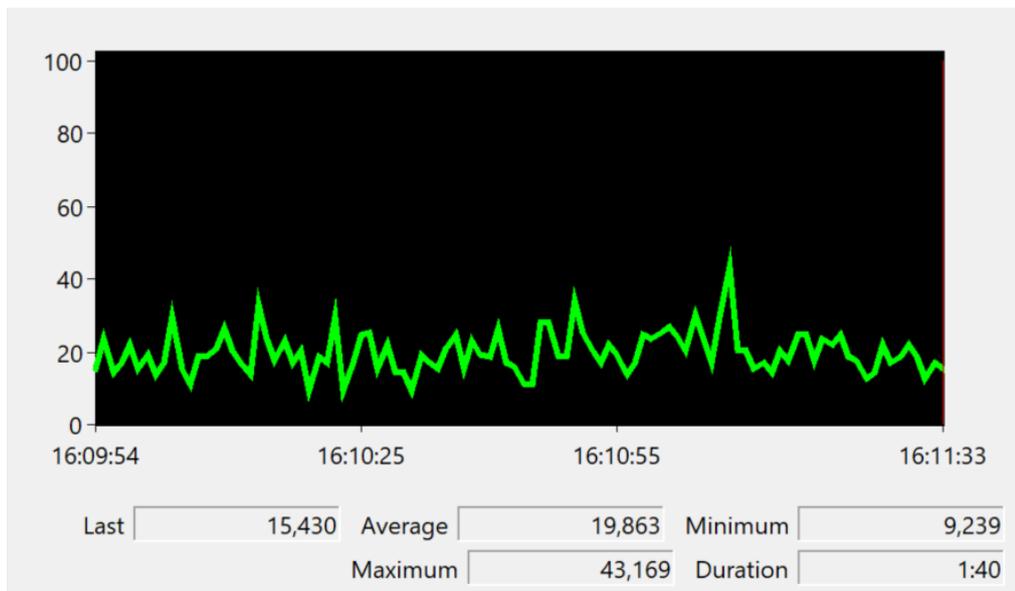
Bei den oben beschriebenen Parametern handelt es sich um Einstellungen, die der Client üblicherweise selbst in der Hand hat und beim Server anfragt. Und genau hier lassen sich, viele Optimierungen vornehmen, denn beide Parameter haben einen starken Einfluss darauf, wie viel CPU-Zeit der OPC UA Client und Server benötigen, da entsprechend viele Informationen verarbeitet bzw. angefragt werden müssen.

Abhängig vom verwendeten Einsatzszenario sollten die beiden Parameter entsprechend sinnvoll eingestellt werden. Wenn es sich bei dem OPC UA Client zum Beispiel um eine Visualisierung handelt, dann machen schnelle PublishingIntervalle und SamplingRaten nur bedingt Sinn, da das menschliche Auge ohnehin keine Informationen schneller als ~200ms verarbeiten kann. Auch die Verwendung der QueueSize sollte in Abhängigkeit zum Einsatzszenario sinnvoll gewählt werden. Wenn der OPC UA Client ohnehin keine Werte aus der Queue verarbeitet, ist eine QueueSize von 1 ausreichend und sinnvoll, da entsprechend weniger Informationen übertragen werden müssen und dies das System weiter optimiert.

In dem folgenden Beispiel hat ein OPC UA Client eine Subscription mit 10.000 Variablen auf dem TwinCAT OPC UA Server angelegt. Als PublishingInterval wurde hierbei 500ms und als SamplingInterval 250ms gewählt. Die CPU-Auslastung des TwinCAT OPC UA Servers lag hierbei bei einem Durchschnittswert von ca. 6,9%, siehe folgenden Screenshot vom Windows Performance Monitor.



Anschließend wurde das PublishingInterval auf 200ms und das SamplingInterval auf 100ms eingestellt. Die CPU-Auslastung des TwinCAT OPC UA Servers erhöhte sich hierdurch auf einen Durchschnittswert von ca. 19%.



StructuredTypes

Mit Hilfe von OPC UA StructuredTypes kann der TwinCAT OPC UA Server [IEC61131 Datenstrukturen](#) [► 57] in seinem Adressraum für Clients zur Verfügung stellen. Eine Datenstruktur kann hierbei jedoch auch als nicht-StructuredType bereitgestellt werden, d.h. das Root-Element ist ein FolderType (ohne Value) und ein Zugriff kann dann auf die einzelnen Membervariablen erfolgen. Bei der ADS-Kommunikation mit der SPS verhalten sich beide Varianten grundlegend unterschiedlich.

Bei dem Zugriff eines OPC UA Clients auf einzelne Membervariablen eines nicht-StructuredTypes kann es, je nach Anzahl der Variablen, durchaus passieren, dass die ADS-Kommunikation auf mehrere ADS Read/Write Requests aufgeteilt wird. Dies kann wiederum zur Folge haben, dass die ADS Read/Write Requests in unterschiedlichen SPS-Zyklen von der SPS abgearbeitet werden. Die Datenkonsistenz der jeweiligen Datenstruktur kann hierbei also nicht garantiert werden.

Bei dem Zugriff eines OPC UA Clients auf einen StructuredType wird hingegen sichergestellt, dass der StructuredType in einem einzigen ADS Read/Write Request von der SPS abgearbeitet wird, wodurch eine Datenkonsistenz sichergestellt wird.

Bei der Verwendung von StructuredTypes für große SPS-Datenstrukturen sollte beachtet werden, dass der resultierende ADS Read/Write Request bzw. Response entsprechend groß sein kann und entsprechend viel Speicher im TwinCAT ADS Router benötigt. Zusätzlich müssen StructuredTypes vom OPC UA Server entsprechend kodiert bzw. dekodiert werden, was zusätzliche CPU-Zeit benötigt. Gerade im Zusammenhang mit den bei einer Subscription einstellbaren Parametern zum SamplingInterval und PublishingInterval lassen sich hier weitere Optimierungen vornehmen.

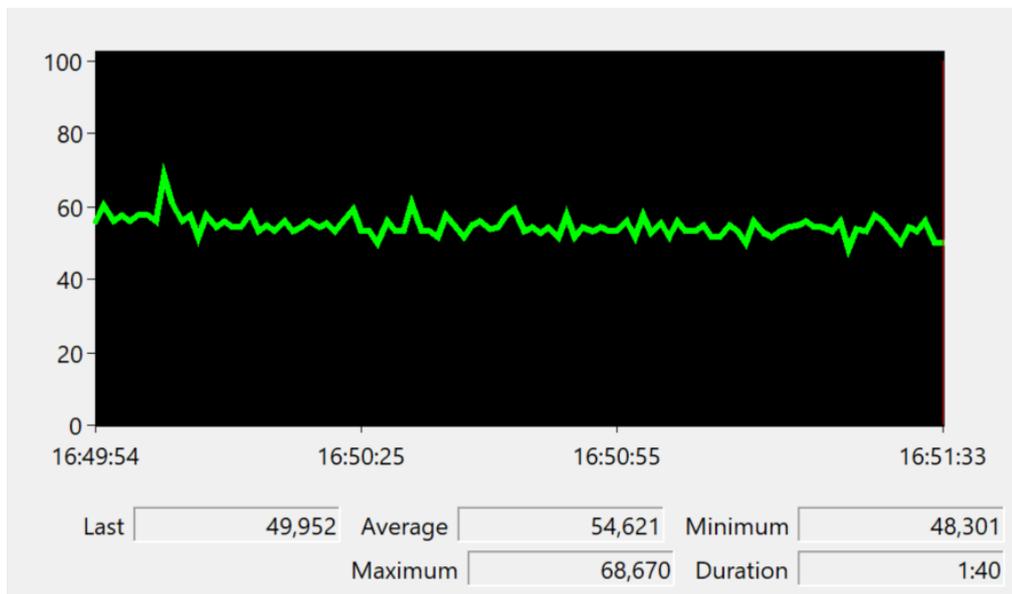
In dem folgenden Beispiel hat ein OPC UA Client eine Subscription mit einem StructuredType auf dem TwinCAT OPC UA Server angelegt. Die unterlagerte SPS-Datenstruktur ist hierbei wie folgt aufgebaut:

```

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

```

Die als Membervariable verwendete Struktur ST_Complex_1 ist ca. 91 Byte groß. Insgesamt handelt es sich hierbei also um eine Datenstruktur mit einer Größe von ca. 1 Mbyte. Als PublishingInterval wurde 500ms und als SamplingInterval 250ms gewählt. Die CPU-Auslastung des TwinCAT OPC UA Servers lag nach dem Anlegen der Subscription bei einem Durchschnittswert von ca. 54,6%, siehe folgenden Screenshot vom Windows Performance Monitor.



Entsprechend des gewählten SamplingIntervals wird bei der unterlagerten ADS-Kommunikation ca. alle 250ms ein ADS Read Request abgesetzt. Bei dem zugehörigen Response erkennt man deutlich die Größe der Datenstruktur von ca. 1 Mbyte, d.h. dass alle 250ms ein 1 Mbyte großes Datenpaket durch den TwinCAT ADS Router transportiert wird (und vom Server entsprechend verarbeitet werden muss).

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

StructuredTypes und deren Membervariablen

Standardmäßig werden die Membervariablen eines StructuredType im Adressraum des Servers als eigene Nodes dargestellt und verfügbar gemacht. Dies benötigt zusätzlichen Arbeitsspeicher, da der TwinCAT OPC UA Server für jede Node Arbeitsspeicher allokiert. Ein OPC UA Client, der jedoch ausschließlich mit dem StructuredType an sich, also „dem Root Element“, arbeitet, benötigt diese zusätzlichen Nodes nicht. Diese lassen sich daher über das OPC.UA.DA:=2 Attribut explizit ausblenden. Beispiel:

```

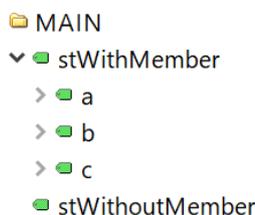
TYPE ST_Test :
STRUCT
  a : DINT;
  b : STRING;
  c : DINT;
END_STRUCT
END_TYPE

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

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

```

Diese Deklaration hat den unten dargestellten Adressraum zur Folge:



4.1.6 Data Access

4.1.6.1 SPS

In diesem Abschnitt wird beschrieben, wie Sie den Namensraum des OPC UA Servers konfigurieren, so dass dieser Variablen aus einer TwinCAT SPS Laufzeit enthält. Der TwinCAT OPC UA Server kann hierbei mehrere Namensräume, also mehrere SPS Laufzeiten darstellen. Damit eine SPS-Variable über den jeweiligen Namensraum erreichbar ist, muss sie im SPS-Programm explizit hierfür freigegeben werden.

● QuickStart

i Beachten Sie, dass sich der OPC UA Server standardmäßig immer mit dem ersten lokalen SPS-Laufzeitsystem verbindet, eine Konfiguration ist also in den meisten Fällen nicht erforderlich (siehe auch [Schnelleinstieg \[► 35\]](#)). Nur in wenigen Fällen ist eine separate Konfiguration notwendig, z. B. wenn im Server zusätzliche Laufzeiten angezeigt werden sollen.

Dieser Dokumentationsartikel beinhaltet folgende Themen:

- [Allgemeine Informationen \[► 49\]](#)
- [Schritt 1: SPS-Variablen auswählen, die über OPC UA öffentlich zugänglich werden sollen \[► 50\]](#)
- [Schritt 2: Symbolbeschreibung herunterladen und im SPS-Projekt aktivieren \[► 53\]](#)
- [Optional] [Schritt 3: Datenzugriffgerät in OPC UA Server konfigurieren \[► 54\]](#)
- [Optional] [Schritt 4: Explizites Ausblenden von Variablen \[► 54\]](#)

Allgemeine Informationen

Für die Konfiguration des Servers und den Zugriff auf SPS-Variablen stehen mehrere Parameter zur Verfügung, welche Sie über den TwinCAT OPC UA Konfigurator einstellen können.

Parameter	Beschreibung	Mögliche Werte
ADS Port	Definiert den ADS-Port, unter dem die SPS-Laufzeit erreichbar ist. Der ADS-Port kann in den Eigenschaften des SPS-Projekts gelesen werden.	800 (BC Controller) 801 (TwinCAT 2) 811 (TwinCAT 2) ... 851 (TwinCAT 3 - standardmäßig) 852 (TwinCAT 3) ...
AutoCfg	Definiert den Typ der Laufzeit, z. B. SPS, C++, I/O. Einige AutoCfg Optionen stehen sowohl als ungefilterte als auch gefilterte Option zur Verfügung. Gefiltert bedeutet, dass der Nutzer bestimmen kann, welche Symbole über OPC UA veröffentlicht werden sollen (siehe unten). Bei Verwendung einer ungefilterten Option wird jedes Symbol über OPC UA zur Verfügung gestellt.	7 TwinCAT 2 (TPY) 8 TwinCAT 2 (TPY) gefiltert 4040 TwinCAT 3 (TMC) 4041 TwinCAT 3 (TMC) gefiltert
AutoCfgSymFile	Symboldatei der jeweiligen SPS-Laufzeit. Standardmäßig wird die automatisch erzeugte Symboldatei der ersten SPS-Laufzeit des lokalen Systems importiert.	Pfad zur Symboldatei. Zeigt standardmäßig auf die Symboldatei (TMC) der ersten lokalen SPS-Laufzeit.
IoMode	Definiert die Methode für den Zugriff auf Symbole. Dies ist besonders wichtig für den Zugriff auf BC-Geräte.	1 (Zugriff über Handle - standardmäßig) 3 (Zugriff auf BC Controller)
Array Expansion	Unterelemente eines Arrays werden standardmäßig nicht als separate Knoten im UA-Namensraum abgebildet. Stattdessen wird lediglich das Array als ein einzelnes Element abgebildet. Trotzdem können UA Clients über deren sogenannten „IndexRange“ auf Unterelemente zugreifen. (Einige ältere OPC UA Clients unterstützen diese Möglichkeit noch nicht.) Das Flag wurde eingeführt, damit der Zugriff dennoch für diese Clients möglich wird. Es sorgt dafür, dass jede Array-Position als separater Knoten im UA-Namensraum angezeigt wird. Dies führt zu einem höheren Speicherbedarf des OPC UA Servers.	0 (deaktiviert - standardmäßig) 1 (aktiviert)
Disabled	Deaktiviert die SPS-Laufzeit im UA-Namensraum, woraufhin der entsprechende Knoten nicht angezeigt wird. Es wird empfohlen, diesen Parameter zu aktivieren, wenn bestimmte SPS-Laufzeiten zum Zeitpunkt der Projektplanung noch nicht verfügbar sind, weil z. B. die entsprechenden Geräte noch nicht ans Netzwerk angeschlossen sind.	0 (deaktiviert - standardmäßig) 1 (aktiviert)

Im nachfolgenden Abschnitt wird anhand eines Beispiels gezeigt, wie Sie die Variablen aus einem SPS-Programm in den Server importieren können. Hierbei wird vorausgesetzt, dass sich der TwinCAT OPC UA Server in seiner Standardkonfiguration (Auslieferungszustand nach der Installation) und die SPS-Laufzeit auf dem gleichen Computer befinden.

Schritt 1: SPS-Variablen konfigurieren

Der TwinCAT OPC UA Server stellt automatisch eine Verbindung zur ersten SPS-Laufzeit auf dem lokalen System her. Die im SPS-Programm für OPC UA gekennzeichneten SPS-Symbole werden beim Starten des Servers berücksichtigt. Eine Kennzeichnung erfolgt hierbei über einen Kommentar an der entsprechenden Stelle (Instanz, Struktur, Variable) im SPS-Programmcode (siehe nachfolgende Beispiele).

Beispiel 1

In diesem Beispiel sind die SPS-Variablen bMemFlag1, bMemFlag2, bMemAlarm2 und iReadOnly über OPC UA freigegeben. Die SPS-Variable bMemAlarm1 soll nicht über den OPC UA Server zugänglich sein.

TwinCAT 3 (TMC-Import):

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

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

bMemAlarm1 : BOOL;

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

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

TwinCAT 2 (TPY-Import):

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

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

bMemAlarm1 : BOOL;

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

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

Wegen des zusätzlichen Kommentars `OPC.UA.DA.Access` wird die Zugriffsebene für die Variable `iReadOnly` auf „ReadOnly“ gesetzt. Die verschiedenen Möglichkeiten finden Sie in der vollständigen Liste der SPS-Kommentare (siehe Liste der Attribute und Kommentare).

● TPY-Import for TwinCAT 3



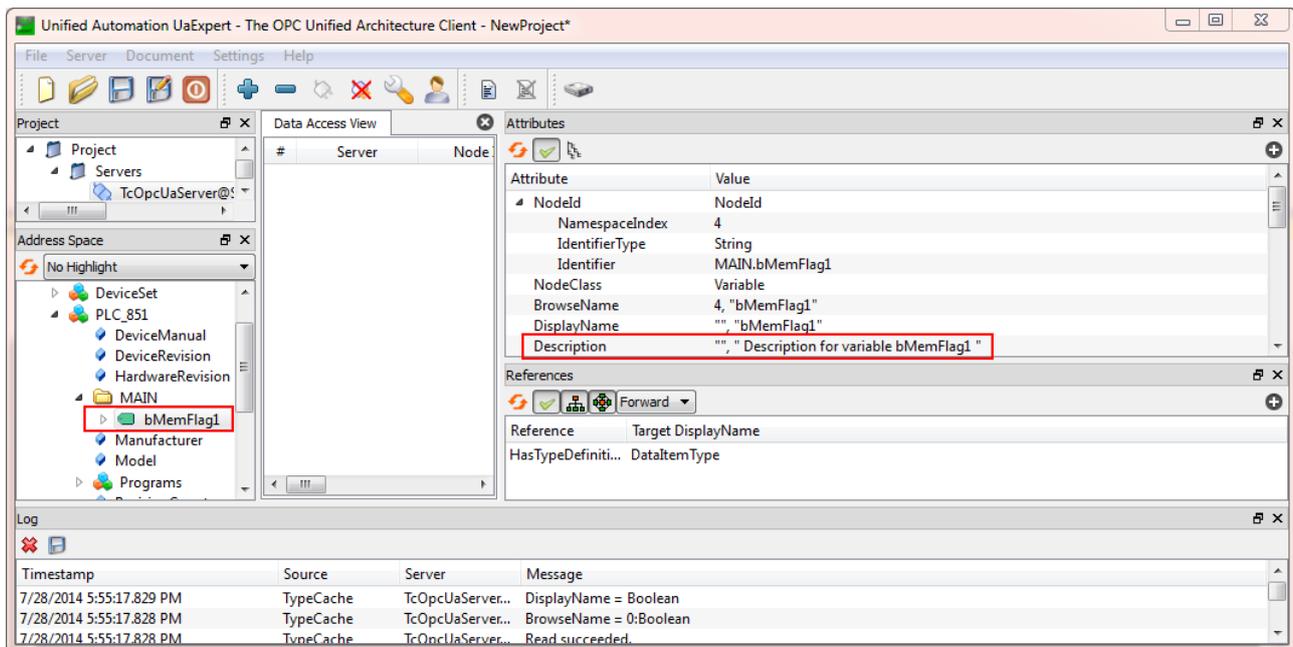
Der TPY-Import kann auch in TwinCAT 3 SPS-Projekten für Migrationszwecke verwendet werden, z. B. wenn Sie ein TwinCAT 2 Projekt in ein TwinCAT 3 Projekt konvertieren und keine Änderungen an den SPS-Kommentaren durchführen möchten. Bitte beachten Sie jedoch, dass erweiterte Funktionen teilweise nur für den neueren Importmechanismus (TMC) zur Verfügung stehen.

Sie können einer Variablen im Server eine Beschreibung geben, was in dem entsprechenden OPC UA Attribut ("Description") hinterlegt wird. Diese Funktion steht nur für TwinCAT 3 zur Verfügung.

TwinCAT 3 (TMC-Import):

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

Der Kommentar wird automatisch zum Beschreibungsattribut des Knotens im UA-Namensraum.



Beispiel 2:

In diesem Beispiel sollen die beiden Instanzen fbTest1 und fbTest2 des Funktionsbausteins FB_BLOCK1 über OPC UA verfügbar sein. Wenn eine ganze Instanz freigegeben wird, sind über OPC UA auch alle ihre Symbole verfügbar. Das SPS-Programm sieht wie folgt aus:

TwinCAT 3 (mit TMC-Import):

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

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

TwinCAT 2 (mit TPY-Import):

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

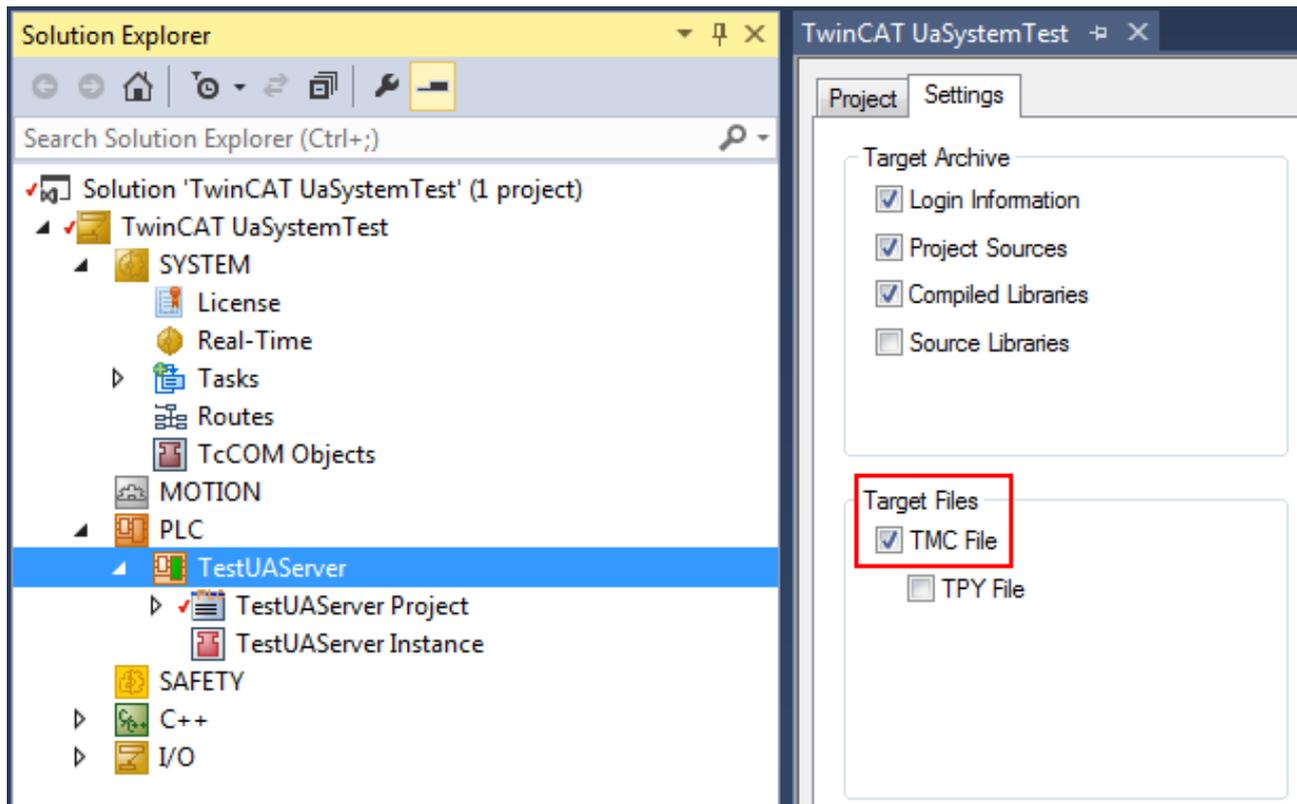
FUNCTION_BLOCK FB_BLOCK1
VAR_INPUT
    ni1 : INT; (*~ (OPC:1:some description) *)
    ni2 : INT;
END_VAR
VAR_OUTPUT
    no1 : INT; (*~ (OPC:1:some description) *)
    no2 : INT;
END_VAR
VAR
    nx1 : INT; (*~ (OPC:1:some description) *)
    nx2 : INT;
END_VAR
```

Die Instanz fbTest1 wird für OPC UA freigegeben, wodurch alle enthaltenen Symbole automatisch auch für OPC UA freigegeben werden, sprich fbTest.ni1, fbTest.ni2, usw. Die Instanz fbTest2 ist nicht für OPC UA gekennzeichnet, allerdings wurden die drei enthaltenen Variablen ni1, no1 und nx1 im Funktionsbaustein gekennzeichnet. Diese sind demzufolge in allen Instanzen über OPC UA verfügbar.

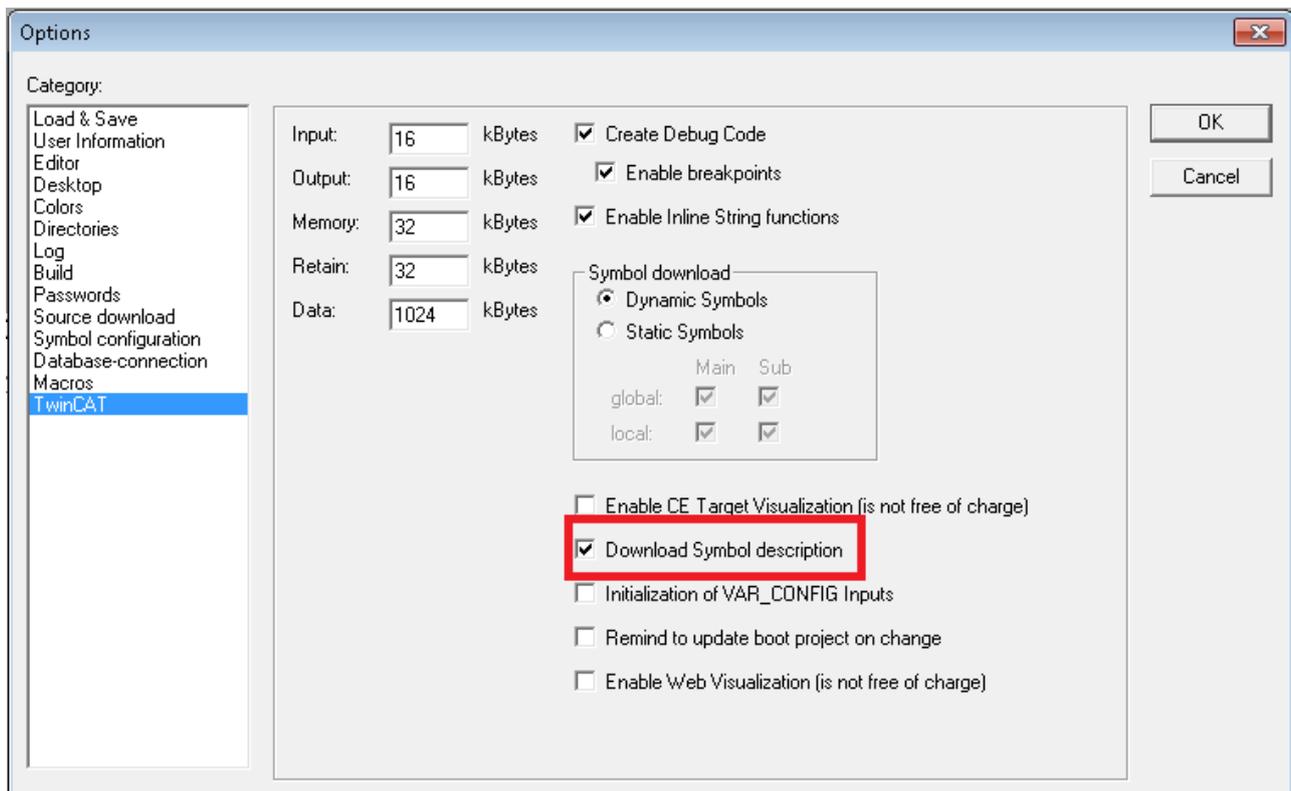
Schritt 2: Herunterladen der Symboldatei aktivieren

Die sogenannte Symboldatei enthält Informationen über alle in einem SPS-Projekt verfügbaren Variablen. Der TwinCAT OPC UA Server benötigt diese Informationen für den Aufbau seines Namensraums. Die aktuellen Symbolinformationen werden standardmäßig automatisch in eine Symboldatei generiert. Diese befindet sich im Projektordner des entsprechenden TwinCAT-Projekts. Damit die Symboldatei in die Ziel-Laufzeit übertragen wird, aktivieren Sie den Download der Symboldatei in den Einstellungen des SPS-Projekts.

TwinCAT 3:



TwinCAT 2:



[Optional] Schritt 3: Weitere Laufzeiten anbinden

Standardmäßig verbindet sich der TwinCAT OPC UA Server mit der ersten SPS-Laufzeit auf dem lokalen System. Wenn Sie mehr als eine SPS-Laufzeit im Server bereitstellen möchten, oder sich die SPS-Laufzeit auf einem anderen System befindet, dann müssen Sie entsprechende Einstellungen im TwinCAT OPC UA Konfigurator vornehmen.

Konfigurieren Sie alle weiteren notwendigen Parameter wie im Abschnitt [Allgemeine Information](#) [► 49] beschrieben.

i Anbindung von Remote Computern

Wenn Sie eine Laufzeit konfigurieren möchten, die sich auf einem remote Industrie-PC oder Embedded-PC befindet, so müssen Sie die richtigen Parameter für AmsNetId und AdsPort eingeben und die entsprechende Symboldatei des SPS-Programms auf diesem PC bereitstellen. Stellen Sie auch sicher, dass Sie eine ADS Route zu dem Remotesystem hergestellt haben.

[Optional] Schritt 4: Explizites Ausblenden von Symbolen

Neben den regulären SPS-Attributen zum Aktivieren eines SPS-Symbols können Sie SPS-Symbole auch explizit von einer Veröffentlichung im OPC-UA-Namensraum ausschließen. Dafür kann es mehrere Gründe geben:

- Sie möchten eine Datenstruktur veröffentlichen, aber nicht alle ihre Unterelemente (Beispiel 1).
- Sie haben eine Datenstruktur bei der Definition aktiviert und möchten z. B. eine einzelne Instanz dieser Datenstruktur ausblenden (Beispiel 2).

Beispiel 1:

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

PROGRAM MAIN
VAR
```

```

instance1 : ST_TEST;
instance2 : ST_TEST;
END_VAR

```

In diesem Fall wird das SPS-Attribut zur Strukturdefinition hinzugefügt, wodurch jede Instanz dieser Struktur auf dem OPC UA Server verfügbar ist. Das Unterelement b (und all seine Unterelemente, falls b eine weitere Datenstruktur ist) wird (werden) von der Veröffentlichung auf dem OPC UA Server ausgeschlossen.

Beispiel 2:

```

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

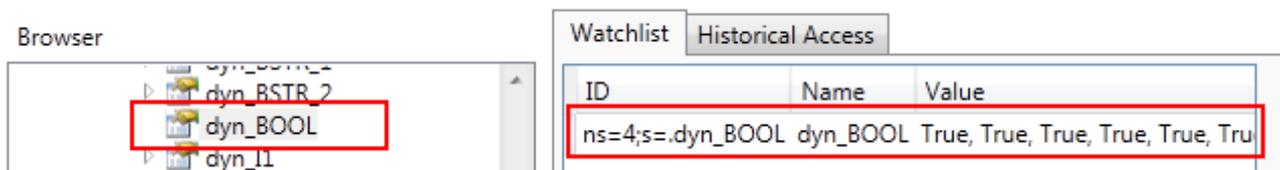
PROGRAM MAIN
VAR
  instance1 : ST_TEST;
  {attribute 'OPC.UA.DA' := '0'}
  instance2 : ST_TEST;
END_VAR

```

Obwohl das SPS-Attribut zur Strukturdefinition hinzugefügt wird, wodurch jede Instanz dieser Struktur auf dem OPC UA Server verfügbar ist, erhält instance2 das SPS-Attribut zum Deaktivieren dieser Vererbung. Dadurch ist nur instance1 im UA-Namensraum verfügbar, instance2 nicht.

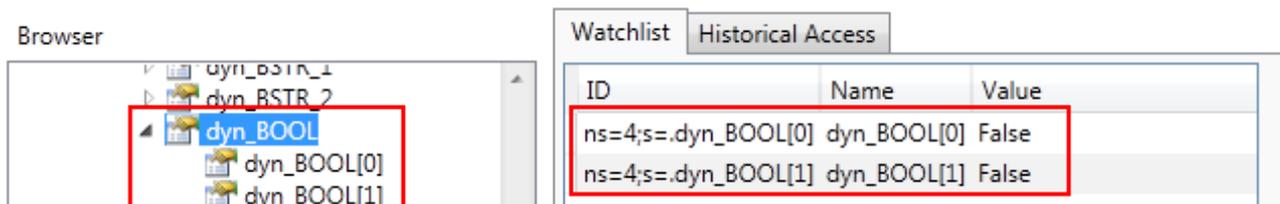
4.1.6.1.1 Arrays

Standardmäßig werden Arrays im UA-Namensraum als einzelner Knoten betrachtet. Dies bedeutet, dass, wenn Sie z. B. in der SPS ein Array dyn_BOOL[10] definieren (und dieses auch für OPC UA freigegeben haben), es anschließend im UA-Namensraum wie folgt auftaucht:



Der Vorteil dieser Vorgehensweise ist eine deutliche Reduzierung der Komplexität des UA-Namensraums und des Speicherverbrauchs, da nicht jede Position eines Arrays als einzelner Knoten im Namensraum zur Verfügung gestellt werden muss. Moderne UA Clients können jedoch weiterhin auf die einzelnen Array-Positionen über den sogenannten „RangeOffset“ zugreifen.

Damit jedoch auch ältere UA Clients unterstützt werden, welche dieses Feature nicht bieten, können Sie die Positionen eines Arrays auch als einzelne Knoten im UA-Namensraum verfügbar machen. Dies stellt sich wie folgt dar:



Diese Einstellung ist durch Aktivierung der Option **Legacy Array Handling** im UA-Konfigurator innerhalb der jeweiligen Namensraum-Konfiguration verfügbar.

Je nach Umfang des SPS-Projekts gewinnt der UA-Namensraum hierdurch deutlich an Komplexität, was sich wiederum in einer erhöhten Speicherauslastung des UA Servers widerspiegelt.

Eine Änderung der oben genannten Einstellungen wird erst aktiv, wenn Sie den UA Server neu starten.

4.1.6.1.2 Enums

Aufzählungen in OPC UA haben immer den Datentyp Int32. Die Norm IEC 61131-3 ermöglicht jedoch die Definition größerer Datentypen als Int32. Damit diese Aufzählungen ordnungsgemäß behandelt werden, bietet der TwinCAT OPC UA Server die Konfigurationsoption <ImportBigEnumsNumeric>, die in seiner Konfigurationsdatei Data Access aktiviert werden kann.

Diese Option ist standardmäßig auf FALSE gesetzt. Dies bedeutet, dass eine Statuscode-Ausnahme BadOutOfRange ausgelöst wird, wenn der Aufzählungswert außerhalb des Int32-Bereichs liegt.

Wenn die Option auf TRUE gesetzt wird, werden Aufzählungen mit größeren Datentypen als Int32 als reguläre Variablen mit diesem bestimmten Datentyp behandelt.

Angenommen, wir haben die folgenden Aufzählungsdefinitionen in unserem SPS-Code:

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

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

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

Bei Aktivierung der vorstehenden Konfigurationsoption werden Instanzen von E_Enum_VeryBig als reguläre Variable vom Datentyp Int64 im Namensraum des Servers behandelt, während Instanzen von E_Enum_Normal und E_Enum_NotSoBig als OPC UA-Aufzählung (mit dem Datentyp Int32) behandelt werden:

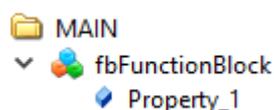
#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Status
1	TcOpcUaServer...	NS4 String MAI...	eEnumVeryBig	0	Int64	09:05:56.115	09:05:56.115	Good
2	TcOpcUaServer...	NS4 String MAI...	eEnumNormal	0 (enum_member_0)	Int32	09:05:59.115	09:05:59.115	Good
3	TcOpcUaServer...	NS4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

4.1.6.1.3 Properties

Die Anzeige von SPS Properties im Namensraum des Servers wird seit TwinCAT 3.1 Build 4024 unterstützt. Hierzu müssen am Property lediglich zwei spezielle SPS Attribute gesetzt werden, damit das Property in den Namensraum importiert und dort als UA Property dargestellt wird. Beispiel:

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

Der Funktionsbaustein an dem das Property definiert ist, muss das normale SPS Attribut zur Freigabe von Symbolen [► 49] enthalten.



In der Konfigurationsdatei TcUaDaConfig.xml kann global das Handling von SPS Properties festgelegt werden. Hierzu dient das Flag „ImportPlcProperties“.

Wert	Beschreibung
false	Zeigt alle Properties im OPC UA Namensraum an, bei denen sowohl das OPC.UA.DA.Property als auch monitoring-Attribut gesetzt wurden.
true	Zeigt alle Properties im OPC UA Namensraum an, bei denen das monitoring-Attribut gesetzt wurde.

4.1.6.1.4 StructuredTypes

StructuredTypes erlauben es, Strukturen zu lesen oder zu schreiben ohne dafür jedes Byte zu interpretieren, weil der UA Server den Informationstyp jedes Elements der Struktur zurückgibt. Durch komplexe Funktionen in modernen OPC UA SDKs können OPC UA Clients diese strukturellen Informationen durchsuchen und interpretieren.

Ab Version 2.2.x des OPC UA Servers werden Strukturen von der TwinCAT-3-Laufzeit (nur TMC- und TMI-Import) als ein StructuredType im UA-Namensraum erzeugt.

Beispiel:

STRUCT ST_Communication:

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

Programm MAIN:

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

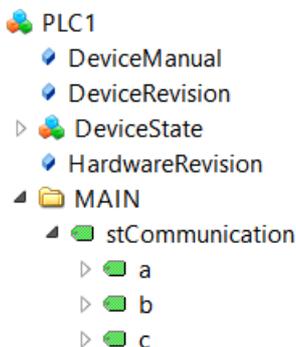
● Gefilterter Modus

i Werden Filter eingesetzt, um Symbole über OPC UA verfügbar zu machen, muss ein STRUCT oder Funktionsbaustein im UA-Namensraum voll verfügbar sein, um als StructuredType angezeigt zu werden.

● Pointer und Referenzen

i Werden in der Struktur Pointer- und Referenztypen verwendet, dann können diese nicht in einen StructuredType überführt werden. Der OPC UA Server stellt diese Strukturen dann als reguläre FolderTypes mit der entsprechenden Membervariablen dar.

Die Instanz stCommunication wird dann im UA-Namensraum als StructuredType angezeigt, inklusive aller Membervariablen:



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

Alternativ kann die STRUCT-Definition auch das SPS Attribut erhalten, um alle Instanzen von STRUCT als StructuredType verfügbar zu machen. Sollen die Membervariablen nicht explizit dargestellt werden, so können diese mit dem Attribut OPC.UA.DA:=2 ausgeblendet werden.

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

Um StructuredType einer bestimmten Instanz zu deaktivieren, verwenden Sie das folgende Attribut:

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

Funktionsbaustein StructuredType

Zusätzlich enthält jeder Funktionsbaustein der TwinCAT 3 SPS auch einen Kindknoten FunctionBlock, der den gesamten Funktionsbaustein als StructuredType beinhaltet.

Beispiel:

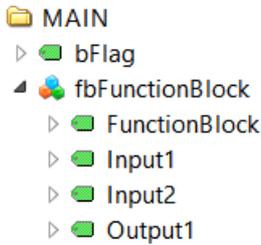
Funktionsbaustein:

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

Instanz des Funktionsbausteins:

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

Instanz des Funktionsbausteins im OPC-UA-Namensraum:



FunctionBlock-Knoten mit StructuredType:

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

Alternativ kann der Funktionsbaustein auch ein SPS-Attribut erhalten, um alle Instanzen des Funktionsbausteins als StructuredType zur Verfügung zu stellen.

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

Um StructuredType einer bestimmten Instanz zu deaktivieren, verwenden Sie das folgende Attribut:

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

● Maximale Größe der Struktur

i Die maximale Größe einer Struktur ist standardmäßig auf 16 kByte festgelegt. Jedes STRUCT tauscht ständig Daten mit dem Basis-ADS-Gerät aus, d. h. mit jedem Lese-/Schreibbefehl eines StructuredTypes wird eine große ADS-Meldung verschickt. Damit der ADS-Router nicht mit großen Meldungen überflutet wird, ist die maximale Größe beschränkt. Sie können diese Vorgabe in der Datei *TcUaDaConfig.xml* ändern. Dazu muss der Schlüssel <MaxStructureSize> in der Datei hinzugefügt und ein neuer Wert für die maximale Größe einer Struktur in Bytes festgelegt werden. Wenn eine Struktur die Größe <MaxStructureSize> überschreitet, wird sie als FolderType importiert, wo jedes Strukturelement als einzelner Knoten verfügbar ist.

4.1.6.1.5 AnalogItemTypes

AnalogItemTypes sind ein Bestandteil der OPC-UA-Spezifikation und ermöglichen es, Metainformationen wie z. B. Einheiten an eine Variable zu heften. In der TwinCAT 3 SPS können Sie diese Metainformationen in Form von SPS-Attributen definieren.

Folgende Einstellungen sind möglich:

- EngineeringUnits: Einheiten, definiert über die OPC-UA-Spezifikation
- EURange: Maximaler Wertebereich der Variablen
- InstrumentRange: Normaler Wertebereich der Variablen
- WriteBehavior: Verhalten, wenn bei einem Schreibvorgang der Wertebereich überschritten wird.

Das nachfolgende Beispiel zeigt, wie die Variable fillLevel als AnalogItemType konfiguriert wird. Folgenden Parameter werden hierbei gesetzt:

- Einheit: 20529 („Prozent“, definiert in der OPC-UA-Spezifikation)
- Max. Wertebereich: 0 bis 100
- Normaler Wertebereich: 10 bis 90
- Schreibverhalten: 1 (Clamping)

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

EngineeringUnits können hierbei anhand der in OPC UA spezifizierten IDs (Teil 8 der OPC-UA-Spezifikation) konfiguriert werden. Die IDs orientieren sich hierbei nach den weit verbreiteten und akzeptierten „Codes for Units of Measurement (Recommendation N.20)“, die vom „United Nations Centre for Trade Facilitation and Electronic Business“ veröffentlicht wurden. CommonCode, der die dreistellige, alphanumerische ID angibt, wird von OPC UA laut Spezifikation in einen Int32-Wert konvertiert und referenziert (Auszug aus OPC-UA-Spezifikation v1.02, Pseudo-Code):

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

Schreibverhalten

Beim Schreiben einer AnalogItemType-Variablen können Sie definieren, wie der OPC UA Server mit dem neuen Wert in Bezug auf den Wertebereich umgehen soll. Hierbei gibt es die folgenden Möglichkeiten:

- 0: Alle Werte sind zugelassen und werden bei einem Schreibvorgang übernommen.
- 1: Der zu schreibende Wert wird passend zum Wertebereich abgeschnitten.
- 2: Der zu schreibende Wert wird abgewiesen, falls er den Wertebereich überschreitet.

4.1.6.1.6 Pointer und Referenzen

Pointer

Pointervariablen (z.B. POINTER TO) werden grundsätzlich nicht vom Server im Namensraum dargestellt. Falls sich eine Pointervariable in einer Struktur befindet und diese als [Structured Data Type](#) [► 57] konfiguriert wurde, so wird die Struktur nicht als Structured Data Type sondern als FolderType dargestellt.

Referenzen

Referenzvariablen (REFERENCE TO) werden als Einzelvariablen vom Server im Namensraum dargestellt und können ohne Einschränkungen gelesen werden. Befindet sich eine Referenz innerhalb einer Struktur, kann diese Struktur nicht mehr als [StructuredTypes](#) [► 57] im Server zur Verfügung gestellt werden, sondern nur als FolderType. Der Zugriff auf die einzelnen Referenzvariablen innerhalb der Struktur funktioniert hingegen.

4.1.6.1.7 Typsystem

Einer der größten Vorteile von OPC UA ist das Meta-Modell, das zur Bereitstellung von Basistypen genauso verwendet werden kann wie zur Erweiterung des Typsystems durch eigene Modelle. Derselbe Mechanismus wird zur Darstellung realer Objekte (Nodes) verwendet, sodass OPC UA Clients einen Objekttyp bestimmen können.

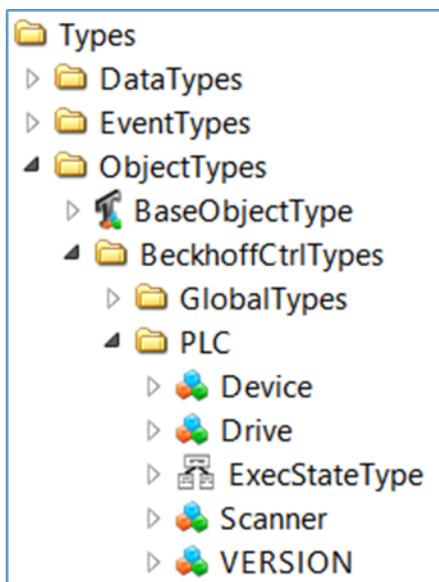
Der OPC UA Server veröffentlicht Typinformationen von der IEC61131-Welt in seinen Namensraum. Dies umfasst nicht nur Basistypen, wie z. B. BOOL, INT, DINT oder REAL, sondern auch erweiterte Typinformationen, wie z. B. die aktuelle Klasse (Funktionsbaustein) oder Struktur, die ein Objekt repräsentiert.

Typinformationen

Typinformationen sind Bestandteil des UA-Namensraums. Der OPC UA Server erweitert die Basistypinformationen wie folgt:

- Lokale Typinformationen, die nur für eine Laufzeit gültig ist, werden in demselben Namensraum wie die Laufzeitsymbole gespeichert.
- Globale Typinformationen, die über verschiedene Laufzeiten gültig sein können, werden in einem eigenen globalen Namensraum gespeichert.

Das Typsystem ist ebenfalls virtuell verfügbar und kann im Bereich Types des OPC UA Servers eingesehen werden:

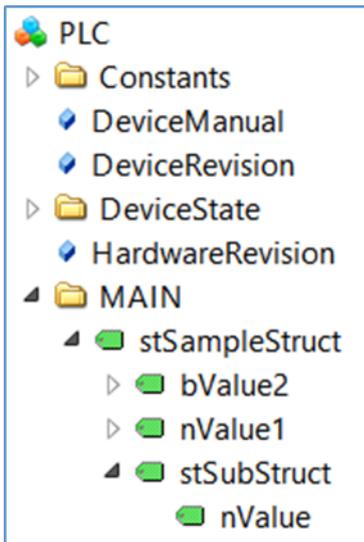


Jeder Nicht-Standarddatentyp wird im Bereich BeckhoffCtrlTypes eingetragen.

Grundlagen

Angenommen, die TwinCAT 3 SPS besteht aus einem SPS-Programm mit verschiedenen STRUCTs. Jeder STRUCT wird als Knoten in einem UA-Namensraum repräsentiert, mit jedem Element der Struktur als untergeordneten Knoten.

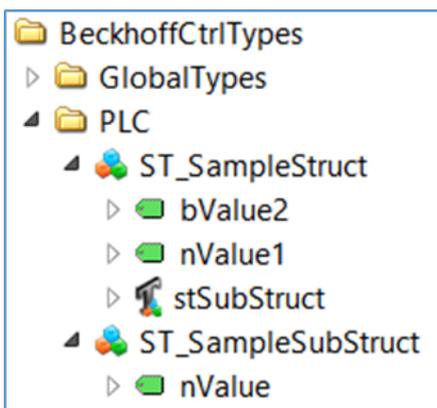
In dem Beispiel besteht der STRUCT stSampleStruct aus drei untergeordneten Elementen: einer Variable nValue1 vom Typ INT, einer Variable bValue2 vom Typ BOOL und einem weiteren STRUCT stSubStruct, das mit der Variablen nValue vom Typ INT lediglich ein untergeordnetes Element enthält.



Die Struktur selbst ist eine reguläre Variable im UA-Namensraum, die den Datentyp ByteString hat. Die Clients können daher einfach mit dem Wurzelement (der Struktur selbst) verbunden werden und dessen Werte durch Interpretation des ByteString gelesen/geschrieben werden. Damit die Interpretation jedes untergeordneten Elements vereinfacht wird, enthält das Typsystem mehr Informationen über die Struktur selbst, in erster Linie in der Referenz zur Instanz:

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

Außerdem enthält das Typsystem mehr Informationen über ST_SampleStruct:



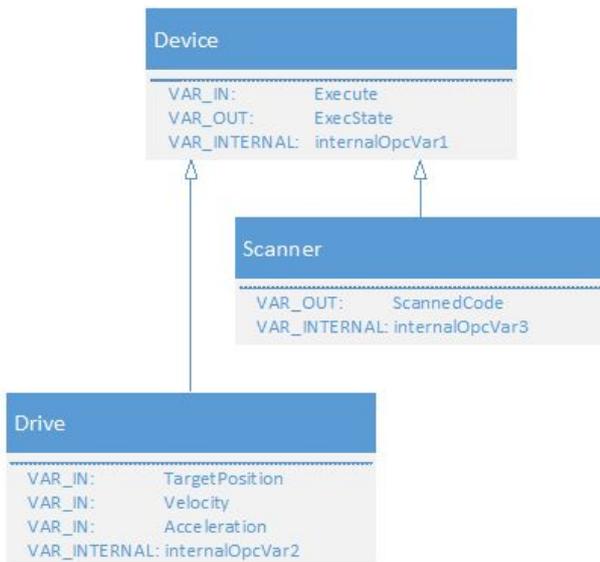
Und in den Referenzen von ST_SampleStruct:

Reference	Target DisplayName
HasTypeDe...	DataItem Type
HasCompo...	nValue1
HasCompo...	bValue2
HasCompo...	stSubStruct

Insgesamt kann das Typsystem sehr nützliche Informationen bieten, wenn ein Client die Struktur weiter interpretieren will.

Objektorientierte Erweiterungen

Angenommen, die TwinCAT-3-SPS-Laufzeit enthält ein SPS-Programm, dessen Struktur wie folgt visualisiert werden kann:

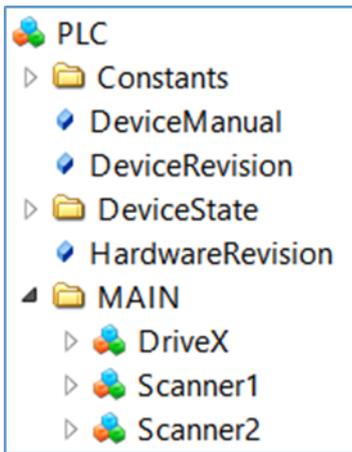


Die zwei Funktionsbausteine Scanner und Drive sind von der Basisklasse Device abgeleitet, indem objektorientierte Erweiterungen der IEC61131-3 verwendet werden. Das MAIN-Programm enthält nun die folgenden Deklarationen:

```

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

Alle drei Objekte sind vom Typ Device, aber auch von ihrem speziellen Datentyp. Der OPC UA Server importiert die Objekte wie folgt:



Der zugrunde liegende Datentyp kann nun in der Referenz jedes Objekts bestimmt werden, z. B. Objekt Scanner1:

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

Entsprechend des zugrundeliegenden IEC61131-Programms ist das Objekt Scanner1 vom Datentyp Scanner und zeigt ebenfalls, welche Variablen enthalten sind und welcher Art die Variable ist: Eingangs-, Ausgangs- oder interne (lokale) Variable. Das Diagramm oben zeigt, dass nicht nur die Variablen des tatsächlichen Funktionsbausteins hier wiedergegeben werden, sondern auch die abgeleiteten Variablen der Basisklasse. Die gesamte IEC61131-3-Vererbungskette wird im UA-Namensraum repräsentiert.

4.1.6.1.8 StatusCode

Der OPC UA Server ermöglicht einer SPS-Anwendung die Änderung des OPC UA StatusCode.

Führen Sie die folgenden Schritte aus, um den StatusCode einer Variablen zu konfigurieren und auf diesen einzuwirken:

- [SPS-Struktur erstellen](#) [► 64]
- [StatusCode überschreiben](#) [► 65]

SPS-Struktur erstellen

Damit die Datenkonsistenz gewährleistet ist, basiert das Konzept der Änderung des OPC UA StatusCode auf StructuredTypes (siehe [StructuredTypes](#) [► 57]). Jede Variable, auf deren UA StatusCode eingewirkt werden soll, muss in ein STRUCT eingeschlossen sein.

Erzeugen Sie ein neues STRUCT und fügen Sie ein SPS-Attribut vor der STRUCT-Definition hinzu. Das STRUCT enthält die Variable selbst und eine Variable vom Datentyp DINT, die den StatusCode darstellt und auf die im Attribut vor der STRUCT-Definition verwiesen wird.

```

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

Erstellen Sie nun eine Instanz von diesem STRUCT, z. B. im MAIN-Programm, und fügen Sie das reguläre SPS-Attribut hinzu, damit die Instanz über OPC UA verfügbar wird.

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

Diese Instanz ist nun innerhalb des OPC-UA-Namensraums als ein StructuredType verfügbar.

PLC1

- DeviceManual
- DeviceRevision
- DeviceState
- HardwareRevision
- MAIN
 - stStatusCodeOverride
 - quality
 - value

Value	ST_StatusCodeOverride
value	0
quality	0

StatusCode überschreiben

Um den UA StatusCode für das STRUCT zu überschreiben, bearbeiten Sie einfach den Wert der Variablen „quality“. Wenn Sie diesen z. B. auf „-2147155968“ setzen, ändert sich der StatusCode des STRUCT zu „BadCommunicationError“.

MAIN [Online]

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

Data Access View

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

Der Wert muss entsprechend der Definition in der OPC-UA-Spezifikation festgelegt werden.

In der nachfolgenden Tabelle werden nur einige der verfügbaren StatusCodes und deren entsprechende dezimale Darstellung aufgelistet. Konsultieren Sie die offizielle OPC-UA-Spezifikation für eine umfassendere Liste der StatusCodes.

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

UA StatusCodes

Beachten Sie bei der Berechnung der dezimalen Darstellung anderer UA StatusCodes auf der Grundlage ihrer hexadezimalen Darstellung, dass Ihr Rechner auf DWORD eingestellt ist, z. B. der Windows-Rechner („Programmierer“-Ansicht).

4.1.6.1.9 Liste der Attribute und Kommentare

Die Laufzeitvariablen für verschiedene OPC-UA-Funktionen (z. B. Data Access oder Historical Access) konfigurieren Sie direkt im SPS-Programm oder im TMC-Code-Editor (bei Verwendung von TwinCAT 3 C++). Der Vorteil ist, dass ein SPS-Entwickler direkt in dem Programm, mit dem er vertraut ist, entscheiden kann, ob und wie eine Variable für OPC UA freizugeben ist. Sie aktivieren eine Variable, indem Sie einen Kommentar und das entsprechenden OPC-UA-Tags vor der Variablen einfügen, z. B.:

TwinCAT 3 SPS (TMC):

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

TwinCAT 2 SPS (TPY):

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

Eine detaillierte Beschreibung zur Verwendung von Attributen und Kommentaren erhalten Sie in den Abschnitten über die entsprechenden Laufzeitkomponenten:

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

Die folgende Tabelle zeigt einen Überblick aller definierbaren Tags und ihrer Bedeutung. In den Unterschnitten der entsprechenden Funktion finden Sie eine ausführliche Beschreibung des Funktionsprinzips.

TwinCAT 3 (TMC):

OPC-UA-Funktion	SPS-Tag	C++ TMC-Code-Editor (Optionale Eigenschaften)	Bedeutung
Data Access (DA)	{attribute 'OPC.UA.DA' := '0'}	Name: OPC.UA.DA Wert: 0	Sperrt eine Variable für OPC UA, woraufhin diese im UA-Namensraum nicht mehr sichtbar ist.
Data Access (DA)	{attribute 'OPC.UA.DA' := '1'}	Name: OPC.UA.DA Wert: 1	Aktiviert eine Variable für OPC UA, woraufhin diese im UA-Namensraum sichtbar wird. Dieses Tag muss immer gesetzt sein, wenn eine Variable für UA verwendet werden soll.
Data Access (DA)	{attribute 'OPC.UA.DA.Access' := 'x'}	Name: OPC.UA.DA.Access Wert: siehe rechte Spalte	Setzt Lese-/Schreib-Zugriff für eine Variable, je nach Parameter „x“. 0 = Kein 1 = Schreibgeschützt 2 = Nur Schreibzugriff 3 = Lese- und Schreibzugriff (Standardwert, wenn kein Tag verwendet wird)
Data Access (DA)	{attribute 'OPC.UA.DA.Alias' := '1'}	Name: OPC.UA.DA.Alias Wert: siehe rechte Spalte	Bestimmt x als Knotennamen im UA-Namensraum, sogenanntes Alias Mapping.
Data Access (DA)	{attribute 'OPC.UA.DA.Description' := 'x'}	Name: OPC.UA.DA.Description Wert: siehe rechte Spalte	Setzt einen Text für das OPC-UA-Attribut „Description“.
Data Access (DA)	{attribute 'OPC.UA.DA.StructuredType' := '0'}	Name: OPC.UA.DA.StructuredType Wert: 0	Deaktiviert StructuredType für ein STRUCT oder einen Funktionsbaustein
Data Access (DA)	{attribute 'OPC.UA.DA.StructuredType' := '1'}	Name: OPC.UA.DA.StructuredType Wert: 1	Aktiviert StructuredType für ein STRUCT oder einen Funktionsbaustein
Data Access	{attribute 'OPC.UA.DA.Status' := 'quality'}	Name: OPC.UA.DA.Status Wert: quality	StatusCode eines Symbols im UA-Namensraum manuell festlegen. Nur für Datenstrukturen. Der Wert „quality“ legt fest, welches DINT Unterelement der Datenstruktur den StatusCode festlegt. Siehe entsprechenden Artikel der Dokumentation für weitere Informationen.
<u>Historical Access (HA)</u> [► 76]	{attribute 'OPC.UA.HA' := '1'}	Name: OPC.UA.HA Wert: 1	Aktiviert eine Variable für Historical Access. Muss mit {attribute 'OPC.UA.DA' := '1'} verwendet werden.

OPC-UA-Funktion	SPS-Tag	C++ TMC-Code-Editor (Optionale Eigenschaften)	Bedeutung
Historical Access (HA) [► 76]	{attribute 'OPC.UA.HA.Storage' := 'x'}	Name: OPC.UA.HA.Storage Wert: siehe rechte Spalte	Definiert den Speicherort für Historical Access, abhängig von Parameter „x“ 1 = Speicher 2 = Datei 3 = SQL Compact Database 4 = SQL Server Database
Historical Access (HA) [► 76]	{attribute 'OPC.UA.HA.Sampling' := 'x'}	Name: OPC.UA.HA.Sampling Wert: siehe rechte Spalte	Legt die Abtastrate fest, mit der die Variablenwerte zu speichern sind, in Abhängigkeit des Parameters „x“ in [ms]
Historical Access (HA) [► 76]	{attribute 'OPC.UA.HA.Buffer' := 'x'}	Name: OPC.UA.HA.Buffer Wert: siehe rechte Spalte	Definiert die maximale Anzahl Werte, die im Datenspeicher bleiben, in Abhängigkeit des Parameters „x“.

TwinCAT 2 (TPY):

OPC-UA-Funktion	SPS-Tag	Bedeutung
Data Access (DA)	(*~ (OPC:0:not available) *)	Sperrt eine Variable für OPC UA, woraufhin diese im UA-Namensraum nicht mehr sichtbar ist.
Data Access (DA)	(*~ (OPC:1:available) *)	Aktiviert eine Variable für OPC UA, woraufhin diese im UA-Namensraum sichtbar wird. Dieses Tag muss immer gesetzt sein, wenn eine Variable für UA verwendet werden soll.
Data Access (DA)	(*~ (OPC_PROP[0005]:1:Schreibgeschützt) *)	Setzt den Schreibschutz für eine Variable. Muss zusammen mit (*~ (OPC:1: available) *) verwendet werden.
Data Access (DA)	(*~ (OPC_UA_PROP[5100] : x: Alias name) *)	Bestimmt x als Knotennamen im UA-Namensraum, sogenanntes Alias Mapping.
Historical Access (HA) [► 76]	(*~ (OPC_UA_PROP[5000]:x:Storage media) *)	Aktiviert eine Variable für „Historical Access“. Muss zusammen mit (*~ (OPC:1: available) *) verwendet werden. x definiert das Speichermedium für die Speicherung der Datenwerte: 1 = Speicher 2 = Datei 3 = SQL Compact Database 4 = SQL Server Database
Historical Access (HA) [► 76]	(*~ (OPC_UA_PROP[5000] [1]:x:SamplingRate) *)	Legt die Abtastrate fest, mit der die Variablenwerte zu speichern sind, in Abhängigkeit des Parameters „x“ in [ms]
Historical Access (HA) [► 76]	(*~ (OPC_UA_PROP[5000][2]:x:Buffer) *)	Definiert die maximale Anzahl Werte, die im Datenspeicher bleiben, in Abhängigkeit des Parameters „x“.

4.1.6.2 C++

In diesem Abschnitt wird beschrieben, wie Sie den Namensraum des TwinCAT OPC UA Servers konfigurieren, um Zugriff auf die Symbole eines TwinCAT-3-C++-Moduls zu erhalten. Hierzu können Sie entweder den OPC-UA-Konfigurator verwenden oder die Konfiguration direkt in der Datei *ServerConfig.xml* des OPC UA Servers vornehmen.

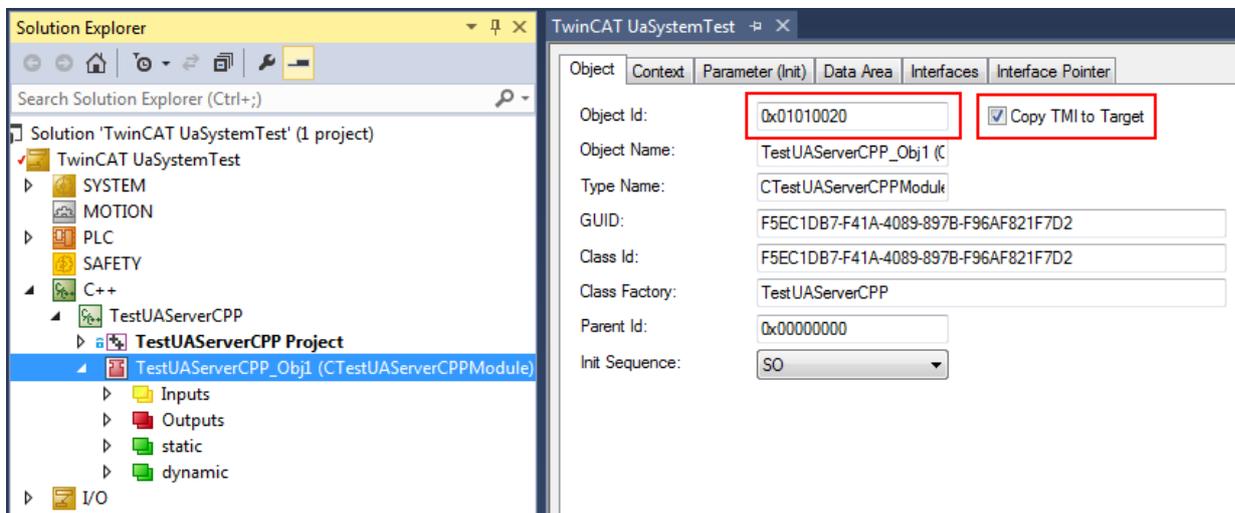
Dieser Abschnitt beinhaltet folgende Themen:

- Schritt 1: [Projektbezogene Einstellungen](#) [► 69]
- Schritt 2: [UA-Server konfigurieren](#) [► 70]

Schritt 1: Projektbezogene Einstellungen

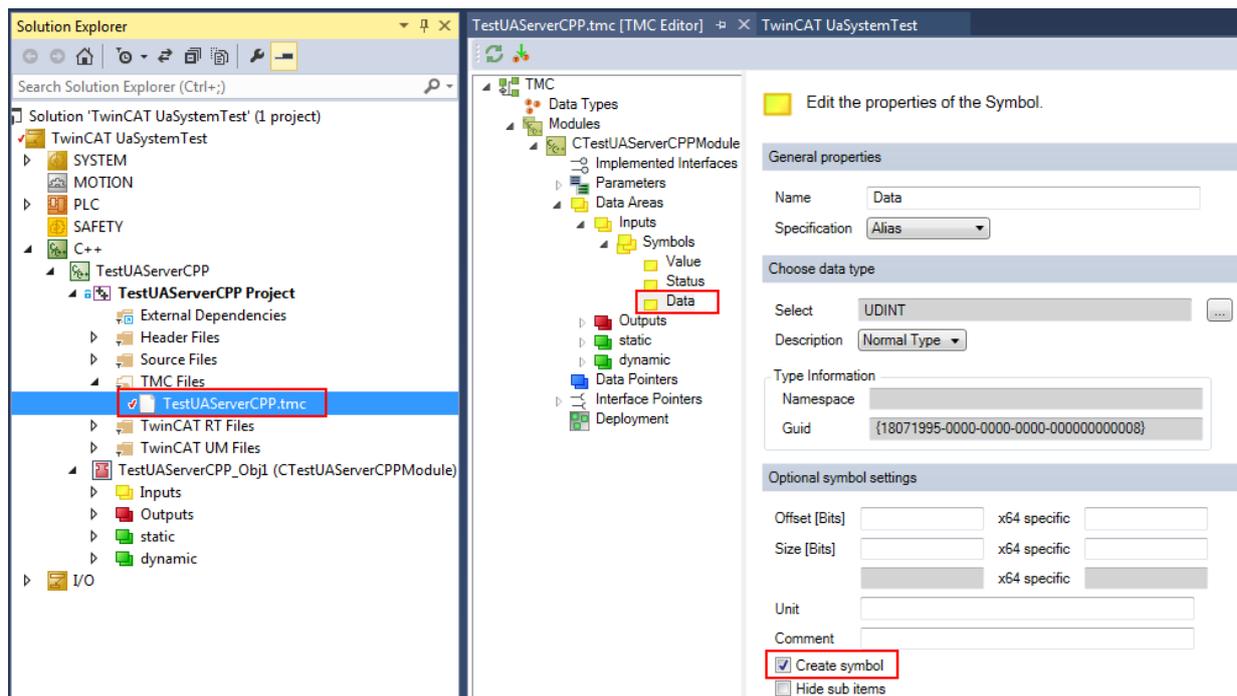
Um bestimmte, in einer Instanz eines C++-Moduls enthaltene Symbole so zu konfigurieren, dass sie über OPC UA zugänglich werden, sind folgende Einstellungen in der entsprechenden C++-Modulinstantz in TwinCAT XAE erforderlich.

1. Der OPC UA Server benötigt die TMI-Symboldatei, die standardmäßig nicht an die Ziel-Laufzeit übergeben wird. Aktivieren Sie die Übertragung durch Setzen der folgenden Optionen:



- ⇒ Die erzeugte TMI-Datei wird nach der Object Id benannt, z. B. „Obj_01010020.tmi“ und in das TwinCAT-Bootverzeichnis, z. B. *C:\TwinCAT\3.1\Boot\Tmi*, abgelegt.

2. Legen Sie fest, welche Symbole für die entsprechenden Variablen zugänglich gemacht werden sollen, indem Sie das Kontrollkästchen **Create symbol** im TMC-Code-Generator aktivieren. Führen Sie den TMC-Code-Generator anschließend aus.



Schritt 2: UA Server konfigurieren

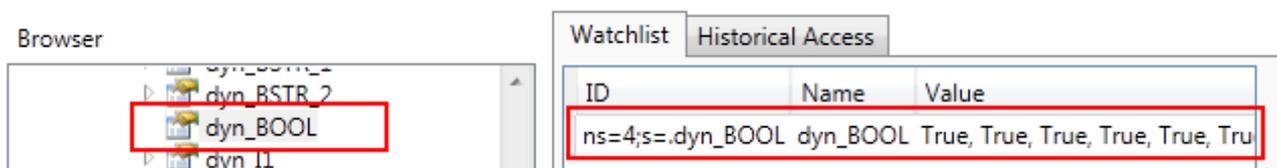
Sie können den Zugang zu TwinCAT-3-C++-Modulen einfach mithilfe des OPC-UA-Konfigurators konfigurieren und parametrisieren. Fügen Sie hierzu im Bereich **Datenzugriff** ein neues Gerät vom Typ „CPP TwinCAT 3 (TMI) gefiltert“ hinzu. Das Feld **SymbolFile** muss auf die TMI-Datei zeigen, die für die C++-Modulinstantz erzeugt wurde. Diese TMI-Datei befindet sich im TwinCAT-Bootverzeichnis, z. B. C: \TwinCAT\3.1\Boot\Tmi, und wurde nach der ObjectID der TwinCAT-3-C++-Modulinstantz benannt.

Die einstellbaren Parameter beschreiben die folgenden Funktionen:

Parameter	Beschreibung	Mögliche Werte
ADS Port	Definiert den ADS-Port, unter dem die C++-Modulinanz zugänglich ist. Der ADS-Port kann in den Eigenschaften des C++-Tasks gelesen werden.	351 352 ...
AutoCfg	Definiert zunächst den Typ der für die Kommunikation verwendeten Ziel-Laufzeit, z. B. SPS, C++, I/O. Anschließend kann eine weitergehende Unterscheidung innerhalb dieser Kategorien stattfinden. Jede AutoCfg-Option steht sowohl als ungefilterte, als auch gefilterte Option zur Verfügung. Gefiltert bedeutet, dass der Nutzer bestimmen kann, welche C++-Symbole über OPC UA öffentlich zugänglich gemacht werden. Bei Verwendung einer ungefilterten Option wird jedes C++-Symbol an OPC UA veröffentlicht.	4020 CPP TwinCAT 3 (TMI) 4021 CPP TwinCAT 3 (TMI) gefiltert
AutoCfgSymFile	Symboldatei (TMI) der entsprechenden C++-Modulinanz.	Pfad zur Symboldatei. (TMI)
IoMode	Definiert die Methode für den Zugriff auf Symbole.	1 (Zugriff über Handle - standardmäßig)
ArraySubItemLegacy Support	Unterelemente eines Array werden standardmäßig nicht als separate Knoten im UA-Namensraum abgebildet. Stattdessen wird lediglich das Array als ein einzelnes Element abgebildet. Nichtsdestotrotz können UA Clients über deren sogenannten „IndexRange“ auf Unterelemente zugreifen. (Einige ältere OPC UA Clients unterstützen diese Möglichkeit noch nicht.) Das Flag wurde eingeführt, damit der Zugriff dennoch für diese Clients möglich wird. Es sorgt dafür, dass jede Array-Position als separater Knoten im UA-Namensraum angezeigt wird. Dies führt zu einem höheren Speicherbedarf des OPC UA Servers.	0 (deaktiviert - standardmäßig) 1 (aktiviert)
Disabled	Deaktiviert die C++-Modulinanz im UA-Namensraum, woraufhin der entsprechende Knoten nicht angezeigt wird. Es wird empfohlen, diesen Parameter zu aktivieren, wenn bestimmte C++-Laufzeiten zum Zeitpunkt der Projektplanung noch nicht verfügbar sind, weil z. B. die entsprechenden Geräte noch nicht ans Netzwerk angeschlossen sind.	0 (deaktiviert - standardmäßig) 1 (aktiviert)

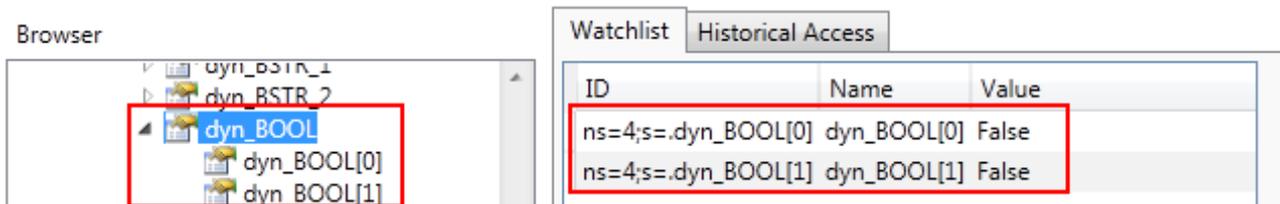
4.1.6.2.1 Arrays

Standardmäßig werden Arrays im UA-Namensraum als einzelner Knoten betrachtet. Dies bedeutet, dass, wenn Sie z. B. in der SPS ein Array `dyn_BOOL[10]` definieren (und dieses auch für OPC UA freigegeben haben), es anschließend im UA-Namensraum wie folgt auftaucht:



Der Vorteil dieser Vorgehensweise ist eine deutliche Reduzierung der Komplexität des UA-Namensraums und des Speicherverbrauchs, da nicht jede Position eines Arrays als einzelner Knoten im Namensraum zur Verfügung gestellt werden muss. Moderne UA Clients können jedoch weiterhin auf die einzelnen Array-Positionen über den sogenannten „RangeOffset“ zugreifen.

Damit jedoch auch ältere UA Clients unterstützt werden, welche dieses Feature nicht bieten, können Sie die Positionen eines Arrays auch als einzelne Knoten im UA-Namensraum verfügbar machen. Dies stellt sich wie folgt dar:



Diese Einstellung ist durch Aktivierung der Option **Legacy Array Handling** im UA-Konfigurator innerhalb der jeweiligen Namensraum-Konfiguration verfügbar.

Je nach Umfang des SPS-Projekts gewinnt der UA-Namensraum hierdurch deutlich an Komplexität, was sich wiederum in einer erhöhten Speicherauslastung des UA Servers widerspiegelt.

Eine Änderung der oben genannten Einstellungen wird erst aktiv, wenn Sie den UA Server neu starten.

4.1.6.3 Matlab/Simulink

In diesem Abschnitt wird beschrieben, wie Sie TwinCAT-3-TMI-Dateien für TcCOM-Module zum Aufbau des Namensraums des OPC UA Servers nutzen.

Dieser Abschnitt beinhaltet folgende Themen:

- [Allgemeine Informationen](#) [► 72]
- [TMI-Datei für TcCom-Module generieren und importieren](#) [► 72]
- [Gefilterter oder ungefilterter Modus](#) [► 74]

Allgemeine Informationen

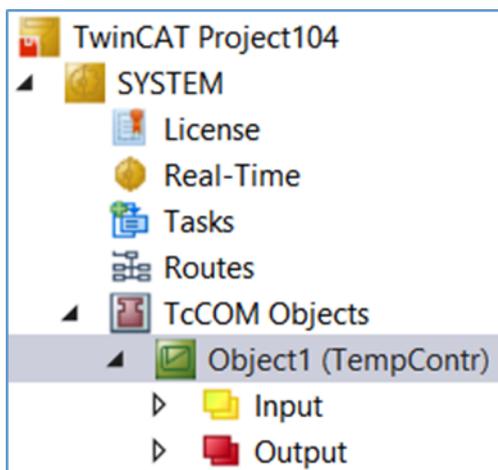
Wenn Sie TwinCAT 3 C++ oder die Matlab/Simulink-Integration verwenden, können Sie für die entstehenden TcCom-Module aus dem TwinCAT 3 XAE heraus eine Symboldatei (TMI-Datei) generieren.

Diese TMI-Datei importiert der OPC UA Server und erzeugt mit den darin enthaltenen Symbolinformationen seinen Namensraum. Dieser kann dann aus Variablen (=Symbolen) bestehen, die in dem jeweiligen TcCom-Modul enthalten sind.

Sie konfigurieren den Import von TMI-Dateien im OPC-UA-Konfigurator.

TMI-Datei für TcCOM-Module generieren und importieren

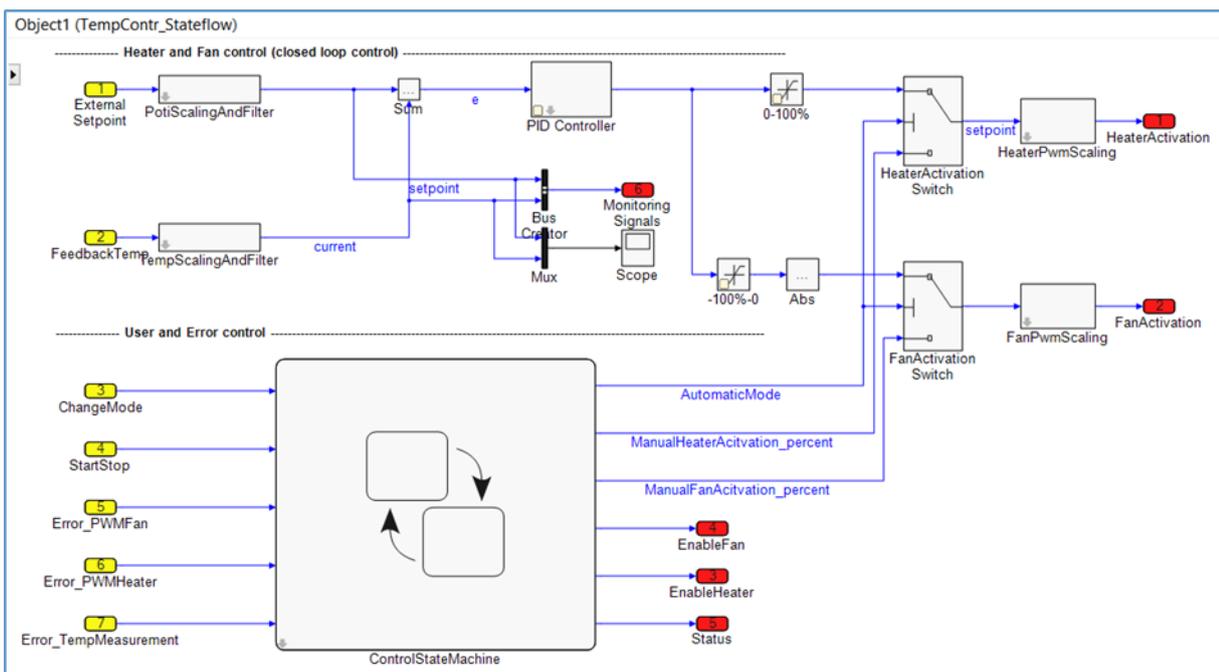
Damit eine TMI-Datei generiert wird und TwinCAT automatisch die TMI-Datei zum Zielsystem kopieren kann, aktivieren Sie in den Einstellungen des TcCOM-Moduls die Option **Copy TMI to Target**.

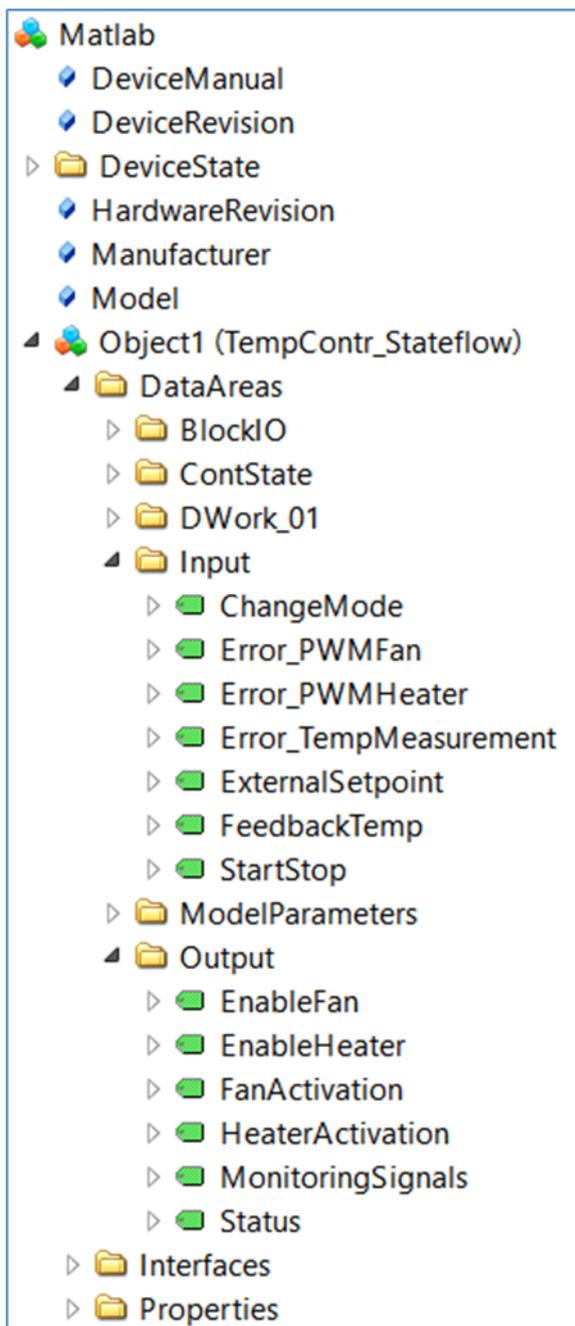


Object	Context	Parameter (Init)	Parameter (Online)	Data Area	Interfaces	Block Diagram
Object Id:		<input type="text" value="0x01010010"/>			<input checked="" type="checkbox"/> Copy TMI to Target	
Object Name:		<input type="text" value="Object1 (TempContr)"/>			<input type="checkbox"/> Share TMC Description	
Type Name:		<input type="text" value="TempContr"/>				
GUID:		<input type="text" value="8F5FDCFF-EE4B-4EE5-80B1-25EB23BD1B45"/>				
Class Id:		<input type="text" value="8F5FDCFF-EE4B-4EE5-80B1-25EB23BD1B45"/>				
Class Factory:		<input type="text" value="TempContr"/>				
Parent Id:		<input type="text" value="0x00000000"/>				
Init Sequence:		<input type="text" value="PSO"/>				

Nach Aktivierung des Projekts enthält das Bootverzeichnis des Zielsystems die TMI-Datei für das TcCOM-Modul. Diese können Sie dann mithilfe des OPC-UA-Konfigurators in den OPC-UA-Namensraum importieren.

Der konfigurierte OPC-UA-Namensraum enthält dann eine Repräsentation des Matlab/Simulink-Blockdiagramms:





Gefilterter oder ungefilterter Modus

Das Produkt TE1400 Target for Simulink bietet diverse Einstellmöglichkeiten zur Selektion von Variablen, die über OPC UA freigegeben werden sollen. Diese Einstellungen werden in der entsprechenden [TE1400 Produktdokumentation](#) beschrieben.

4.1.6.4 I/O Task

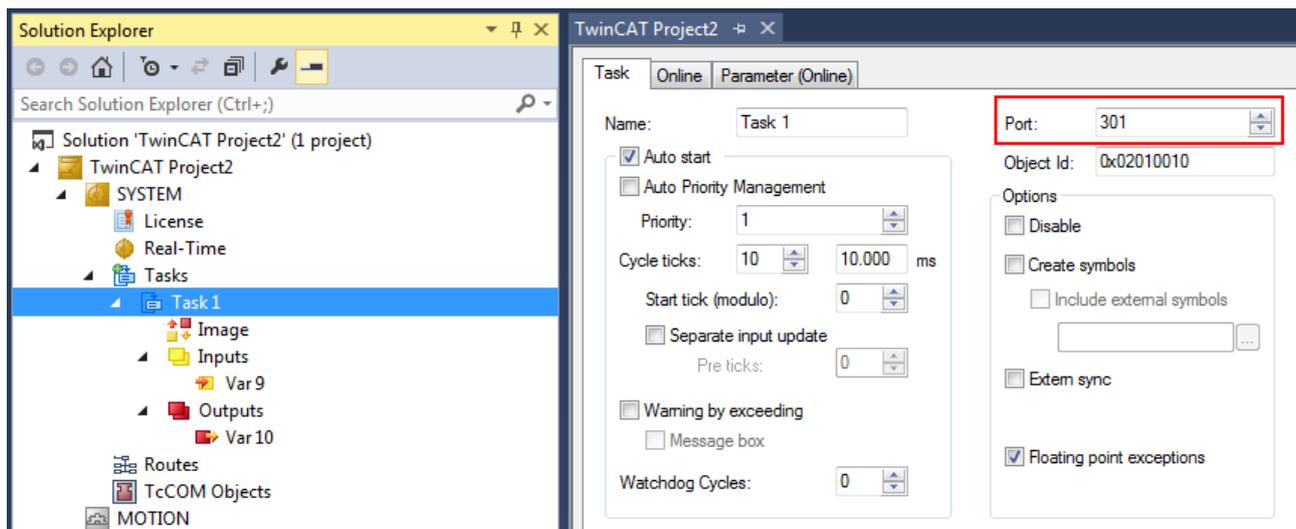
In diesem Abschnitt wird beschrieben, wie Sie IO-Taskvariablen über die Datenzugriffsfunktionen des TwinCAT OPC UA Servers zugänglich machen.

Dieser Abschnitt beinhaltet folgende Themen:

- [Allgemeine Informationen \[► 75\]](#)
- [Schritt 1: TwinCAT-I/O-Task konfigurieren, um das Symbolhandling zu aktivieren. \[► 75\]](#)
- [Schritt 2: I/O-Task dem OPC-UA-Namensraum hinzufügen \[► 76\]](#)

Allgemeine Informationen

Sie können die Variablen eines Tasks mit Prozessabbild über OPC UA veröffentlichen.



Die Art und Weise, wie die Variablen angezeigt und wie mit ihnen kommuniziert wird, wird anhand verschiedener Parameter festgelegt. Sie können die Parameter mithilfe des OPC-UA-Konfigurators oder direkt in der Konfigurationsdatei *ServerConfig.xml* festlegen.

Die folgende Tabelle gibt einen Überblick über alle Parameter, die beim Zugriff auf das I/O-Task von Bedeutung sind:

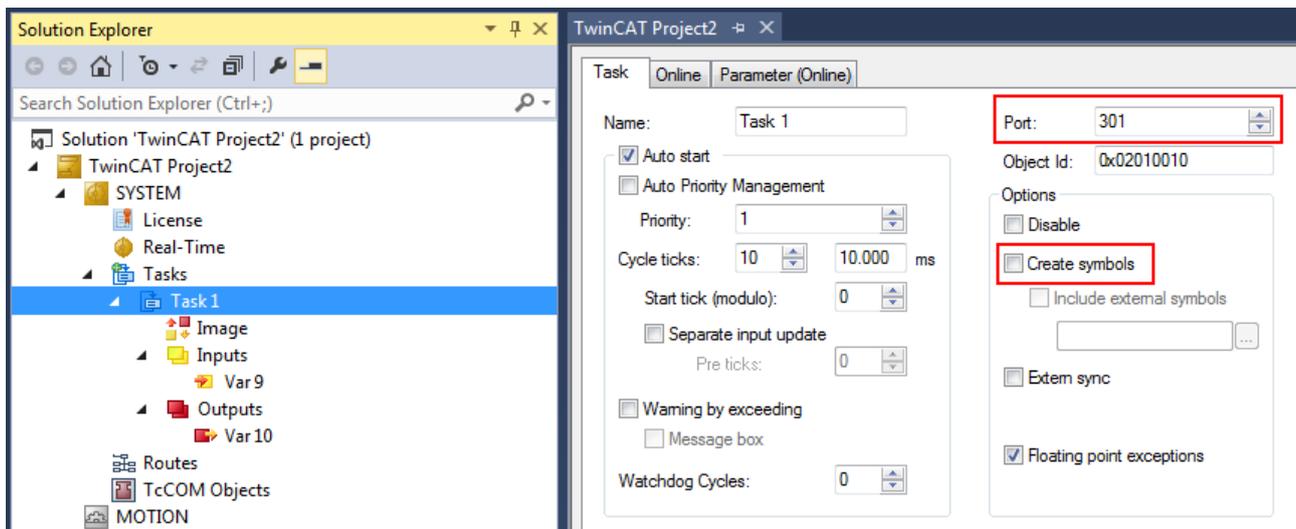
Parameter	Beschreibung	Mögliche Werte
ADS Port	Definiert den ADS-Port, unter dem das I/O-Task zugänglich ist. Der ADS Port kann in den Eigenschaften des I/O-Tasks ausgelesen werden.	301 302 ...
AutoCfg	Definiert zunächst den Typ der für die Kommunikation verwendeten Ziel-Laufzeit, z. B. SPS, C++, I/O. Anschließend kann eine weitergehende Unterscheidung innerhalb dieser Kategorien stattfinden. Jede AutoCfg-Option steht sowohl als ungefilterte, als auch gefilterte Option zur Verfügung. Gefiltert bedeutet, dass die Nutzer bestimmen können, welche I/O-Variablen dem OPC UA über Kommentare öffentlich zugänglich gemacht werden. Bei Verwendung einer ungefilterten Option wird jede Task-Variable an OPC UA veröffentlicht.	107 I/O TwinCAT 2/3 108 I/O TwinCAT 2/3 gefiltert
AutoCfgSymFile	Bestimmt den Pfad der CurrentConfig.xml-Datei, die sich normalerweise im Ordner <i>C:\TwinCAT\3.1\Boot\</i> befindet.	Pfad von CurrentConfig.xml
Disabled	Deaktiviert die I/O Task im UA-Namensraum, woraufhin der entsprechende Knoten nicht angezeigt wird. Wir empfehlen die Aktivierung dieses Parameters wenn bestimmte Laufzeiten zum Zeitpunkt der Projektplanung noch nicht verfügbar sind, weil z. B. die entsprechenden Geräte noch nicht ans Netzwerk angeschlossen sind.	0 (deaktiviert - standardmäßig) 1 (aktiviert)

Die folgenden Schritte beschreiben, wie Sie Variablen von einem I/O-Task in den UA-Namensraum importieren können. Hierbei wird vorausgesetzt, dass sich der OPC UA Server und die Laufzeit sich auf dem gleichen Computer befinden.

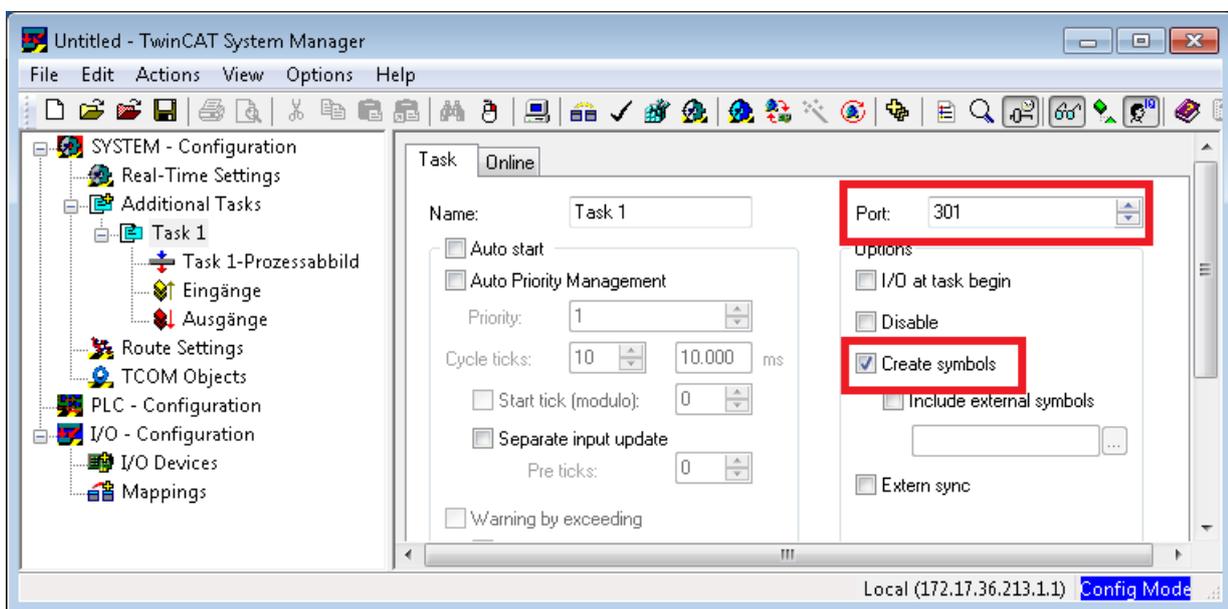
Schritt 1: TwinCAT-I/O-Task konfigurieren, um das Symbolhandling zu aktivieren.

Öffnen Sie die I/O-Task-Einstellungen und aktivieren Sie die Option **Symbole erzeugen**. Notieren Sie an dieser Stelle auch den ADS-Port des I/O-Tasks (im Beispiel ist das 301).

TwinCAT 3:



TwinCAT 2:



Schritt 2: I/O-Task dem OPC-UA-Namensraum hinzufügen

Konfigurieren Sie den OPC UA Server so, dass er Zugriff auf das I/O-Task bietet. Fügen Sie mithilfe des OPC-UA-Konfigurators ein neues Gerät vom Typ „IO TwinCAT 2/3“ hinzu und konfigurieren Sie dessen Einstellungen für AdsNetId, AdsPort und SymbolFile (Pfad zur Datei *CurrentConfig.xml*).

Starten Sie, nachdem Sie die entsprechenden Eigenschaften eingestellt haben, den OPC UA Server neu, damit die Einstellungen aktiv werden.

4.1.7 Historical Access

In diesem Abschnitt werden die notwendigen Schritte zur Konfiguration der Variablen im Namensraum des OPC UA Servers für Historical Access (HA) beschrieben.

Historical Access ist eine Funktion von OPC UA, bei der die Variablenwerte entweder dauerhaft auf einem Datenträger (Datei oder Datenbank) oder in der Geräte-RAM gespeichert werden, damit sie später vom Client abgerufen werden können. Die Art und Weise wie der OPC UA Server die Variablenwerte liest und speichert, kann konfiguriert werden.

Die Historical Access Konfiguration kann für Variablen aus beliebigen Laufzeitsystemen (SPS, C++, ...) durchgeführt werden. Als Voraussetzung muss die jeweilige Variable zunächst einmal für OPC UA freigegeben werden. Wie dies geschieht erfahren Sie im jeweiligen Dokumentartikel zum Thema "Data Access".

Voraussetzungen und Empfehlungen

Es gelten die folgenden Voraussetzungen zur Nutzung von Historical Access:

- Das Laufzeitsystem, dessen Symbole für Historical Access gespeichert werden sollen, muss für Data Access konfiguriert sein (ebenso müssen die jeweiligen Variablen freigegeben sein).
- Um eine SQL-Compact-Datenbank als Speichermedium nutzen zu können, muss auf dem Rechner, auf dem der OPC UA Server läuft, SQL Compact Runtime 3.5 SP2 installiert sein.
- SQL Compact Datenbanken werden auch unter Windows CE unterstützt.
- MS SQL Server Datenbanken werden unter Windows CE nicht unterstützt.
- Es werden die folgenden MS SQL Server Versionen unterstützt: 2017, 2019
- Es gelten die folgenden Empfehlungen bei Verwendung des jeweiligen Speichertyps:
 - Arbeitsspeicher: Anzahl der Einträge < 5000
 - Dateisystem: Anzahl der Einträge < 10000
 - Datenbank: Anzahl der Einträge >= 10000

Es werden die folgenden Speichertypen unterstützt:

Speichertyp	TF6100 Server Setup Version	Betriebssysteme	Beschreibung
Arbeitsspeicher	4.x und 5.x	Windows, Windows CE, TwinCAT/BSD	Speichert die Werte im Arbeitsspeicher des Geräts, auf dem der OPC UA Server läuft. Es sind keine weiteren Parameter erforderlich. Nach einem Neustart des Geräts sind die gespeicherten Werte nicht mehr verfügbar. Das Speichermedium ist demzufolge nicht persistent. Es gelten die oben aufgeführten Anforderungen und Empfehlungen.
Dateisystem	4.x	Windows, Windows CE	Speichert die Werte in mehreren Dateien, deren Speicherort festgelegt werden kann. Jedem für dieses Speichermedium konfigurierten Symbol wird eine eigene Datei in diesem Verzeichnis zugeordnet. Darüber hinaus besteht eine Sicherungskopie der Datei für jedes Symbol, die dann erzeugt wird, sobald die Puffergröße erreicht ist. Der Inhalt der Datendatei wird dann als Sicherungskopie gespeichert und es wird eine neue Datei erzeugt. Es gelten die oben aufgeführten Anforderungen und Empfehlungen.
SQL Compact Datenbank	4.x	Windows, Windows CE	Speichert die Werte in eine SQL Compact Database, dessen Speicherort festgelegt werden kann. Es gelten die oben aufgeführten Anforderungen und Empfehlungen.
MS SQL Server Datenbank	4.x	Windows	Speichert die Werte in einer SQL Server Database, die mit verschiedenen Parametern referenziert wird. Es gelten die oben aufgeführten Anforderungen und Empfehlungen.
TwinCAT Analytics	5.x	Windows, TwinCAT/BSD	Speichert die Werte in einem Dateiformat, welches dem TwinCAT Analytics Speicherformat entspricht. Die Daten können somit mit der TwinCAT Analytics Toolchain weiter verwendet werden. Dieses Format ersetzt seit TF6100 Version 5.x das alte Datei-basierte Speicherformat (s.o.).

Die einzelnen Speichertypen werden als sogenannte HistoryAdapter im Server konfiguriert. Dies geschieht über den TwinCAT OPC UA Server Konfigurator. Jeder HistoryAdapter (außer „Volatile“ -> Arbeitsspeicher) kann hierbei mehrfach verwendet werden, z. B. wenn die historischen Werte für einzelne Variablen in unterschiedlichen Datenspeichern abgelegt werden sollen. Im Bereich HistoryNodes werden die einzelnen Nodes konfiguriert, einem Datenspeicher zugewiesen und mit einer SamplingRate sowie einer maximalen Größe für den zu verwendenden Ringbuffer im Datenspeicher versehen.

Das Attribut `HistoryWriteable="true"` bewirkt, dass der Datenspeicher für diese Node über einen Aufruf `HistoryUpdate()` mit Werten befüllt werden kann. Ein solcher Aufruf wird z. B. von dem TwinCAT OPC UA Client über den Funktionsbaustein `UA_HistoryUpdate` zur Verfügung gestellt. Wenn Sie einen Node mit diesem Attribut konfigurieren, dann „sampled“ der Server die Werte üblicherweise nicht selbst, sondern bekommt diese von anderen Clients gereicht. Die SamplingRate ist daher bei einer solchen Konfiguration üblicherweise gleich 0.

Voraussetzungen

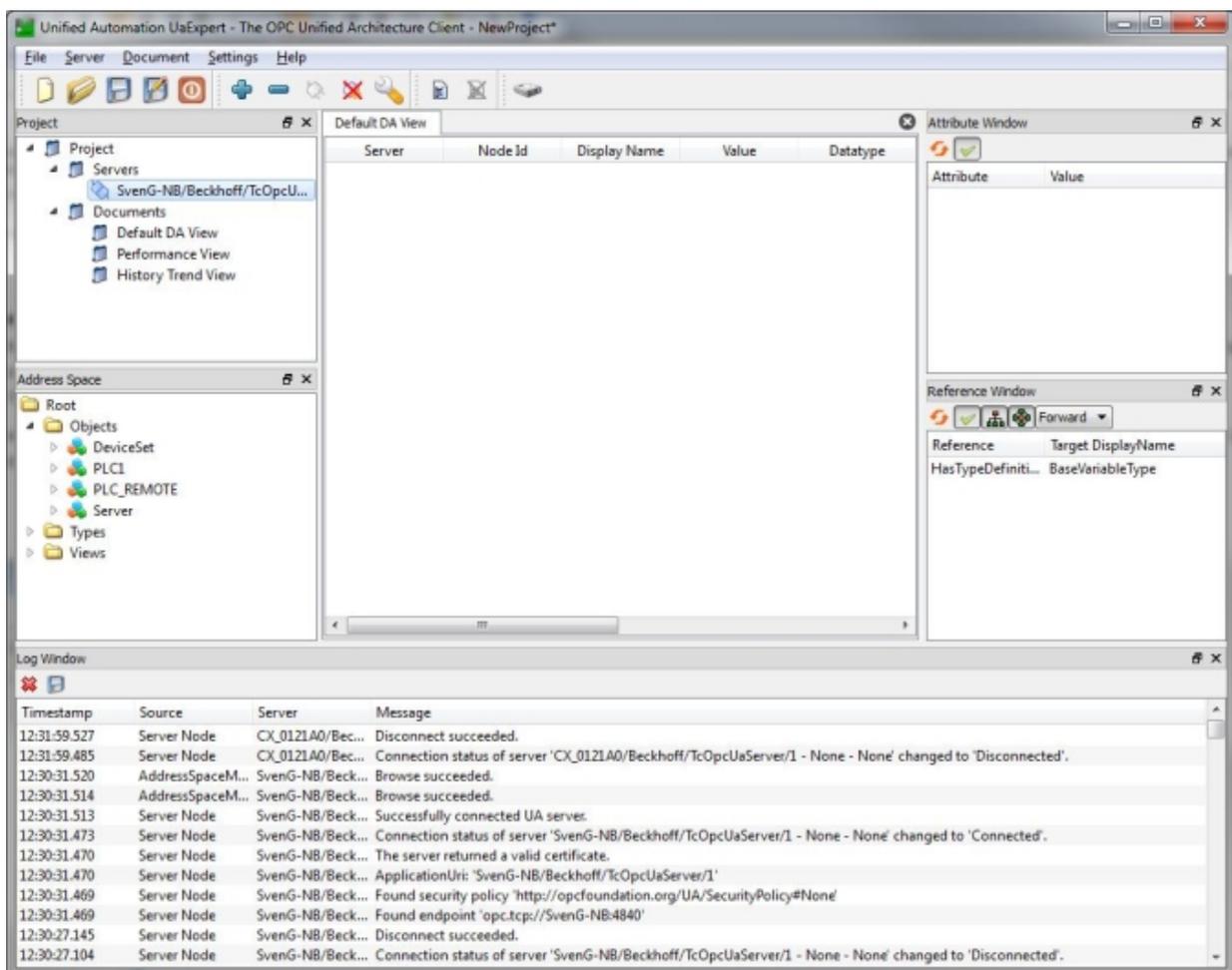
Produkte	Setup-Versionen	Zielplattform
TF6100	4.x.x	IPC oder CX (x86, x64, ARM)

4.1.7.1 Historische Daten anzeigen

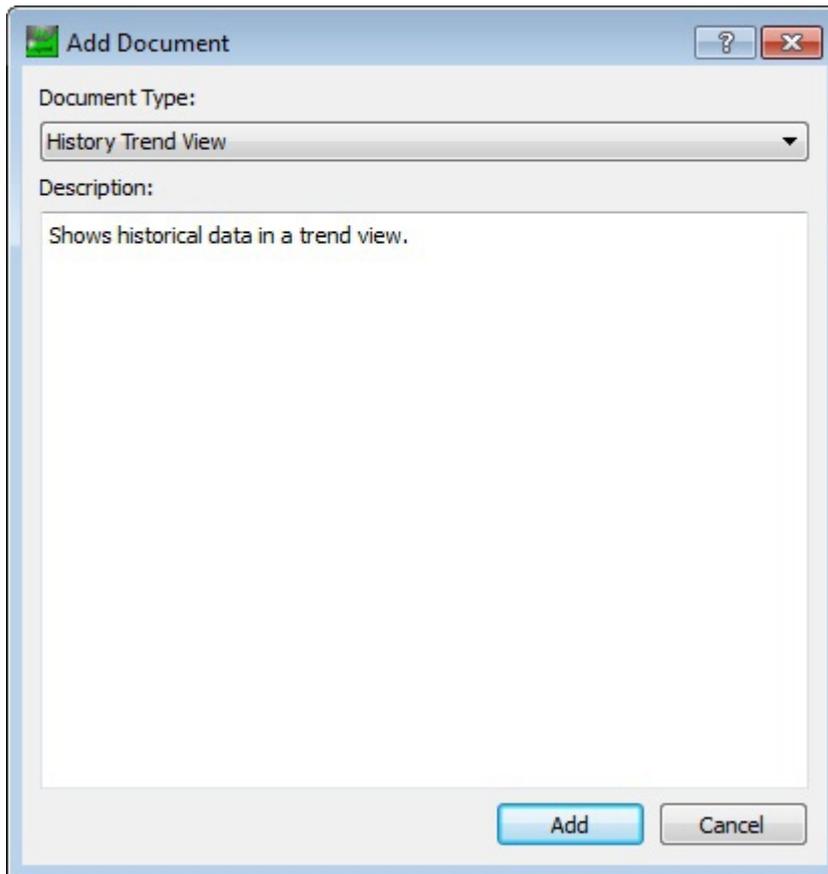
Historical-Access-Werte in einem OPC UA Client anzeigen

Die folgende Schritt-für-Schritt-Anleitung beschreibt, wie Sie die UA-Expert-Software konfigurieren, um auf historische Daten zuzugreifen.

1. Starten Sie die UA-Expert-Software und stellen Sie eine Verbindung mit dem OPC UA Server her.

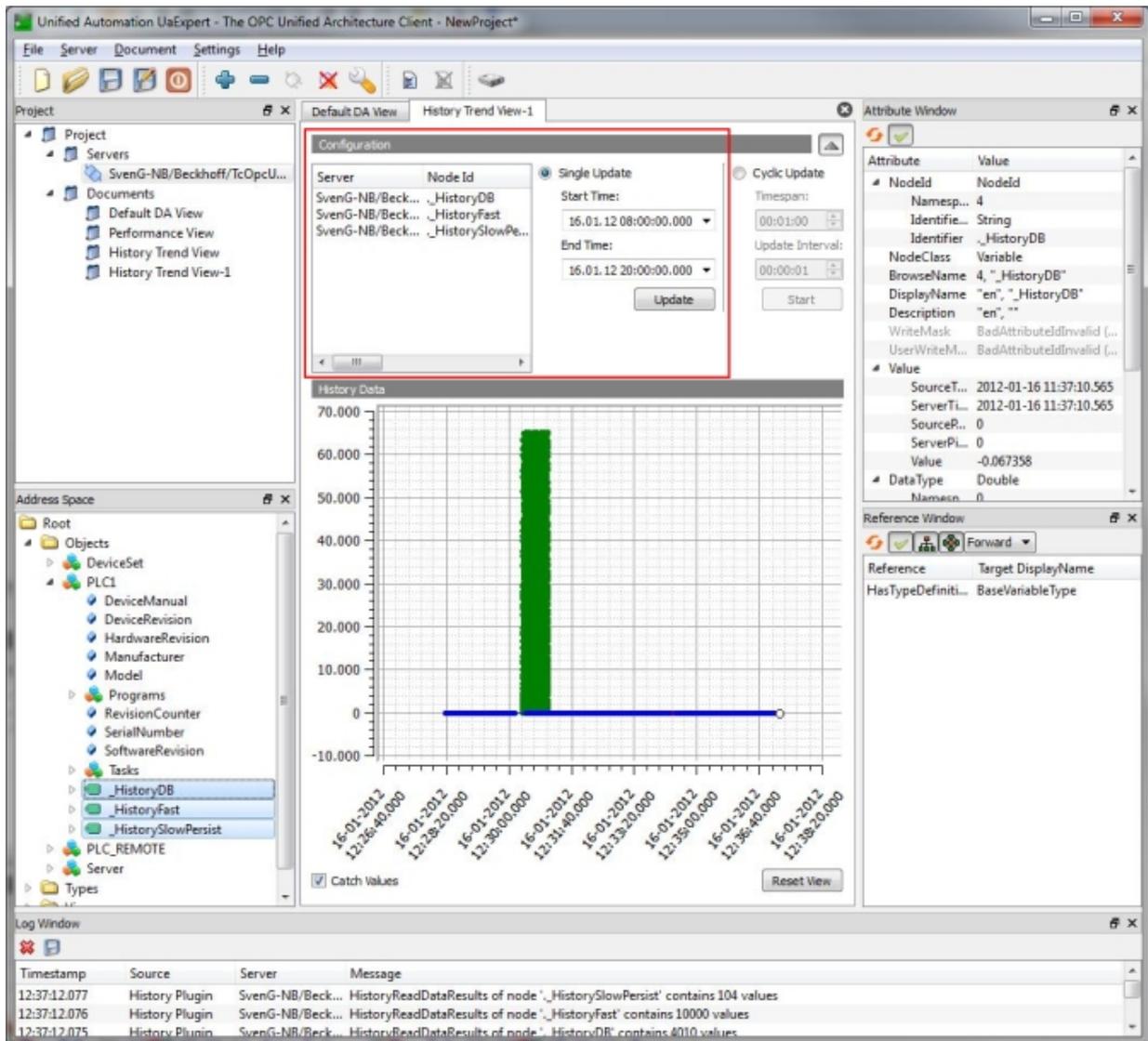


2. Fügen Sie eine neue **History Trend Ansicht** hinzu.



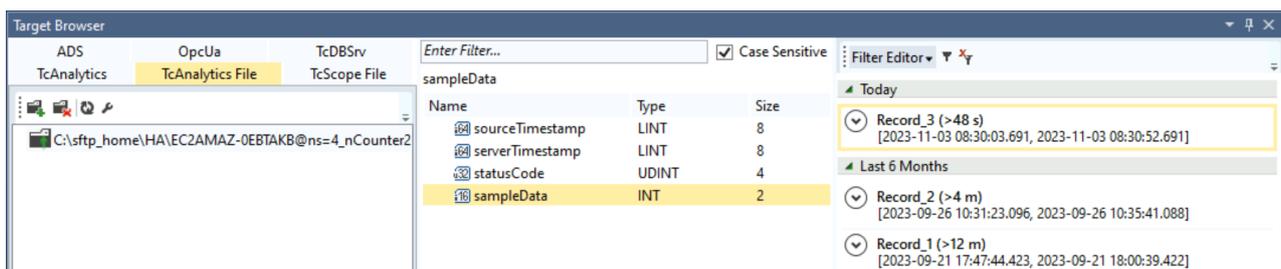
3. Durchsuchen Sie den PLC1-Namensraum und fügen Sie die SPS-Variablen `_HistoryDB`, `_HistoryDBcompact`, `_HistoryFast` und `_HistorySlowPersist` per Drag-and-drop hinzu.

⇒ Sie können nun den gewünschten Zeitraum, für den die Symbolwerte angezeigt werden sollen, mithilfe der Steuerelemente **Start Time** und **End Time** festlegen, oder falls erforderlich ein **Cyclic Update** für diese Variablen starten.



Historical-Access-Werte im TwinCAT Target Browser anzeigen

Wenn Sie in Ihrer Historical Access Konfiguration das TwinCAT Analytics Speicherformat verwenden, so können Sie die gespeicherten Werte auch über die TwinCAT Analytics Toolchain abrufen, zum Beispiel dem TwinCAT Target Browser. Fügen Sie hierzu das Verzeichnis, welches Sie in Ihrer Konfiguration zur Speicherung der historischen Werte festgelegt haben, in der Registerkarte **TcAnalytics File** vom TwinCAT Target Browser hinzu. Die gespeicherten Werte werden dann in der gewohnten TwinCAT Analytics Form als Records dargestellt und können weiterverarbeitet werden.



4.1.8 Alarms and Conditions

In diesem Abschnitt werden die notwendigen Schritte zur Konfiguration von OPC UA Alarms and Conditions (A&C) auf dem OPC UA Server beschrieben. Das Grundkonzept ist unabhängig von der TwinCAT-Laufzeit. Demzufolge sind die Konfigurationsschritte für SPS, C++, TcCOM-Laufzeit oder nur I/O-Task gleich.

OPC UA Alarms and Conditions (Teil 9 der OPC-UA-Spezifikation) beschreibt ein Modell für die Überwachung von Prozesswerten und das Ausgeben von Alarmen und Ereignissen bei Zustandsänderungen eines Laufzeit-Symbols.

Voraussetzungen

Die folgenden Voraussetzungen gelten für die Verwendung von OPC UA Alarms and Conditions:

- Das zu überwachende Laufzeitsymbol muss im Namensraum verfügbar sein.
- Der OPC UA Client muss Alarms and Conditions unterstützen. In diesem Abschnitt wird der UA Expert (von Unified Automation) als Referenz-UA-Client verwendet.

Allgemeine Informationen

Führen Sie die folgenden Schritte einmalig aus, um ein Symbol für Alarms and Conditions freizugeben:

- Schritt 1: [Laufzeitsymbol für Datenzugriff aktivieren](#) [► 82] (damit das Symbol generell via OPC UA zugänglich wird.)
- Schritt 2: [A&C für ein Symbol aktivieren](#) [► 82]
- Schritt 3: [Eigene Benutzerdaten mit einem Ereignis übermitteln](#) [► 83]
- Schritt 4: [Ereignis über die FireEvent-Methode auslösen](#) [► 85]
- Schritt 5: [Mehrsprachige Alarmtexte konfigurieren](#) [► 86]
- Schritt 6: [A&C bei einem Referenz-OPC-UA-Client anmelden](#) [► 86]

Diese Schritte werden nachfolgend ausführlicher erläutert. Am Ende des Abschnitts finden Sie Informationen zum Empfang von konfigurierten Alarmen über A&C mit dem UA-Expert-Referenz-Client.

Unterstützte Alarmtypen

Die Implementierung von OPC UA Alarms and Conditions unterstützt derzeit die folgenden Alarmtypen:

- LimitAlarmType: Verschiedene Grenzen für ein Symbol definieren. Wird eine Grenze erreicht, gibt der UA Server einen Alarm aus.
- OffNormalAlarmType: Einen Wert definieren, der „normal“ ist. Wenn der aktuelle Wert vom „normalen“ Wert abweicht, gibt der UA Server einen Alarm aus.

Schritt 1: Laufzeitsymbol für Datenzugriff aktivieren

Damit eine Variable für A&C konfiguriert werden kann, muss diese im OPC UA Server zur Verfügung stehen. Im Falle einer SPS-Variablen versehen Sie diese dazu mit einem Attribut (siehe [SPS](#) [► 49]).

Schritt 2: A&C für ein Symbol aktivieren

Mit dem OPC-UA-Server-Konfigurator können Sie ein Laufzeitsymbol für A&C konfigurieren. Der Konfigurator hat eine einfache grafische Bedienoberfläche, um die dahinter stehende XML-Datei zu bearbeiten. Der Konfigurator steht, je nach Setup-Version, in zwei Varianten zur Verfügung: Standalone und [integriert in das Visual Studio](#) [► 127].

Der folgende Programmausschnitt zeigt ein Beispiel dieser XML-Datei, um das allgemeine Verhalten und den Aufbau der A&C-Implementierung besser zu verstehen.

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

```

NodeId="s=MAIN.nCounter1" />
</Condition>
<Condition Name="Switch" Severity="500">
  <OffNormalAlarmType Normal="0" MessageNormal="100" MessageOffNormal="20" />
  <ItemToMonitor SamplingRate="100" NS="urn:[NodeName]:BeckhoffAutomation:Ua:PLC1"
NodeId="s=MAIN.bSwitch" />
</Condition>
<Condition Name="Struct" Severity="300">
  <LimitAlarmType LowLowLimit="-10" LowLimit="0" HighLimit="10" HighHighLimit="20"
MessageNormal="100" MessageLowLow="10" MessageLow="11" MessageHigh="12" MessageHighHigh="13"/>
  <ItemToMonitor SamplingRate="100" NS="urn:[NodeName]:BeckhoffAutomation:Ua:PLC1"
NodeId="s=MAIN.stStruct" />
</Condition>
</ConditionController>
<ConditionController Name="ConditionController2" >
  <Condition Name="Counter2" Severity="200">
    <LimitAlarmType LowLowLimit="-10" LowLimit="0" HighLimit="10" HighHighLimit="20"
MessageNormal="100" MessageLowLow="10" MessageLow="11" MessageHigh="12" MessageHighHigh="13"/>
    <ItemToMonitor SamplingRate="100" NS="urn:[NodeName]:BeckhoffAutomation:Ua:PLC1"
NodeId="s=MAIN.nCounter2" />
  </Condition>
</ConditionController>
</TcUaAcConfig>

```

Eine „Bedingung“ definiert das zu überwachende Laufzeitsymbol und die Alarmgrenzen und -texte. Jede Bedingung ist in einem sogenannten „ConditionController“ organisiert, dem Objekt, das die OPC UA Clients später abonnieren.

Bei der Erstellung einer Bedingung (Condition) müssen der NamespaceName (NS) und die NodeID spezifiziert werden, um auf den zu überwachenden UA-Knoten zu verweisen. Der Standalone-Konfigurator bietet einen einfachen Browsing-Mechanismus, um einen Knoten auszuwählen. Beim in das Visual Studio integrierten Konfigurator wird der Target Browser genutzt (Extension - OPC UA). In XML kann durch den Platzhalter [NodeName] in NamespaceName die XML-Datei einfach zwischen zwei verschiedenen Hardwaresystemen hin und her wechseln. NamespaceName beinhaltet immer den Hostnamen des IPC oder Embedded-PC, auf dem der OPC UA Server läuft. Wird [NodeName] gewählt, wird dieser Tag durch den Hostnamen des aktuellen IPCs oder Embedded-PCs ersetzt, auf dem der UA Server läuft.

Die SamplingRate bestimmt, wie oft der UA Server einen Wert vom Knoten abfragen soll, um festzustellen, ob eine der Alarmgrenzen erreicht wurde.

Die Alarmtexte werden über eine ID identifiziert. Die ID identifiziert eindeutig einen Alarmtext aus der Ressourcendatei (siehe [Schritt 5: Mehrsprachige Alarmtexte konfigurieren](#) [► 86]).

Schritt 3: Eigene Benutzerdaten mit einem Alarm übermitteln

Alarme können Felder mit eigenen Benutzerdaten enthalten, die die mit dem Alarm ausgegebenen Daten ergänzen. Diese Benutzerdatenfelder können Sie in der Laufzeitanwendung erzeugen und ausfüllen. Erstellen Sie dazu ein STRUCT und nennen Sie dessen erstes Element „value“. Bei einem Alarm werden dann alle folgenden Elemente in einem zusätzlichen Benutzerdatenfeld gesendet.

Beispiel SPS-Anwendung:

```

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

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

```

Die Instanz von ST_CustomStruct wird über den regulären Mechanismus, z. B. in TwinCAT 3, für den Datenzugriff aktiviert. Das STRUCT muss dazu als [StructuredType](#) [► 57] aktiviert sein.

```

PROGRAM MAIN
VAR
  {attribute 'OPC.UA.DA' := '1'}

```

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

Bei der Anmeldung bei einem ConditionController müssen die OPC UA Clients besondere AlarmConditionTypes, sogenannte „BkUaLimitAlarmType“ und „BkUaOffNormalAlarmType“, abonnieren, damit sie die besonderen Benutzerdatenfelder beim Eingang eines Alarms empfangen können.

- AlarmConditionType
 - ShelvingState
 - ActiveState
 - EnabledState
 - InputNode
 - MaxTimeShelved
 - SuppressedOrShelved
 - SuppressedState
 - LimitAlarmType
 - HighHighLimit
 - HighLimit
 - LowLimit
 - LowLowLimit
 - BkUaLimitAlarmType
 - DiscreteAlarmType
 - OffNormalAlarmType
 - NormalState
 - BkUaOffNormalAlarmType

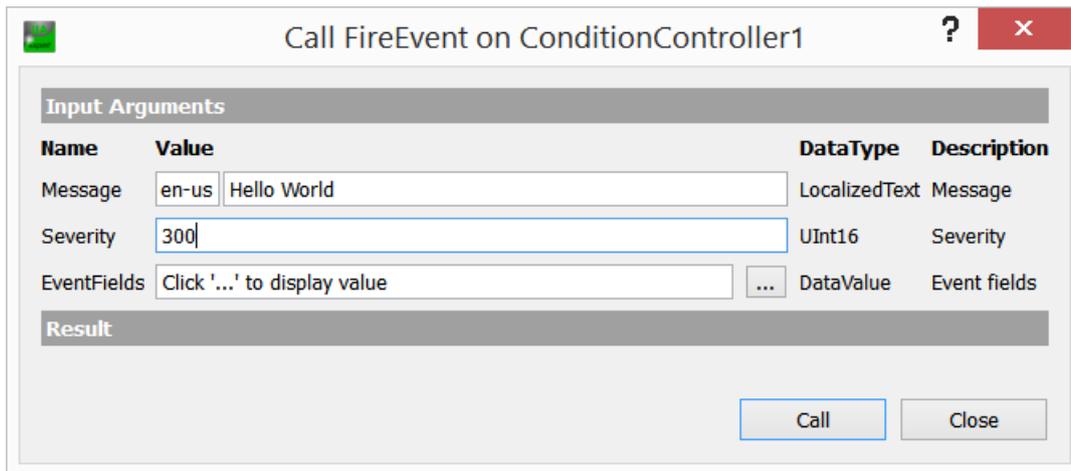
Der OPC UA Client empfängt dann die Benutzerdaten in den Feldern BkUaEventData und BkUaEventValue des eingehenden Alarms. Im obigen Beispiel sind das der Wert der SPS-Variablen sowie die Benutzerdaten, die mit der SPS-Struktur ST_SomeStruct dargestellt werden.

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

Schritt 4: Ereignis über die FireEvent-Methode auslösen

Jeder ConditionController umfasst eine FireEvent-Methode, mit der die OPC UA Clients ein allgemeines Ereignis mit benutzerdefinierten EventFields auslösen können.

- 📁 A&C
 - 📁 ConditionController1
 - ▶ Counter1
 - ▶ Counter2
 - ▶ CustomStruct
 - ▶ FireEvent
 - ▶ nCounter1
 - ▶ nCounter2
 - ▶ stCustomStruct



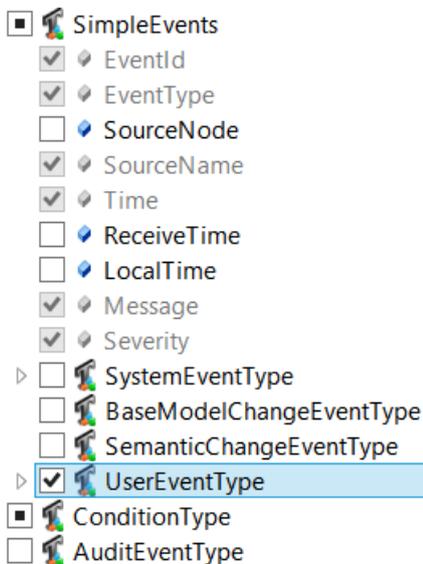
Wenn die Methode ausgeführt wird, dann wird ein Ereignis auf dem OPC UA Server ausgegeben. Andere OPC UA Clients können diese Ereignisse empfangen, wenn sie den entsprechenden ConditionController abonnieren.

Events | Alarms | Event History

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

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

Die benutzerdefinierten EventFields werden als „UserEventData“ an das Ereignis angehängt. Diese Daten können von OPC UA Clients empfangen werden, die beim SimpleEventType „UserEventType“ angemeldet sind.



Schritt 5: Mehrsprachige Alarmtexte konfigurieren

Die A&C-Implementierung im OPC UA Server unterstützt die Verwendung von mehrsprachigen Alarmtexten. Abhängig von der Sprache, mit der sich der UA Client mit dem Server verbindet, wird ein entsprechender Alarmtext verwendet.

Alarmtexte werden in XML-Dateien konfiguriert, für jede Sprache gibt es eine eigene Datei. Diese Dateien befinden sich im Ordner res (Ressource) des OPC UA Servers. Mit dem Konfigurator können Sie Alarmtexte einfach hinzufügen oder entfernen, ohne dass Sie die XML-Dateien direkt bearbeiten müssen. Jeder Alarmtext hat eine eigene ID, die in der Datei nur einmal vorkommt.

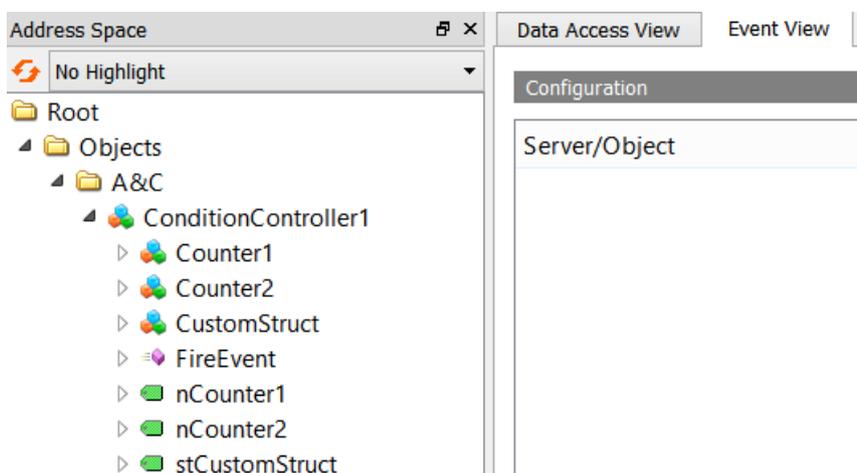
Beispiel:

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

Schritt 6: A&C bei einem Referenz-OPC-UA-Client anmelden

Ein OPC UA Client muss sich bei einem ConditionController anmelden, damit er Ereignisse für Bedingungen empfangen kann, die für diesen besonderen ConditionController konfiguriert sind. Der UA Expert stellt Funktionen zur Verfügung, um UA-Ereignisse zu abonnieren und zu empfangen.

Nachdem Sie den UA Expert gestartet und eine Verbindung zum OPC UA Server hergestellt haben, fügen Sie Ihrem Arbeitsbereich eine neue Dokumentenansicht Event View hinzu.



Sie können einen ConditionController durch Ziehen des entsprechenden Objekts in den Event View abonnieren. Die Ereignisse für diesen ConditionController werden im Event Window angezeigt.

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

Möglicherweise müssen spezielle Alarm- und/oder Ereignistypen abonniert werden, damit Sie alle Felder eines eingehenden Ereignisses oder Alarms empfangen können.

4.1.9 Method Call

RPC-Methoden können über den TwinCAT OPC UA Server in der [SPS \[►_90\]](#) oder in [C++ \[►_92\]](#) aufgerufen werden. Das Handling eines RPC-Methodenaufrufs über OPC UA wird auf OPC UA-Ebene für beide Optionen in gleicher Weise durchgeführt:

- Wenn eine RPC-Methode erfolgreich ausgeführt wurde, gibt der Server als Rückmeldung für den OPC UA-Methodenaufruf den Statuscode „Good“ zurück.
- Wenn die RPC-Methode nicht aufgerufen wurde, gibt der Server eine Fehlermeldung im Format „Bad_XYZ“ zurück, abhängig vom aufgetretenen Fehler.
- Wenn die RPC-Methode erfolgreich aufgerufen wurde, aber die Antwort nicht im Server gelesen werden konnte, wird der Statuscode „OpcUa_GoodPostActionFailed“ vom Server zurückgegeben.

4.1.9.1 Job-Methoden

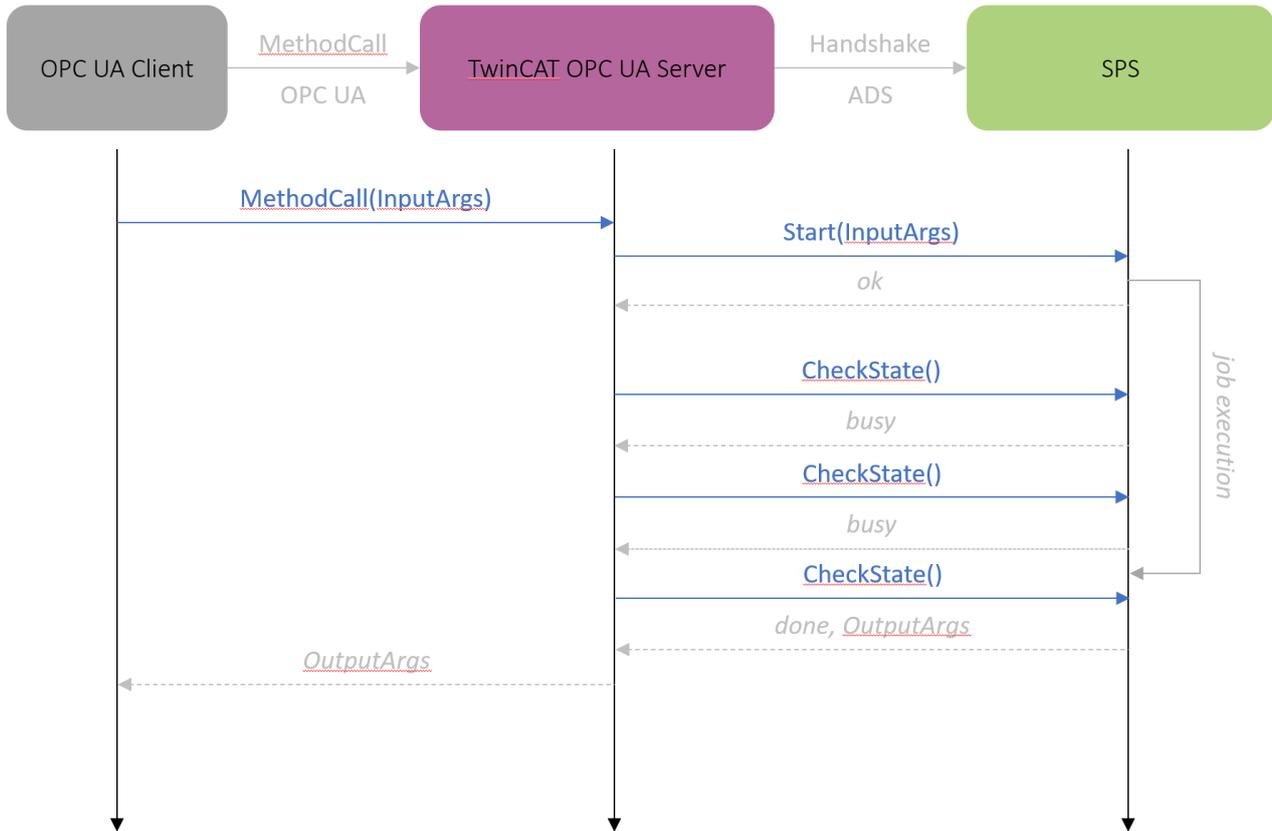
Das Konzept der Job-Methoden hat im Vergleich zu den regulären Methodenaufrufen einen grundlegenden Unterschied: die OPC UA Methoden werden nicht mehr 1:1 auf eine SPS-Methode abgebildet, sondern anstelle dessen auf einen Funktionsbaustein mit einer bestimmten Signatur. Hierdurch können auch Methodenaufrufe realisiert werden, welche aus Sicht einer SPS-Anwendung länger als einen Zyklus dauern.

Der SPS-seitige Aufbau einer solchen Job-Methode ist hierbei wie folgt definiert. Es existiert ein Funktionsbaustein, welcher über ein SPS-Attribut als Job-Methode definiert wird. Der Funktionsbaustein beinhaltet dann verschiedene SPS-Methoden, auf welche vom TwinCAT OPC UA Server in Form eines Handshake-Mechanismus zugegriffen wird, um diese als OPC UA Methode bereitstellen zu können.

Methode	Beschreibung
Start	<p>Wird vom Server aufgerufen, sobald ein OPC UA Client die OPC UA Methode aufruft. Beinhaltet die Eingabeparameter des OPC UA Methodenaufrufs als VAR_INPUT.</p> <p>Über den HRESULT Rückgabewert der Methode kann direkt ein OPC UA Status Code in dessen Dezimalrepräsentation zurückgegeben werden, z.B. „0“ für den Status Code „Good“. Als Wert wird hierbei der entsprechend in der OPC UA Spezifikation definierte numerische Wert eines Status Codes verwendet. Eine Definition aller verfügbaren Status Codes kann hier eingesehen werden:</p> <p>http://www.opcfoundation.org/UA/schemas/StatusCode.csv</p> <p>Typischerweise returniert diese Methode den Wert „0“ (Good). Der SPS-Entwickler kann sich jedoch auch dazu entscheiden, dass zum Beispiel die Eingabeparameter validiert werden sollen. Im Fehlerfall könnte man den OPC UA Methodenaufruf dann fehlschlagen lassen, z.B. mit dem Rückgabewert „2158690304“ (BadInvalidArguments).</p>
CheckState	<p>Wird zyklisch vom Server aufgerufen, um zu überprüfen, ob der Job noch in Bearbeitung ist oder nicht. Solange der Job noch in Bearbeitung ist, gibt diese Methode den Wert „Busy“ zurück, andernfalls „Done“.</p> <p>Ausgabeparameter für die OPC UA Methode werden hier als VAR_OUTPUT deklariert.</p> <p>Über den HRESULT Rückgabewert der Methode kann direkt ein OPC UA Status Code in dessen Dezimalrepräsentation zurückgegeben werden, z.B. „0“ für den Status Code „Good“. Als Wert wird hierbei der entsprechend in der OPC UA Spezifikation definierte numerische Wert eines Status Codes verwendet. Eine Definition aller verfügbaren Status Codes kann hier eingesehen werden:</p> <p>http://www.opcfoundation.org/UA/schemas/StatusCode.csv</p> <p>Bei einer erfolgreichen Abarbeitung des Jobs returniert diese Methode typischerweise den Wert „0“ (Status Code „Good“). Der SPS-Entwickler kann sich jedoch auch dazu entscheiden, dass im Falle eines Fehlers der OPC UA Methodenaufruf fehlschlagen soll, z.B. mit dem Rückgabewert „2151415808“ (BadOutOfRange) oder dem allgemeineren „2147483648“ (Bad).</p>
Abort	<p>Diese Methode wird vom Server aufgerufen, falls ein Job abgebrochen werden muss, zum Beispiel, wenn der Server heruntergefahren wird oder neu startet. Der SPS-Entwickler hat dann die Möglichkeit seinen SPS-Code entsprechend aufzuräumen.</p>

Workflow

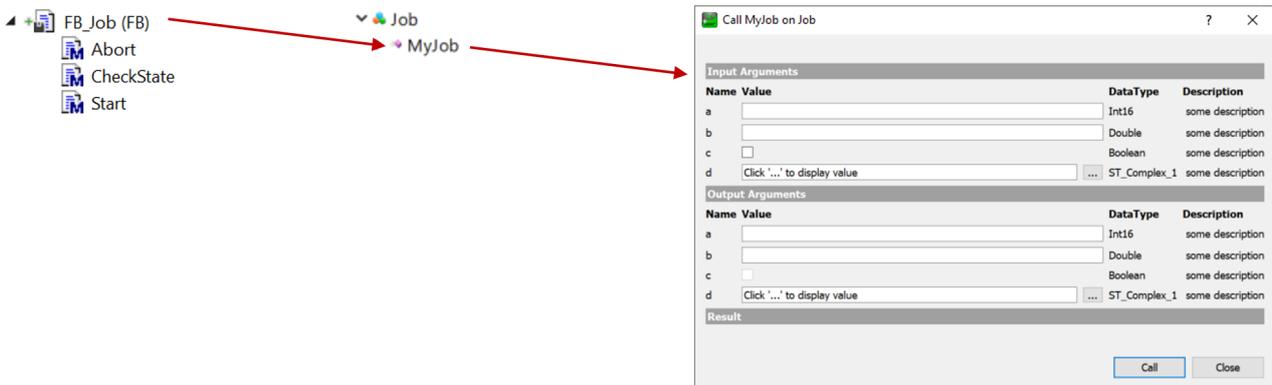
Der Handshake-Mechanismus zwischen Server und SPS kann vereinfacht wie folgt dargestellt werden.



Beispiel

Das folgende Beispiel ist in ausführbarer Form auch in den [Samples \[▶ 255\]](#) enthalten. Dieser Teil der Doku soll noch einmal die grundlegenden Zusammenhänge zur Funktionsweise erläutern. Das Beispiel „simuliert“ einen Job-Methodenaufruf, dessen Abarbeitung in der SPS circa vier Sekunden dauert. Dies wurde mit Hilfe eines Timers realisiert, welcher im Funktionsbaustein-Teil des Jobs deklariert und verwendet wurde. Über die State Machine des Funktionsbausteins wird hierbei der Abschluss des Jobs verzögert (die Handshake-Methode CheckState returniert „busy“) und erst nach Ablauf des Timers wird der Job beendet (die Handshake-Methode CheckState returniert „done“).

In diesem Beispiel wurde für die OPC UA Methode mit dem Namen „MyJob“ ein Funktionsbaustein namens FB_Job erzeugt, welcher die oben genannte, benötigte Signatur aufweist.



Die Funktionsbaustein-Deklaration enthält das Attribut OPC.UA.DA.JobMethod und als dessen Wert den zu verwendenden Namen der OPC UA Methode.

```

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

Die drei Methoden Abort, CheckState und Start enthalten in ihrer Deklaration das Attribut TcRpcEnable (damit die Methoden auch über ADS aufgerufen werden können). Beispiel:

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

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

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

Die Eingabeparameter der OPC UA Methode werden in der SPS-Methode Start() als VAR_INPUT deklariert. Kommentare hinter den Variablen werden als Description des jeweiligen OPC UA Eingabeparameters verwendet. Beispiel:

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

Die Ausgabeparameter der OPC UA Methode werden in der SPS-Methode CheckState() als VAR_OUTPUT deklariert. Kommentare hinter den Variablen werden als Description des jeweiligen OPC UA Ausgabeparameters verwendet. Beispiel:

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

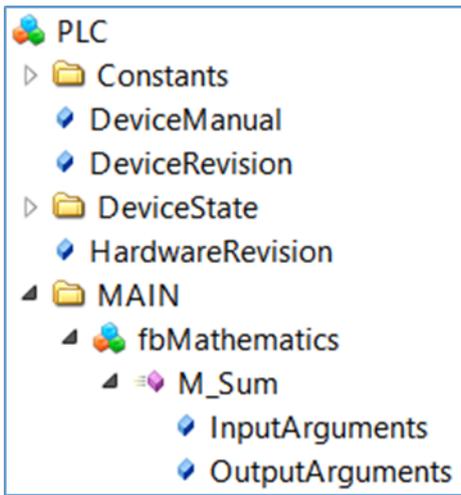
(In diesem Beispiel werden zu Zwecken der Veranschaulichung die Werte der Eingabeparameter 1:1 auf die Ausgabeparameter gelegt. Daher sind die Eingabe- und Ausgabeparameter identisch.)

4.1.9.2 SPS

Methodenaufrufe sind ein grundlegender Teil der OPC-UA-Spezifikation. Mit der Einführung dieser Funktionalitäten in die SPS-Welt bietet TwinCAT 3 die Möglichkeit zur effizienten Ausführung von RPC-Aufrufen in der IEC61131-Welt und verringert somit die klassischen Handshake-Pattern bei der Kommunikation zwischen den Geräten. Der OPC UA Server importiert SPS-Methoden wie OPC-UA-Methoden über seinen TMC-Import.

● IEC61131-Methode

I Auch wenn die SPS-Methode eine normale Methode im UA-Namensraum zu sein scheint, ist sie immer noch eine IEC61131-Methode, die innerhalb des Echtzeit-Kontexts ausgeführt wird und somit unter den Kontext einer Echtzeit-Task fällt. Der SPS-Entwickler muss daher Vorsichtsmaßnahmen treffen, sodass die Ausführungszeit der Methode in die Task-Zykluszeit passt.



Methoden in IEC61131-3

Methoden in der IEC61131-Welt werden immer unterhalb eines Funktionsbausteins konfiguriert. Auf Hochsprachenebene kann der Funktionsbaustein als umgebende Klasse der Methode betrachtet werden. Die Methode selbst müssen Sie mit einem speziellen SPS-Attribut deklarieren, sodass das TwinCAT-System weiß, dass die Methode für einen remote Methodenaufruf aktiviert werden soll.

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

Gefilterter oder ungefilterter Modus

In Abhängigkeit davon, welcher Importmodus verwendet wird, müssen Sie den umgebenden Funktionsbaustein ebenfalls für den OPC-UA-Zugriff aktivieren. Dies kann durch Verwendung der normalen SPS-Attribute für den OPC-UA-Zugriff erfolgen.

Beachten Sie dabei, dass Sie nur dann die SPS-Attribute verwenden müssen, wenn der gefilterte Modus verwendet wird, um Symbole aus der SPS für den Zugriff über OPC UA freizugeben. Dies ist die Standardeinstellung des OPC UA Servers.

Beispiel:

Die Methode M_Sum befindet sich im Funktionsbaustein FB_Mathematics. Die Deklaration der Funktionsbausteininstanz verwendet das SPS-Attribut, das den Funktionsbaustein und damit die Methode ebenfalls für den OPC-UA-Zugriff aktiviert hat.

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

● **Pointervariablen als VAR_IN_OUT**

I Pointer-Variablen, die als VAR_IN_OUT definiert wurden, werden weder von der SPS noch vom TwinCAT OPC UA Server behandelt. Ein entsprechendes Trace-Ereignis wird in den Server-Trace geschrieben.

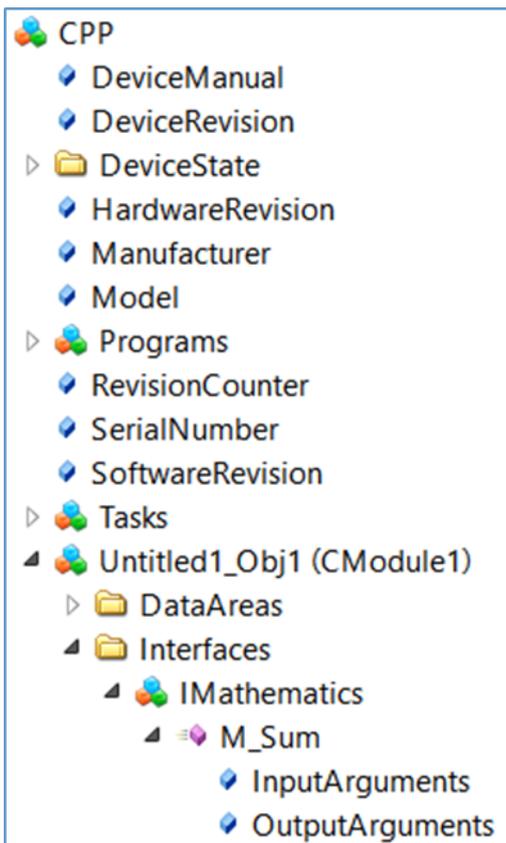
08:47:37.677Z|1|11A0* Fehler beim Importieren der Methode 'METH_PArray': VAR_IN_OUT Zeigervariablen sind nicht erlaubt!

4.1.9.3 C++

Methodenaufrufe sind ein grundlegender Teil der OPC-UA-Spezifikation. Mit der Einführung dieser Funktionalitäten in die SPS-Welt bietet TwinCAT 3 die Möglichkeit zur effizienten Ausführung von RPC-Aufrufen im C++-Echtzeitkontext und verringert somit die klassischen Handshake-Pattern bei der Kommunikation zwischen den Geräten. Der OPC UA Server importiert C++-Methoden wie OPC-UA-Methoden über dessen TMI-Import.

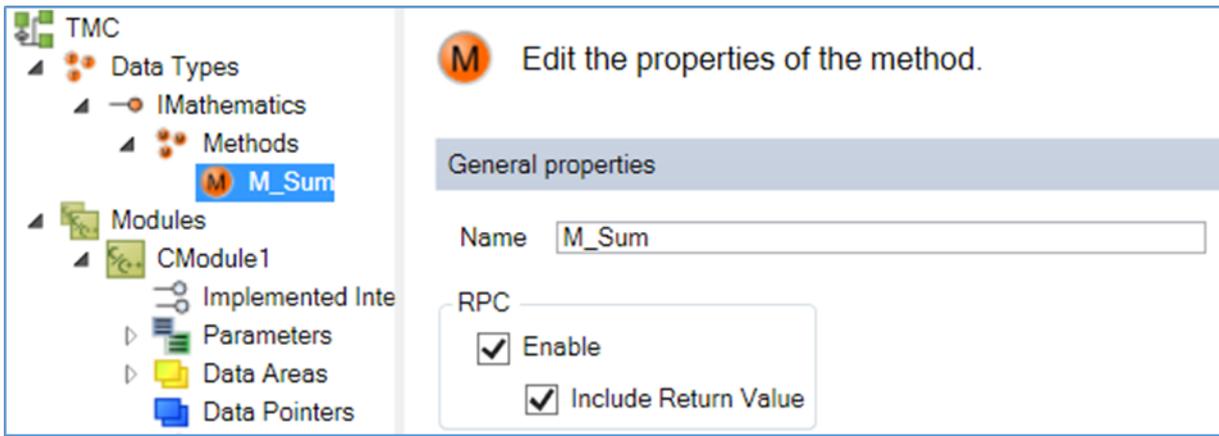
● Echtzeit-Methode

i Auch wenn die C++-Methode eine normale Methode im UA-Namensraum zu sein scheint, ist sie immer noch eine Echtzeit-Methode, die innerhalb des Echtzeit-Kontexts ausgeführt wird und somit unter den Kontext einer Echtzeit-Task fällt. Der TwinCAT-C++-Entwickler muss daher Vorsichtsmaßnahmen treffen, sodass die Ausführungszeit der Methode in die Task-Zykluszeit passt.

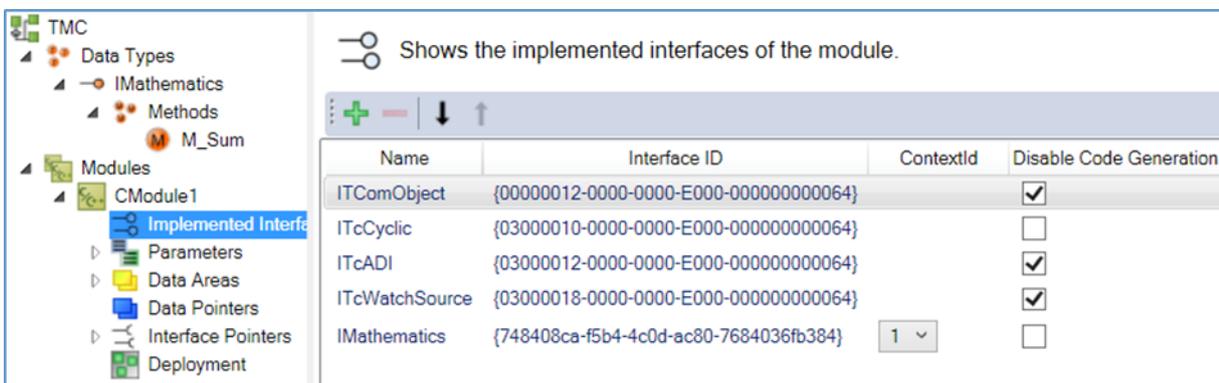


Methoden in TwinCAT 3 C++

TwinCAT-Module können Schnittstellen implementieren, die über vordefinierte Methoden verfügen (siehe TcCOM-Module). Die Methode selbst muss für RPC-Aufrufe während ihrer Definition aktiviert sein (siehe Dokumentation TwinCAT Module Class Wizard), sodass der OPC UA Server weiß, dass sie zur Ausführung bereitsteht.



Damit auch der Rückgabewert der Methode zur Verfügung steht, muss die entsprechende Option **Include Return Value** aktiviert sein. Beachten Sie, dass die Schnittstelle, unter der die Methode erstellt wurde, implementiert werden muss.



Gefilterter oder ungefilterter Modus

In Abhängigkeit davon, welcher Importmodus verwendet wird, müssen Sie die Methode für den Zugriff über OPC UA deklarieren. Dies kann durch Verwendung des TMC-Code-Editors und der üblichen OPC-UA-Attribute als optionale Eigenschaften erfolgen.

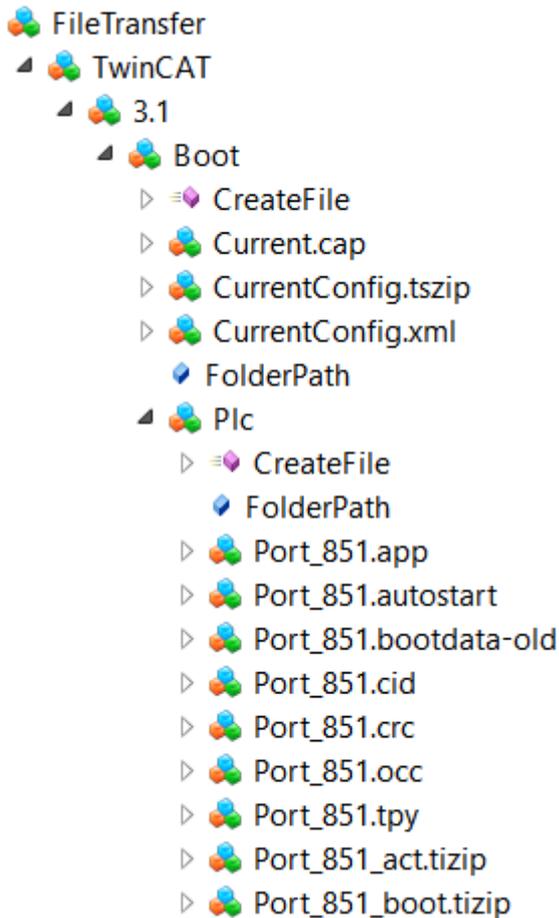


4.1.10 File Transfer

4.1.10.1 Zugriff auf Dateien und Ordner über OPC UA

Ab OPC-UA-Spezifikation Version 1.02 enthält OPC UA einen spezialisierten ObjectType zur Dateiübertragung, der in Anlage C der Spezifikation beschrieben ist. Dieser spezielle ObjectType namens „FileType“ beschreibt das Informationsmodell für die Datenübertragung. Dateien können in OPC UA mit ByteStrings als einfache Variablen modelliert werden. FileType ist eine Datei mit Methoden zum Zugriff auf die Datei. In der OPC-UA-Spezifikation erhalten Sie weitere Informationen zu FileType sowie Aufbau und Handhabung der zugrunde liegenden Methoden und Eigenschaften zum Zugriff auf eine Datei im OPC-UA-Namensraum.

Beckhoff hat einen generischen Weg implementiert, um Dateien und Ordner von einer lokalen Festplatte in den OPC-UA-Namensraum zu laden. Jede Datei wird durch einen FileType repräsentiert und ermöglicht Lese- und Schreibvorgänge für diese Datei. Zusätzlich enthält jeder Ordner eine Methode CreateFile(), um neue Dateien auf der Festplatte zu erstellen und einen eigenen FolderPath, um den tatsächlichen Pfad zum Ordner auf dem OPC UA Server festzulegen.



● FileTransfer im Device Manager OPC UA Server

i Diese Funktion hat nur der OPC UA Server des Beckhoff Device Managers (IPC Diagnose). Der TwinCAT OPC UA Server stellt ebenfalls einige Teile dieser Dateiübertragung bereit. Die allgemeine Funktion, die eine Offenlegung aller Dateien und Ordner ermöglicht, steht aber nur im OPC UA Server zur Verfügung, der zum Gerätemanager gehört, der automatisch auf jedem Beckhoff Industrie-PC oder Embedded-PC verfügbar ist. In der [Gerätemanagerdokumentation](#) erhalten Sie weitere Informationen.

Konfiguration

FileType-Objekte werden in einem separaten Namensraum mit der Bezeichnung „FileTransfer“ erstellt. Zur Konfiguration des Namensraums und zur Auswahl der über OPC UA verfügbaren Dateien und Ordner dient eine XML-Datei (*files.xml*), die in demselben Verzeichnis wie die ausführbare Datei des OPC UA Servers sein muss. Um die Konfiguration zu aktivieren, muss das System neu gestartet werden. Die XML-Datei enthält Informationen über den Ordnerpfad und eine Suchmaske, die definiert, welche Dateien im OPC-UA-Namensraum veröffentlicht werden:

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

Beispiel: Lesen einer Datei mit UA Expert

Der allgemeine Umgang mit Dateien ist in Anhang C der OPC-UA-Spezifikation beschrieben. Das Lesen einer Datei via UA kann in folgende Schritte unterteilt werden:

- Aufruf der Open-Methode einer Datei. Diese Methode gibt ein Dateihandle zurück, das für den späteren Zugriff gespeichert werden muss. Der Modus legt fest, ob die Datei gelesen oder in sie geschrieben wird (siehe [Dateimodi](#) [[▶ 95](#)]).
- Bestimmen der Größe der Datei mit der Eigenschaft „Size“. So kann die ganze Datei bei Aufruf der Read-Methode gelesen werden.
- Aufruf der Read-Methode. Dateihandle und Dateigröße als Eingaben einfügen. Zielordner wählen, in den der Dateinhalt NACH dem Aufruf der Methode zu speichern ist.
- Aufruf der Close-Methode zur Freigabe des Dateihandles.

Dateimodi

Die folgende Tabelle zeigt alle verfügbaren Dateimodi.

Feld	Bit	Beschreibung
Lesen	1	Die Datei wird zum Lesen geöffnet. Wenn dieses Bit nicht gesetzt ist, kann Read nicht ausgeführt werden.
Schreiben	4	Die Datei wird zum Schreiben geöffnet. Wenn dieses Bit nicht gesetzt ist, kann Write nicht ausgeführt werden.
EraseExisting	6	Der vorhandene Dateinhalt wird gelöscht und es wird eine leere Datei zur Verfügung gestellt.
Append	10	Die Datei wird geöffnet und ans Ende positioniert, sonst auf den Anfang. Diese Position kann mit SetPosition geändert werden.

Generelles Verhalten

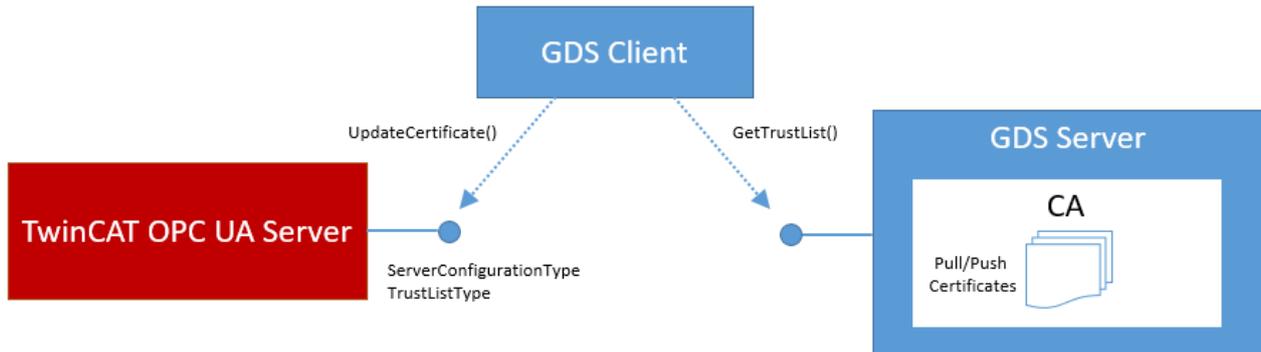
Die Anzahl an parallel geöffneten Dateien ist im Prinzip unlimitiert und unterliegt ggf. nur etwaigen Einschränkungen des zugrunde liegenden Betriebssystems. Dateien unterliegen jedoch einem 60-Sekunden Timeout. Nach diesem Timeout werden offene Dateien nicht sofort automatisch geschlossen. Anstelle dessen werden sie als „zu-schließend“ markiert. Wenn während dieser Zeit das entsprechende FileHandle für eine Read/Write Operation verwendet wird, so wird der Timeout zurückgesetzt und das FileHandle bleibt gültig. Wenn während dieser Zeit eine Open Operation auf derselben Datei durchgeführt wird, so wird das alte FileHandle freigegeben. Wenn ein OPC UA Client seine Verbindung zum Server trennt und noch Dateien geöffnet hat, so werden alle FileHandle die zu dieser Sitzung gehören automatisch geschlossen.

4.1.11 Global Discovery Service

Der TwinCAT OPC UA Server erlaubt die Integration der Serverapplikation in einen Global Discovery Service (GDS), damit dieser Zertifikate für den Server ausstellen und Zertifikatssperlisten (CRL) bereitstellen kann. Hierbei werden zwei Modelle unterstützt, Push und Pull, welche im Folgenden näher erläutert werden sollen:

Push

Bei diesem Modell beinhaltet der Server ein standardisiertes Interface, welches ein GDS Client verwenden kann, um sich im Auftrag des Servers mit einem Global Discovery Service zu verbinden, dort die Serverapplikation zu registrieren und ein Server-Zertifikat inklusive der aktuellen CRL anzufordern. Das Zertifikat wird dann auf dem Server aktiviert.



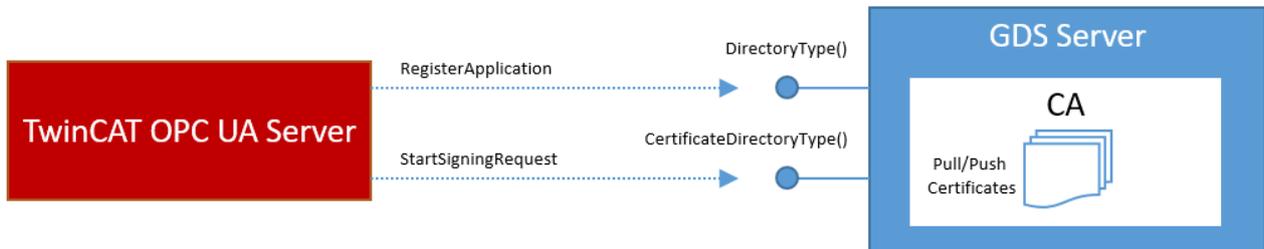
Als Voraussetzung zur Nutzung dieser Funktionalität muss sich der GDS Client am Server mit einem Benutzerkonto authentifizieren, welches über Administrator-Berechtigung [▶ 135] (IsRoot = true) verfügt.

i **GDS Client**
 Als GDS Client kommt zur Verwendung jeder Client in Frage, welcher das Push-Modell unterstützt. Diverse OPC UA Toolkithersteller bieten hier entsprechende Softwarepakete an. Alternativ stellt auch die OPC Foundation einen GDS Sample Client auf Github zur Verfügung.

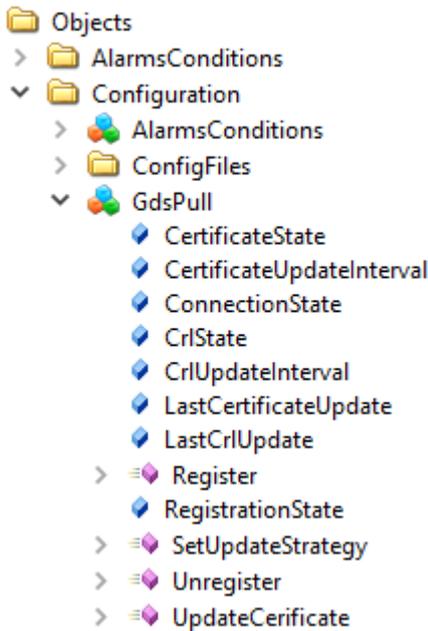
i **GDS Server**
 Als Global Discovery Service kommt zur Verwendung prinzipiell jeder GDS in Frage. Diverse OPC UA Toolkithersteller bieten hier entsprechende Softwarepakete an. Alternativ stellt auch die OPC Foundation einen GDS Sample Server auf Github zur Verfügung.

Pull

Bei diesem Modell stellt der Server eigenständig eine Verbindung zum Global Discovery Service her, registriert sich dort als Serverapplikation und bezieht ein passendes Serverzertifikat.



Der TwinCAT OPC UA Server bietet über seinen Konfigurations-Namensraum die Möglichkeit, die GDS Pull Funktionen zu aktivieren und zu konfigurieren.

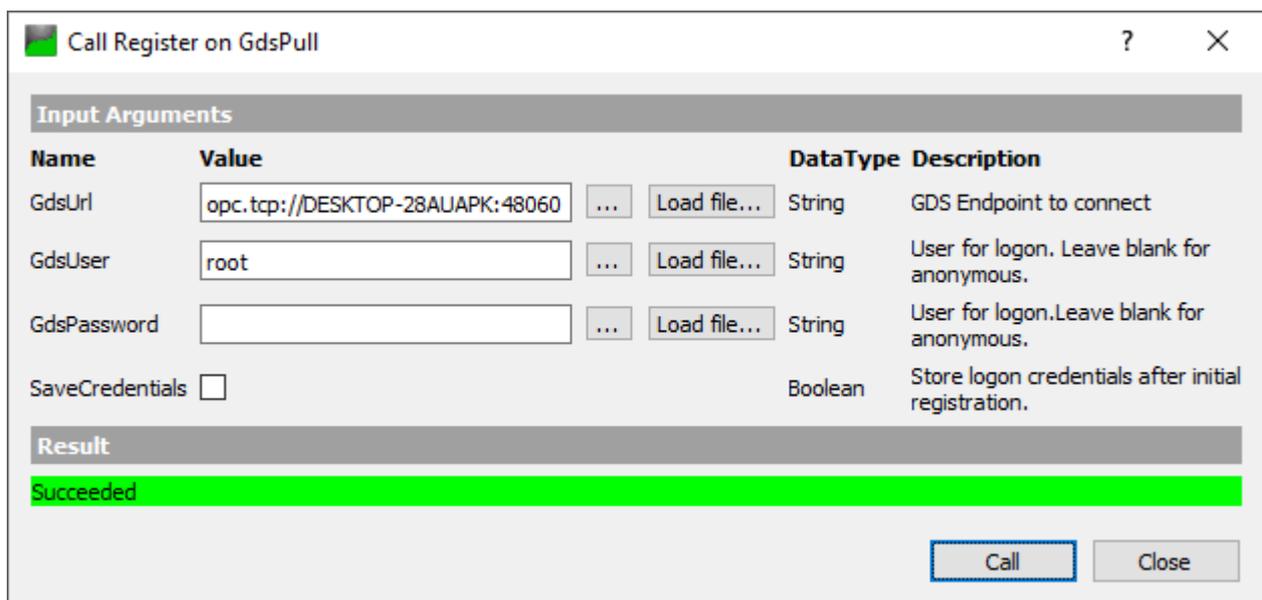


Registrierung am Global Discovery Service

Im ersten Schritt muss der TwinCAT OPC UA Server am GDS als Applikation registriert werden. Dies geschieht über die Methode Register(). Durch die Registrierung am GDS wird automatisch auch ein Serverzertifikat für die Serverapplikation beantragt. Je nach Implementierung der GDS Applikation wird ein solches Zertifikat entweder automatisch oder nach manueller Freigabe durch einen Administrator ausgestellt. Anhand der Variablen RegistrationState und CertificateState lässt sich überprüfen, ob der Server bereits an einem Global Discovery Service registriert wurde und ein Zertifikat von diesem erhalten hat. Die Variable CrlState gibt den Status der Certificate Revocation List an und ob diese vom GDS bezogen werden konnte.

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

Die Methode erwartet die folgenden Eingabeparameter:



Eingabeparameter	Bedeutung
GdsUrl	Die URL des Global Discovery Service im Format opc.tcp://hostnameOrIpAddress:port
GdsUser	Benutzername eines GDS Benutzers mit der Berechtigung neue Applikationen registrieren zu dürfen.
GdsPassword	Password des Benutzers
SaveCredentials	Speichert das Password des Benutzers in einer Konfigurationsdatei des TwinCAT OPC UA Servers. Aus Sicherheitsgründen wird nicht empfohlen diese Einstellung zu setzen, sie wurde ausschliesslich für Global Discovery Service entwickelt, welche eine Benutzername/Password Authentifizierung zwingend benötigen. Üblicherweise wird diese jedoch nur einmalig bei der Registrierung der Applikation benötigt. Bei allen darauf folgenden Verbindungen zum GDS wird dann das ausgestellte Server-Zertifikat verwendet.

● Reinitialisierung der Server-Endpunkte

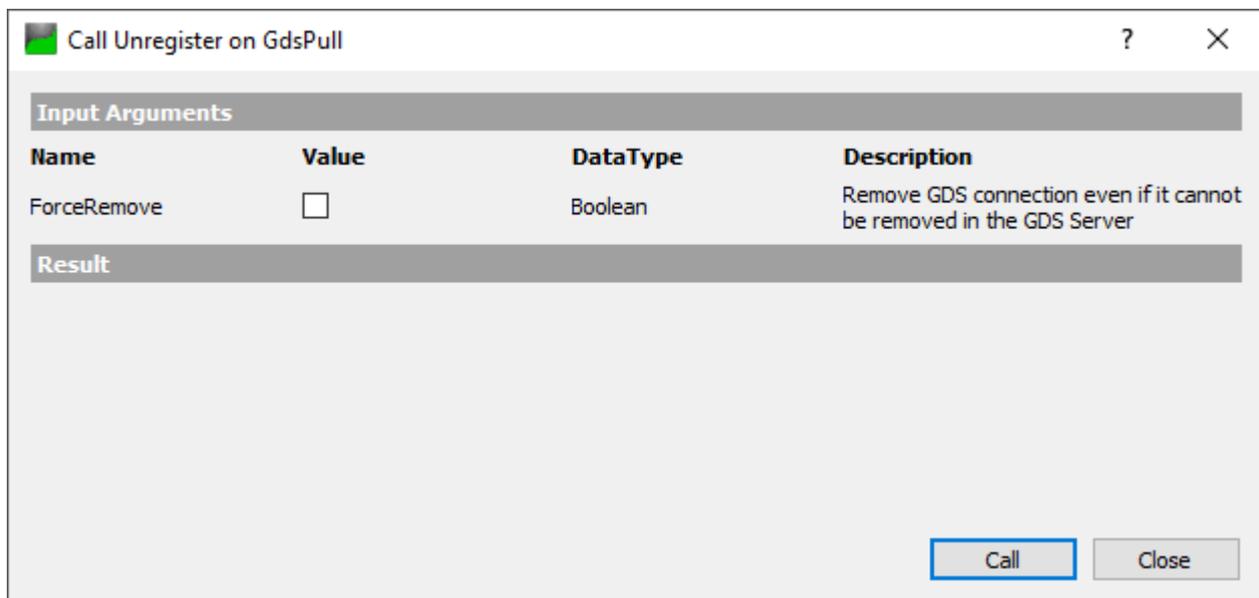
I Nach der Registrierung der Serverapplikation am Global Discovery Service und dem Erhalt eines Serverzertifikats, initialisiert der Server einmal seine Endpunkte neu, wodurch verbundene Clients kurzzeitig die Verbindung verlieren.

Nachdem die Serverapplikation bei einem Global Discovery Service registriert wurde, wird im Installationsverzeichnis des TwinCAT OPC UA Servers eine neue Datei namens "TcUaGdsClientConfig.xml" erzeugt. Diese beinhaltet sowohl die Verbindungsinformationen des konfigurierten GDS als auch die von dort bezogenen Registrierungsinformationen und Zeitstempelinformationen für die Serverapplikation, z.B. wann das Zertifikat und die CRL das letzte Mal aktualisiert wurden.

De-Registrierung am Global Discovery Service

Um die TwinCAT OPC UA Server Applikation beim GDS zu de-registrieren, kann die Methode Unregister() verwendet werden. Die erfolgreiche Ausführung der Methode bewirkt, dass die Serverapplikation auf dem GDS de-registriert und der Inhalt der Datei TcUaGdsClientConfig.xml gelöscht wird.

Die Methode erwartet die folgenden Eingabeparameter:



Eingabeparameter	Bedeutung
ForceRemove	Falls die Verbindung zum Global Discovery Service nicht mehr zur Verfügung steht, kann durch das Setzen dieses Eingabeparameter die Verbindung zum GDS entfernt werden. Der TwinCAT OPC UA Server entfernt hierbei den GDS aus seiner Konfiguration.

Nachdem der TwinCAT OPC UA Server vom GDS entkoppelt wurde, bleiben das ausgestellte Serverzertifikate und die CRL bestehen. Falls Sie diese löschen und den Server mit einem self-signed Zertifikat betreiben möchten, müssen Sie die entsprechenden Dateien im PKI-Verzeichnis des Servers entfernen und den Server anschließend neu starten. Der Server erstellt sich daraufhin wieder ein self-signed Zertifikat.

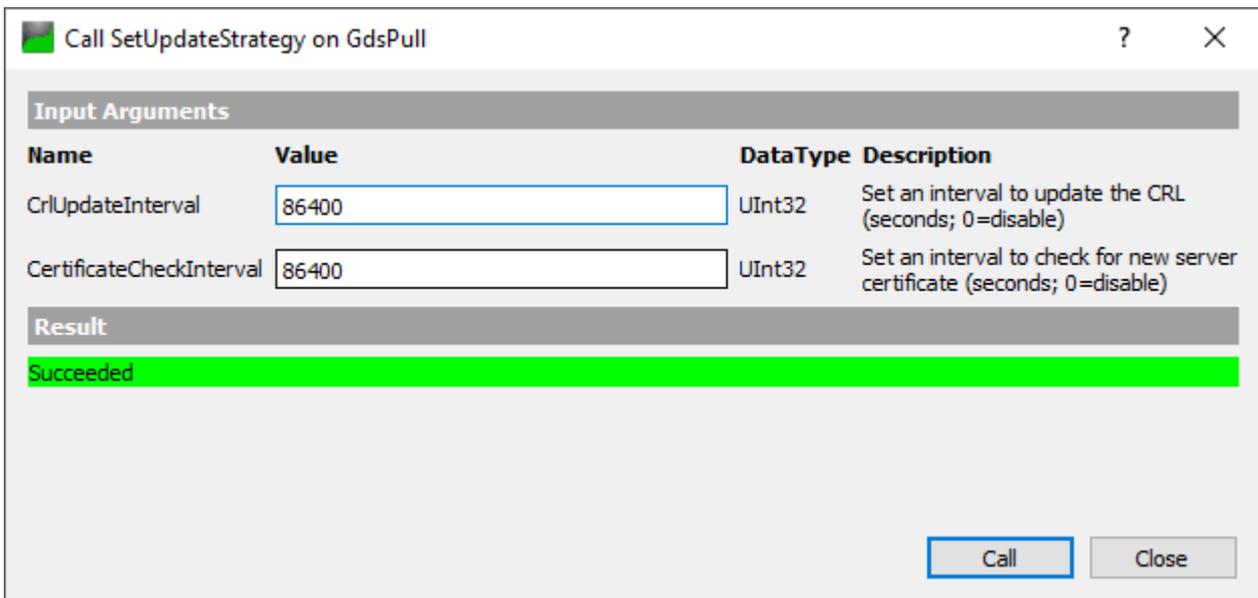
Aktualisierung des Serverzertifikats

Durch Ausführen der Methode UpdateCertificate() kann eine Aktualisierung des Serverzertifikats außerhalb des regulären Updateintervalls beim GDS angefragt werden. Die Methode erwartet keine weiteren Eingabeparameter.

Setzen der Updateintervalle für Serverzertifikat und CRL

Durch Ausführen der Methode SetUpdateStrategy() können die Updateintervalle für das Serverzertifikat und die Zertifikatssperlliste gesetzt werden.

Die Methode erwartet die folgenden Eingabeparameter:



Eingabeparameter	Bedeutung
CrlUpdateInterval	Setzt das Aktualisierungsintervall für die Zertifikatssperlliste. (Sekunden)
CertificateCheckInterval	Setzt das Aktualisierungsintervall für das Serverzertifikat. (Sekunden)

4.1.12 TwinCAT Eventlogger

Der TwinCAT OPC UA Server ermöglicht eine Integration des TwinCAT Eventloggers um Alarm und Events zur Versenden. Hierbei wird ein Alarm/Event vom TwinCAT Eventlogger in einen OPC UA Alarm bzw. Event umgewandelt.

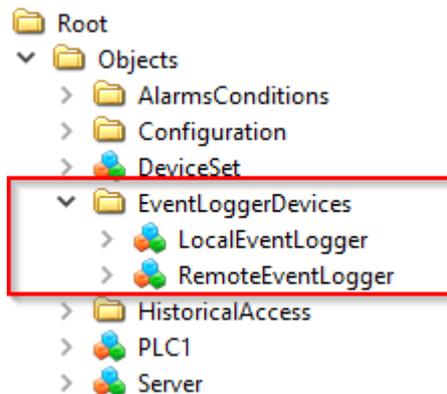
i Die Integration des TwinCAT Eventloggers ist nicht auf Windows CE verfügbar. Das heißt, dass auf Windows CE-Geräten betriebene TwinCAT OPC UA Server den EventLoggerDevices-Ordner nicht anzeigen. Hier ist es dann nur möglich, den TwinCAT EventLogger als Remote-Eventlogger in einem TwinCAT OPC UA Server auf einem anderen Betriebssystem einzubinden.

Konfiguration

Zur Konfiguration des Servers für eine Verbindung mit dem Eventlogger, müssen Sie im Serververzeichnis eine neue Konfigurationsdatei mit dem Namen „TcUaEventLogConfig.xml“ anlegen. Diese Datei beinhaltet eine Liste mit TwinCAT Eventlogger-Geräten, welche anhand ihrer AMS NetID identifiziert werden. Die Konfiguration hat hierbei den folgenden Aufbau:

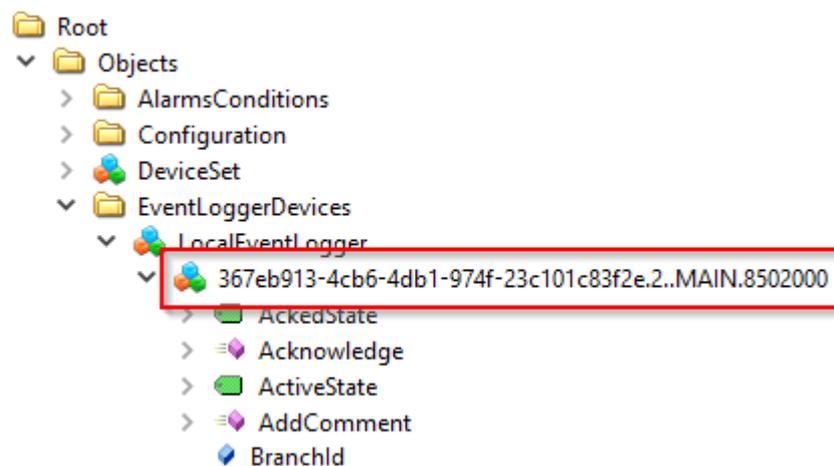
```
<TcUaEventLogConfig>
  <EventLoggerDevice Name="LocalEventLogger" AmsAddr="127.0.0.1.1.1"/>
  <EventLoggerDevice Name="RemoteEventLogger" AmsAddr="192.168.0.56.1.1"/>
</TcUaEventLogConfig>
```

Die einzelnen Geräteeinträge werden anschließend im Folder „EventLoggerDevices“ im Namensraum des Servers angezeigt.



Ein OPC UA Client kann sich nun auf das entsprechende Objekt subscriben um Events und/oder Alarme von dem jeweiligen TwinCAT Eventlogger-Gerät zu empfangen.

Im Gegensatz zu einem Event wird ein Alarm zusätzlich als Kindelement von dem jeweiligen Eventlogger-Gerät angezeigt.

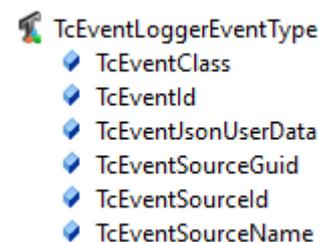


OPC UA Alarm/Event Types

Beim Subscriben auf das Objekt und den Empfang von Alarmen oder Events muss ein Client berücksichtigen, dass hierbei spezifische Objekttypen verwendet werden. Die jeweiligen Typen sind wie folgt definiert.

TcEventLoggerEventType

Der TcEventLoggerEventType ist abgeleitet vom BaseEventType und erweitert diesen mit TwinCAT Eventlogger-spezifischen Properties:



NodeID: i=4200

NamespaceName: urn:BeckhoffAutomation:Ua:Types:GlobalTypes

TcEventLoggerAlarmConditionType

Der TcEventLoggerAlarmConditionType ist abgeleitet vom AlarmConditionType und erweitert diesen mit TwinCAT Eventlogger-spezifischen Properties:

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

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

Beispiel

Das folgende Beispiel basiert auf dem Standard Code-Sample vom TwinCAT Eventlogger, welches aus dem Beckhoff Information System bezogen werden kann. Dieses Sample beinhaltet Code Snippets für die SPS, mit welchen man sowohl ein Event als auch einen Alarm feuern kann.

Schritt 1: Konfiguration des Servers

TwinCAT OPC UA Server und die SPS laufen in diesem Beispiel nun lokal auf demselben System. Dementsprechend wurde auch die TcEventLogConfig.xml erstellt:

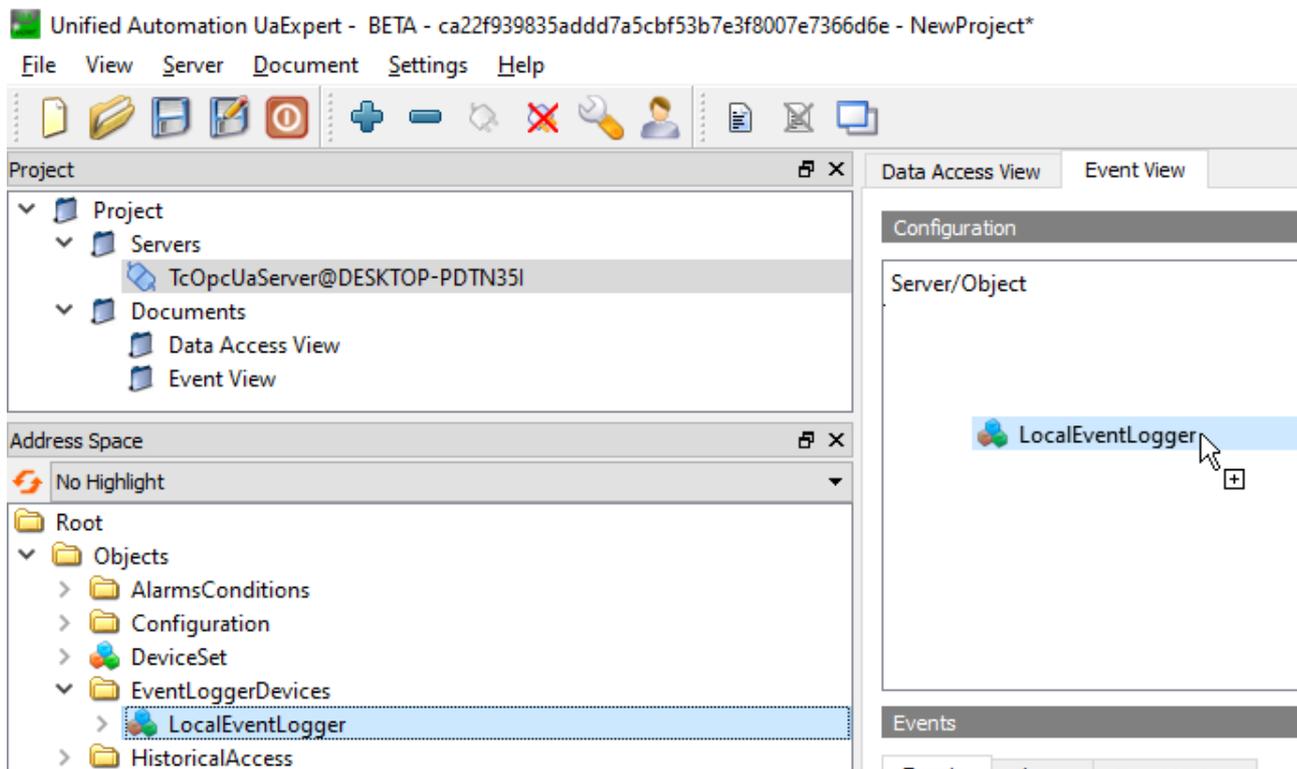
```
<TcUaEventLogConfig>
  <EventLoggerDevice Name="LocalEventLogger" AmsAddr="127.0.0.1.1.1"/>
</TcUaEventLogConfig>
```

Schritt 2: Aktivieren des TwinCAT Eventlogger Samples

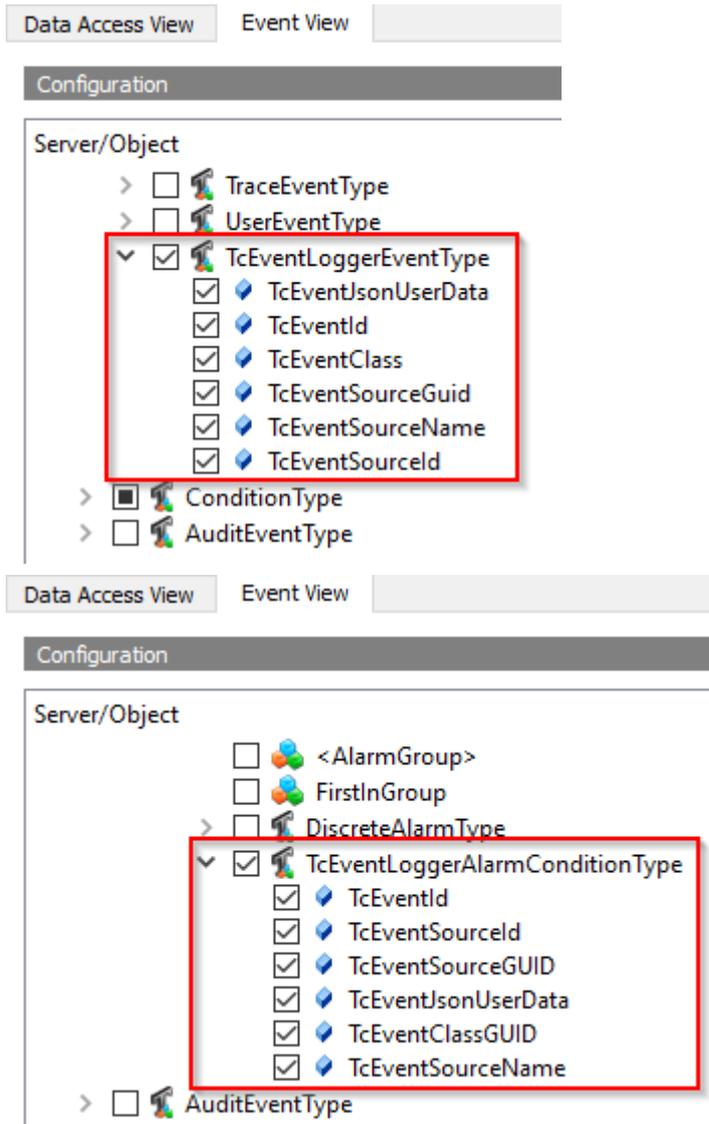
Vor dem Aktivieren des TwinCAT Eventlogger Samples haben wir das automatische Feuern eines Events bzw. Alarms zunächst deaktiviert. In der vorliegenden Sample-Version erfolgt dies durch das Initialisieren von bSend und bAlmRaise mit dem Wert FALSE. Anschließend wird das Projekt aktiviert und in der lokalen SPS-Laufzeit ausgeführt.

Schritt 3: Verbinden eines OPC UA Clients mit dem Server

Als OPC UA Client wurde nun der UA Expert gewählt. Nach dem Aufbau einer Verbindung mit dem Server fügen Sie nun im UA Expert ein neues „Event View“ Dokument hinzu. Anschließend ziehen Sie per Drag&Drop das konfigurierte TwinCAT Eventlogger-Gerät in das Event View.



Damit die TwinCAT Eventlogger-spezifischen Properties empfangen werden können, muss der UA Expert den entsprechend Event- bzw. AlarmType filtern. Sie finden den TcEventLoggerEventType bzw. TcEventLoggerAlarmConditionType in der Typliste des Event View. Beispiel:

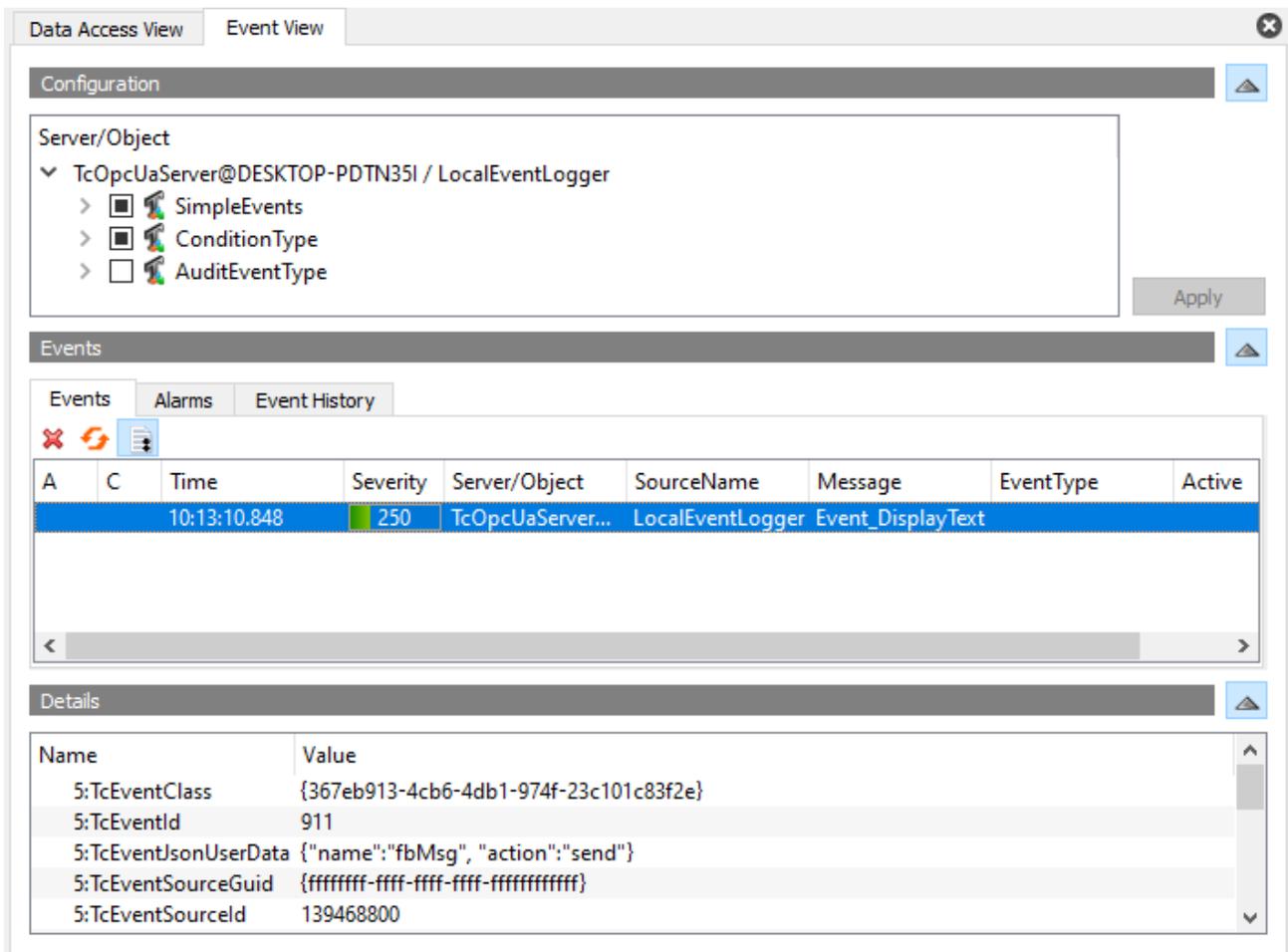


Durch einen Klick auf den **Apply**-Button werden diese Filter übernommen.

Schritt 4: Feuere ein Events

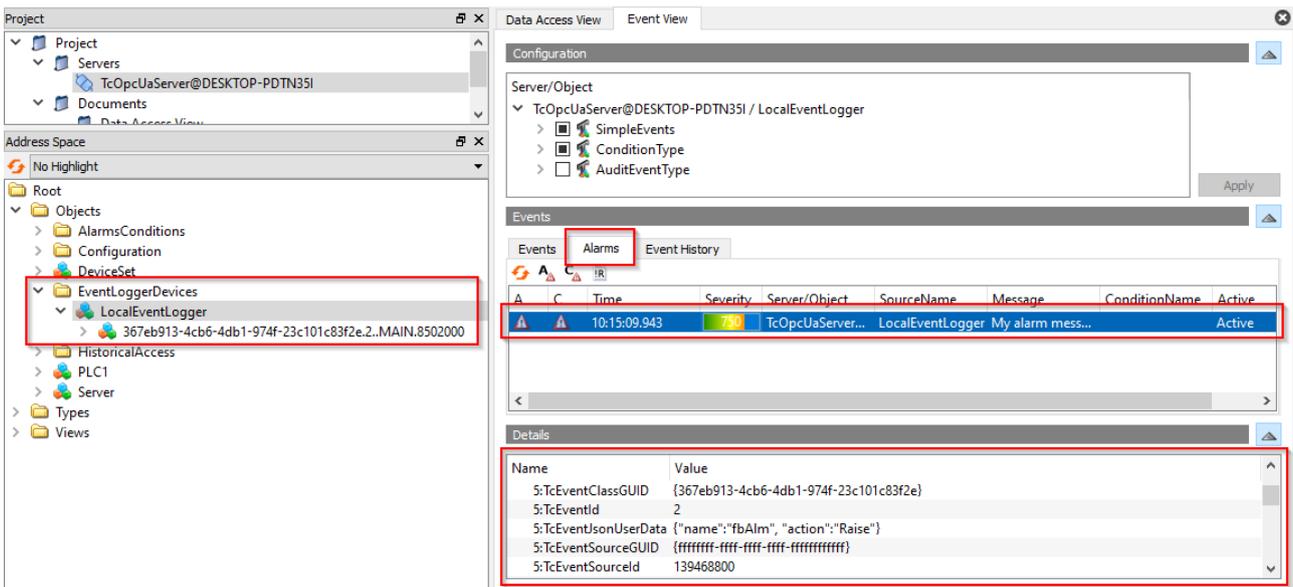
Im TwinCAT Eventlogger Sample setzen wir nun die Variable bSend auf den Wert TRUE, um ein Event abzufeuern. Der UA Expert empfängt dieses Event automatisch und zeigt es im Event View mitsamt der TwinCAT Eventlogger spezifischen Properties an.

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



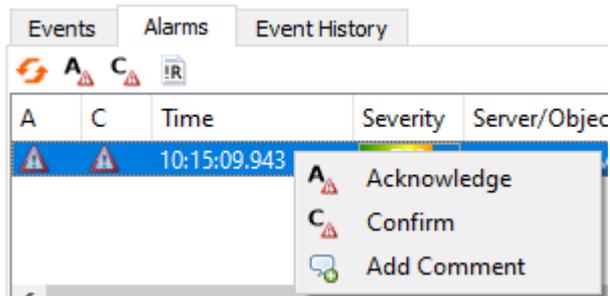
Schritt 5: Feuern eines Alarms

Im TwinCAT Eventlogger Sample setzen wir nun die Variable bAlmRaise auf den Wert TRUE, um einen Alarm abzufeuern. Der UA Expert empfängt diesen Alarm automatisch und zeigt ihn im Event View mitsamt der TwinCAT Eventlogger-spezifischen Properties an. Zusätzlich wird der Alarm als eigenes Objekt im Namespace unterhalb des Eventlogger Geräts angezeigt.



Schritt 6: Quittieren eines Alarms

Zusätzlich zum Empfang eines Alarms kann dieser auch quittiert werden. Die Quittierung wird hierbei vom Server auch an den TwinCAT Eventlogger zurückgemeldet. Im UA Expert lässt sich ein Alarm über das Kontextmenü im Event View quittieren.



● Acknowledge und Confirm

I Bitte beachten Sie, dass nur ein Confirm zurück an den TwinCAT Eventlogger gemeldet wird. Die Information zu einem Acknowledge bleibt hierbei im Server, da der TwinCAT Eventlogger aktuell nur das Konzept eines Confirms vorsieht.

Nach einem Confirm wird die entsprechende Variable `eConfirmationState` in der TwinCAT Eventlogger Instanz vom Typ `FB_TcAlarm` gesetzt.

<code>bRaised</code>	BOOL	TRUE
<code>eConfirmationState</code>	TCEVENTCONFIRMA...	Confirmed

4.1.13 Security

4.1.13.1 Übersicht

Einer der Gründe für den Erfolg von OPC UA als Kommunikationstechnologie sind die verschiedenen, integrierten Sicherheitsmechanismen. Eine auf OPC UA basierte Datenkommunikation lässt sich auf zwei Ebenen absichern:

1. Transportebene
2. Applikationsebene

Endpunkte

Ein Server bietet dem Client eine Liste mit verschiedenen Endpunkten [► 105] an mit denen sich der Client verbinden kann. Ein Endpunkt beschreibt hierbei unter Anderem welche Sicherheitsfunktionen (z. B. Message Security Mode, Security Policy und zur Verfügung stehende Identity Token) die Kommunikationsverbindung über diesen Endpunkt erfüllen soll. So kann ein Endpunkt z. B. eine Signierung und Verschlüsselung der Datenpakete erfordern (Transportebene), sowie eine zusätzliche Authentifizierung des Clients auf Basis von Benutzername/Password (Applikationsebene).

Transportebene

Eine auf OPC UA basierte Kommunikationsverbindung kann auf Transportebene abgesichert werden. Dies geschieht durch die Verwendung von Client/Server Zertifikaten und eine gegenseitige Vertrauensstellung zwischen Client- und Serverapplikation. Hierbei muss der Client dem Server-Zertifikat vertrauen und umgekehrt, damit eine Kommunikationsverbindung hergestellt werden kann. Hierfür ist ein gegenseitiger Zertifikatsaustausch [► 108] notwendig.

Applikationsebene

Zusätzlich zur Transportebene lässt sich eine Kommunikationsverbindung auch auf Applikationsebene absichern. Hierfür stehen verschiedene Authentifizierungsmechanismen [► 106] zur Verfügung, die vom Serverendpunkt angeboten werden.

Sehen Sie dazu auch

- Zugriffsrechte [► 109]

4.1.13.2 Endpunkte

Der TwinCAT OPC UA Server stellt verschiedene Endpunkte über den Standard-Port 4840/tcp für OPC UA Clients zur Verfügung. Die Endpunkte definieren hierbei die Art der Verbindung zwischen Client und Server und ob diese gesichert oder ungesichert erfolgen soll.

● Standard-Port

i Beachten Sie, dass der Standard-Port 4840 eventuell von anderen OPC UA Servern verwendet wird, z. B. dem Local Discovery Server (LDS) von der OPC Foundation, die von manchen Anbietern mit OPC-UA-Softwarepaketen eingesetzt wird.

● Sichere Endpunkte

i Bitte beachten Sie, dass zur Verwendung der sicheren Endpunkte ein Vertrauensverhältnis zwischen Server und Client hergestellt werden muss, was üblicherweise über deren Zertifikate erfolgt. Wie Sie ein solches Vertrauensverhältnis Server-seitig konfigurieren können, erfahren Sie [hier \[► 108\]](#).

● Deprecated Endpunkte

i Über einen Konfigurationsschalter (<AllowDeprecatedSecurityPolicies>) in der TcUaServerConfig.xml lassen sich veraltete und als unsicher eingestufte Security Policies wieder aktivieren. Wir empfehlen jedoch aus Sicherheitsgründen diesen Konfigurationsschalter deaktiviert zu lassen.

Liste der Endpunkte

Die Liste fasst die Endpunkte des OPC UA Servers zusammen. Dabei sind auch bereits abgekündigte Endpunkte enthalten. Der folgende Screenshot zeigt die aktuell im OPC UA Server (Version 3.2.0.62) enthaltenen Endpunkte. Dabei kann entweder nur Signierung oder Signierung in Verbindung mit Verschlüsselung genutzt werden.

Seit dem Setup mit der Version 4.3.28 (Server-Version 3.2.0.62) ist außerdem der unverschlüsselte Endpunkt aus Security- und Zertifizierungsgründen standardmäßig deaktiviert.

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

Die verfügbaren Endpunkte orientieren sich an der Sicherheit der Security-Mechanismen. Werden Security-Profile durch den Lauf der Zeit als potenziell unsicher eingestuft, werden sie standardmäßig nicht mehr im TwinCAT OPC UA Server eingesetzt, sondern durch neuere und sichere Verschlüsselungsalgorithmen ersetzt.

Security-Profil	Security-Modus	Kurzbeschreibung
None	None	Das ist der unverschlüsselte Endpunkt, mit diesem kann ohne Security kommuniziert werden. Seit Version 4.3.28 standardmäßig deaktiviert. Kann bei Bedarf durch Konfiguration des Servers wieder aktiviert werden.
Basic128Rsa15 (veraltet)	Sign / Sign & Encrypt	Dieser Endpunkt wird sicherheitstechnisch als veraltet eingestuft und ist standardmäßig deaktiviert. Bei Bedarf durch Konfiguration des Servers wieder freischaltbar.
Basic256 (veraltet)	Sign / Sign & Encrypt	Dieser Endpunkt wird sicherheitstechnisch als veraltet eingestuft und ist standardmäßig deaktiviert. Bei Bedarf durch Konfiguration des Servers wieder freischaltbar.
Basic256Sha256	Sign / Sign & Encrypt	Aktuell im Server vorhandener Endpunkt für sichere Signierung und Verschlüsselung.
Aes256_Sha256_RsaPss	Sign / Sign & Encrypt	Aktuell im Server vorhandener Endpunkt für sichere Signierung und Verschlüsselung.
Aes256_Sha256_RsaOaep	Sign / Sign & Encrypt	Aktuell im Server vorhandener Endpunkt für sichere Signierung und Verschlüsselung.

Alle in der Liste aufgeführten Endpunkte können über die Konfiguration des Servers aktiviert oder deaktiviert werden. In der folgenden Abbildung sind alle Endpunkte aktiviert.

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

4.1.13.3 Authentifizierung

Eine OPC UA Client Applikation kann sich über verschiedene IdentityToken am Server authentifizieren:

- Anonymous
- Benutzername/Password
- Benutzerzertifikat

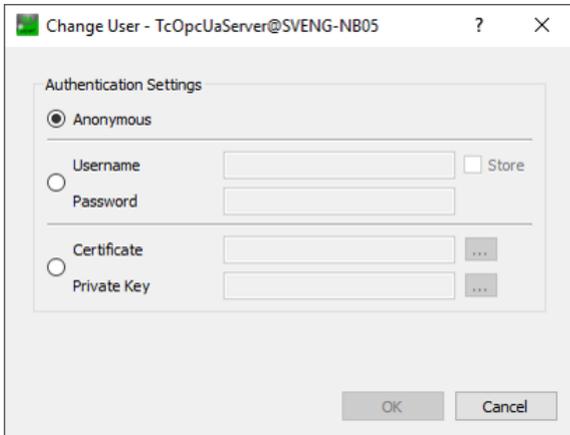
Auslieferungszustand



Im Auslieferungszustand des Servers ist das Anonymous IdentityToken aktiviert, der Server erfordert jedoch eine einmalige [Initialisierung](#) [► 37], um verwendet werden zu können. Anschließend wird das Anonymous IdentityToken deaktiviert und Client-Applikationen müssen sich mit einem gültigen User IdentityToken am Server authentifizieren.

Anonymous

Diese Art der Authentifizierung ermöglicht es, beliebigen OPC UA Clients eine Verbindung zur Serverapplikation herzustellen. Die Angabe einer Benutzeridentität ist hierbei nicht erforderlich, wodurch sich auch keinerlei Möglichkeiten bieten, Zugriffsrechte auf dem Server zu definieren. Wir empfehlen diese Authentifizierungsart nach der Inbetriebnahme des Servers zu deaktivieren. Dies kann über den TwinCAT OPC UA Konfigurator erfolgen. Im Folgenden finden Sie einen Beispiel-Screenshot aus der OPC UA Client Applikation "UA Expert":



Benutzername/Passwort

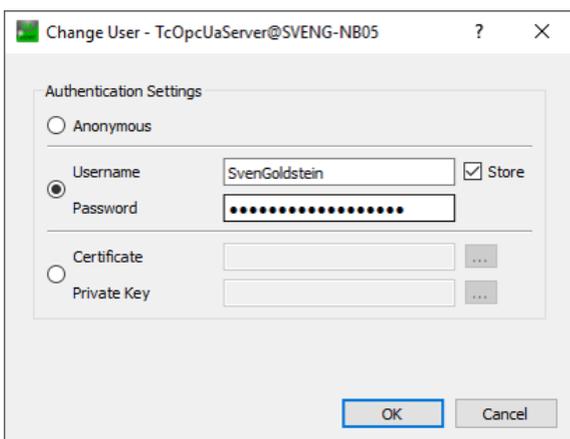
Diese Art der Authentifizierung verwendet eine Benutzername/Passwort-Kombination zum Authentifizieren des Clients an der Serverapplikation. Auf dem Server lassen sich dann Zugriffsrechte für die jeweilige Benutzeridentität definieren. Die Benutzeridentität kann hierbei auf verschiedenen Ebenen definiert sein:

- Benutzeridentität ist im Server definiert
- Benutzeridentität kommt aus dem unterlagerten Betriebssystem (z. B. ein lokaler Windows Benutzer)
- Benutzeridentität kommt aus dem Active Directory (z. B., wenn der Industrie-PC Teil einer Windows Domäne ist)

i Empfehlung bei Verwendung von User IdentityToken

Sollen User IdentityToken zur Authentifizierung von Client-Applikationen verwendet werden, so empfehlen wir die Verwendung von Betriebssystem-Benutzern.

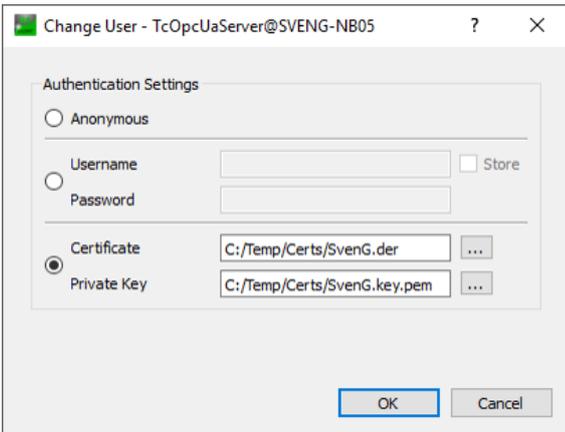
Im Folgenden finden Sie einen Beispiel-Screenshot aus der OPC UA Client Applikation "UA Expert":



Benutzerzertifikat

Diese Art der Authentifizierung verwendet ein Zertifikat, um sich an der Serverapplikation zu authentifizieren. Die Handhabung der Benutzerzertifikate auf Serverseite ist identisch zur Verwendung von Zertifikaten auf Transportebene, d. h. der Server muss dem (Benutzer-) Zertifikat vertrauen, bevor sich der Client mit dem

Zertifikat erfolgreich am Server authentifizieren kann. Ein separates Verzeichnis ("pkuser") zur Verwaltung der Benutzerzertifikate steht hierfür im Server zur Verfügung. Im Folgenden finden Sie einen Beispiel-Screenshot aus der OPC UA Client Applikation "UA Expert":



HINWEIS

Authentifizierung und Serverzertifikat

Der TwinCAT OPC UA Client benötigt bei der Verwendung des unverschlüsselten Endpunkts in Verbindung mit Authentifizierung dennoch den Public-Key vom OPC UA Server Zertifikat, um das Passwort bei der Übertragung zu verschlüsseln. Dazu muss diesem Zertifikat im TwinCAT OPC UA Client vertraut werden (siehe [Zertifikatsaustausch \[▶ 108\]](#)).

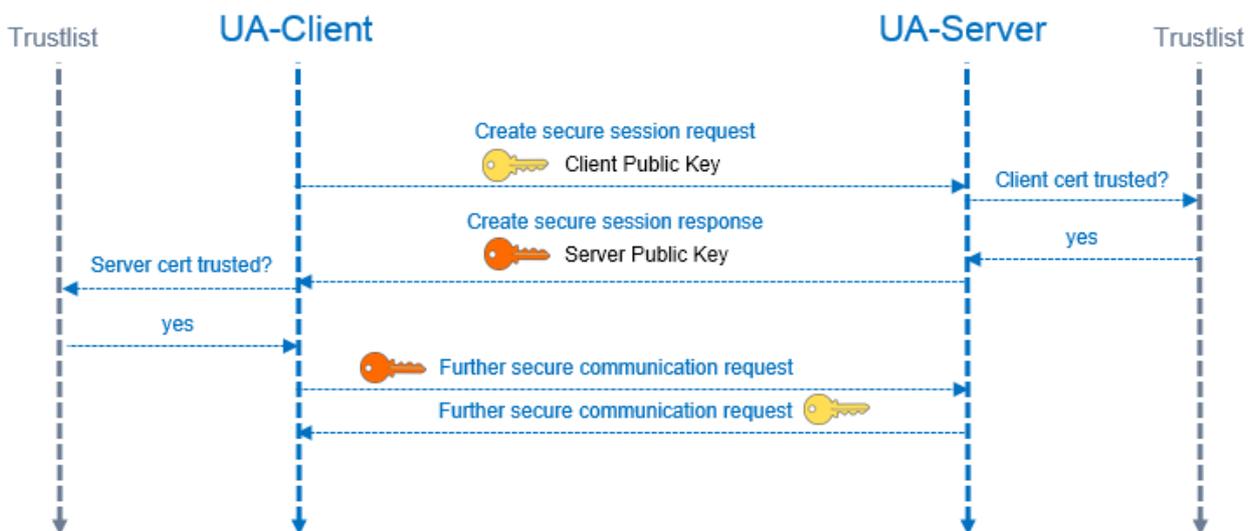
Sehen Sie dazu auch

[Zugriffsrechte \[▶ 109\]](#)

4.1.13.4 Zertifikatsaustausch

Für eine Absicherung der Kommunikationsverbindung auf Transportebene über einen sicheren Endpunkt [▶ 105] ist die Herstellung einer gegenseitigen Vertrauensstellung zwischen Client und Server notwendig.

Standardmäßig generieren sowohl der TwinCAT OPC UA Server als auch der TwinCAT OPC UA Client beim ersten Start ein maschinenspezifisches, selbstsigniertes Zertifikat zur Authentifizierung der jeweiligen Applikation.



Vertrauensstellung auf dem Server einrichten

Zur Einrichtung einer Vertrauensstellung zwischen einem beliebigen OPC UA Client und dem TwinCAT OPC UA Server, benötigen Sie den öffentlichen Schlüssel des Clientzertifikats. Der Server muss diesem vertrauen. Dies kann zum Beispiel über das Dateisystem erfolgen. Der Server verwaltet die Vertrauenseinstellungen für Client-Zertifikate im Unterverzeichnis PKI.

- Vertrauenswürdige Zertifikate: %InstallDir%\Server\PKI\CA\trusted\certs
- Nicht-Vertrauenswürdige Zertifikate: %InstallDir%\Server\PKI\CA\rejected\certs

Durch Verschieben von Client-Zertifikaten zwischen diesen Verzeichnissen können die Vertrauenseinstellungen entsprechend angepasst werden. Der öffentliche Schlüssel eines Clientzertifikats wird beim ersten Verbindungsversuch des Clients mit einem sicheren Endpunkt automatisch im oben genannten Verzeichnis für nicht-Vertrauenswürdige Zertifikate abgelegt. Durch das anschließende Verschieben des öffentlichen Schlüssels in das Verzeichnis für vertrauenswürdige Zertifikate, wird dem Client beim nächsten Verbindungsversuch vertraut.

i AutomaticallyTrustAllClientCertificates

Ist diese Option in der TcUaServerConfig.xml aktiviert, so vertraut der Server automatisch allen Clientzertifikaten. Diese werden in diesem Fall nicht in einem der oben genannten Verzeichnisse aufgelistet.

Vertrauensstellung auf dem Client einrichten

Abhängig vom verwendeten OPC UA Client müssen eventuell unterschiedliche Schritte vorgenommen werden, damit der OPC UA Client dem OPC UA Server vertraut. Typischerweise erfolgt bei Client-Applikationen mit grafischer Benutzeroberfläche ein Warnhinweis bei der ersten Verbindung mit dem Server, wobei das Server-Zertifikat dann als vertrauenswürdige eingestuft werden kann.

Die folgende Anweisung ist daher nur für den TwinCAT OPC UA Client gültig.

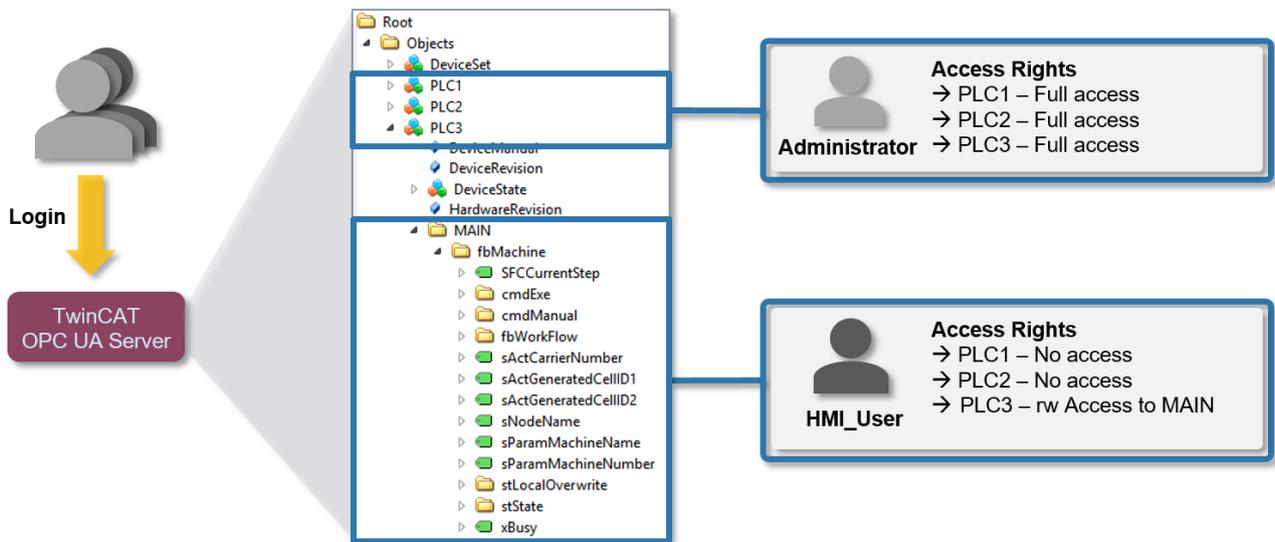
Der öffentliche Schlüssel des OPC UA Servers befindet sich als DER-Datei in dem folgenden Verzeichnis:
`%InstallDir%\Server\PKI\CA\own\certs`

Kopieren Sie die Datei beim TwinCAT OPC UA Client in das entsprechende „Trusted“-Verzeichnis:
`%InstallDir%\Client\PKI\CA\trusted\certs`

4.1.13.5 Zugriffsrechte

Der TwinCAT OPC UA Server ermöglicht die Konfiguration von Zugriffsrechten für bestimmte authentifizierte Benutzeridentitäten [► 106]. Diese Zugriffsrechte können sowohl für ganze Namespaces als auch auf individuelle Nodes konfiguriert werden.

Hierdurch kann sowohl der Zugriff auf ADS-Geräte (z. B. auf verschiedene SPS-Laufzeiten) als auch Variablen feingranular eingestellt werden. Diese Sicherheitseinstellungen sind für alle ADS-Geräte verfügbar, die im Server-Namespaces dargestellt werden können.



Siehe auch:

Konfiguration > Setup-Version 4.x.x. > [Konfiguration der Sicherheitseinstellungen im Konfigurator \[▶ 135\]](#).

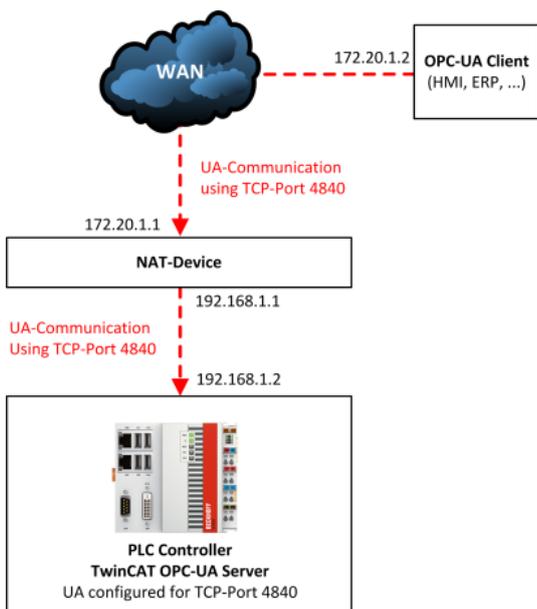
Voraussetzungen

Produkte	Setup-Versionen	Zielplattform
TF6100	4.x.x	IPC oder CX (x86, x64, ARM)

4.1.14 Verschiedenes

4.1.14.1 Firewalls konfigurieren

Um eine OPC-UA-Kommunikation auch über ein NAT-Gerät, z. B. einen Internet-Router, zu ermöglichen, muss dieses den verwendeten UA-Port an den TwinCAT OPC UA Server weiterleiten können (sogenanntes „Port Forwarding“). Standardmäßig wird der TwinCAT OPC UA Server für eine UA-Kommunikation über den TCP-Port 4840 konfiguriert, diese Konfiguration kann jedoch bei Bedarf über die Server-Konfigurationsdatei oder den OPC-UA-Konfigurator selbst angepasst werden. Die folgende Abbildung verdeutlicht den Zusammenhang von Port Forwarding und dem UA Server.

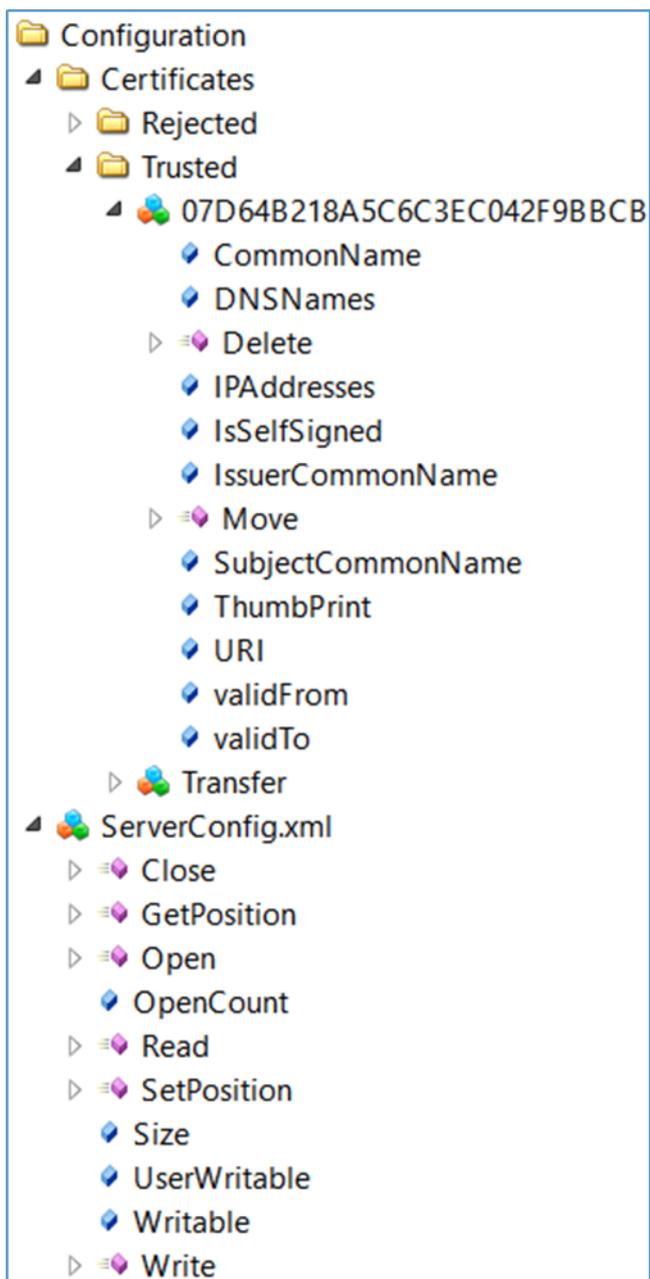


In diesem Beispiel baut der OPC UA Client eine UA-Verbindung über den TCP-Port 4840 zum NAT-Gerät auf, welches dann diese Kommunikationsverbindung über ein Port Forwarding an den TwinCAT OPC UA Server weiterleitet. Das NAT-Gerät muss also nur den im TwinCAT OPC UA Server konfigurierten UA-Port an das Zielgerät weiterleiten. Den vom UA Server verwendeten Port können Sie entweder in der Server-Konfigurationsdatei oder auch ganz bequem über den UA-Konfigurator (im Auslieferungszustand ist dieser immer der TCP-Port 4840) einsehen. Die entsprechende Konfiguration Ihres NAT-Geräts für ein Port Forwarding entnehmen Sie bitte dessen Dokumentation.

4.1.14.2 Namespace zur Konfiguration des Servers

Ab der Serverversion 2.1.x bietet der TwinCAT OPC UA Server einen sogenannten Konfigurationsnamensraum welcher die folgenden Funktionalitäten enthält:

- Verwaltung der Serverkonfigurationsdateien
- Verwaltung der Zertifikate



Verwaltung der Serverkonfigurationsdateien

Die Serverkonfigurationsdateien werden im Namensraum als OPC UA FileType veröffentlicht und bieten daher die entsprechenden Methoden und Eigenschaften, um Zugriff auf die jeweilige Datei zu erhalten.

Verwaltung von Zertifikaten

Jedes Client-Zertifikat, das dem Server bekannt ist, wird im Namensraum als OPC-UA CertificateType veröffentlicht. Zertifikate werden in „abgelehnte“ und „vertrauenswürdige“ Zertifikate unterteilt, was durch einen separaten Ordner im Namensraum repräsentiert wird.

Durch Aufruf der Methode Move() kann ein Zertifikat zwischen den Vertrauenslisten verschoben werden.

Zudem bieten verschiedene Eigenschaften zur einfacheren Identifikation weitere Informationen über die Zertifikate selbst.

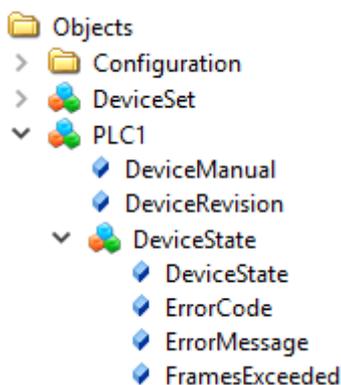
Verwendung des Konfigurationsnamensraums

Der Konfigurationsnamensraum ist aus Gründen der einfachen Benutzung standardmäßig aktiviert und steht Anwendern zur Verfügung. Der TwinCAT OPC UA Konfigurator kann sich hierdurch mit einem Server verbinden und die entsprechende Konfiguration des Servers vornehmen.

Wir empfehlen den Namensraum nach Abschluß der Serverkonfiguration nur authentifizierten Benutzern zugänglich zu machen. Dies bedeutet, dass ein OPC UA Client sich gegenüber dem OPC UA Server durch Bereitstellung einer gültigen Benutzername-Passwort-Kombination authentifizieren muss, um auf den Namensraum zugreifen zu können. Wie Sie dies einstellen können erfahren Sie [hier \[► 135\]](#).

4.1.14.3 DeviceState

Jeder Namespace im TwinCAT OPC UA Server enthält ein sogenanntes DeviceState Objekt.



Dieses Objekt zeigt über diverse Properties den Zustand des unterlagerten ADS Gerätes an.

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

Wenn sich das Gerät in einem ERROR Zustand befindet, dann liefert das ErrorCode Property folgende Werte:

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

Eine entsprechend lesbare Fehlermeldung wird dann im ErrorMessage Property dargestellt.

4.1.14.4 ReverseConnect

Der TwinCAT OPC UA Server unterstützt die ReverseConnect Funktion von OPC UA, um eine rückwärtsgerichtete Kommunikationsverbindung vom Server zum Client aufzubauen. Um diese Funktion zu aktivieren, muss im Server eine Liste mit Client-Adressen hinterlegt werden. Anschliessend baut der Server für jeden Client in der Liste eine OPC UA TCP Verbindung auf.

● Kompatibilität von Clients

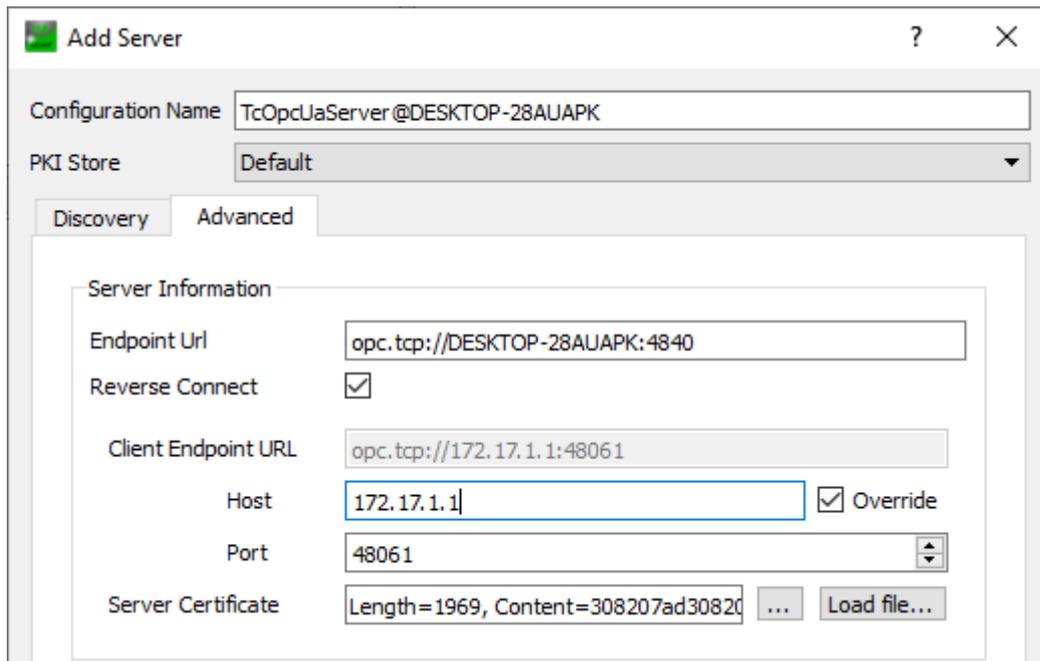
i Bitte beachten Sie, dass der OPC UA Client ebenfalls diese Funktion unterstützen und über seine ReverseConnect URL erreichbar sein muss.

Konfiguration im Server

Eine Liste mit OPC UA Clients kann in der TcUaServerConfig.xml innerhalb des <UaEndpoint> konfiguriert werden.

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

Beispiel-Konfiguration im UA Client (UA Expert)



Im UA Expert können Sie in der Registerkarte **Advanced** den ReverseConnect aktivieren und den Hostnamen oder die IP-Adresse des Clientrechners, sowie den vom Client zu öffnenden Port und das Serverzertifikat zum Herstellen der Vertrauensstellung konfigurieren.

4.1.14.5 DI Components

Jeder SPS Namensraum auf dem Server beinhaltet eine Anzahl an Nodes, über welche sich statische Metainformationen zur SPS angeben lassen. Diese optionalen Informationen können in der TcUaDaConfig.xml für jeden Namenraum angegeben werden.

- PLC1
 - DeviceManual
 - DeviceRevision
- > DeviceState
 - HardwareRevision
 - Manufacturer
 - Model
- > Programs
 - RevisionCounter
 - SerialNumber
 - SoftwareRevision
- > Tasks

Die folgende Tabelle gibt weitere Informationen zu diesen Nodes. Die konkreten Wertebelegungen der einzelnen Nodes können zu einem großen Teil anwendungsspezifisch sein, daher werden im Auslieferungszustand des Servers nur Beispielwerte verwendet. Der Server selbst stellt diese Nodes in seinem Namensraum zur Verfügung, ändert jedoch nicht selbstständig deren Wertebelegungen.

Node	Beschreibung
DeviceManual	Erlaubt die Angabe einer Adresse unter der sich das Gerätehandbuch finden lässt, z.B. ein Pfad im Dateisystem oder eine Webadresse.
DeviceRevision	Beinhaltet den Revisions-Level einer Hardwarekomponente oder des gesamten Geräts.
HardwareRevision	Beinhaltet den Revisions-Level der Hardware.
Manufacturer	Beinhaltet den Namen des Geräteherstellers, üblicherweise als FQDN (Fully Qualified Domain Name), z.B. beckhoff.com.
Model	Beinhaltet den Namen des „Produkts“ (falls anwendbar).
RevisionCounter	Kann einen Zähler beinhalten, wie oft die Konfiguration des Geräts aktualisiert wurde.
SerialNumber	Eindeutige Seriennummer des Geräts, wie vom Gerätehersteller vergeben.
SoftwareRevision	Beinhaltet die Version oder den Revisions-Level der Softwarekomponente, der Firmware einer Hardwarekomponente oder auch der Firmware des Geräts.

4.1.14.6 ServerState

Die ServerState Variable im Server-Namensraum gibt den aktuellen Zustand des Servers an. Die folgende Tabelle gibt einen Überblick über die möglichen Variablenwerte.

Variablenwert	Beschreibung
Running	Der Server wurde erfolgreich hochgefahren.
Failed	Es wurde ein Problem in einer der Server Konfigurationsdateien gefunden, z.B. eine ungültige Konfiguration in der TcUaSecurityConfig.xml.
NoConfiguration	Der Server wurde noch nicht <u>initialisiert</u> [► 37].
Suspended	Der Server wurde noch nicht komplett hochgefahren, d.h. es sind unter Umständen noch nicht alle Funktionen verfügbar.

- ▼ Server
 - > AlarmsConditions
 - ◆ Auditing
 - > EventLoggerDevices
 - > GetMonitoredItems
 - ◆ NamespaceArray
 - > Namespaces
 - > RequestServerStateChange
 - > ResendData
 - ◆ ServerArray
 - > ServerCapabilities
 - > ServerConfiguration
 - > ServerDiagnostics
 - > ServerRedundancy
 - ▼ ServerStatus
 - > BuildInfo
 - > CurrentTime
 - > SecondsTillShutdown
 - > ShutdownReason
 - > StartTime
 - State
 - ◆ ServiceLevel
 - > Trace
 - > VendorServerInfo

4.1.14.7 Logging

Sie können für eine erweiterte Diagnose eine Protokolldatei im Server aktivieren, in welcher dann auf Basis von unterschiedlichen Protokollleveln verschiedene Informationen mitgeschrieben werden. Diese Protokolldatei wird üblicherweise nur zu Diagnosezwecken benötigt.

Das Aktivieren/Deaktivieren des Loggings erfolgt im Regelfall über den [Visual Studio Konfigurator \[► 143\]](#) oder [Standalone Konfigurator \[► 158\]](#).

Der Standardpfad für die erstellten Protokolldateien lautet:

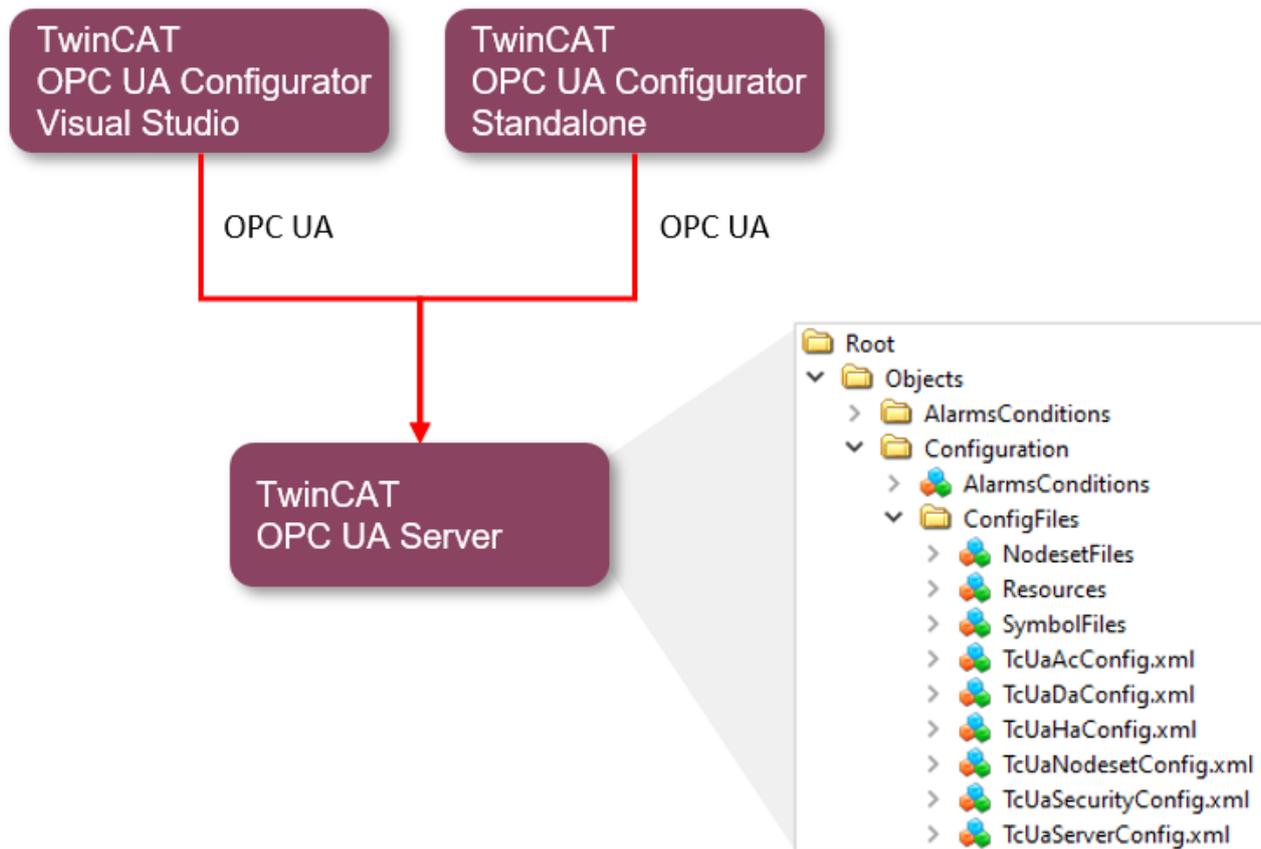
```
Windows: %ProgramData%\Beckhoff\TF6100-OPC-UA-Server\logs
TwinCAT/BSD: /var/log/TF6100-OPC-UA-Server
```

4.2 Konfigurator

Der TwinCAT OPC UA Configurator bietet ein grafisches Frontend zum Editieren der TwinCAT OPC UA Server Konfigurationsdateien. Er wird in zwei Varianten ausgeliefert:

- Als [Visual Studio \[► 116\]](#) Projektvorlage
- Als [Standalone \[► 145\]](#) Applikation

Beide Varianten sind Bestandteil des TwinCAT OPC UA Configurator Setups und ermöglichen eine Online-Konfiguration des TwinCAT OPC UA Servers indem sie sich per OPC UA mit dem Server verbinden und dessen Konfigurationsdateien editieren. Dieser Zusammenhang ist in folgendem Schaubild noch einmal dargestellt.

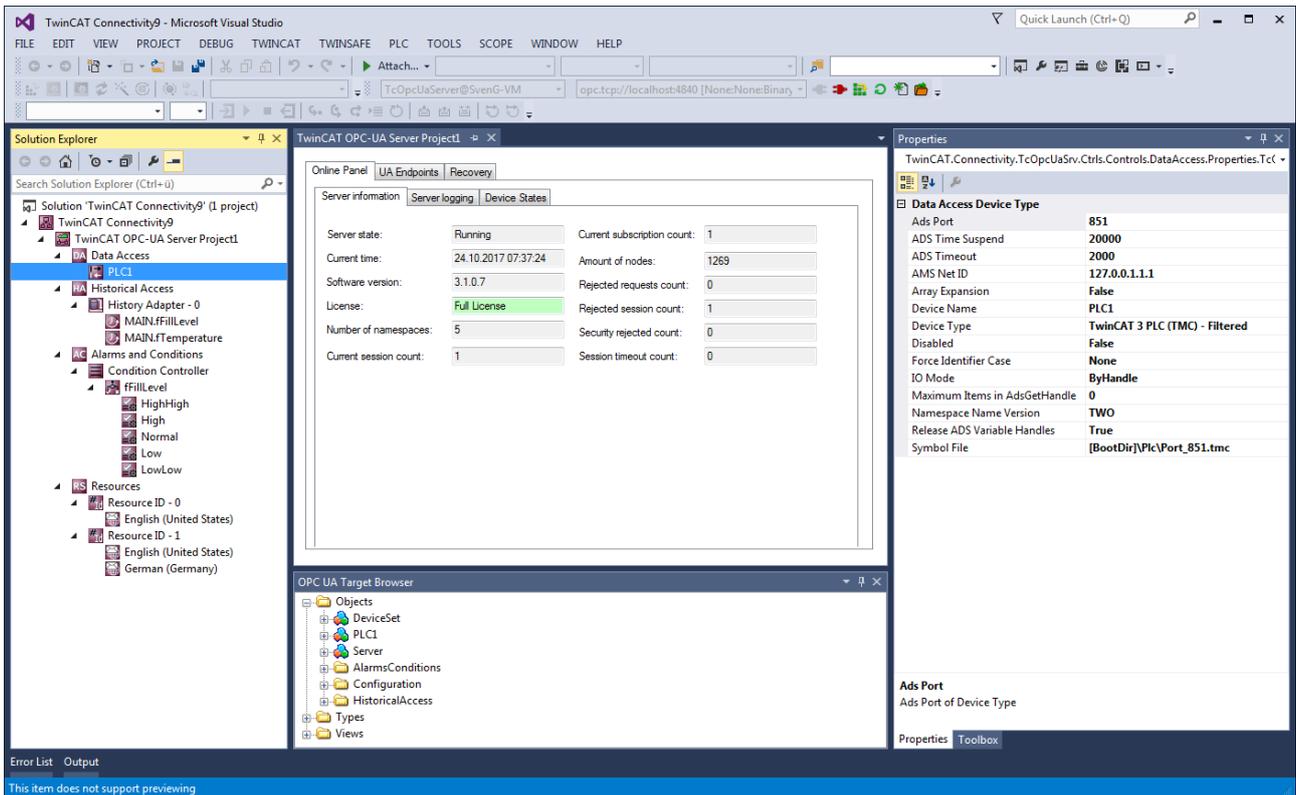


Grundlage für die Konfiguration über OPC UA ist der sogenannte Konfigurations-Namensraum [► 111] des TwinCAT OPC UA Servers. Hierin werden die Konfigurationsdateien des Servers als OPC UA Objekte bereitgestellt.

4.2.1 Visual Studio

4.2.1.1 Übersicht

Das TF6100-Setup (Version 4.x.x und höher) beinhaltet die aktuellste Version des OPC-UA-Server-Konfigurators. Dieser wurde für ein durchgängiges und einheitliches Engineering-Konzept in Microsoft Visual Studio als eigener Projekttyp integriert. Sie können alle unterschiedlichen Facetten vom TwinCAT OPC UA Server konfigurieren und hierbei auch Source-Control-Mechanismen wie z. B. Team Foundation Server oder Subversion-Integrationen verwenden.



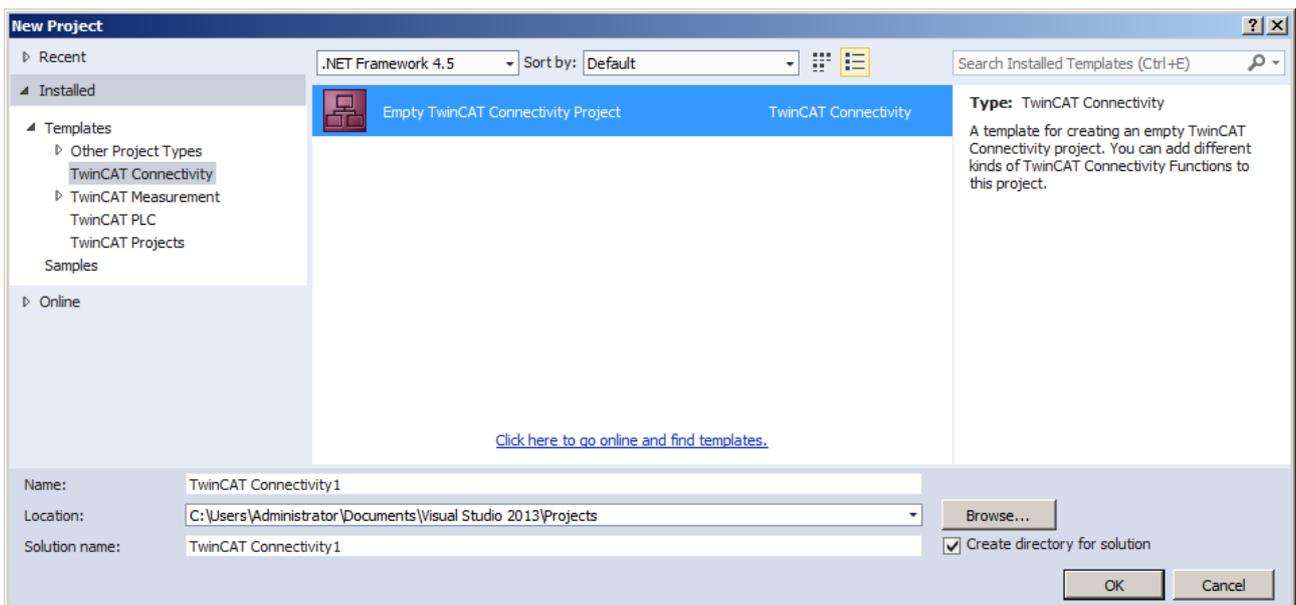
Voraussetzungen

Produkte	Setup-Versionen	Zielplattform
TF6100	4.x.X	IPC oder CX (x86, x64, ARM)

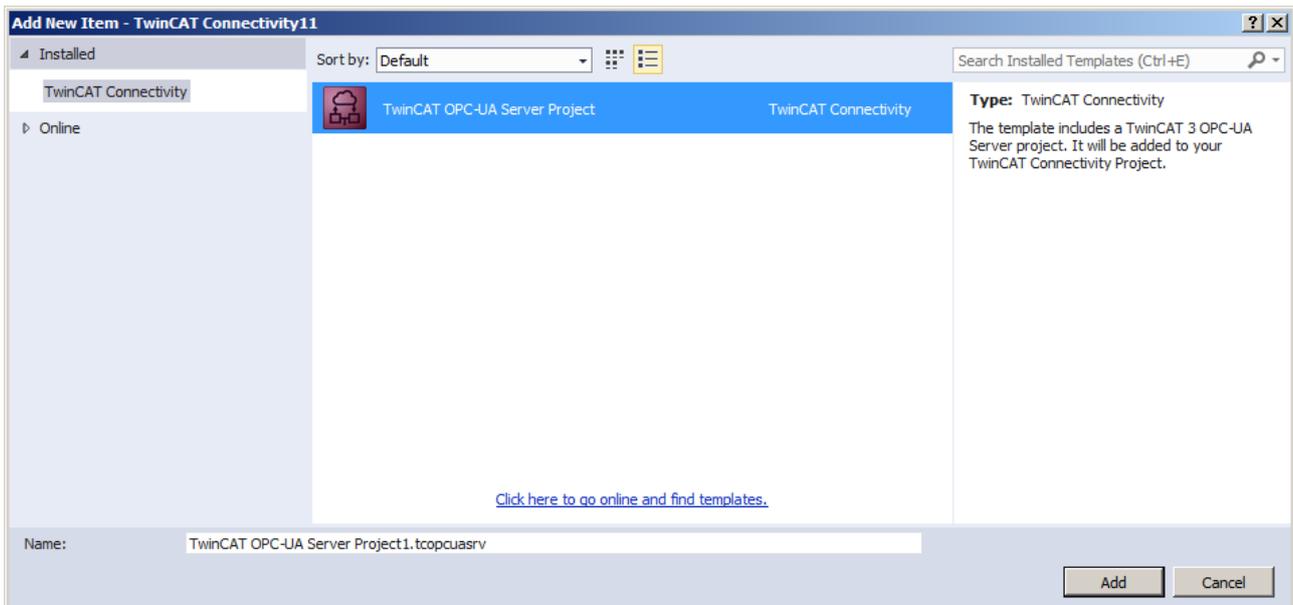
4.2.1.2 Neues Projekt anlegen

Das Projektpaket vom OPC-UA-Konfigurator bindet sich in das sogenannte Connectivity-Paket ein. Sie können dieses beim Anlegen eines neuen Visual-Studio-Projekts auswählen.

Projektvorlage „TwinCAT Connectivity Project“:



Projektvorlage „TwinCAT OPC-UA Server Project“:



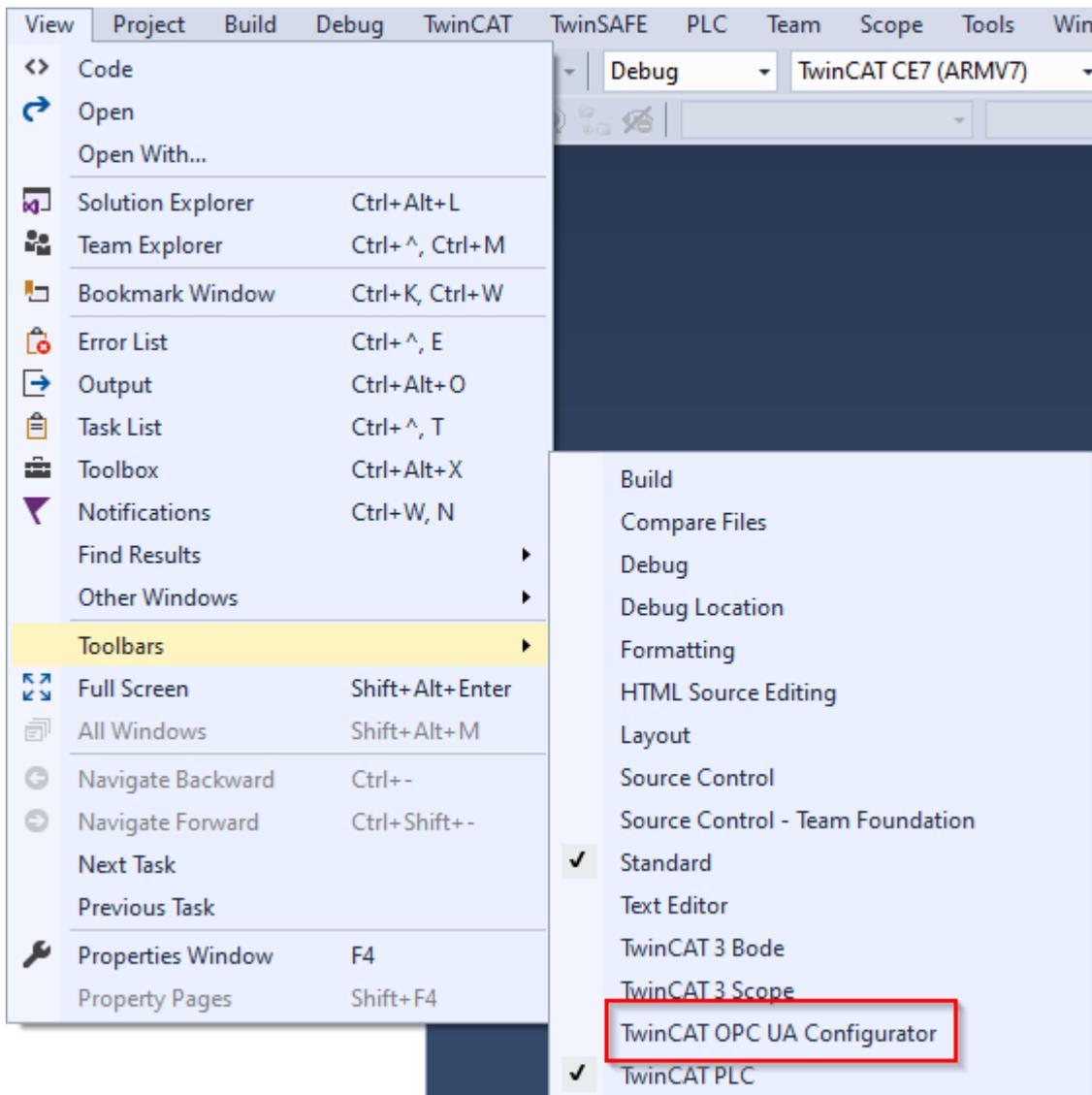
Voraussetzungen

Produkte	Setup-Versionen	Zielplattform
TF6100	4.x.x	IPC oder CX (x86, x64, ARM)

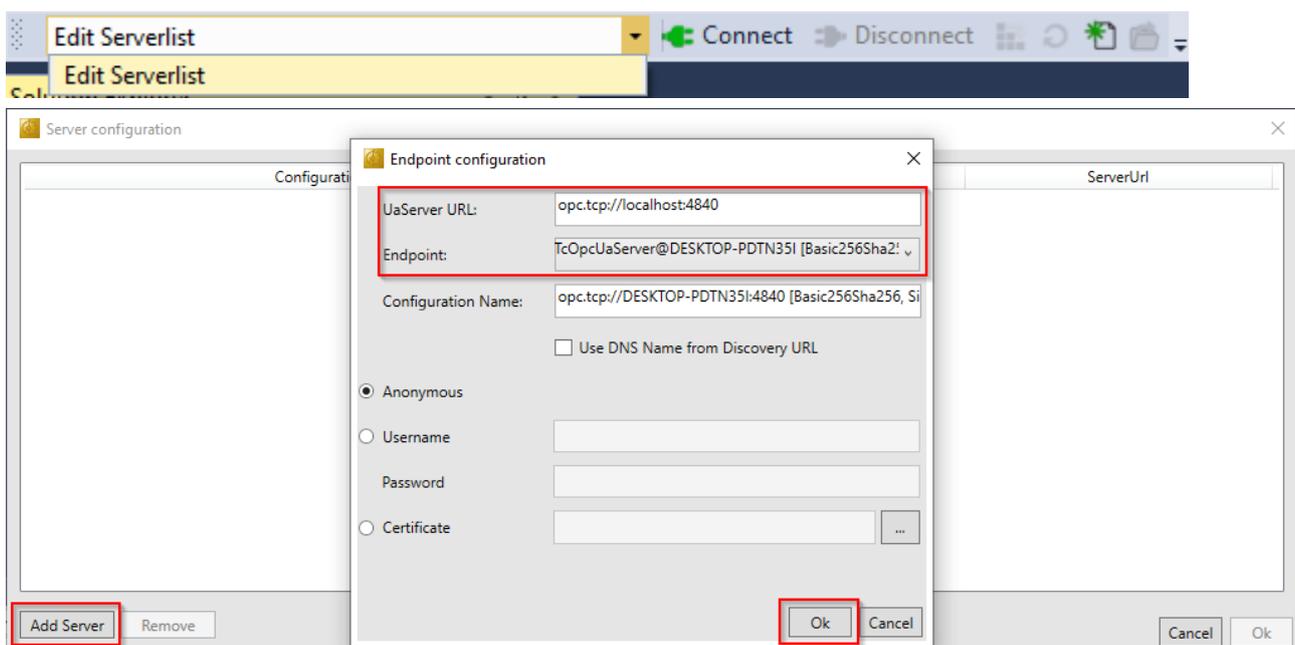
4.2.1.3 Verbinden mit einem Server

Der OPC-UA-Konfigurator ermöglicht die vollständige Parametrierung des Servers über OPC UA. Ähnlich wie im TwinCAT-XAE-System können Sie über die Symbolleiste einen OPC-UA-Server auswählen, mit dem Sie sich verbinden wollen.

Fügen Sie hierfür zunächst die entsprechende Toolbar zu Ihrer Visual Studio Oberfläche hinzu.

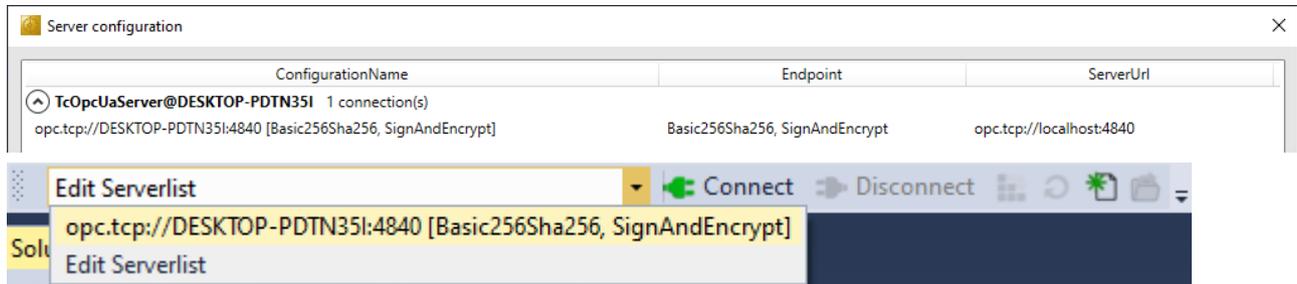


Anschließend können Sie über den Eintrag **Edit Serverlist** in der DropDownBox der Toolbar eine oder mehrere Serververbindungen hinzufügen.



Im Dialog **Endpoint configuration** nehmen Sie hierbei alle Einstellungen für die Verbindung mit dem Server vor, insbesondere die Server URL, die Auswahl eines vom Server angebotenen Endpunkts und optional auch das IdentityToken (z. B. Username/Password), mit dem sich der Konfigurator mit dem Server verbinden soll.

Die Server-Verbindung wird dann unter einem automatisch generierten Konfigurationsnamen zur Serverliste hinzugefügt und ist anschließend in der DropDownListe der Toolbar selektierbar.



Durch einen Klick auf den **Connect**-Button kann nun eine Verbindung mit dem Server hergestellt und dieser konfiguriert werden.

● Online-Konfiguration



Sämtliche Einstellungen die Sie in Ihrem Projekt vornehmen werden für den verbundenen TwinCAT OPC UA Server durchgeführt.

● Initialisierung des Servers



Sollte sich der Server noch im (uninitialisierten) Auslieferungszustand befinden, so erhalten Sie einen entsprechenden Hinweis zur Server-Initialisierung. Dieser Vorgang ist im Artikel zur [Durchführung der Server-Initialisierung \[► 120\]](#) näher beschrieben.

Voraussetzungen

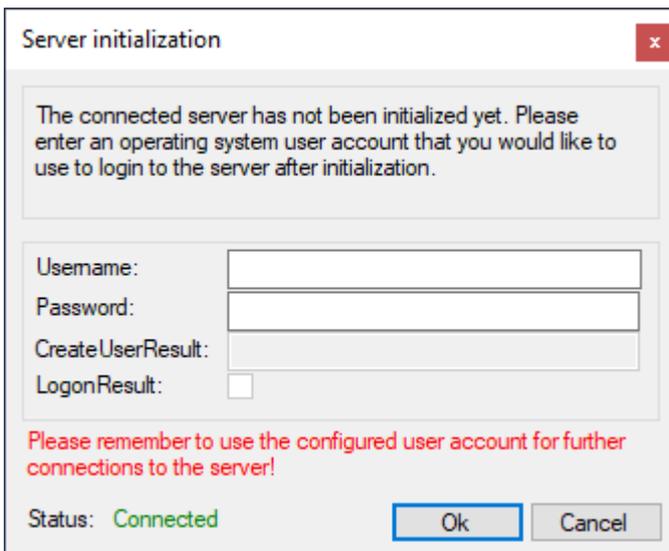
Produkte	Setup-Versionen	Zielplattform
TF6100	4.x.x	IPC oder CX (x86, x64, ARM)

Sehen Sie dazu auch

📄 [Namespace zur Konfiguration des Servers \[► 111\]](#)

4.2.1.4 Durchführen der Server-Initialisierung

Der TwinCAT OPC UA Server wird in einem uninitialisierten Modus ausgeliefert, welcher auf dem sogenannten TOFU (Trust-On-First-Use) Prinzip begründet ist. Detaillierte Informationen zu diesem Server-Feature und die entsprechenden Hintergrundinformationen finden Sie [hier \[► 37\]](#). Der TwinCAT OPC UA Configurator ermöglicht die Initialisierung des Servers beim ersten Verbindungsaufbau. Ein entsprechender Warnhinweis weist auf den uninitialisierten Server hin und ermöglicht eine entsprechende Initialisierung.



Sehen Sie dazu auch

Initialisierung [▶ 37]

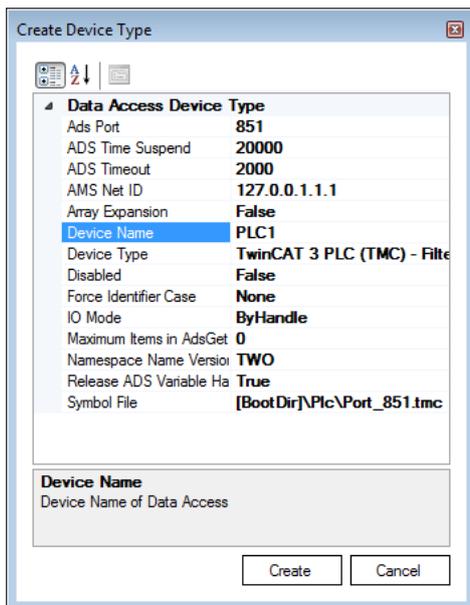
4.2.1.5 ADS-Geräte hinzufügen

Der OPC UA Server kann mit einem oder mehreren ADS-Geräten „sprechen“. Zur Herstellung einer Verbindung ist eine Route zu dem jeweiligen ADS-Gerät erforderlich. Im OPC-UA-Konfigurator werden ADS-Geräte in der Facette **Data Access** angelegt, konfiguriert und somit dem OPC UA Server bekannt gegeben.

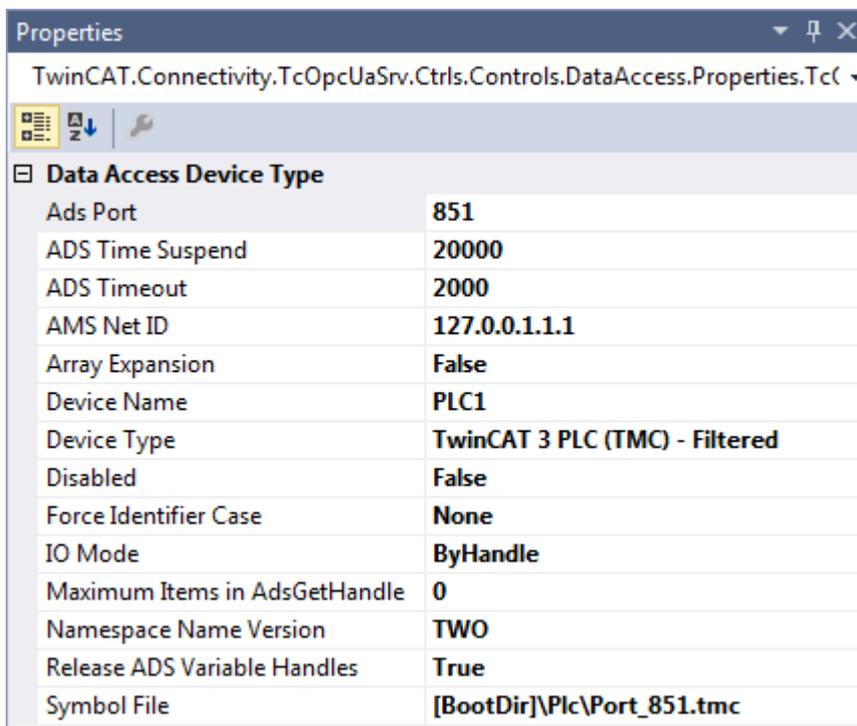
Neue ADS-Geräte fügen Sie der Konfiguration über den Kontextmenübefehl **Add new Device Type** hinzu.



Nach Ausführung des Befehls öffnet sich ein Dialogfenster, in dem Sie die Verbindungsparameter für dieses Gerät konfigurieren können, z. B. AMS Net ID, ADS-Port oder auch die Symboldatei.

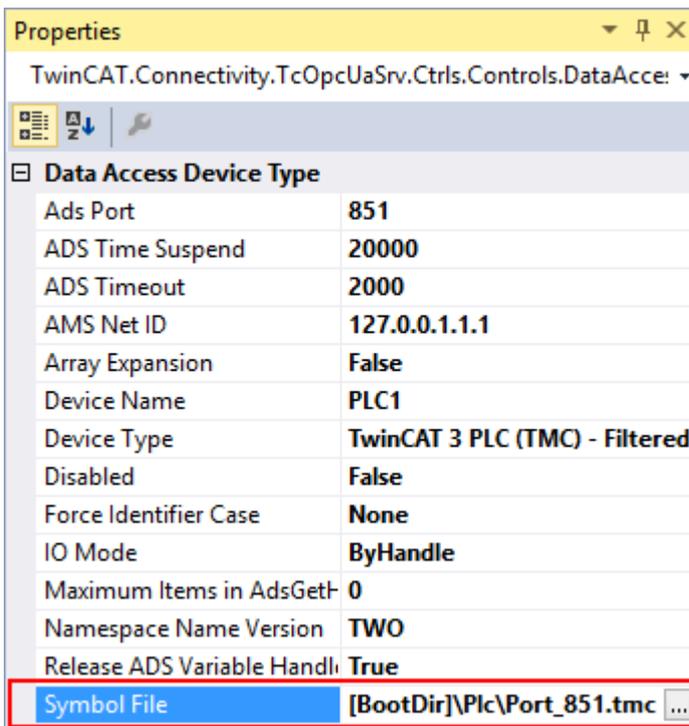


Die Verbindungsparameter können Sie bei Bedarf nachträglich über das Eigenschaftfenster von Visual Studio modifizieren.

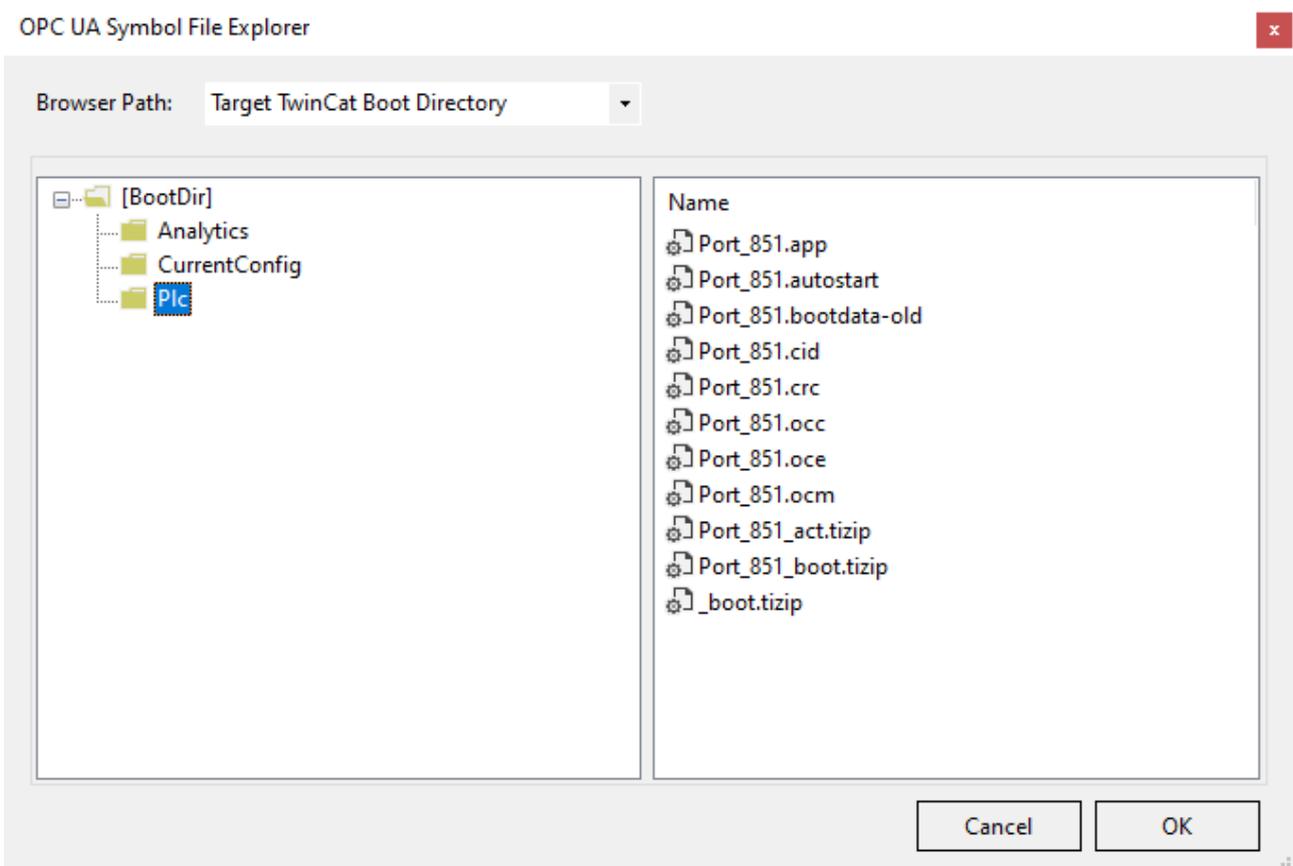


Symboldatei auswählen

Auf dem ausgewählten Zielgerät vorhandene Symboldateien können direkt eingelesen werden. Die Symboldateien können dabei entweder im TwinCAT-Bootverzeichnis oder im Symbolverzeichnis des OPC UA Servers hinterlegt sein. Über den entsprechenden Dialog bei der Symboldatei-Konfiguration können Sie die Dateien selektieren.



Der TwinCAT OPC UA File Explorer kann entweder mit dem lokalen TwinCAT-Verzeichnis oder dem Remote-Bootverzeichnis verbunden werden. Letzteres wird über den Configuration-Namespace des Servers eingelesen (siehe [Namespace zur Konfiguration des Servers](#) [► 111]).



Voraussetzungen

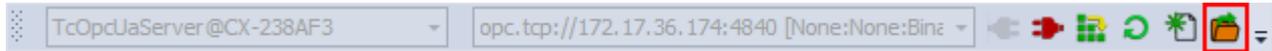
Produkte	Setup-Versionen	Zielplattform
TF6100	4.x.x	IPC oder CX (x86, x64, ARM)

4.2.1.6 Konfiguration lesen und schreiben

Über den Konfigurator können Sie sowohl den Download/Upload von kompletten Server-Konfigurationen anstoßen als auch jede einzelne Facette (Data Access, Historical Access, etc.) einzeln auf ein Zielgerät aufspielen bzw. von dort öffnen. Die hierfür notwendigen Funktionen sind sowohl in die Symbolleiste als auch in das Kontextmenü der jeweiligen Facette eingebunden.

Konfiguration vom Zielgerät öffnen

Über die entsprechende Schaltfläche in der Symbolleiste können Sie die Konfiguration des selektierten Zielgeräts öffnen.



Siehe auch: [Verbinden mit einem Server \[P. 118\]](#)

Konfiguration auf einem Zielgerät aktivieren

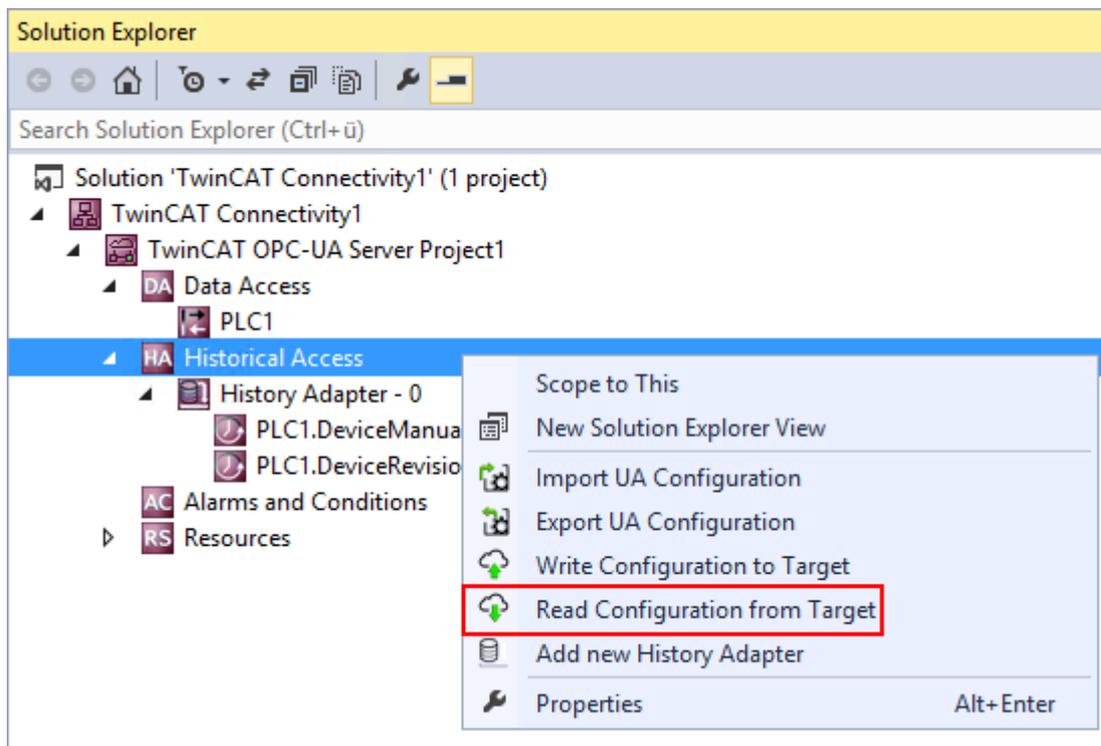
Über die entsprechende Schaltfläche in der Symbolleiste können Sie die aktuell geöffnete Konfiguration auf das selektierte Zielgerät herunterladen.



Siehe auch: [Verbinden mit einem Server \[P. 118\]](#)

Teilkonfiguration öffnen

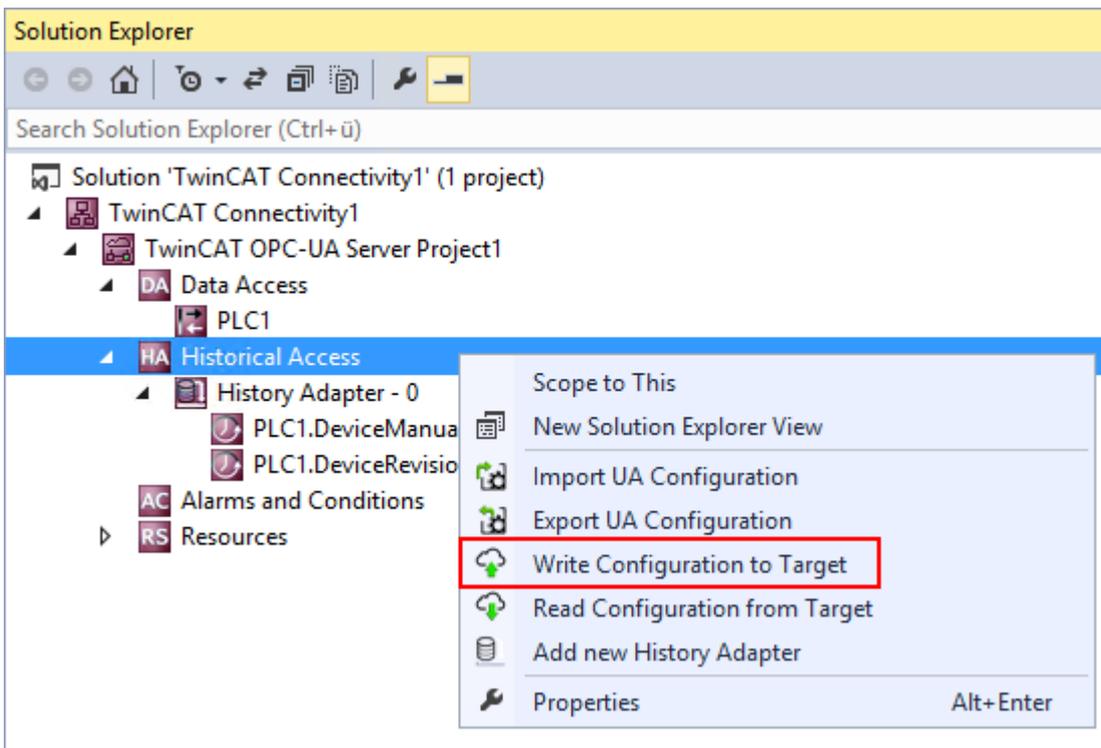
Über den Befehl **Read Configuration from Target** im Kontextmenü einer bestimmten Facette der Konfiguration können Sie die Teilkonfiguration vom selektierten Zielgerät öffnen.



Siehe auch: [Verbinden mit einem Server \[P. 118\]](#)

Teilkonfiguration herunterladen

Über den Befehl **Write Configuration to Target** im Kontextmenü einer bestimmten Facette der Konfiguration können Sie die Teilkonfiguration auf das selektierte Zielgerät herunterladen.



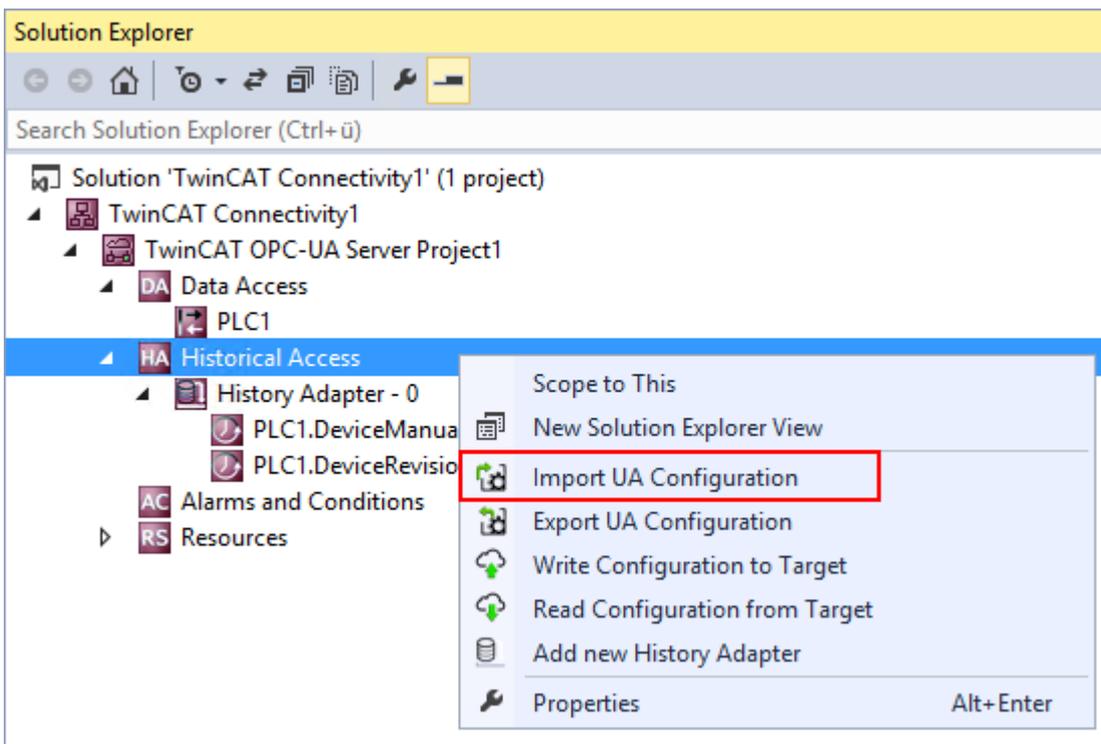
Siehe auch: [Verbinden mit einem Server \[► 118\]](#)

4.2.1.7 Konfigurationsdateien importieren und exportieren

Die Befehle des Kontextmenüs ermöglichen den Import/Export von Konfigurationsdateien des OPC UA Servers.

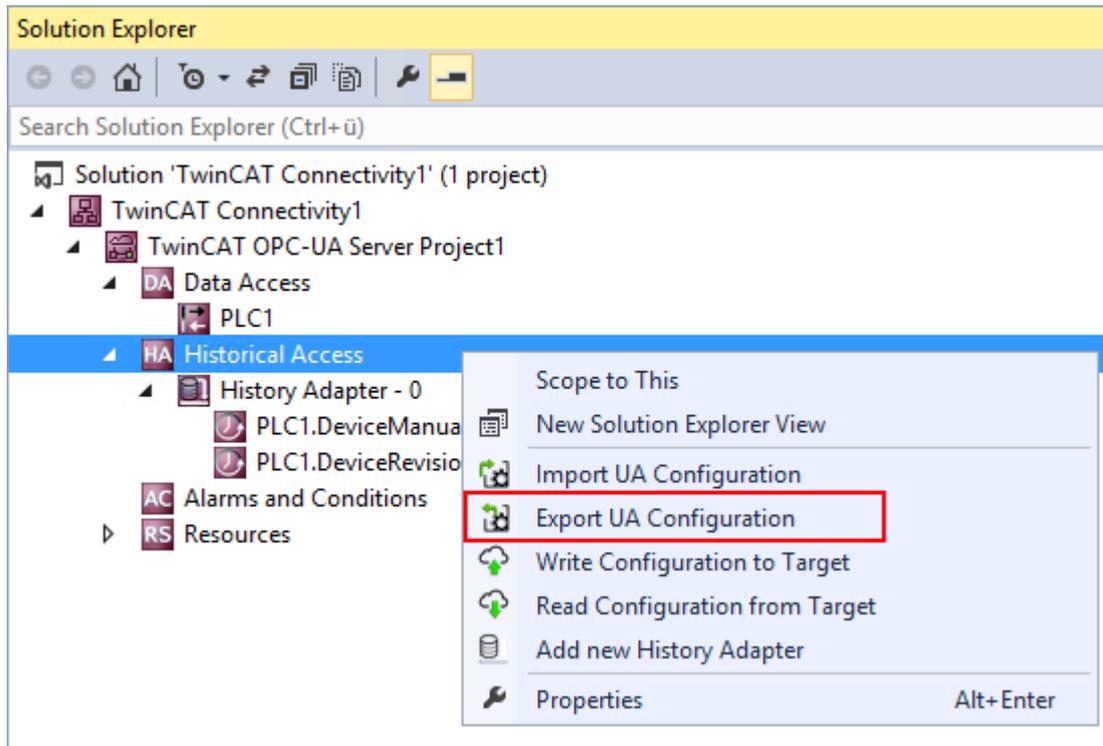
Teilkonfiguration importieren

Über den Befehl **Import UA Configuration** im Kontextmenü einer bestimmten Facette der Konfiguration können Sie die Teilkonfiguration (z. B. Historical Access) aus einer XML-Konfigurationsdatei importieren.



Teilkonfiguration exportieren

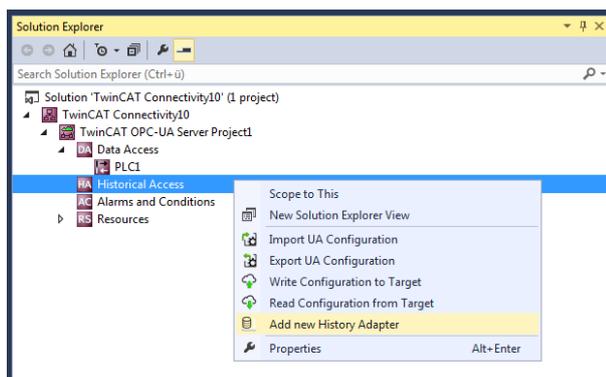
Über den Befehl **Export UA Configuration** im Kontextmenü einer bestimmten Facette der Konfiguration können Sie die Teilkonfiguration (z. B. Historical Access) in eine XML-Konfigurationsdatei exportieren.



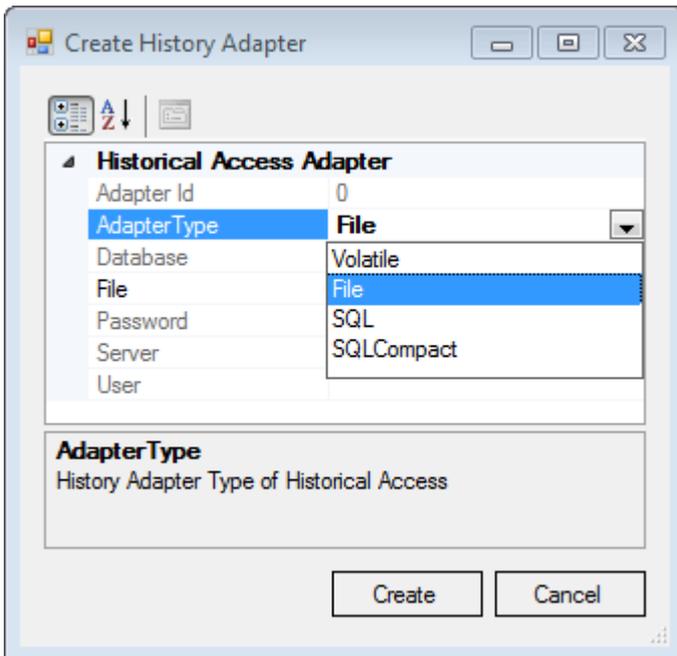
4.2.1.8 Historical Access konfigurieren

Zur Konfiguration von Historical Access müssen zunächst die History Adapter eingerichtet werden. Hierbei handelt es sich um die unterschiedlichen Speicherorte für die historischen Daten, z. B. RAM, Datei, SQL Server.

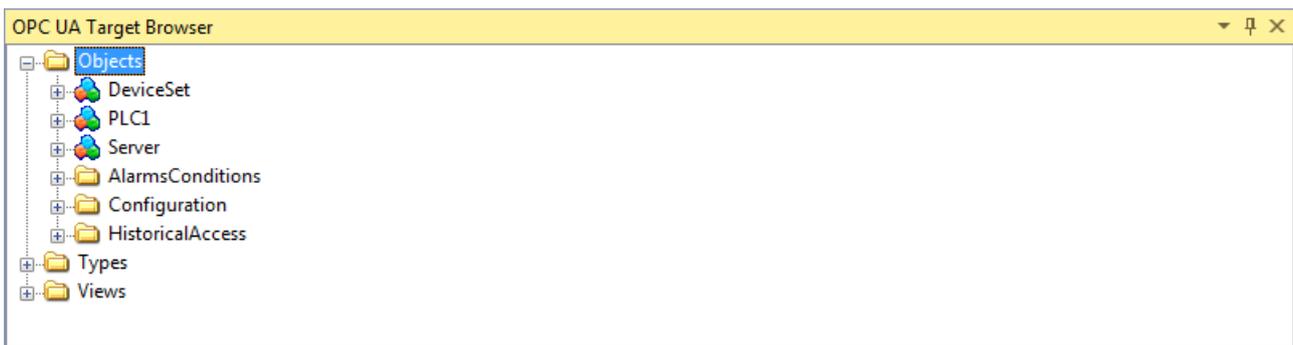
History Adapter fügen Sie der Konfiguration über den Kontextmenübefehl **Add new History Adapter** hinzu.



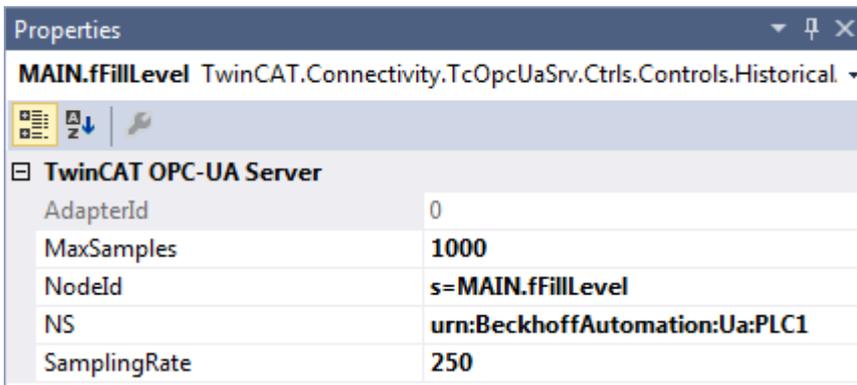
Je nach Adaptertyp müssen Sie weitere Parameter spezifizieren, z. B. den gewünschten Pfad zur Dateiablage oder die Zugangsdaten zum SQL Server.



Nachdem Sie einen History Adapter angelegt haben, können Sie dem Adapter die gewünschten Variablen hinzufügen. Die Variablen müssen zum Zeitpunkt des Engineerings bereits auf dem selektierten OPC UA Server vorliegen. Zur Selektion der Variablen können Sie den integrierten **OPC UA Target Browser** verwenden und die Variablen aus dem Target Browser per Drag-and-drop zum History Adapter hinzufügen.



Im Eigenschaftfenster der neu hinzugefügten Variable können Sie weitere Parameter spezifizieren, z. B. die gewünschte SamplingRate oder die Größe des zu verwendenden Ringbuffers im History Adapter.

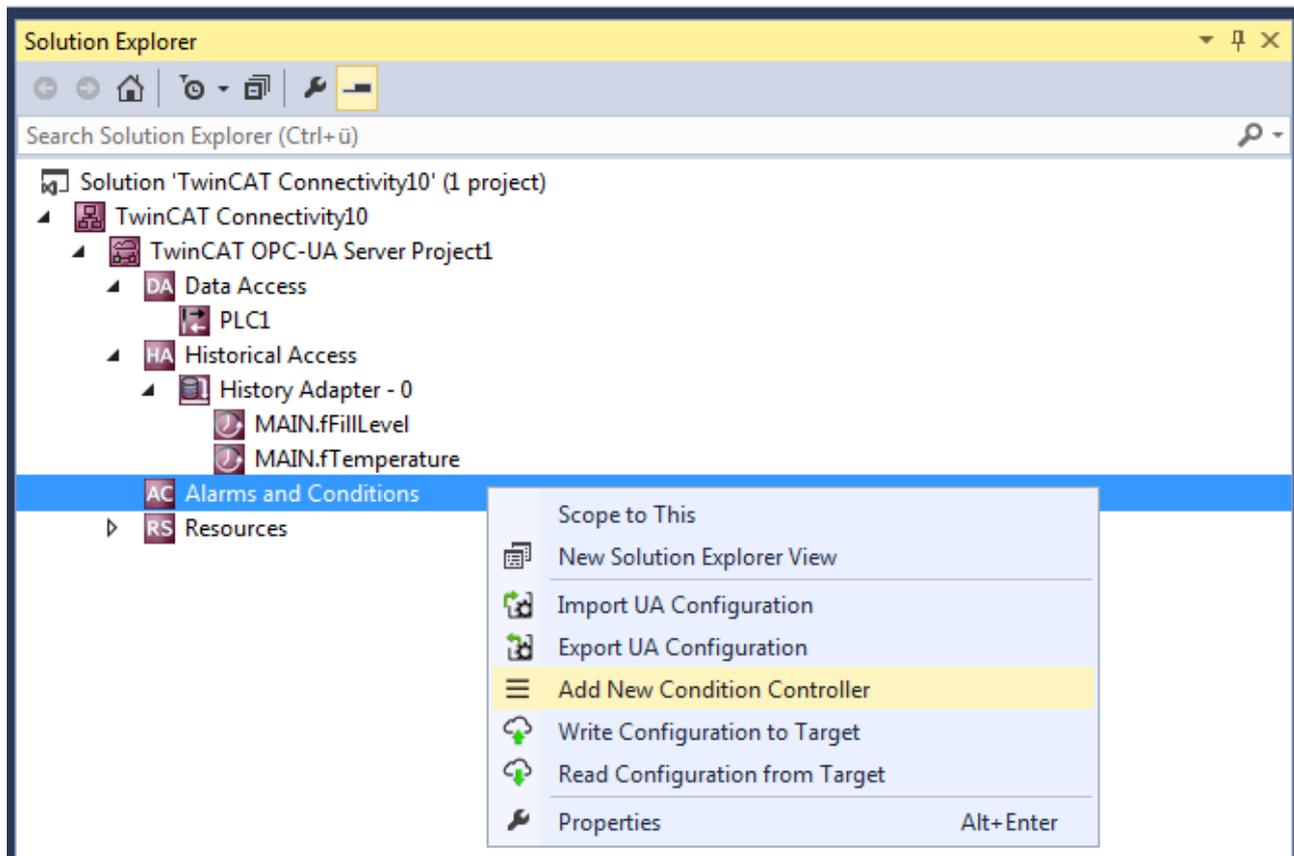


Siehe auch: [Verbinden mit einem Server \[► 118\]](#)

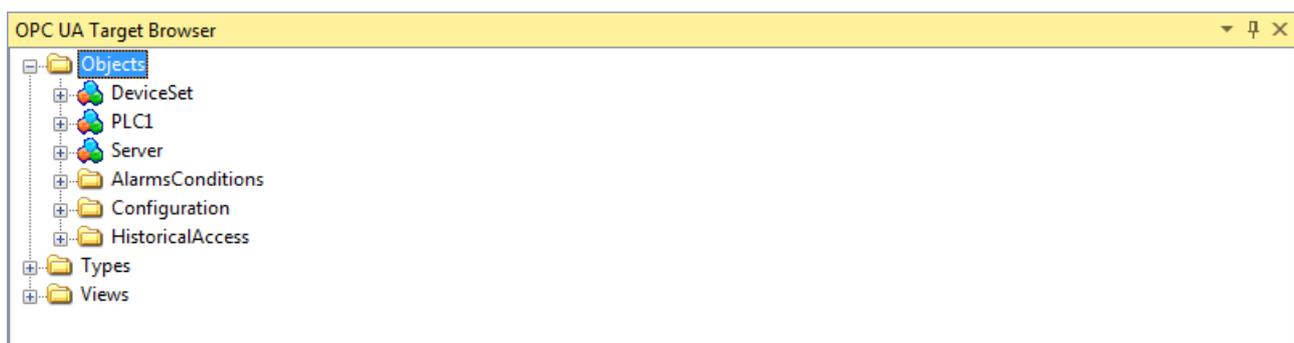
4.2.1.9 Alarms and Conditions konfigurieren

Zur Konfiguration von Alarms and Conditions (A&C) müssen zunächst die Condition Controller eingerichtet werden. Hierbei handelt es sich um Container-Einheiten, die Alarme gruppieren.

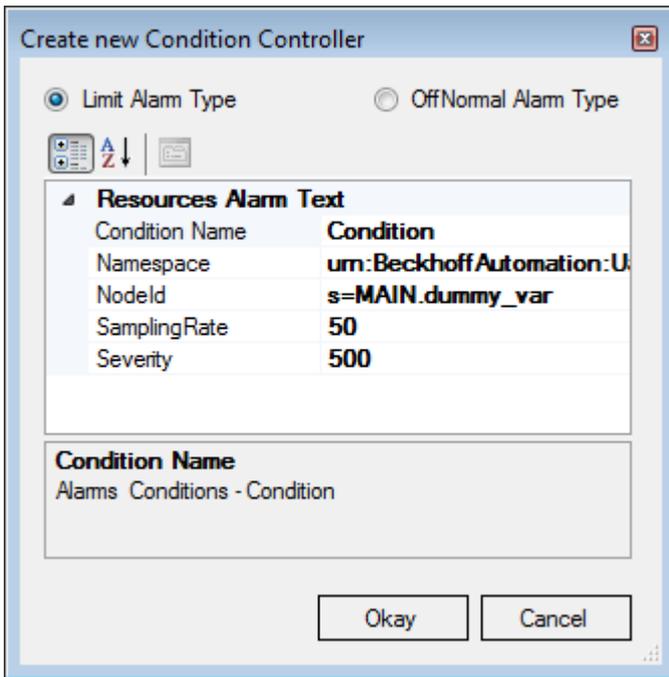
Condition Controller fügen Sie der Konfiguration über den Kontextmenübefehl **Add New Condition Controller** hinzu.



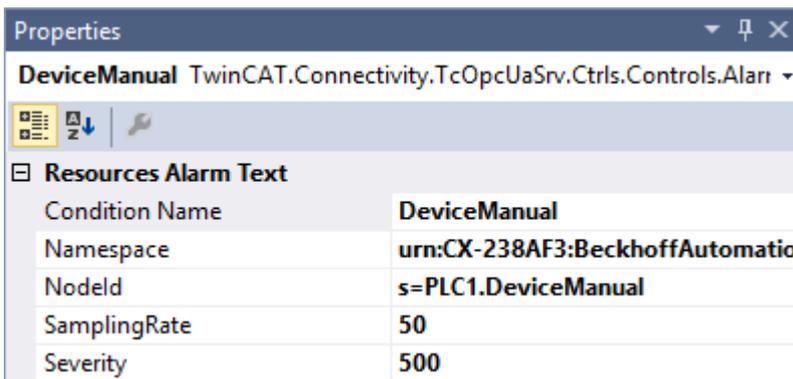
Nachdem Sie einen Condition Controller angelegt haben, können Sie dem Controller die gewünschten Variablen hinzufügen und im Sinne des Alarms and Conditions überwachen. Für jede Variable wird hierbei eine Condition angelegt, welche die Parameter für die Überwachung spezifiziert. Die Variablen müssen zum Zeitpunkt des Engineerings bereits auf dem selektierten OPC UA Server vorliegen. Zur Selektion der Variablen können Sie den integrierten **OPC UA Target Browser** verwenden und die Variablen aus dem Target Browser per Drag-and-drop zum Condition Controller hinzufügen.



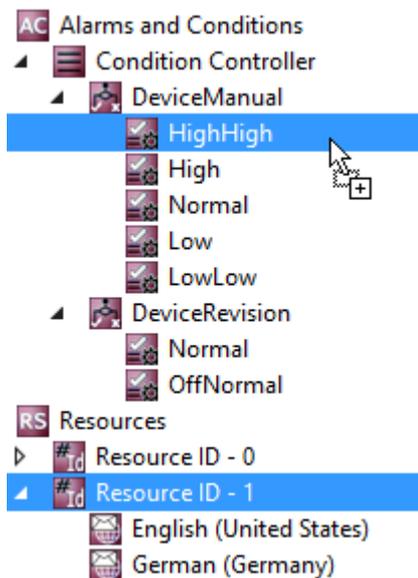
In dem sich öffnenden Dialogfenster können Sie den Condition-Typ und weitere Parameter für die Überwachung definieren, z. B. SamplingRate und Severity.



Je nach ausgewähltem Condition-Typ können Sie zusätzliche Parameter im Eigenschaftfenster der Condition spezifizieren. Die Schwellenwerte für den jeweiligen Condition-Typ werden als einzelne Einträge in der Baumansicht der Konfiguration angezeigt. Auch hier können Sie die entsprechenden Parameter im Eigenschaftfenster konfigurieren.



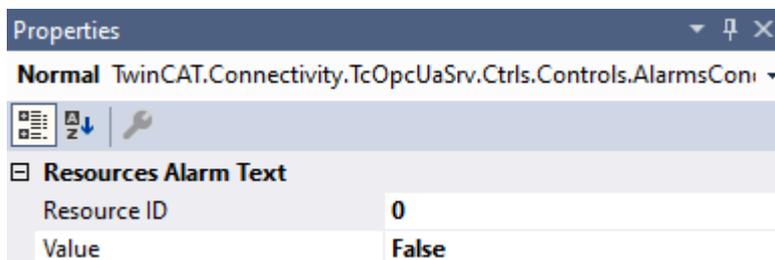
Anschließend müssen Sie die Alarmtexte definieren, die beim Triggern einer Condition an den OPC UA Client versendet werden sollen. Wie Sie Alarmtexte anlegen, wird im Abschnitt [Alarmtexte konfigurieren](#) [131] beschrieben. Die Alarmtexte können Sie per Drag-and-drop auf den jeweiligen Schwellenwert einer Condition ziehen.



Alarmtyp OffNormal

Bei einem Alarmtyp OffNormal definieren Sie, welcher Zustand einer Boolean-Variablen als Normal gewertet wird. Wenn der Variablenwert hiervon abweicht, wird ein Alarm ausgelöst. Für das Arbeiten mit Wertebereichen (bspw. Integer- oder Double-Variablen) muss die SPS verwendet werden. Dort wird dann je nach Wert ein entsprechender TRUE oder FALSE-Zustand an den OPC UA-Server weitergegeben.

Zustand	Wertebereich
Normal	TRUE oder FALSE, je nach Entscheidung des Benutzers.
OffNormal	TRUE oder FALSE, je nach Konfiguration des Normal-Zustands. Kann nicht durch den Benutzer konfiguriert werden.

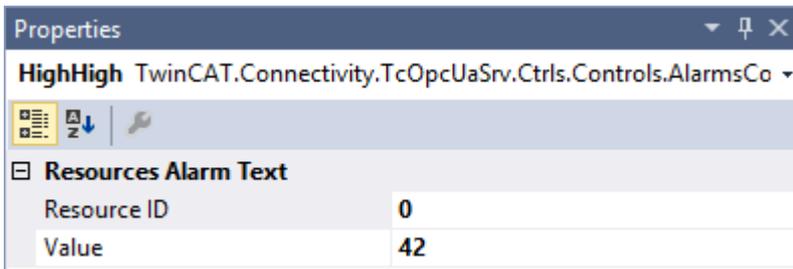


Im ersten Schritt wird wie oben beschrieben der Normalzustand konfiguriert. Anschließend definiert der Benutzer für den jeweiligen Zustand (OffNormal und Normal) einen Alarm-Text über die Resources. Das kann entweder per Drag-and-drop oder per Auswahl in der Dropdown-Liste **Resource ID** geschehen.

Alarmtyp Limit

Bei einem Alarmtyp Limit definieren Sie unterschiedliche Schwellenwerte, bei deren Erreichen ein Alarm verschickt werden soll. Die folgende Tabelle beschreibt die verschiedenen Schwellenwerte anhand einer Beispielkonfiguration.

Zustand	Beispiel-Schwellenwerte	Zugehöriger Wertebereich (INT)
HighHigh	5000	5000-32767
High	2000	2000-4999
Normal	-	1000-1999
Low	1000	500-999
LowLow	500	-32768-499



Im ersten Schritt werden wie oben beschrieben die verschiedenen Schwellwerte konfiguriert. Anschließend definiert der Benutzer für den jeweiligen Zustand (HighHigh, High, Normal, Low, LowLow) einen Alarm-Text über die Resources. Das kann entweder per Drag-and-drop oder per Auswahl in der Dropdown-Liste **Resource ID** geschehen.

Voraussetzungen

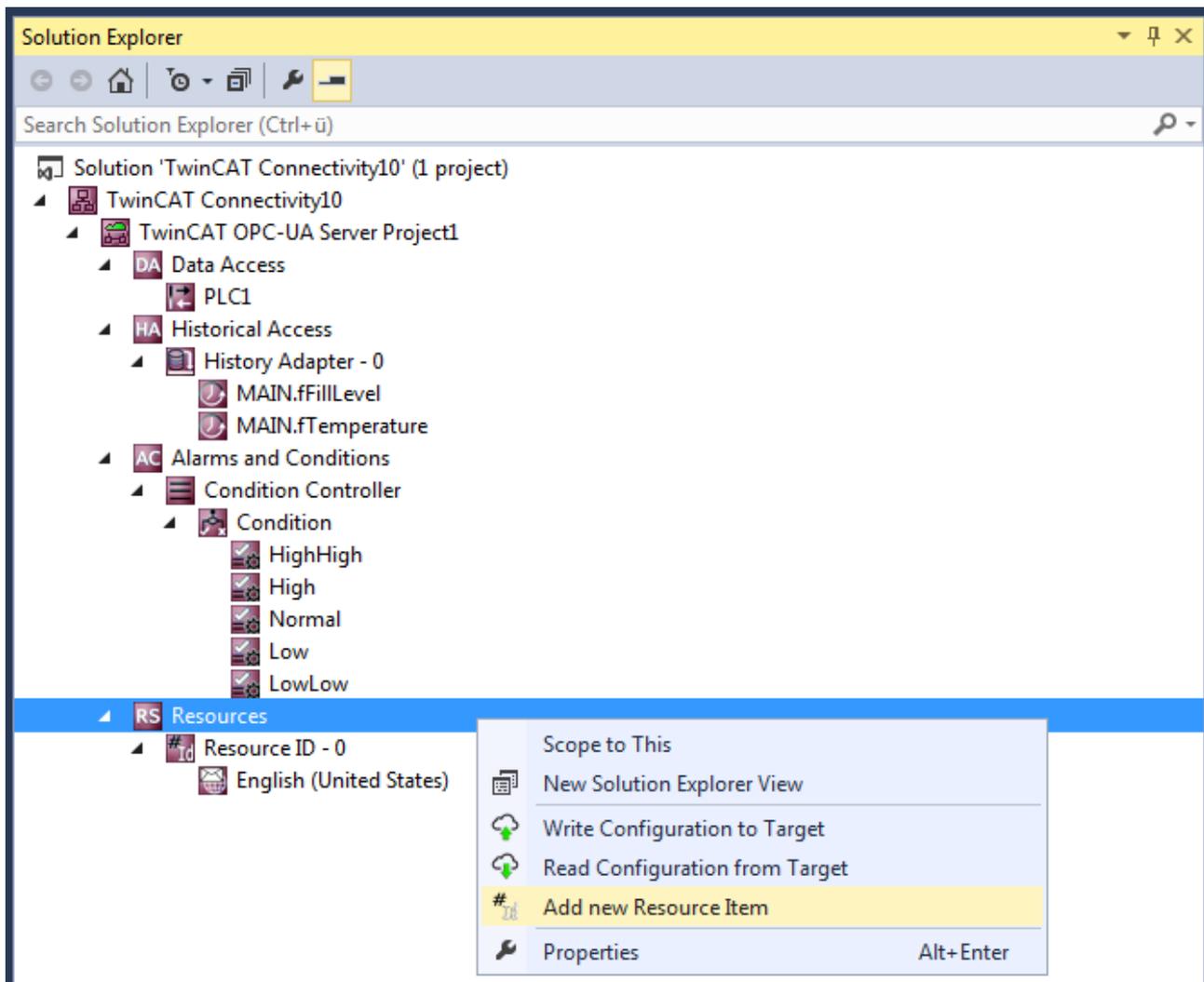
Produkte	Setup-Versionen	Zielplattform
TF6100	4.x.x	IPC oder CX (x86, x64, ARM)

Siehe auch: [Verbinden mit einem Server \[▶ 118\]](#)

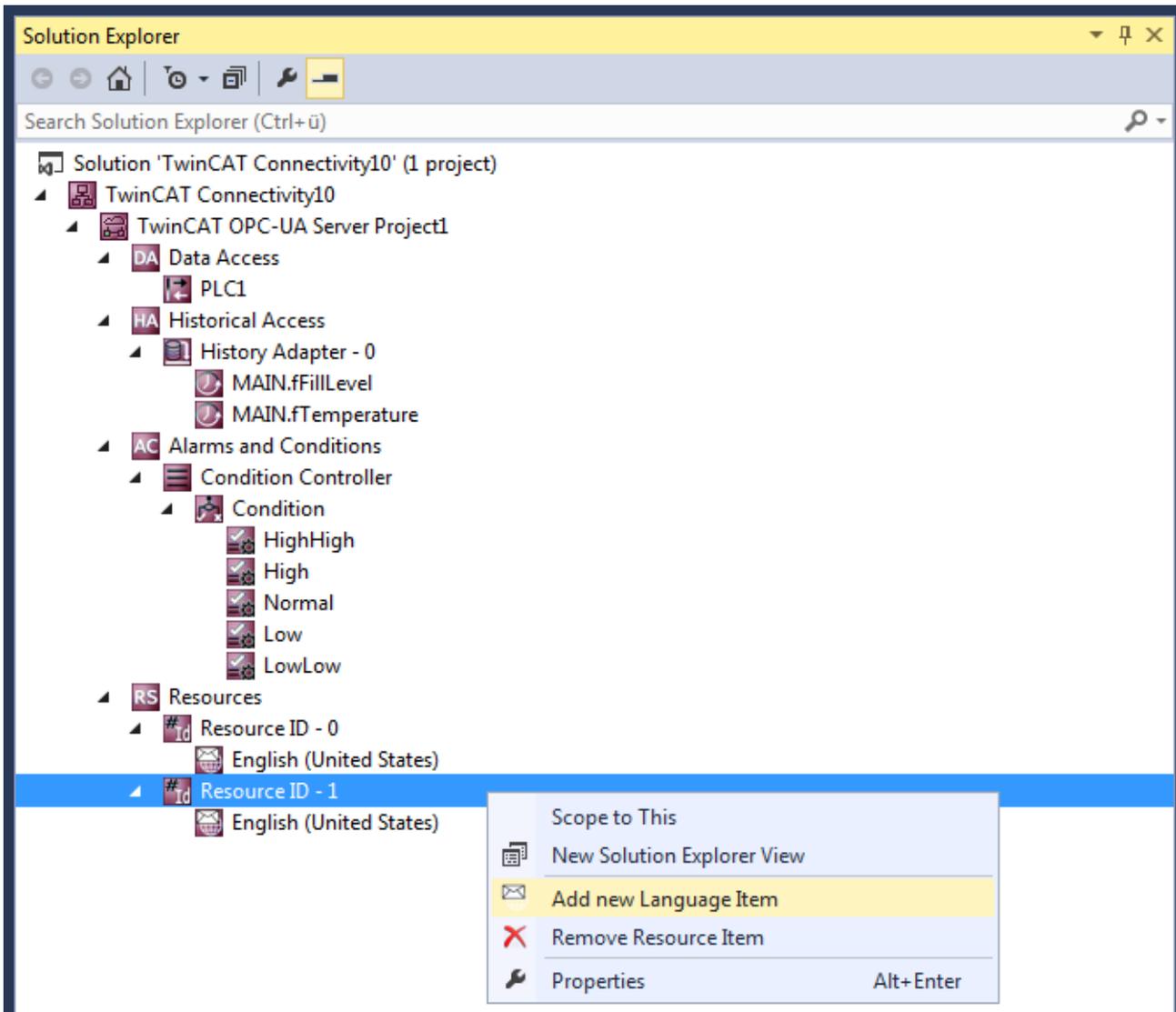
4.2.1.10 Alarmtexte konfigurieren

Der OPC-UA-Konfigurator ermöglicht die (mehrsprachige) Verwaltung von Alarmtexten, die zum Beispiel beim [Alarms and Conditions \[▶ 127\]](#) verwendet werden. Die Konfiguration der Alarmtexte erfolgt in der Facette **Resources**. Jeder Alarmtext wird durch eine eindeutige ID identifiziert. Dieser ID können dann mehrere Sprachtexte zugeordnet werden.

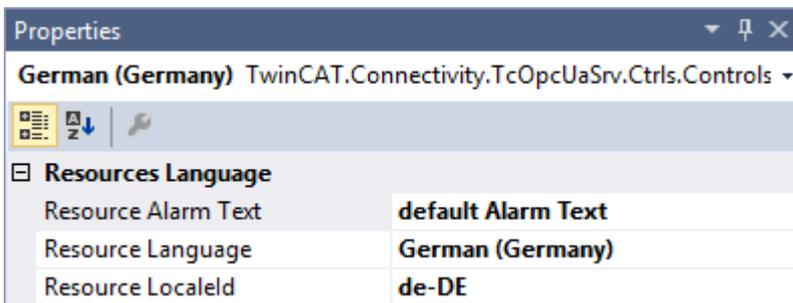
Über den Befehl **Add new Resource Item** im Kontextmenü können Sie sogenannte „Resource Items“ anlegen.



Über den Befehl **Add new Language Item** im Kontextmenü eines „Resource Items“ fügen Sie diesem neue Sprachelemente („Language Items“) hinzu.



Im Eigenschaftsfenster können Sie ein Sprachelement weiter parametrieren, z. B. den Sprachtext und die zugeordnete Sprache. Wenn Sie die Sprache festlegen, wird automatisch die zugehörige LocaleID gesetzt. Die LocaleID wird vom OPC UA Client angefordert, um anzugeben, in welcher Sprache er Alarmtexte erwartet.



Voraussetzungen

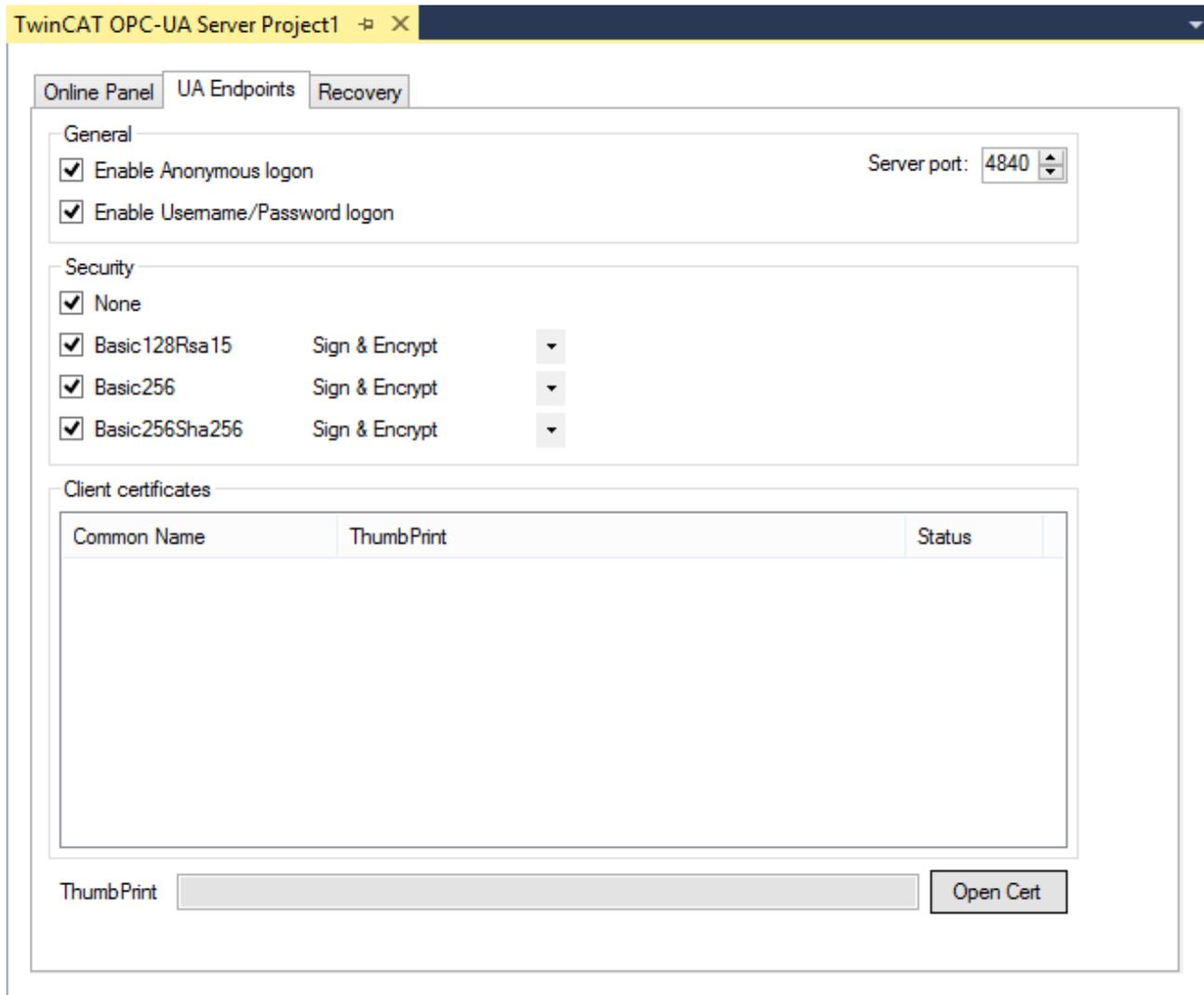
Produkte	Setup-Versionen	Zielplattform
TF6100	4.x.x	IPC oder CX (x86, x64, ARM)

4.2.1.11 Endpunkte konfigurieren

Die Endpunkte des OPC UA Servers geben an, welche Security-Mechanismen bei der Verbindungsherstellung eines Clients benutzt werden sollen. Diese reichen von „unverschlüsselt“ bis zu „verschlüsselt und signiert“, basierend auf verschiedenen Schlüsselstärken.

Die Endpunkte können Sie über den Konfigurator aktivieren und deaktivieren. Es kann z. B. sinnvoll sein, den unverschlüsselten Endpunkt zu deaktivieren, damit sich alle Clients nur mit gültigem und als vertrauenswürdig eingestuftem Zertifikat verbinden können.

Sie konfigurieren die Endpunkte direkt auf Ebene des OPC-UA-Server-Projekts. Durch einen Doppelklick auf das Projekt können Sie in der Registerkarte **UA Endpoints** die entsprechenden Einstellungen vornehmen. Die Einstellungen werden nach einem Aktivieren der Konfiguration und einem anschließenden Neustart des Servers wirksam (siehe [Konfiguration lesen und schreiben \[► 124\]](#) und [Server neu starten \[► 143\]](#)).



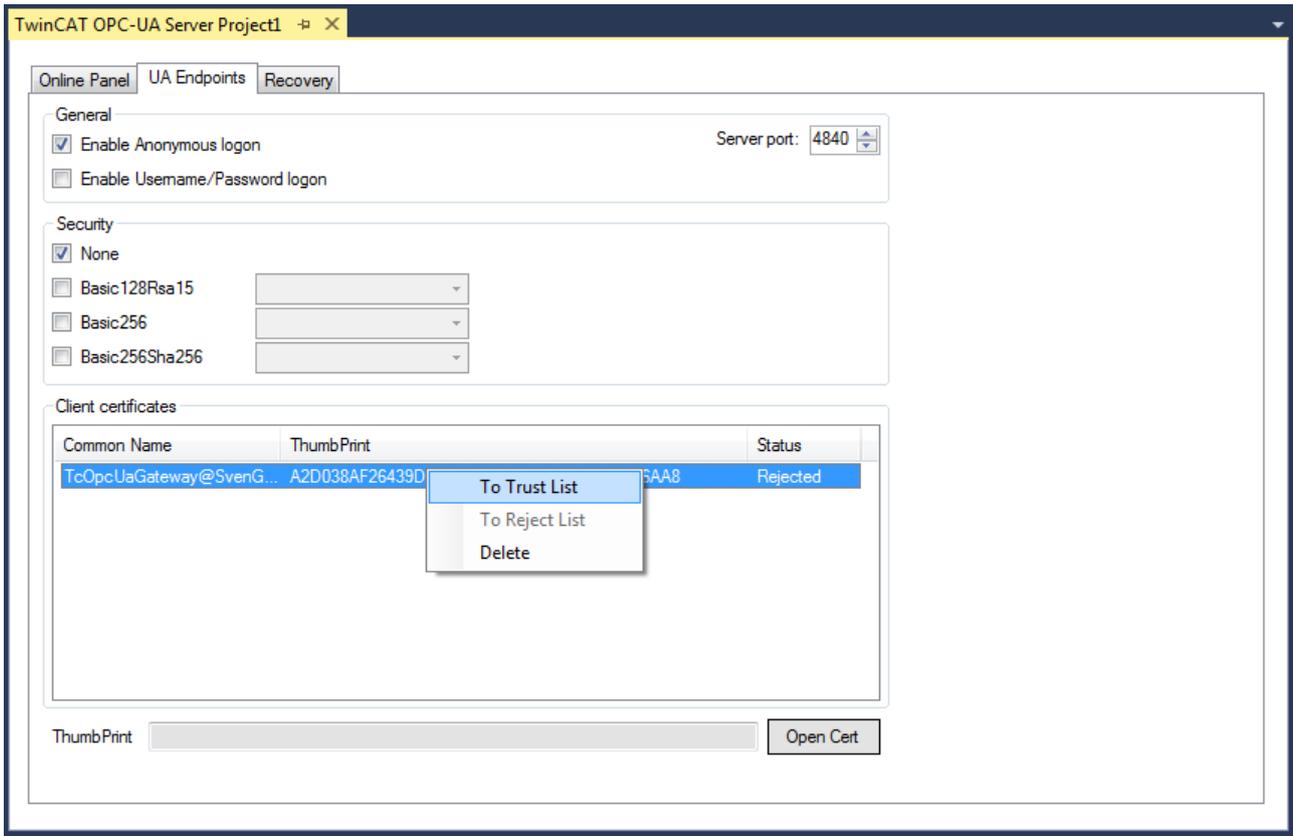
Voraussetzungen

Produkte	Setup-Versionen	Zielplattform
TF6100	4.x.x	IPC oder CX (x86, x64, ARM)

4.2.1.12 Vertrauensstellung für Zertifikate

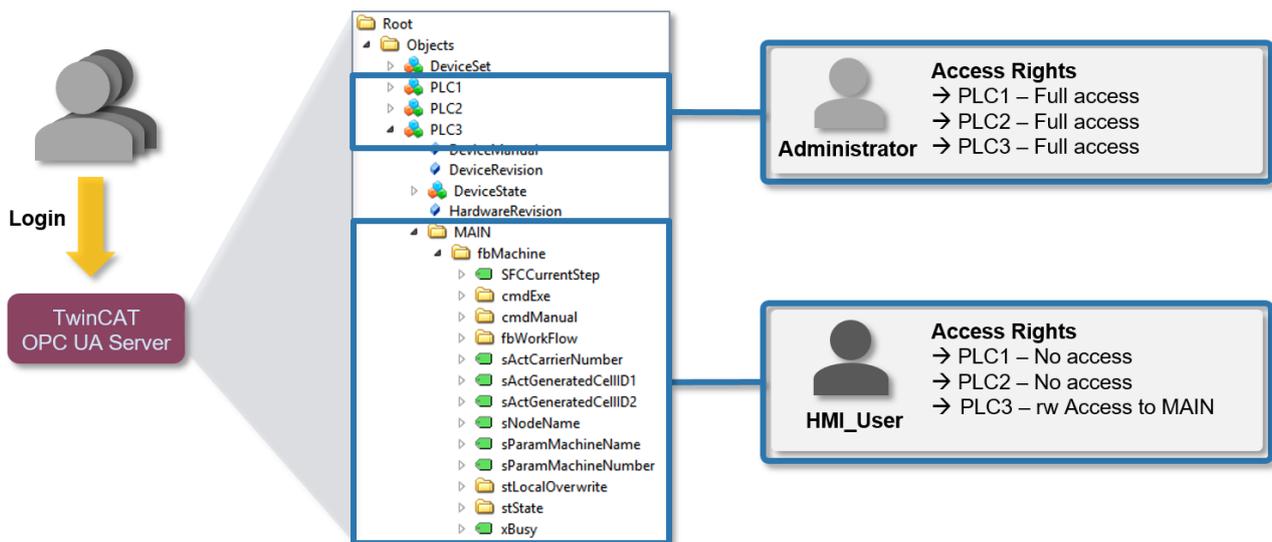
Die auf dem Server vorhandenen Client-Zertifikate können über den Konfigurator komfortabel verwaltet werden. In den Projekteinstellungen können Sie die Zertifikate in der Registerkarte **UA Endpoints** im Bereich **Client certificates** als vertrauenswürdig einstufen oder verweigern.

Nachdem ein OPC UA Client zum ersten Mal versucht hat, sich mit einem sicheren Endpunkt des Servers zu verbinden, wird das Client-Zertifikat auf dem Server hinterlegt und als „rejected“ deklariert. Anschließend kann der Server-Administrator das Zertifikat freischalten. Ein anschließender Verbindungsversuch des Clients mit einem gesicherten Endpunkt wird dann erfolgreich sein.



4.2.1.13 Sicherheitseinstellungen konfigurieren

Der OPC UA Server ermöglicht die Konfiguration von Berechtigungen auf Namespace- und Node-Ebene. Hierdurch kann sowohl der Zugriff auf ADS-Geräte (z. B. auf verschiedene SPS-Laufzeiten) als auch Variablen feingranular eingestellt werden. Diese Sicherheitseinstellungen sind für alle ADS-Geräte verfügbar, die im Server-Namespace dargestellt werden können.



Konfiguration

Die Konfiguration der Berechtigungen erfolgt auf Basis einer XML-basierten Konfigurationsdatei (*TcUaSecurityConfig.xml*), die in demselben Verzeichnis wie der Server liegt. Die Konfigurationsdatei besteht aus den drei Bereichen „Benutzer (Users)“, „Gruppen (Groups)“ und „AccessInfos“.

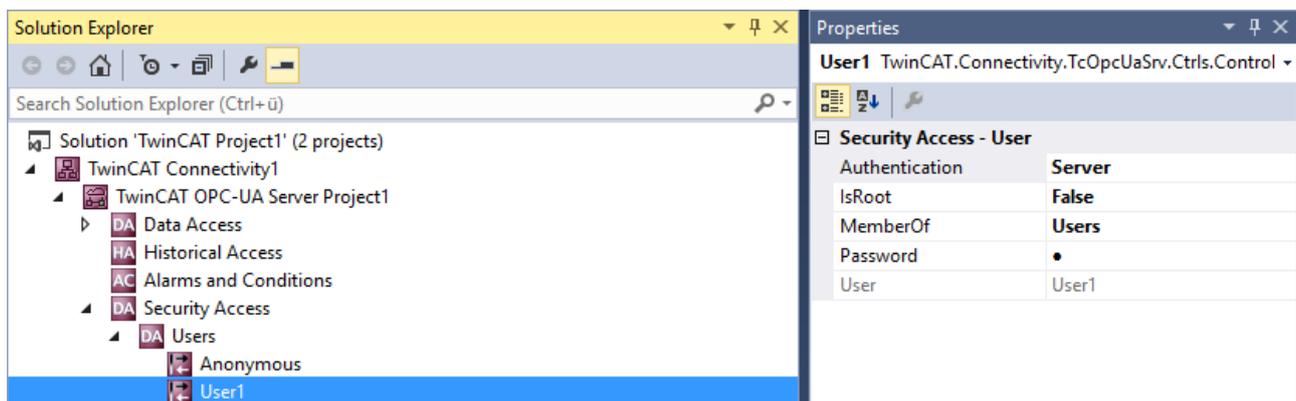
Benutzer (Users)

Im Bereich „Users“ können Sie Benutzerkonten konfigurieren, die vom OPC UA Server als Login akzeptiert werden sollen. Hierbei gibt es drei verschiedene Authentifizierungsmethoden:

OS (empfohlene Authentifizierungsmethode)	Es werden die Mechanismen vom Betriebssystem verwendet, um Benutzername und Passwort zu validieren. Das Benutzerkonto obliegt vollständig der Kontrolle des Betriebssystems und/oder der Domäne.
Server (nicht empfohlen)	Benutzername und Passwort sind nur dem OPC UA Server bekannt. Beide Informationen werden hierbei im Klartext in der XML-Datei hinterlegt.
None	Es wird nur der Benutzername vom Server ausgewertet, das Passwort wird ignoriert.

Benutzer können mit einem Tag <DefaultAccess> konfiguriert werden, das den Standardzugriff des Benutzers auf einen bestimmten Namespace angibt.

Benutzer können Mitglied einer oder mehrerer Gruppen sein. Dies können Sie über das Attribut **MemberOf** spezifizieren. Bei Mitgliedschaft in mehreren Gruppen trennen Sie die Gruppen durch ein Semikolon.

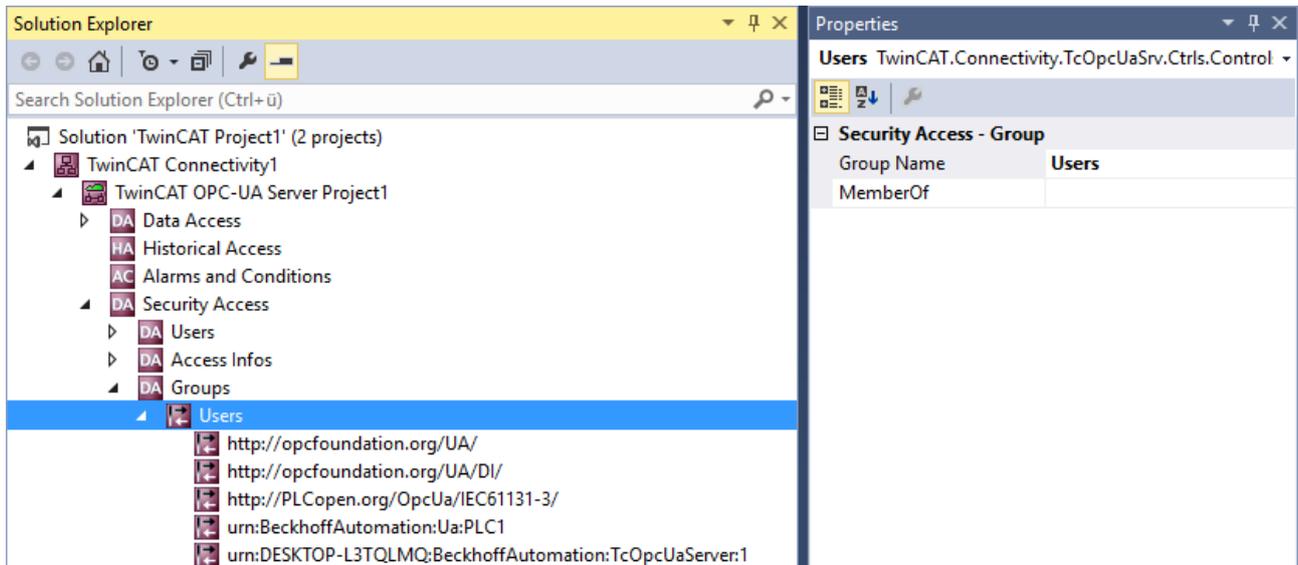


Gruppen (Groups)

Um eine einfachere Konfiguration mit mehreren Benutzeraccounts zu ermöglichen, können Sie die Benutzer in Gruppen zusammenfassen.

Gruppen können ebenfalls mit einem Tag <DefaultAccess> konfiguriert werden.

Über das Attribut **MemberOf** können Sie Gruppen verschachteln. Bei Mitgliedschaft in mehreren Gruppen trennen Sie die Gruppen durch ein Semikolon.

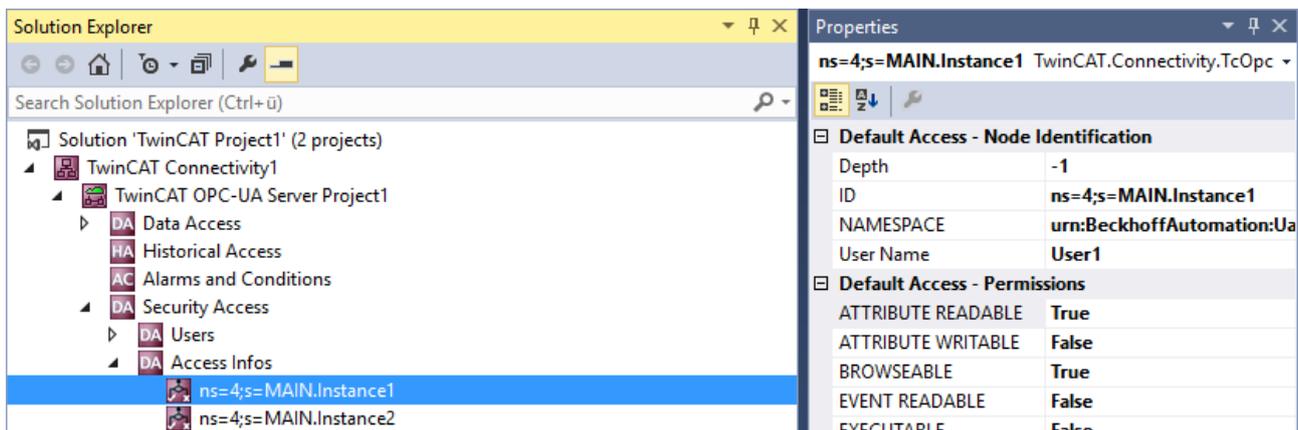


AccessInfos

Wenn eine feingranulare Einstellung von Berechtigungen auf Node-Ebene erfolgen soll, können Sie zusätzlich AccessInfos konfigurieren, welche die Zugriffsberechtigungen auf Nodes spezifizieren. Zugriffsrechte können hierbei auf Unterelemente vererbt werden. Obwohl AccessInfos die feingranularste Konfigurationsmöglichkeit von Berechtigungen ermöglichen, wird eine solche Konfiguration auch schnell unübersichtlich. Prüfen Sie daher, ob eine Konfiguration von Zugriffsrechten auf Namespace-Ebene (siehe oben) nicht ausreichend ist.

Die AccessInfo für eine Node beinhaltet die folgenden Einstellungen:

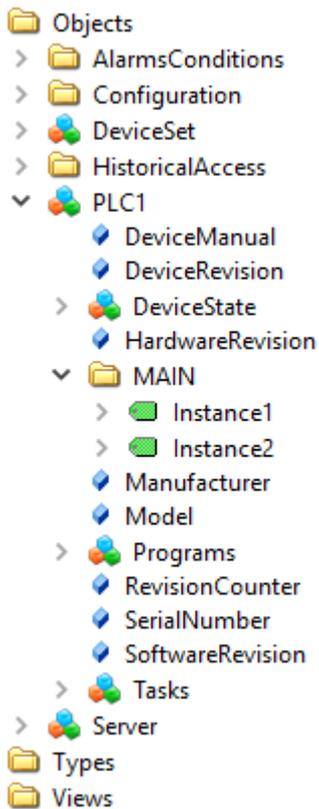
NS	Konfiguriert den NamespaceName, in dem die Node lokalisiert ist
Id	Konfiguriert den Identifier der Node, inklusive des IdentifierTypes (z. B. s = String)
Depth	Vererbungstiefe der Berechtigungen (-1 für unendlich)
User/Group	Benutzer oder Gruppe, der/die auf diese Node Zugriff erhalten soll, inklusive des AccessLevels



AccessInfos können per Drag-and-drop von Variablen aus dem Target Browser konfiguriert werden. Die konfigurierbaren Berechtigungen sind kumulativ.

Beispielkonfiguration

Gegeben sei folgendes einfaches Steuerungsprogramm. Die Variablen sind bereits im OPC-UA-Namespace des Servers veröffentlicht. Der OPC UA Server befindet sich zunächst im Auslieferungszustand.



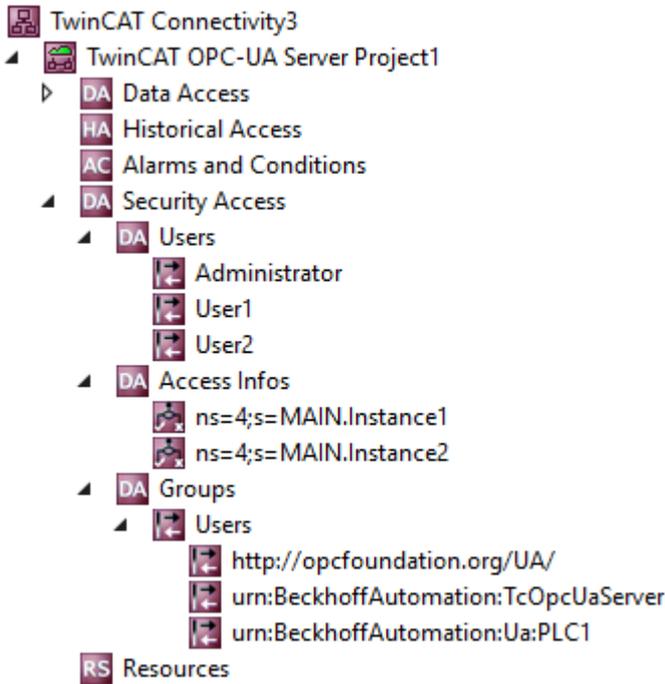
Zugriffbeschränkungen

Der Zugriff auf den Server soll für Clients wie folgt eingeschränkt werden:

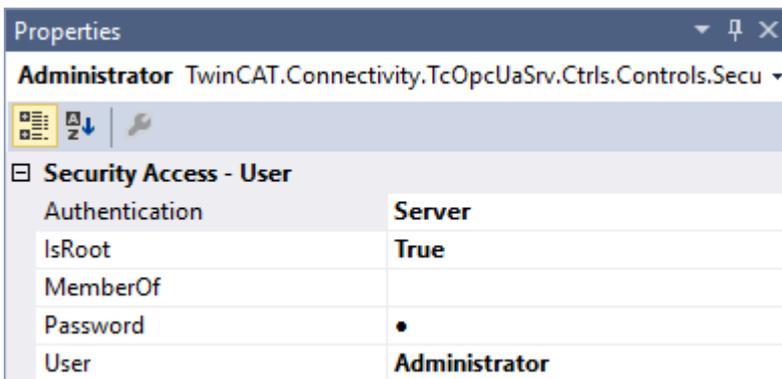
- Anonymous Zugriff soll deaktiviert werden.
- Es soll einen Benutzer „Administrator“ geben, der Vollzugriff auf den kompletten Server hat.
- Es soll einen Benutzer „User1“ geben, der ausschließlich Lesezugriff auf MAIN.Instance1 hat. Der Benutzer soll hierbei nicht aus dem Betriebssystem kommen, sondern nur intern im Server verwendet werden.
- Es soll einen Benutzer „User2“ geben, der ausschließlich Lesezugriff auf MAIN.Instance2 hat. Der Benutzer soll hierbei nicht aus dem Betriebssystem kommen, sondern nur intern im Server verwendet werden.
- Über eine Gruppe "Users" sollen allgemeine Zugriffsberechtigungen für alle Benutzer konfiguriert werden.

Einstellungen

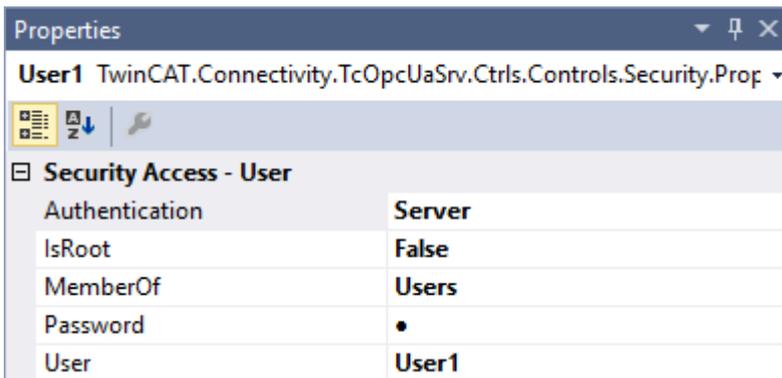
Die Konfiguration des OPC UA Server wird wie folgt eingestellt:



Einstellungen für den Benutzer „Administrator“:



Einstellungen für den Benutzer „User1“:



Einstellungen für den Benutzer „User2“:

Properties

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

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

Einstellungen Access Infos „MAIN.Instance1“:

Properties

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

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

Einstellungen Access Infos „MAIN.Instance2“:

The screenshot shows the 'Properties' dialog box for the object 'ns=4;s=MAIN.Instance2'. It is divided into two sections: 'Default Access - Node Identification' and 'Default Access - Permissions'.

Default Access - Node Identification	
Depth	-1
ID	ns=4;s=MAIN.Instance2
NAMESPACE	urn:BeckhoffAutomation:Ua:PLC1
User Name	User2

Default Access - Permissions	
ATTRIBUTE READABLE	True
ATTRIBUTE WRITABLE	False
BROWSEABLE	True
EVENT READABLE	False
EXECUTABLE	False
HISTORY DELETE	False
HISTORY INSERT	False
HISTORY MODIFY	False
HISTORY READABLE	False
PERMISSION ALL	False
READABLE	True
WRITABLE	False

Einstellungen für die Gruppe „Users“:

Die Benutzergruppe wird sowohl mit grundlegendem Zugriff auf benötigte Server- und Typsystem-Namensräume ausgestattet als auch mit Read- und Browse-Berechtigungen auf den PLC1-Namensraum.

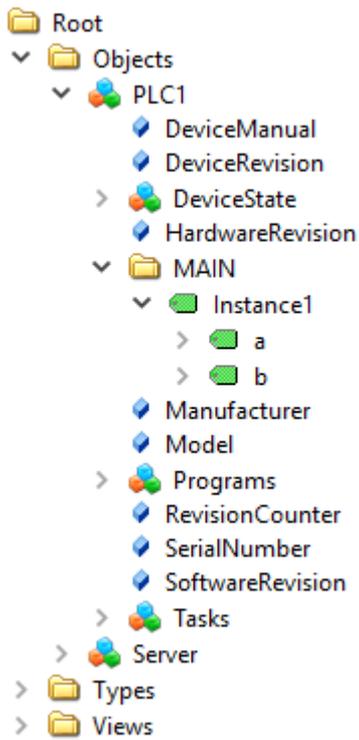
The screenshot shows the 'Properties' dialog box for the object 'urn:BeckhoffAutomation:Ua:PLC1'. It is divided into two sections: 'Default Access - Namespace' and 'Default Access - Permissions'.

Default Access - Namespace	
NAMESPACE	urn:BeckhoffAutomation:Ua:PLC1

Default Access - Permissions	
ATTRIBUTE READABLE	True
ATTRIBUTE WRITABLE	False
BROWSEABLE	True
EVENT READABLE	False
EXECUTABLE	False
HISTORY DELETE	False
HISTORY INSERT	False
HISTORY MODIFY	False
HISTORY READABLE	False
PERMISSION ALL	False
READABLE	False
WRITABLE	False

Ergebnis

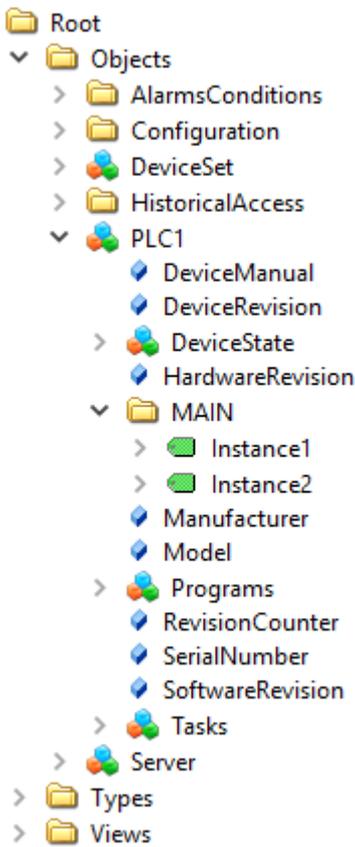
Nach Aktivierung der Konfiguration stellt sich der Namensraum des Servers für „User1“ nach einem Verbindungsaufbau wie folgt dar:



Auf die Node „Instance1“ hat der Benutzer nur Leserechte, was durch das Attribut `UserAccessLevel` deutlich wird:

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

Der Benutzer „Administrator“ hingegen hat volle Zugriffsrechte auf alle Elemente des Namensraums:



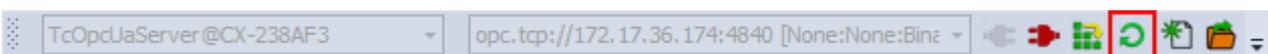
4.2.1.14 Server neu starten

Der OPC-UA-Konfigurator ermöglicht das Triggern eines Neustarts des OPC UA Server. Dies kann lokal oder remote erfolgen und bezieht sich auf das jeweils selektierte Zielgerät.

● Verbindungsverlust

i Ein Neustart des OPC UA Server führt immer zu einem Verbindungsverlust aller verbundenen Clients.

Den Neustart triggern Sie über die Symbolleiste.



Voraussetzungen

Produkte	Setup-Versionen	Zielplattform
TF6100	4.x.x	IPC oder CX (x86, x64, ARM)

Siehe auch: [Verbinden mit einem Server \[▶ 118\]](#)

4.2.1.15 Logging

Für eine erweiterte Diagnose können Sie die Logging-Funktion des OPC UA Servers aktivieren.

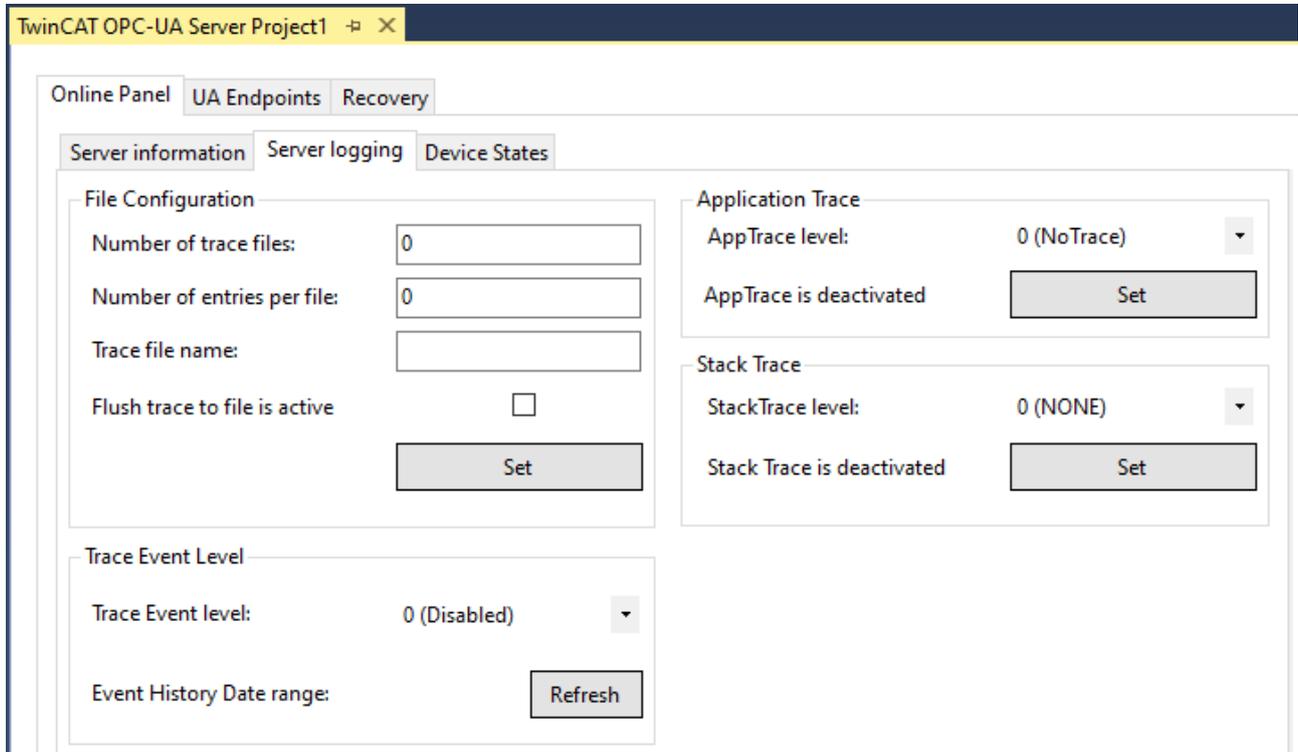
● Schreiben der Log-Datei

i Durch das Aktivieren der Logging-Funktion auf dem Server wird eine Protokolldatei auf dem Dateisystem geschrieben. Stellen Sie sicher, dass ausreichend Speicherplatz zur Verfügung steht und setzen Sie die Logging-Parameter entsprechend (Anzahl Log-Dateien, Größe pro Log-Datei).

i Performance- und Timingverhalten

Durch das Aktivieren der Protokollfunktionen verändert sich das Timing-Verhalten des OPC UA Servers. Hierdurch können je nach Plattform und Projekt Geschwindigkeitseinbußen entstehen.

Sie aktivieren die Logging-Funktion im Konfigurator des Projekts in der Registerkarte **Online Panel** über die Schaltfläche **Activate**. Sie können die Funktion je nach selektiertem Zielgerät lokal oder remote aktivieren. Die Logging-Funktion ist so lange aktiv, bis sie über den Konfigurator wieder deaktiviert oder bis der OPC UA Server neu gestartet wird.



Trace-Level

Generell gilt: Je höher der „Trace level“, desto detailliertere (und mehr) Daten werden geschrieben, desto mehr Last wird jedoch auch auf der Serverapplikation verursacht, wodurch sich das Timingverhalten entsprechend ändert. Bitte aktivieren Sie daher das Logging nur im Diagnosefall und in Absprache mit dem Beckhoff Support.

Activate App Trace

In den meisten Fällen ist es ausreichend ein sogenanntes „AppTrace“ zu erstellen. Hierbei werden Informationen der Serverapplikation protokolliert. Zum Aktivieren des AppTrace tragen Sie bitte die Anzahl an TraceFiles, sowie die Anzahl Einträge pro TraceFile in die zugehörigen Textfelder ein. Anschliessend wählen Sie einen Tracelevel aus und klicken auf den Button zum Aktivieren des AppTrace. Die Werte in den grau hinterlegten Textfeldern stellen die aktuellen Einstellungen auf dem Server dar.

Activate Stack Trace

In einigen wenigen Fällen ist es zusätzlich notwendig ein sogenanntes „StackTrace“ zu erstellen, wodurch Informationen vom OPC UA Stack protokolliert werden. Zum Aktivieren des StackTrace tragen Sie bitte die Anzahl an TraceFiles, sowie die Anzahl Einträge pro TraceFile in die zugehörigen Textfelder ein. Anschliessend wählen Sie einen Tracelevel aus und klicken auf den Button zum Aktivieren des StackTrace. Die Werte in den grau hinterlegten Textfeldern stellen die aktuellen Einstellungen auf dem Server dar.

Voraussetzungen

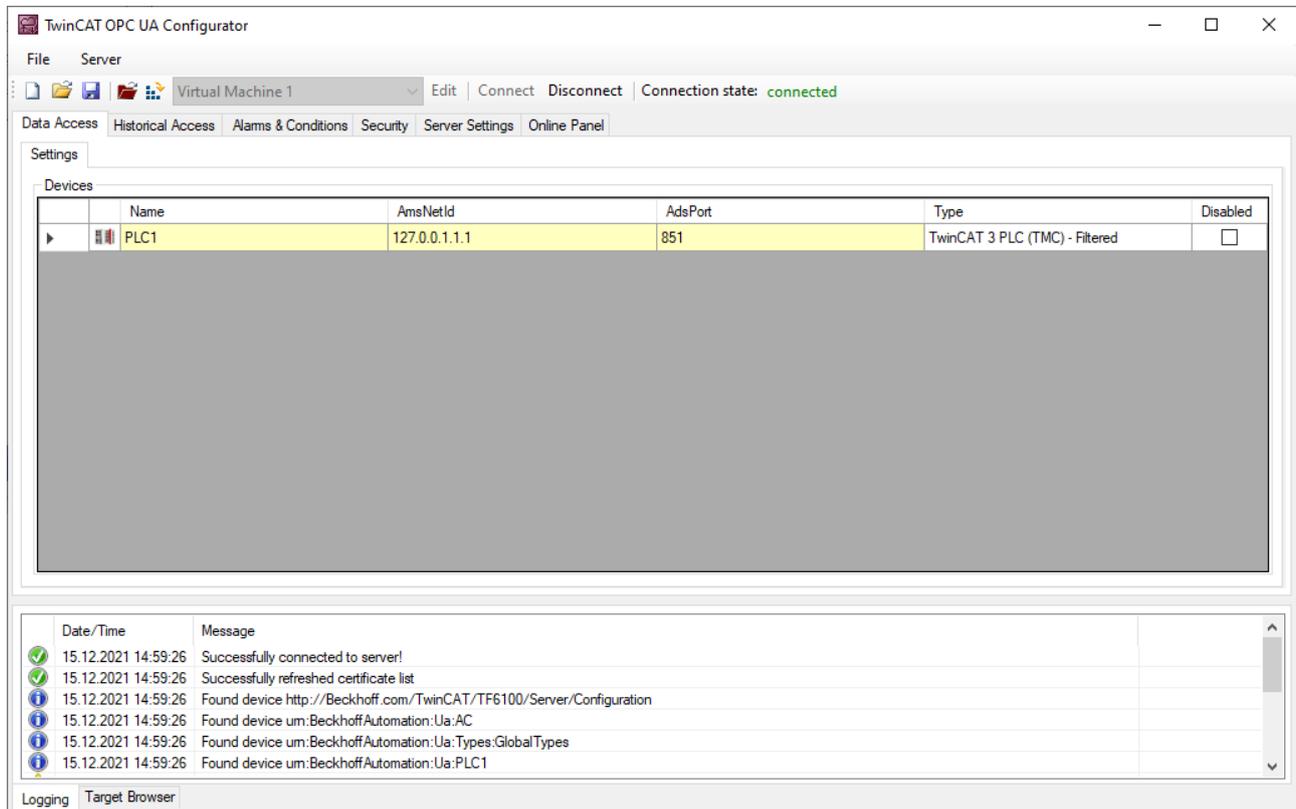
Produkte	Setup-Versionen	Zielplattform
TF6100	4.x.x	IPC oder CX (x86, x64, ARM)

Siehe auch: [Auswahl eines Zielgeräts \[► 118\]](#)

4.2.2 Standalone

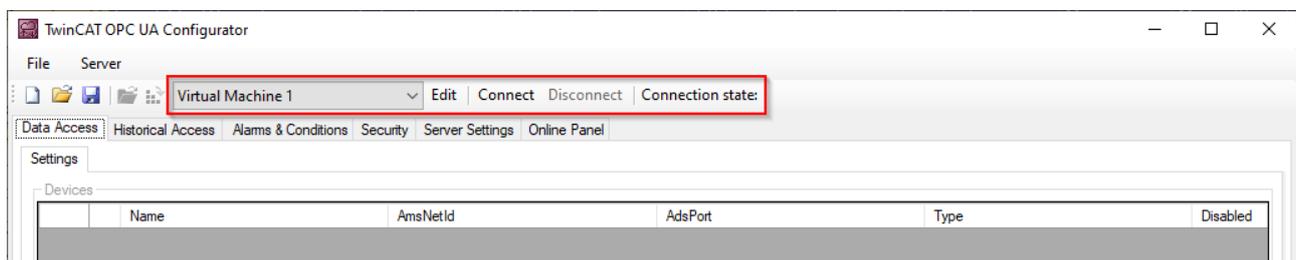
4.2.2.1 Übersicht

Der Standalone-Konfigurator ermöglicht eine Parametrisierung des TwinCAT OPC UA Servers unabhängig vom Visual Studio. Sie können alle unterschiedlichen Features des Servers konfigurieren.

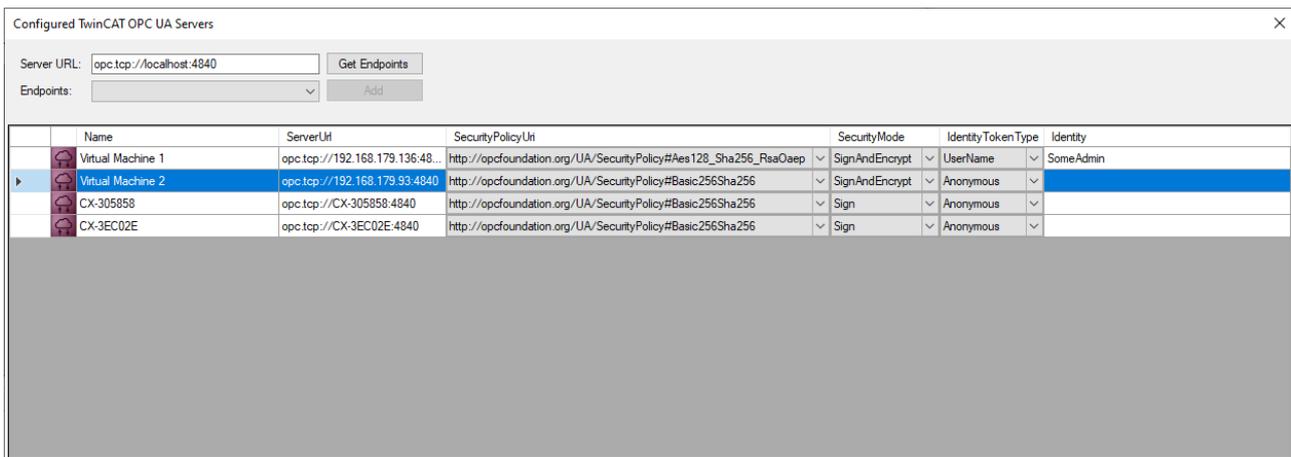


4.2.2.2 Verbinden mit einem Server

Der OPC-UA-Konfigurator ermöglicht die vollständige Parametrierung des Servers über OPC UA. Ähnlich wie im TwinCAT-XAE-System können Sie über die Symbolleiste einen OPC-UA-Server auswählen, mit dem Sie sich verbinden wollen.



Durch einen Klick auf den **Edit**-Button öffnen Sie den Serverlisten-Dialog. In diesem Dialog können Sie eine oder mehrere Server-Verbindungen hinzufügen.



Durch Eingabe einer ServerURL und Aufruf des **Get Endpoints**-Buttons kann eine Server-Verbindung zur Liste hinzugefügt werden. Etwaige Einstellungen zum IdentityToken, z. B., ob sich der Konfigurator als Anonymous-Benutzer oder mit einer Benutzername/Password-Kombination verbinden soll, müssen manuell eingestellt werden.

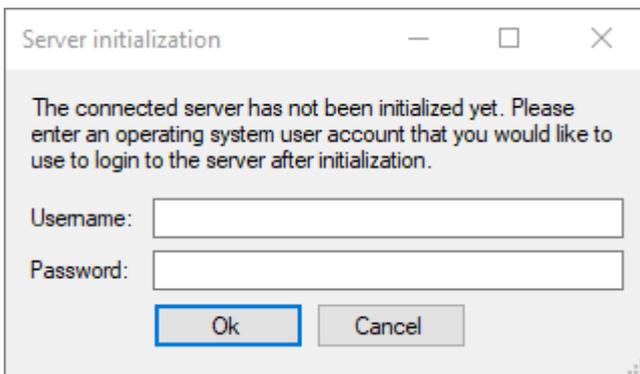
● Bestätigen einer Konfiguration

i Bitte bestätigen Sie Änderungen an den Einträgen immer mit der **ENTER**-Taste, da sie nur dann im Hintergrund automatisch gespeichert werden.

Nach der Konfiguration einer Server-Verbindung ist der entsprechende Eintrag in der DropDownBox verfügbar und die Verbindung kann durch Klicken des **Connect**-Buttons hergestellt werden.

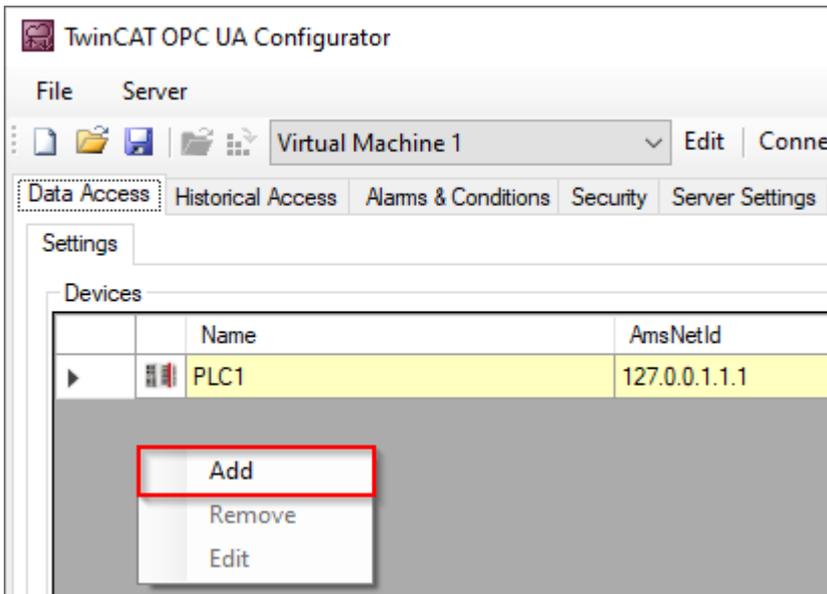
4.2.2.3 Durchführen der Server-Initialisierung

Der TwinCAT OPC UA Server wird in einem uninitialisierten Modus ausgeliefert, welcher auf dem sogenannten TOFU (Trust-On-First-Use) Prinzip begründet ist. Detaillierte Informationen zu diesem Server-Feature und die entsprechenden Hintergrundinformationen finden Sie [hier \[▶ 37\]](#). Der TwinCAT OPC UA Configurator ermöglicht die Initialisierung des Servers beim ersten Verbindungsaufbau. Ein entsprechender Warnhinweis weist auf den uninitialisierten Server hin und ermöglicht eine entsprechende Initialisierung.

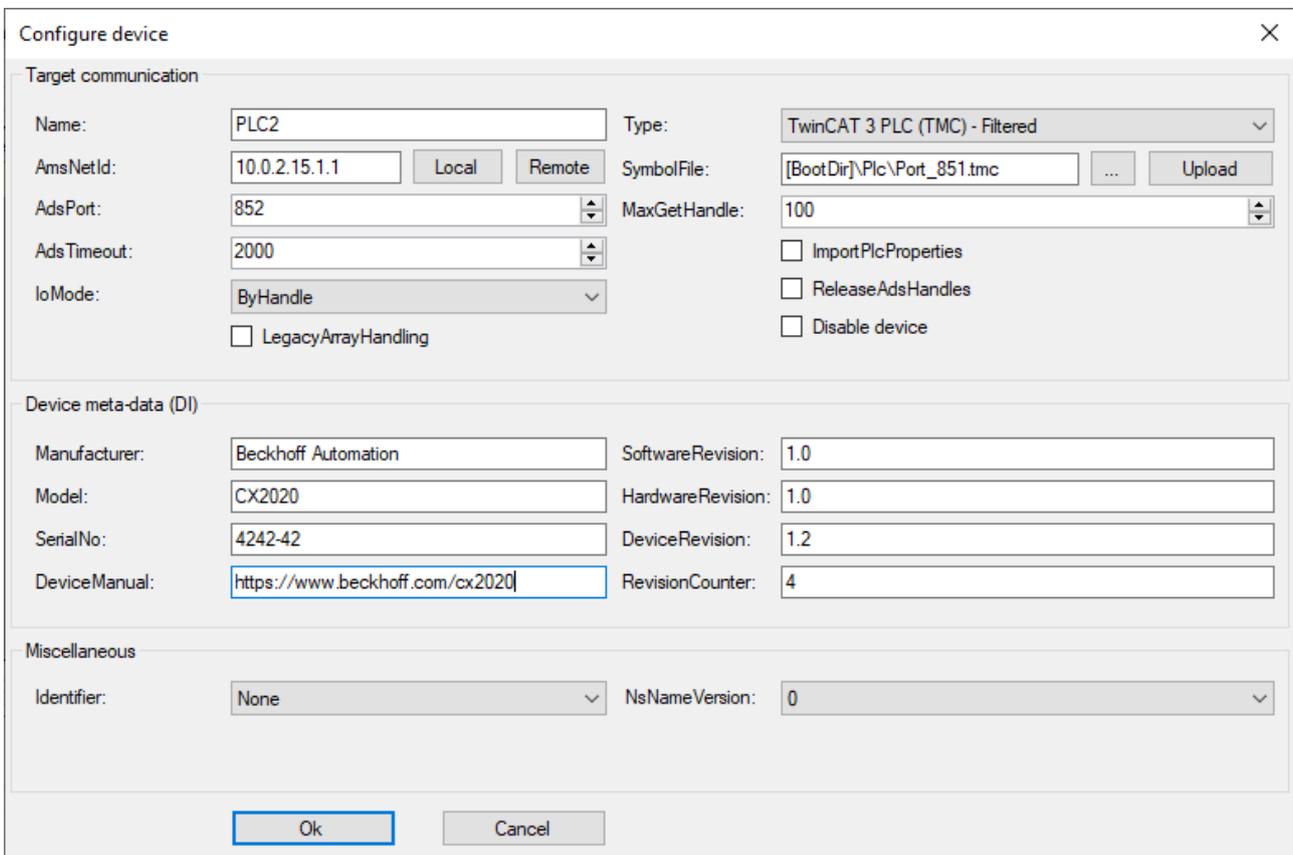


4.2.2.4 ADS-Geräte hinzufügen

Über die Registerkarte **Data Access** lassen sich ADS-Geräte zur TwinCAT OPC UA Server-Konfiguration hinzufügen. Im zugehörigen DataGrid können Sie über das Kontextmenü ein neues Gerät anlegen.



Im anschließenden Dialog können Sie die Geräte-spezifischen Parameter setzen.



Auswählen einer AMS NetID

Zur Auswahl einer AMS NetID können entweder die ADS-Teilnehmer vom lokalen System oder dem verbundenen TwinCAT OPC UA Server selektiert werden. Als ADS-Teilnehmer wird hierbei ein System bezeichnet, welches eine ADS-Route zu dem lokalen oder Server-System besitzt. Durch einen Klick auf den Button **Local** werden die lokalen ADS-Routen angezeigt. Durch einen Klick auf den Button **Remote** werden die ADS-Routen auf dem verbundenen TwinCAT OPC UA Server angezeigt.

Auswählen einer Symboldatei

Die Auswahl einer Symboldatei erfolgt immer vom lokalen System aus. Die Symboldatei kann jedoch über den **Upload**-Button auf den verbundenen TwinCAT OPC UA Server hochgeladen werden. Die Symboldatei wird dabei in dem Unterordner „symbolfiles“ des TwinCAT OPC UA Server-Basisverzeichnisses abgelegt und automatisch über einen Platzhalter in der Konfigurationsdatei referenziert.

4.2.2.5 Konfiguration lesen und schreiben

Der Konfigurator ermöglicht sowohl das Auslesen/Schreiben der Konfigurationsdateien vom TwinCAT OPC UA Server als auch das Laden/Speichern der Konfigurationsdateien auf dem lokalen System. Diese Funktionalitäten stehen sowohl über das Menü als auch die Toolbar zur Verfügung.

Lokales Laden/Speichern

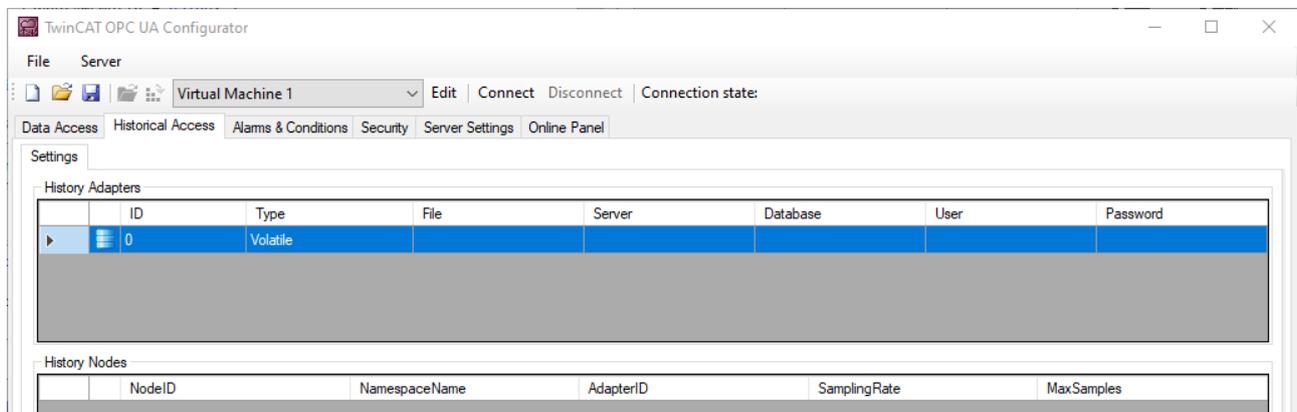
Diese Funktionen stehen über das **File**-Menü zur Verfügung. Die hier verfügbaren Buttons **Open** und **Save** ermöglichen ein Laden und Speichern der Konfigurationsdateien. Es werden hierbei immer alle Konfigurationsdateien geladen oder gespeichert.

Remotes Laden/Speichern

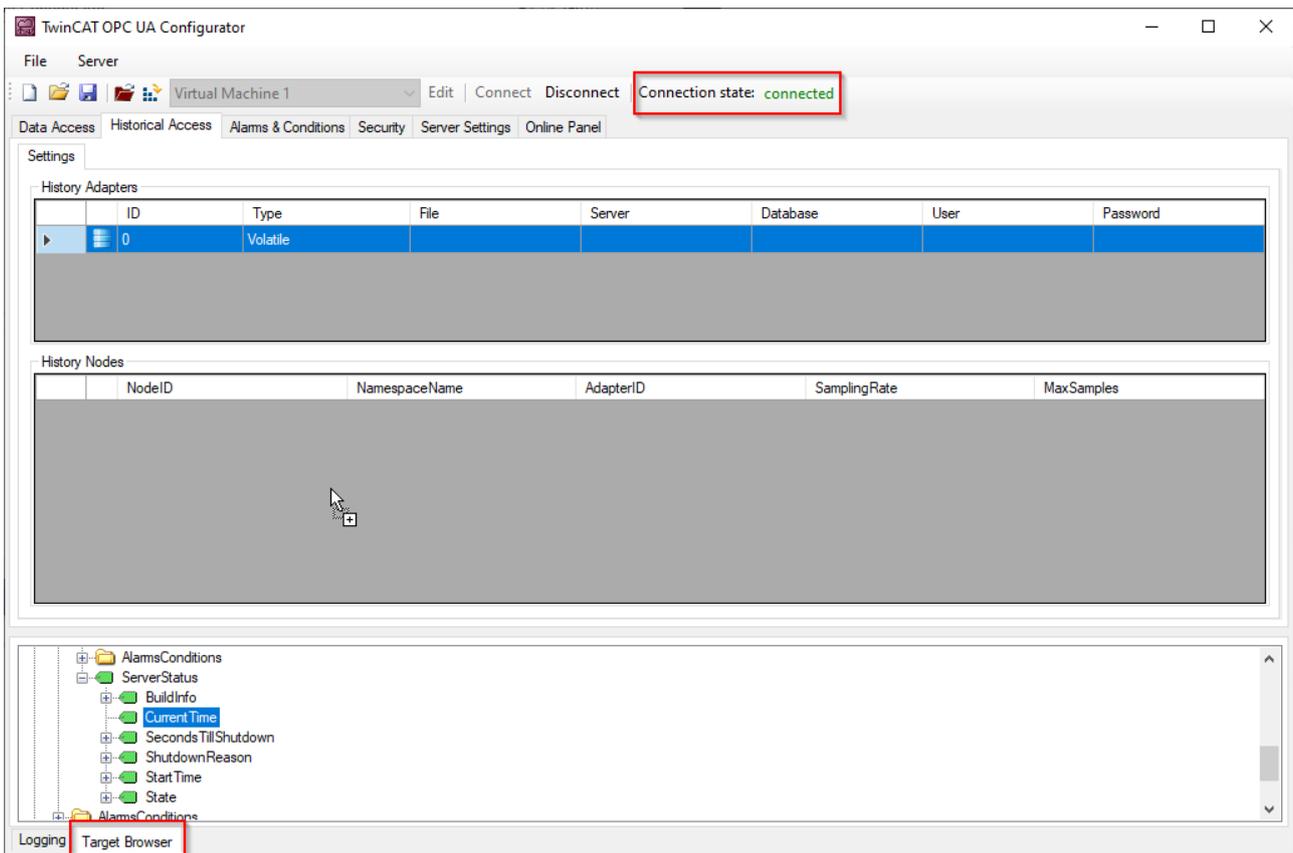
Diese Funktionen stehen über das **Server** Menü zur Verfügung. Die hier verfügbaren Buttons **Open from target** und **Activate from target** ermöglichen ein Laden und Speichern der Konfigurationsdateien vom verbundenen TwinCAT OPC UA Server. Es werden hierbei immer alle Konfigurationsdateien geladen oder gespeichert.

4.2.2.6 Historical Access konfigurieren

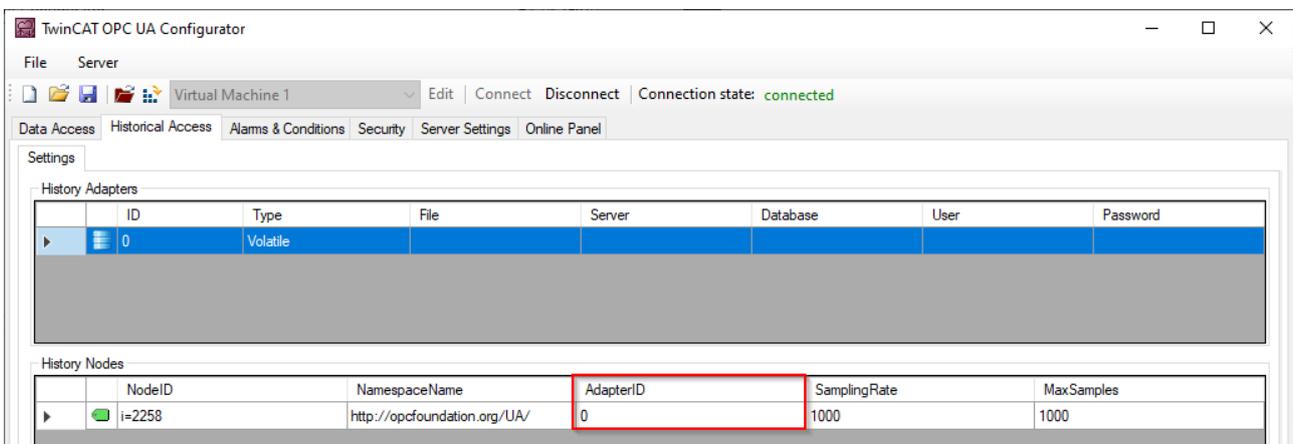
Über die Registerkarte **Historical Access** können Sie die entsprechende Konfiguration vornehmen und sowohl die **History Adapter** als auch die **History Nodes** konfigurieren. Ein **History Adapter** definiert hierbei die Art der Datenablage und eine **History Node** die Variable für die historische Daten in der Datenablage gespeichert werden sollen.



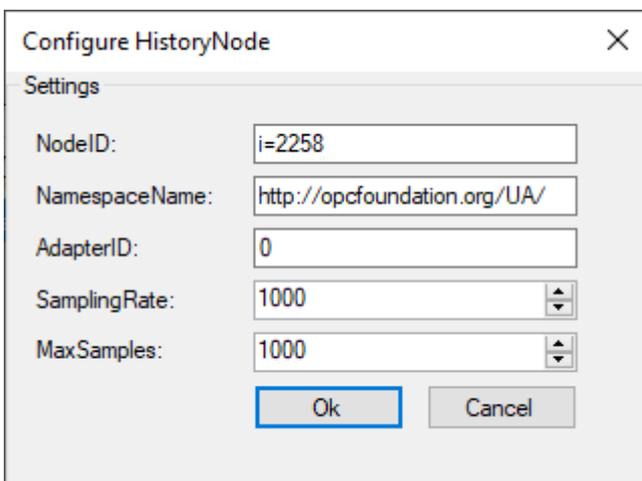
Über das Kontextmenü können Sie sowohl **History Adapter** als auch **History Nodes** anlegen. Wenn Sie mit einem TwinCAT OPC UA Server verbunden sind, dann können Sie auch komfortabel die zu konfigurierenden Nodes per Drag&Drop aus dem **Target Browser** zu den **History Nodes** hinzufügen.



Anschließend kann eine **History Node** über die AdapterID mit dem jeweiligen **History Adapter** verknüpft werden.

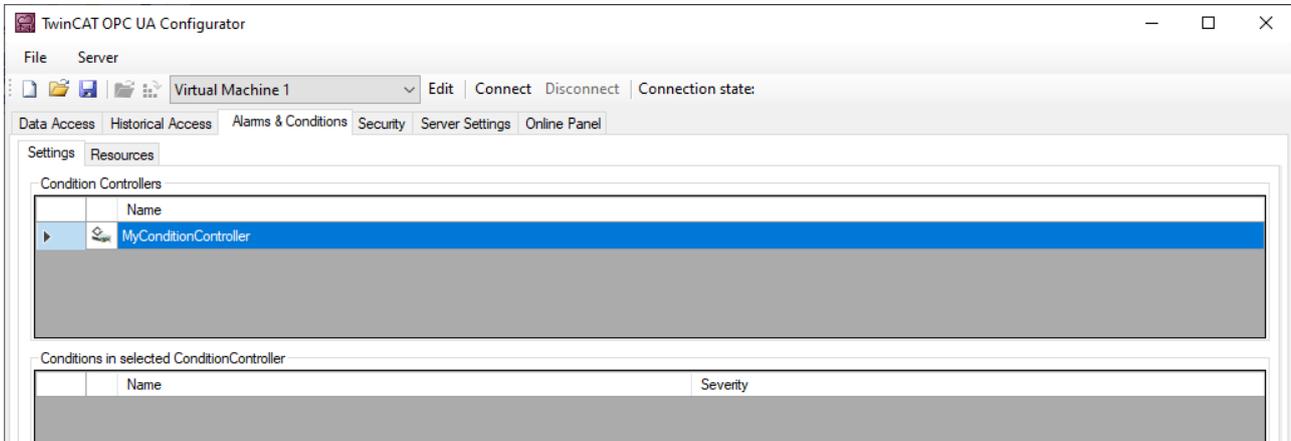


Durch einen Doppelklick auf die History Node gelangen Sie in den entsprechenden Konfigurationsdialog.

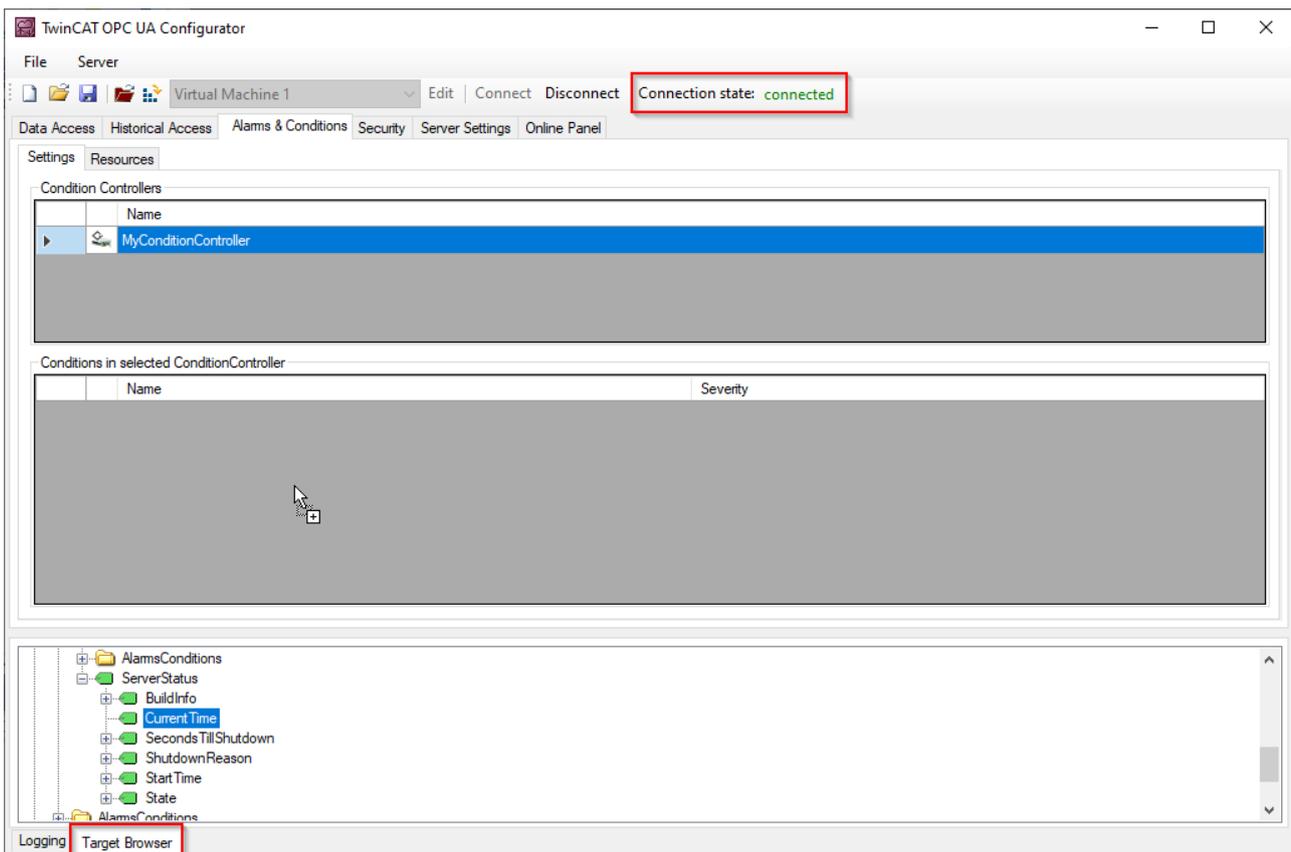


4.2.2.7 Alarms and Conditions konfigurieren

Über die Registerkarte **Alarms & Conditions** können Sie die entsprechende Konfiguration vornehmen und sowohl die **Condition Controller** als auch die **Conditions** konfigurieren. Ein **Condition Controller** ist hierbei eine Verwaltungseinheit zur Organisation der einzelnen **Conditions**. Eine **Condition** hingegen spiegelt eine Variable wider welche im Sinne von **Alarms & Conditions** anhand von konfigurierbaren Schwellenwerten überwacht werden soll.



Über das Kontextmenü können Sie sowohl **Condition Controller** als auch **Conditions** anlegen. Wenn Sie mit einem TwinCAT OPC UA Server verbunden sind, dann können Sie auch komfortabel die zu konfigurierenden Nodes per Drag&Drop aus dem **Target Browser** zu den **Conditions** hinzufügen.

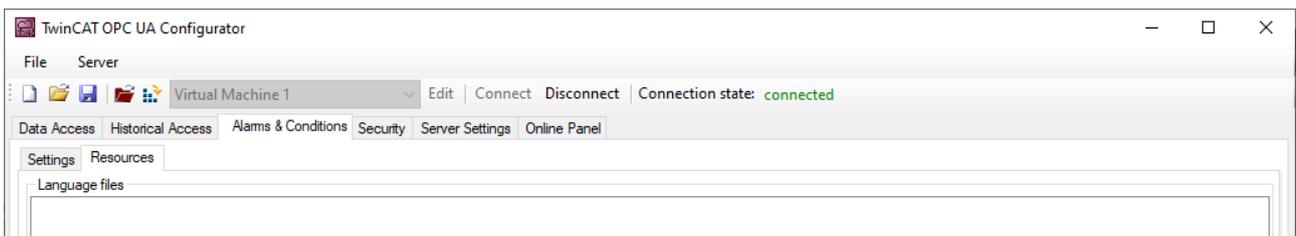


Eine **Condition** wird immer zu dem aktuell selektierten **Condition Controller** hinzugefügt. Bei Verwendung von Drag&Drop öffnet sich der Konfigurationsdialog einer **Condition** automatisch.

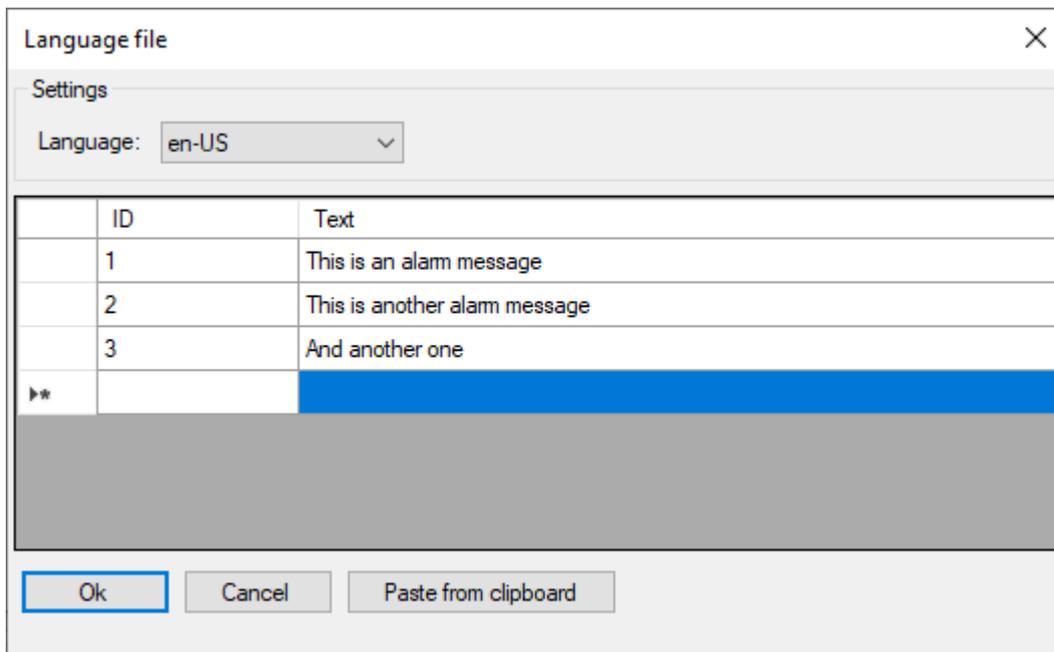
Die bei Selektion des jeweiligen **AlarmTypes** zu konfigurierenden Alarmtexte lassen sich über die entsprechenden DropDown-Boxes auswählen. Bitte beachten Sie, dass die Alarmtexte hierbei schon vorhanden sein müssen. In dem Artikel [Alarmtexte konfigurieren](#) [151] erfahren Sie mehr zu diesem Thema.

4.2.2.8 Alarmtexte konfigurieren

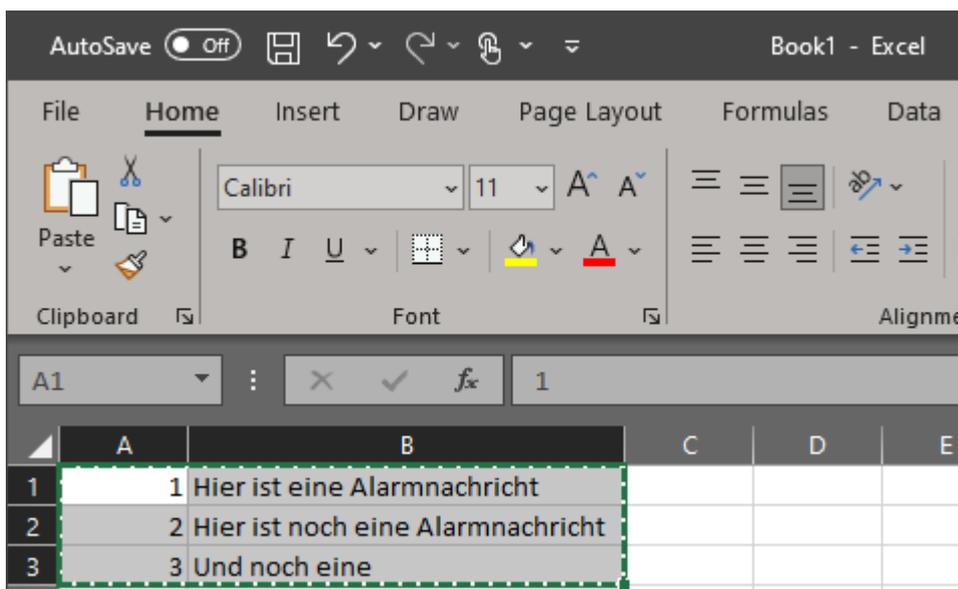
Innerhalb des **Alarms & Conditions**-Bereichs können Sie über die Registerkarte **Resources** Alarmtexte konfigurieren, welche Sie anschließend für eine Condition verwenden können.

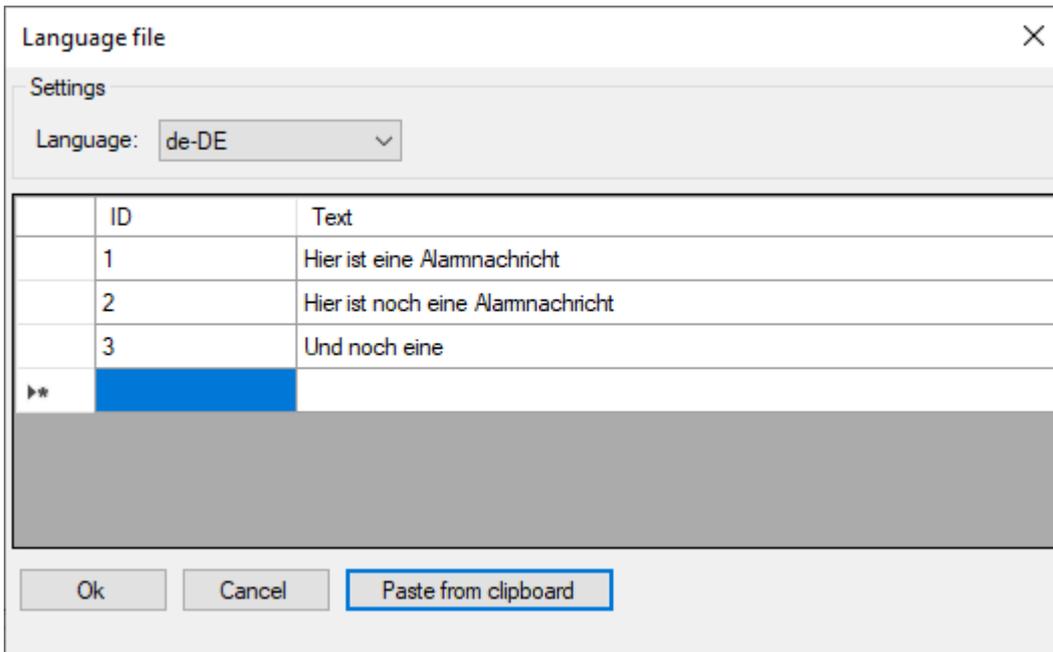


Über das Kontextmenü können Sie eine neue Alarmtextdatei hinzufügen. Diese Dateien sind nach der jeweiligen Sprache gruppiert, für welche die Alarmtexte definiert werden.

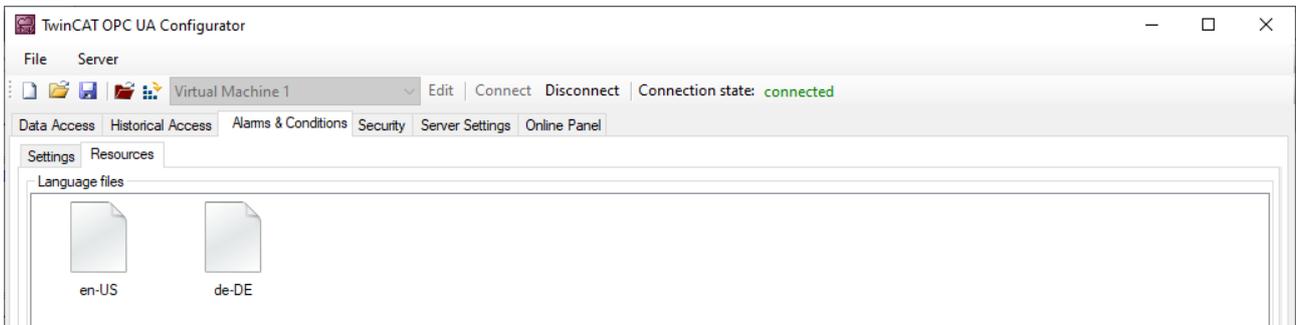


Über den **Paste from clipboard**-Button können ID und Text aus einer Excel-Tabelle übernommen werden, indem Sie von dort zuerst in die Zwischenablage kopiert (STRG+C) und dann über den Button importiert werden.





Nach der Konfiguration der Sprachdateien können Sie die Alarmtexte an einer Condition verwenden.



Condition
✕

General

Name:

Severity:

Node settings

Identifier:

NamespaceName:

SamplingRate:

Alarm type

LimitAlarm Type OffNormalAlarm Type

LimitAlarm Type settings

	Alarm text ID:	Detected languages:
HighHighLimit: <input type="text" value="10"/>	<input type="text" value="1"/> ▼	<input type="text" value="en-US,de-DE"/>
HighLimit: <input type="text" value="20"/>	<input type="text" value="2"/> ▼	<input type="text" value="en-US,de-DE"/>
LowLimit: <input type="text" value="0"/>	<input type="text" value="3"/> ▼	<input type="text" value="en-US,de-DE"/>
LowLowLimit: <input type="text" value="-10"/>	<input type="text" value="1"/> ▼	<input type="text" value="en-US,de-DE"/>

OffNormalAlarmType settings

	Alarm text ID:	Detected languages:
Normal value: <input type="text"/>	<input type="text" value=""/>	<input type="text"/>

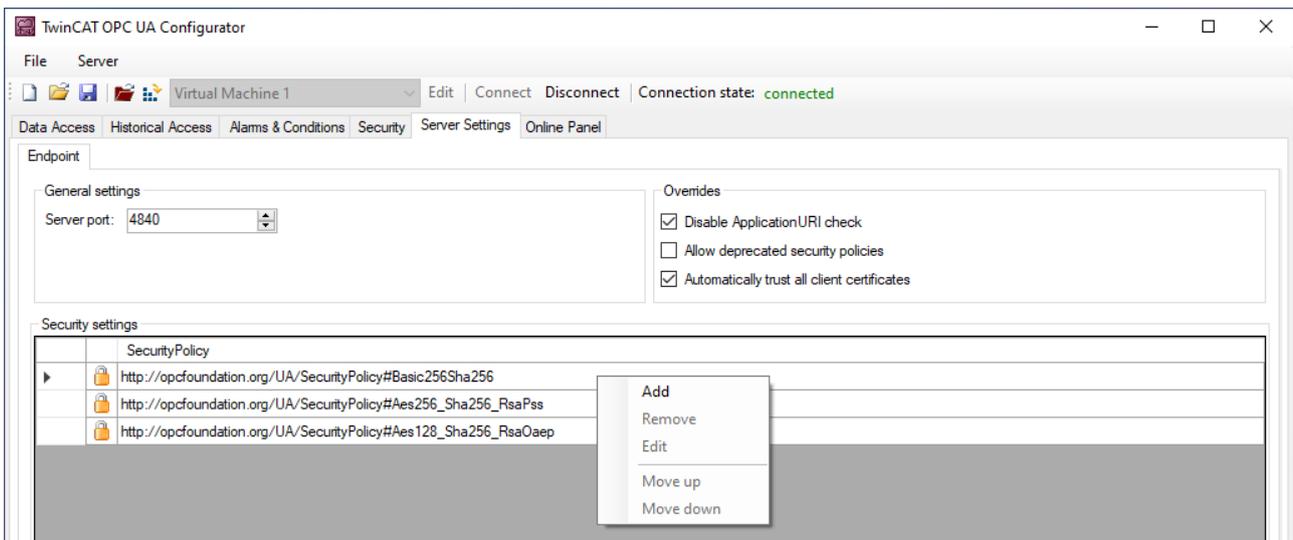
Über die **Detected languages**-Felder können Sie schnell überprüfen, ob Sie die selektierte AlarmtextID auch für alle Sprachen definiert haben, oder ob eventuell eine Sprache vergessen wurde.

4.2.2.9 Endpunkte konfigurieren

Die Endpunkte des OPC UA Servers geben an, welche Security-Mechanismen bei der Verbindungsherstellung eines Clients benutzt werden sollen. Diese reichen von „unverschlüsselt“ bis zu „verschlüsselt und signiert“, basierend auf verschiedenen Schlüsselstärken.

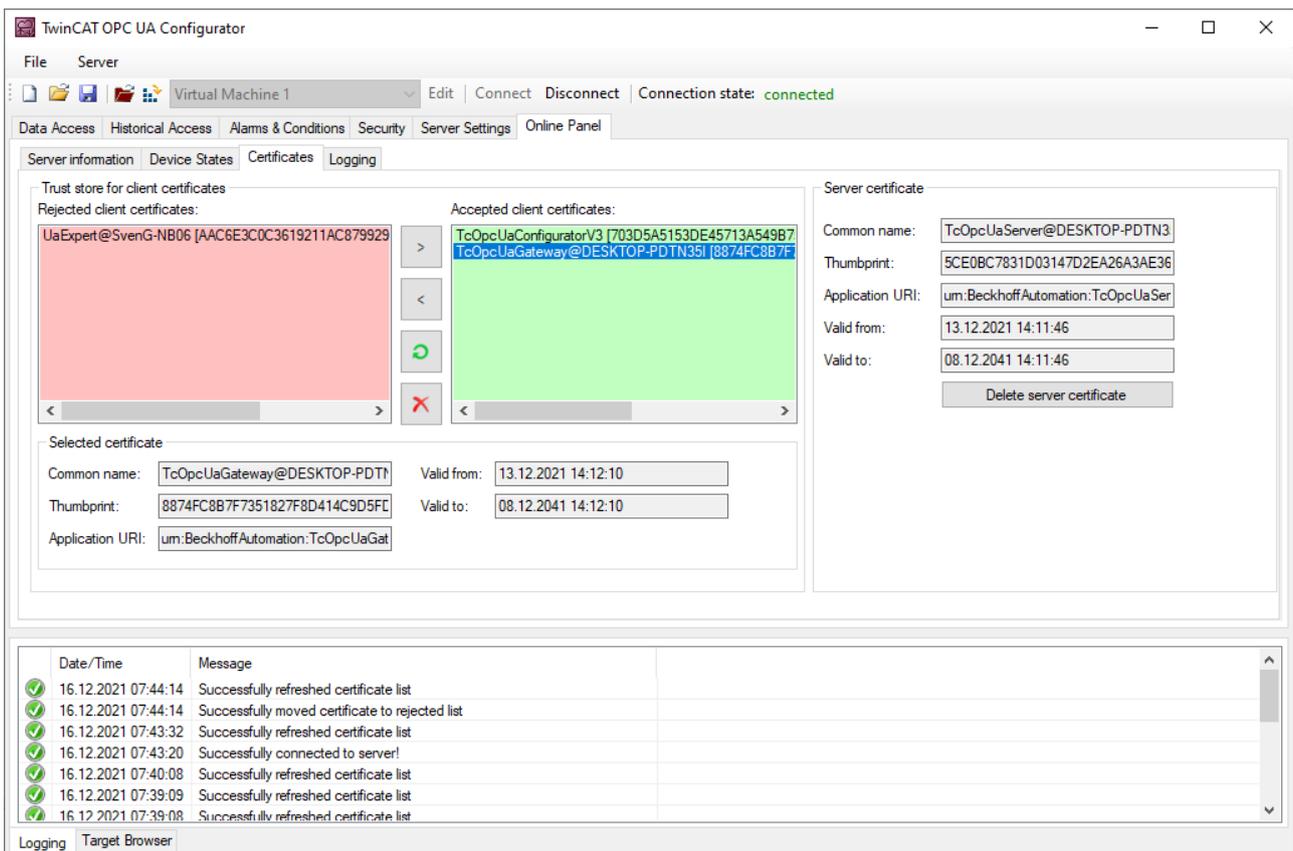
Die Endpunkte können Sie über den Konfigurator aktivieren und deaktivieren. Es kann z. B. sinnvoll sein, den unverschlüsselten Endpunkt zu deaktivieren, damit sich alle Clients nur mit gültigem und als vertrauenswürdig eingestuftem Zertifikat verbinden können.

Über die Registerkarte **Server Settings** können Sie die Endpunkte, sowie einige Zusatzparameter, konfigurieren. Über das Kontextmenü lassen sich Endpunkte hinzufügen oder aus der Konfiguration entfernen.



4.2.2.10 Vertrauensstellung für Zertifikate

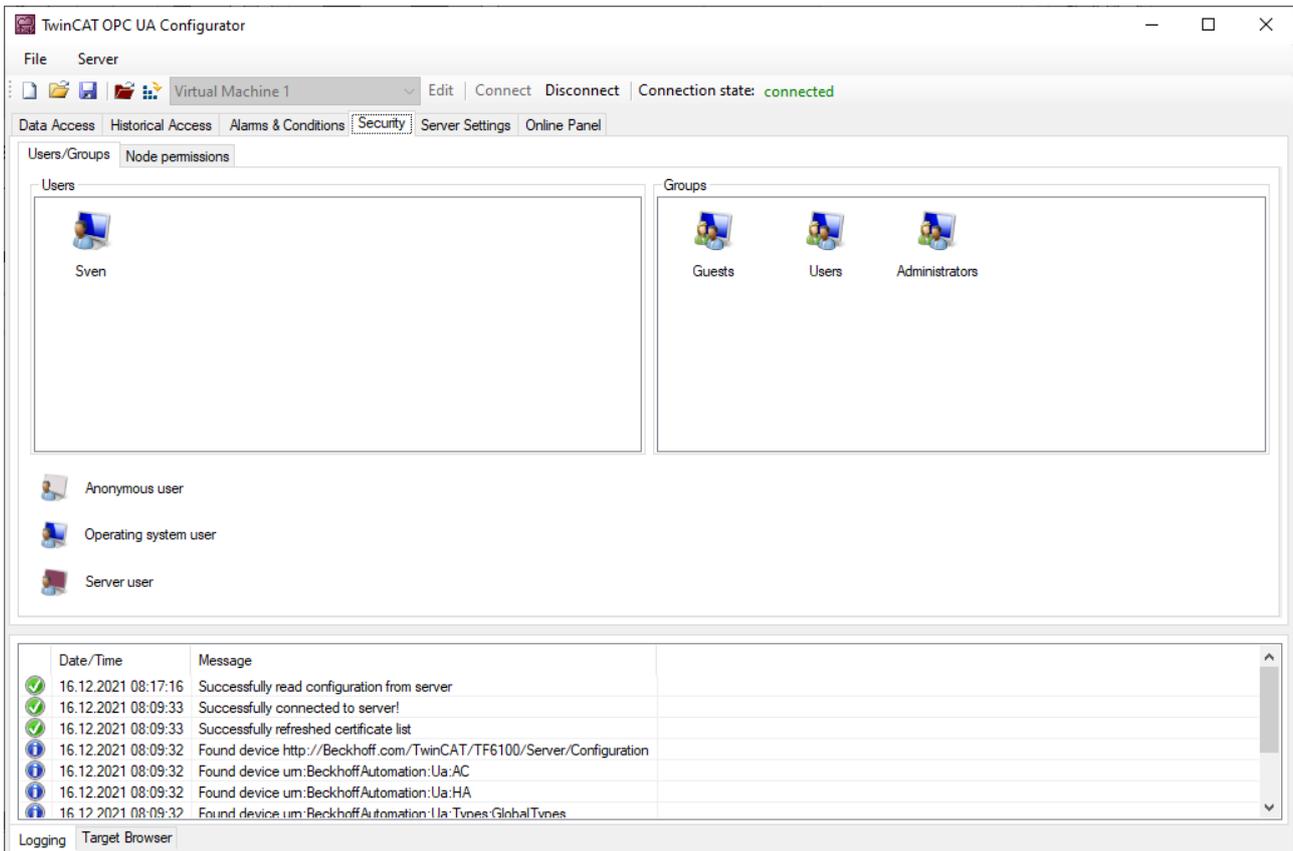
Über die Registerkarte **Online Panel** und den dortigen Bereich **Certificates** lassen sich die Vertrauensstellungen für Clientzertifikate auf dem TwinCAT OPC UA Server konfigurieren. Durch Selektion eines Clientzertifikats in dem jeweiligen TrustStore (Rejected/Accepted) lassen sich Zertifikatsdetails anzeigen und dieses zwischen den TrustStores verschieben.



4.2.2.11 Sicherheitseinstellungen konfigurieren

Über die Registerkarte **Security** lassen sich Sicherheitseinstellungen am Server vornehmen. Diese Sicherheitseinstellungen können die folgenden Punkte beinhalten:

- Benutzer und Gruppen
- Zugriffsrechte für Gruppen auf Namespaces
- Zugriffsrechte für Gruppen auf einzelne Nodes



Benutzer und Gruppen

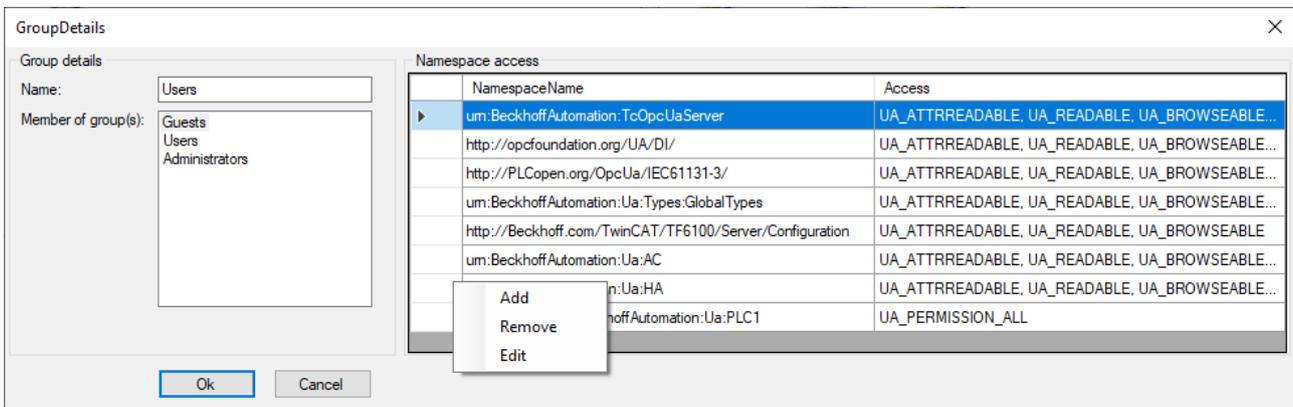
Zur Konfiguration von Zugriffsrechten müssen zunächst einmal Benutzer und Benutzergruppen erstellt werden. Im Auslieferungszustand des Servers sind bereits einige Gruppen vordefiniert. Über das Kontextmenü lassen sich neue Benutzer oder Gruppen zur Konfiguration hinzufügen.

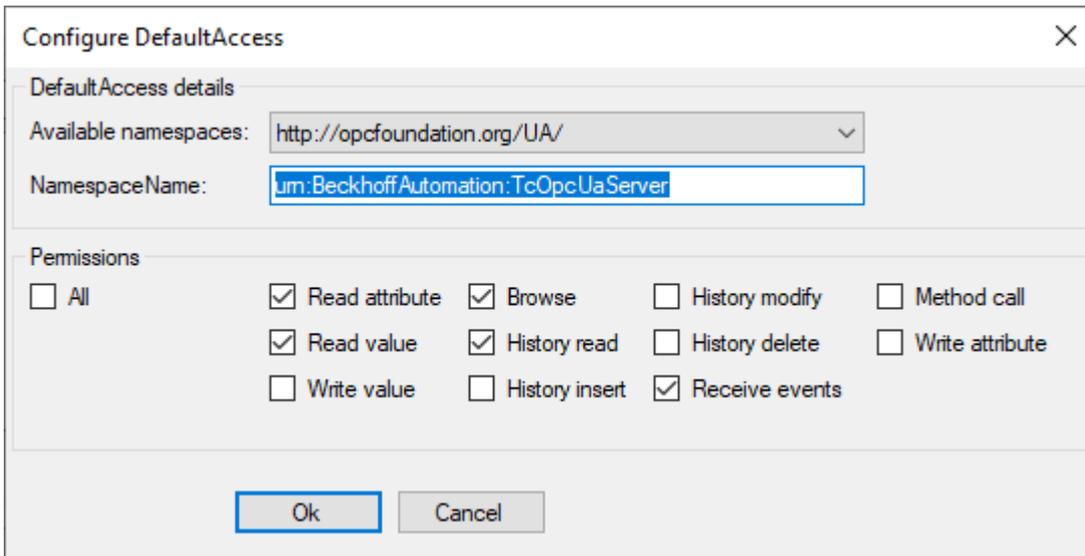
Ein Benutzer kann hierbei entweder der Anonymous-Benutzer, ein Betriebssystembenutzer oder ein Serverbenutzer sein. Wir empfehlen in jedem Fall die Konfiguration von Betriebssystembenutzern.

Eine Benutzergruppe kann einen sogenannten **Default-Access** konfiguriert haben. Hierbei handelt es sich um Zugriffsrechte auf einen bestimmten Namespace.

Zugriffsrechte auf Namespaces

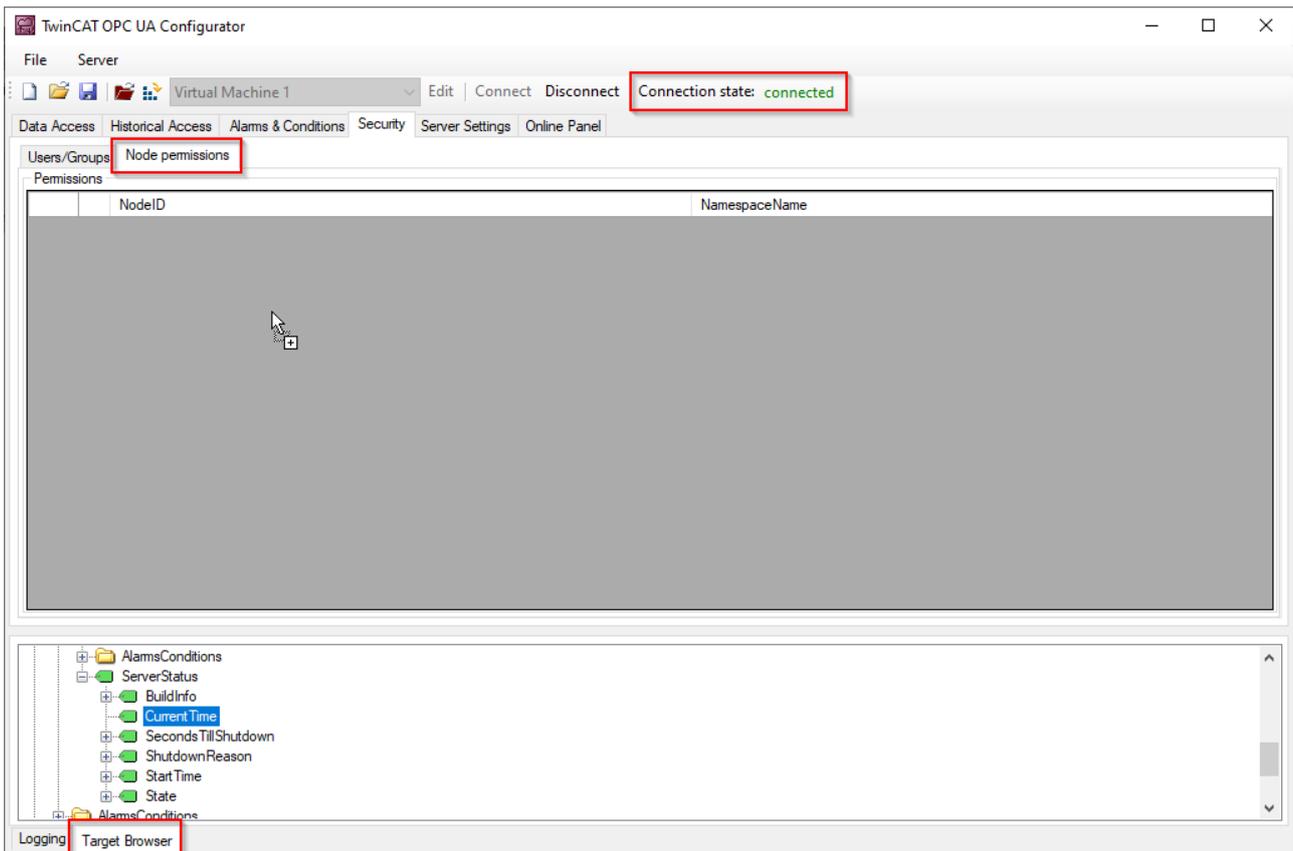
Zugriffsrechte auf bestimmte Namespaces lassen sich an einer Benutzergruppe definieren. In den Einstellungen der Gruppe gibt es hierbei einen entsprechenden Konfigurationsbereich, welcher sich über das Kontextmenü editieren lässt.



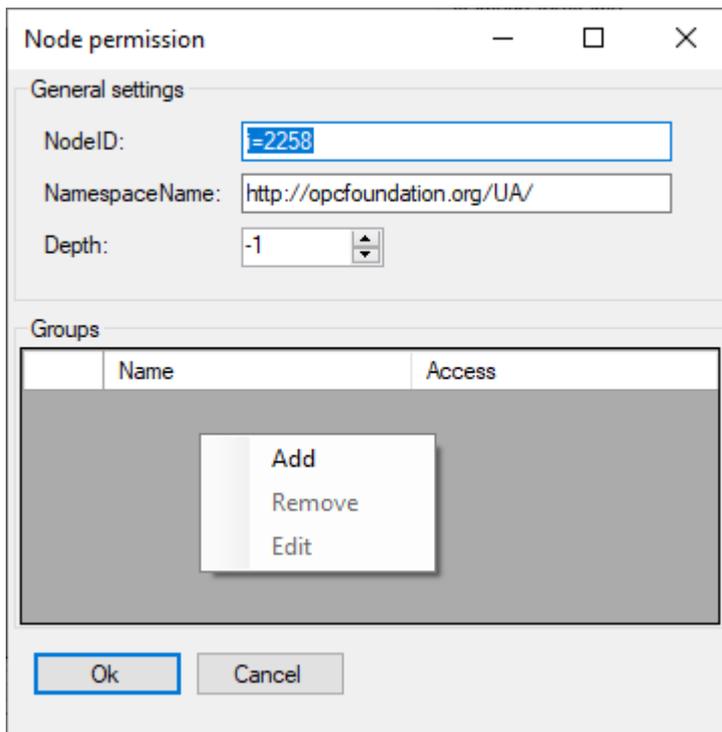


Zugriffsrechte auf einzelne Nodes

Über die Registerkarte **Node permissions** lassen sich Zugriffsrechte auf einzelne Nodes und deren Kindelemente definieren. Sie können die Nodes hierbei manuell über das Kontextmenü konfigurieren oder sie bequem per Drag&Drop aus dem **Target Browser** zur Konfiguration hinzufügen, sofern Sie mit einem Server verbunden sind.



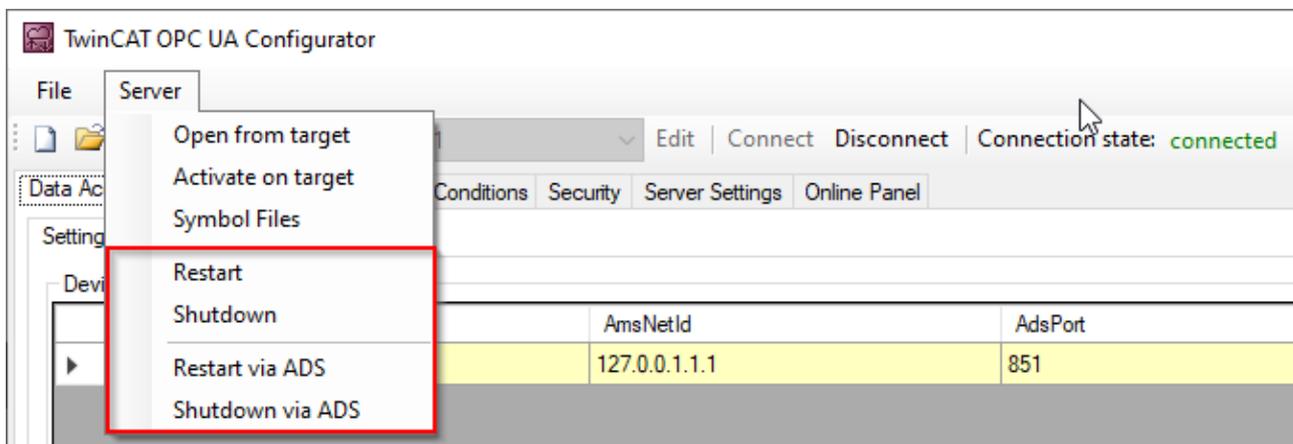
Im Node-Konfigurationsdialog lassen sich dann die Benutzergruppen und Zugriffsrechte der jeweiligen Gruppe definieren.



Über den Parameter **Depth** können Sie einstellen, ob die Berechtigungen auf Kindelemente vererbt werden sollen. Der Wert „-1“ gibt hierbei an, dass alle Kindelemente die Berechtigungen vererbt bekommen sollen.

4.2.2.12 Server neu starten

Über das **Server**-Menü lässt sich ein TwinCAT OPC UA Server neu starten. Üblicherweise möchten Sie den gerade über OPC UA verbundenen Server neu starten. Alternativ können Sie den Neustart auch über ADS antriggern, falls Sie eine ADS-Route zu dem Server-System hergestellt haben.



4.2.2.13 Logging

Für eine erweiterte Diagnose können Sie die Logging-Funktion des OPC UA Servers aktivieren.

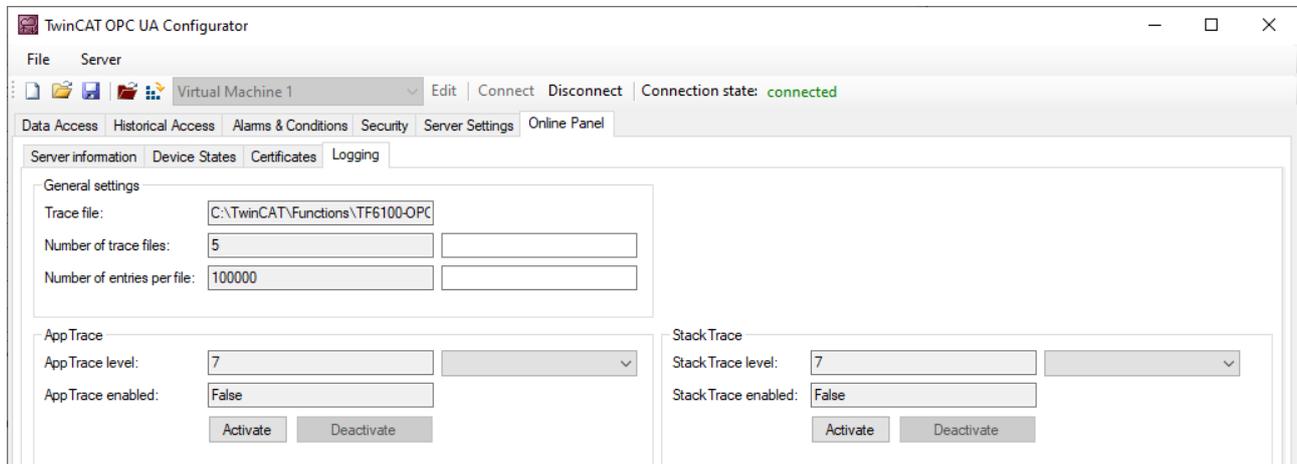
● Schreiben der Log-Datei

i Durch das Aktivieren der Logging-Funktion auf dem Server wird eine Protokolldatei auf dem Dateisystem geschrieben. Stellen Sie sicher, dass ausreichend Speicherplatz zur Verfügung steht und setzen Sie die Logging-Parameter entsprechend (Anzahl Log-Dateien, Größe pro Log-Datei).

i Performance- und Timingverhalten

Durch das Aktivieren der Protokollfunktionen verändert sich das Timing-Verhalten des OPC UA Servers. Hierdurch können je nach Plattform und Projekt Geschwindigkeitseinbußen entstehen.

Über die Registerkarte **Online Panel** und den dortigen Bereich **Logging** lassen sich die Server-Protokollfunktionen aktivieren.



Trace-Level

Generell gilt: Je höher der „Trace level“, desto detailliertere (und mehr) Daten werden geschrieben, desto mehr Last wird jedoch auch auf der Serverapplikation verursacht, wodurch sich das Timingverhalten entsprechend ändert. Bitte aktivieren Sie daher das Logging nur im Diagnosefall und in Absprache mit dem Beckhoff Support.

Activate App Trace

In den meisten Fällen ist es ausreichend ein sogenanntes „AppTrace“ zu erstellen. Hierbei werden Informationen der Serverapplikation protokolliert. Zum Aktivieren des AppTrace tragen Sie bitte die Anzahl an TraceFiles, sowie die Anzahl Einträge pro TraceFile in die zugehörigen Textfelder ein. Anschliessend wählen Sie einen Tracelevel aus und klicken auf den Button zum Aktivieren des AppTrace. Die Werte in den grau hinterlegten Textfeldern stellen die aktuellen Einstellungen auf dem Server dar.

Activate Stack Trace

In einigen wenigen Fällen ist es zusätzlich notwendig ein sogenanntes „StackTrace“ zu erstellen, wodurch Informationen vom OPC UA Stack protokolliert werden. Zum Aktivieren des StackTrace tragen Sie bitte die Anzahl an TraceFiles, sowie die Anzahl Einträge pro TraceFile in die zugehörigen Textfelder ein. Anschliessend wählen Sie einen Tracelevel aus und klicken auf den Button zum Aktivieren des StackTrace. Die Werte in den grau hinterlegten Textfeldern stellen die aktuellen Einstellungen auf dem Server dar.

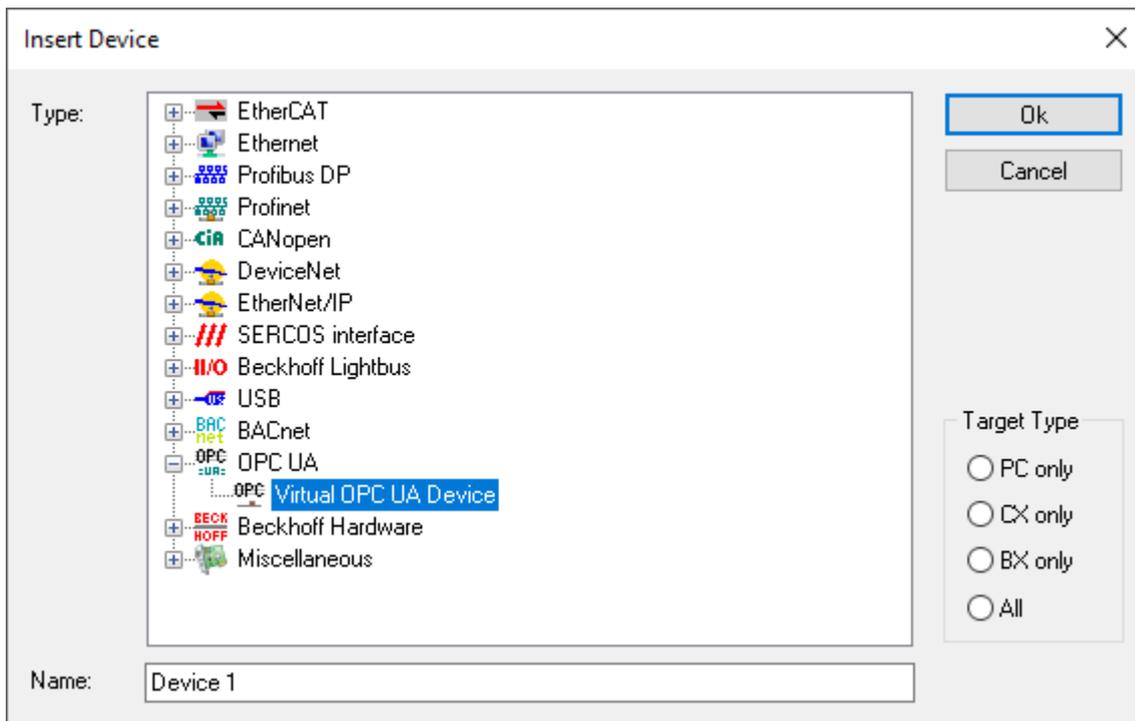
4.3 Client I/O

4.3.1 Übersicht

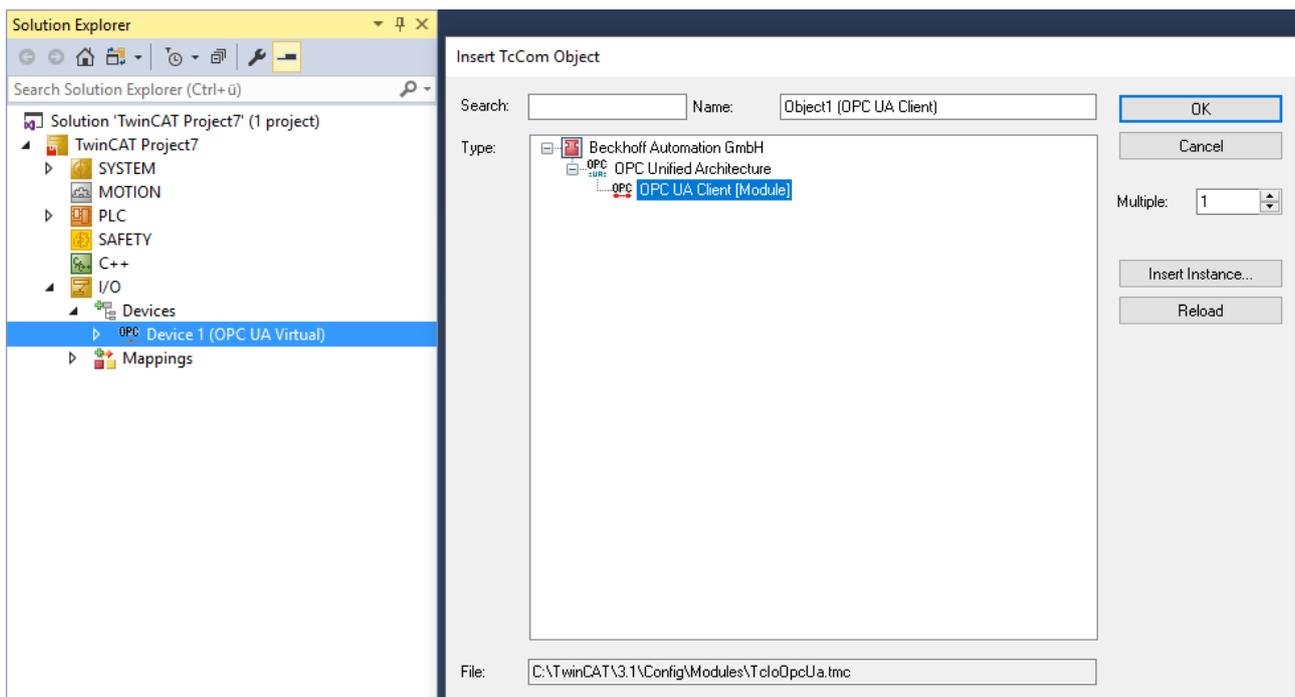


Diese Funktionalität erfordert TwinCAT 3.1 Build 4022.4 oder höher.

Der TwinCAT OPC UA Client ist ebenfalls als ein I/O-Treiber integriert und ist als solcher in der TwinCAT-I/O-Konfiguration verfügbar. Dies verringert nicht nur die Engineering-Zeit, die zur Herstellung der OPC-UA-Verbindung zu einem Remoteserver erforderlich ist, sondern ermöglicht die Realisierung vieler Anwendungsfälle, wie z. B. die Erstellung von einfach zu konfigurierenden Protokollbrücken.



Fügen Sie zunächst einen virtuellen Geräte-Container und anschließend ein OPC-UA-Client-Objekt zur Konfiguration hinzu.



Nachdem Sie das OPC-UA-I/O-Client-Objekt erstellt haben, können Sie seine Verbindung zu einem OPC UA Target Server konfigurieren und Variablen, Methoden und Strukturen zum Prozessabbild hinzufügen. Der Prozess des Datenaustauschs mit einem Target Server wird auf ein einfaches Mapping-Verfahren von Prozessvariablen reduziert.

Voraussetzungen

Produkte	Setup-Versionen	Zielplattform
TF6100	4.x.x	IPC oder CX (x86, x64, ARM)

4.3.2 Schnelleinstieg

Die folgende Anleitung ermöglicht einen Schnelleinstieg in den OPC UA I/O Client. In der Anleitung wird ein SPS-Projekt erzeugt, das eine Variable über den OPC UA Server freigibt (zur Simulation eines UA Servers) und die freigegebene Variable wieder über den OPC UA I/O Client einliest. Dies kann natürlich auch auf zwei Projekte und zwei Steuerungen aufgeteilt werden.

Variablen für Server freigeben

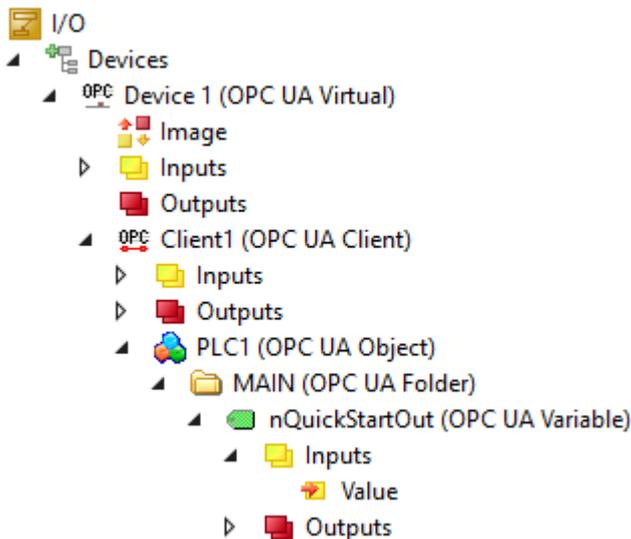
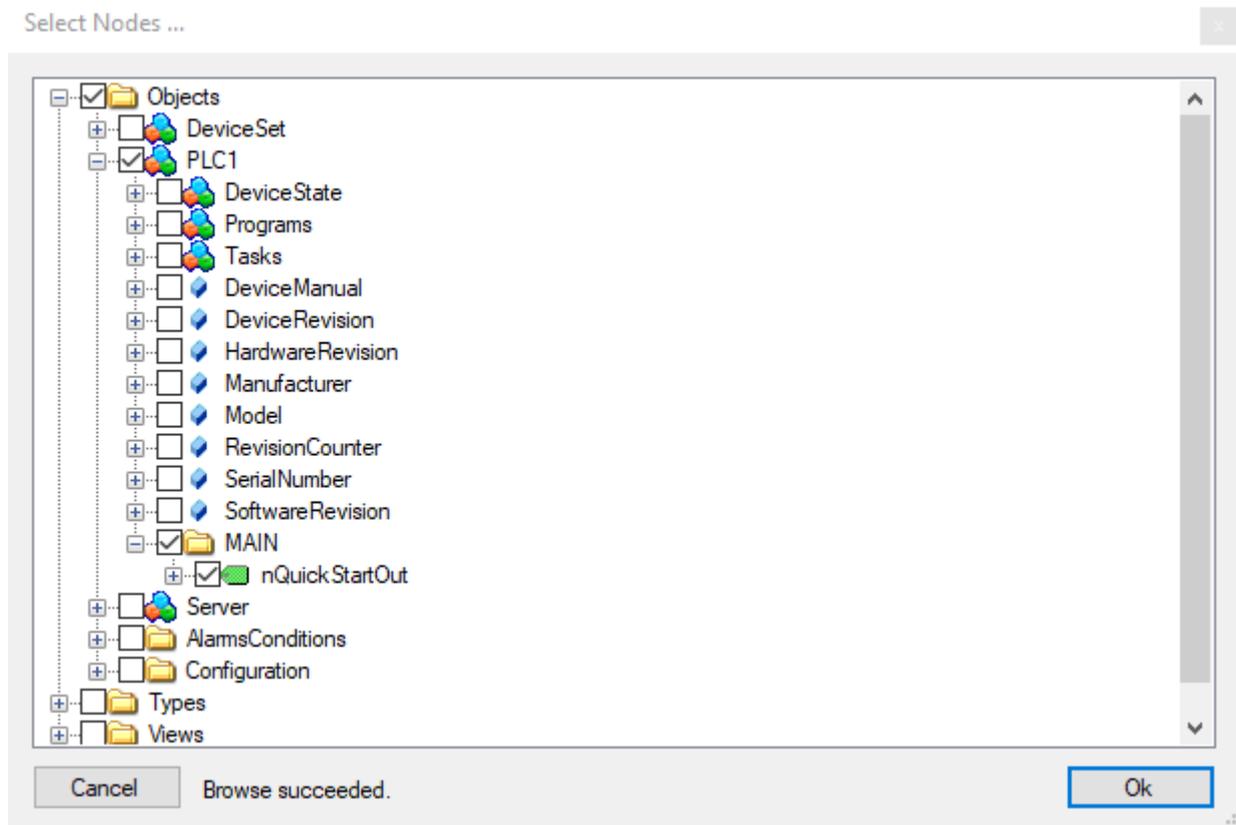
Zur Simulation soll in diesem Schnelleinstieg der OPC UA Server verwendet werden.

1. Erstellen Sie ein neues TwinCAT-Projekt und fügen Sie dem Projekt ein neues SPS-Projekt hinzu.
 2. Legen Sie im SPS-Projekt eine neue Variable „nQuickStartOut“ vom Datentyp INT an. Dies soll die Variable sein, die über den OPC UA Server freigegeben wird.
 3. Setzen Sie das OPC-UA-Attribut [► 49] für diese Variable, um die Variable über den OPC UA Server freizugeben.
 4. Aktivieren Sie das Auswahlkästchen für den Download der TMC-Datei.
 5. Aktivieren Sie die Konfiguration.
- ⇒ Die Variable ist für den Server freigegeben.

I/O UA Client konfigurieren

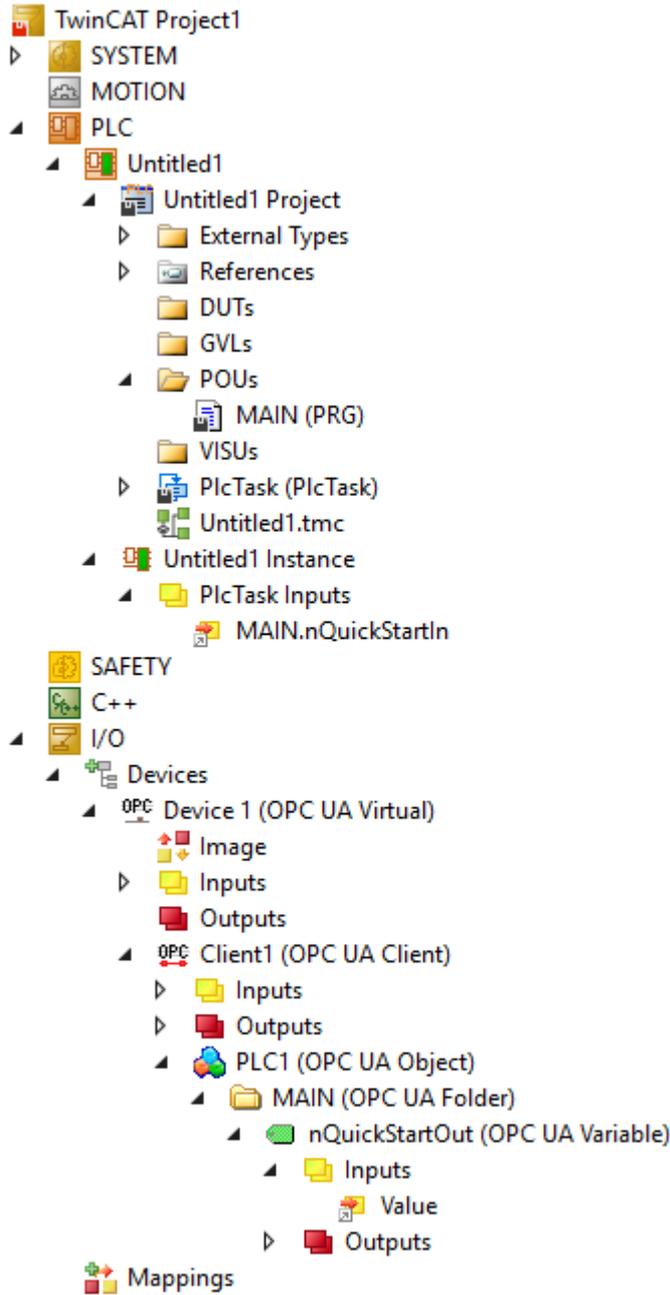
1. Legen Sie in demselben SPS-Projekt eine weitere Variable „nQuickStartIn“ vom Datentyp INT an und definieren Sie diese als Eingangsvariable (AT%I*). Dies soll die Variable sein, die vom OPC UA Client I/O-Gerät über das Mapping mit der Variablen vom Server verknüpft wird.
2. Kompilieren Sie das Projekt, sodass die Eingangsvariable im SPS-Prozessabbild zur Verfügung steht.
3. Fügen Sie ein neues „Virtual OPC UA Device“ hinzu.
4. Fügen Sie einen neuen „OPC UA Client“ zum Device hinzu und öffnen Sie dessen Einstellungen.
5. Navigieren Sie zur Registerkarte **Settings**. Tragen Sie dort die Server-URL des OPC UA Servers ein. In diesem Beispiel lautet diese „opc.tcp://localhost:4840“.
6. Klicken Sie auf **Add Nodes**.

7. Navigieren Sie zur freigegebenen Variable „nQuickStartOut“ auf dem OPC UA Server (zu finden unterhalb von PLC1 > MAIN) und selektieren Sie diese. Die Variable wird nun zum Prozessabbild des „Virtual OPC UA Device“ hinzugefügt.



8. Erweitern Sie die neu hinzugefügte Variable im Prozessabbild und klicken Sie doppelt auf **Input > Value**. Verknüpfen Sie die Variable mit der Variablen „nQuickStartIn“ aus dem SPS-Prozessabbild.
9. Aktivieren Sie die Konfiguration.

⇒ Sie sollten nun die nachfolgend dargestellte Konfiguration vorfinden. Die beiden Prozessabbild-Variablen nQuickStartIn und nQuickStartOut sind miteinander verknüpft.



10. Nach dem Aktivieren der Konfiguration loggen Sie sich in das SPS-Programm ein.

TwinCAT_Project1.Untitled1.MAIN				
Expression	Type	Value	Prepared value	Address
nQuickStartOut	INT	1864		
nQuickStartIn	INT	1856		%I*

Den aktuellen Wert können Sie auch im Prozessabbild-Mapping einsehen.

11. Über die Einstellungen am OPC UA Client können Sie Parameter wie z. B. die Sampling rate modifizieren.

Voraussetzungen

Produkte	Setup-Versionen	Zielplattform
TF6100	4.x.x	IPC oder CX (x86, x64, ARM)

4.3.3 Unterstützte Datentypen

Der OPC UA Client ermöglicht den Zugriff auf einen OPC UA Server direkt aus einer Echtzeitlogik heraus. Zum Lesen und Schreiben von Daten muss der Datentyp des OPC-UA-Knotens der TwinCAT-Umgebung zugeordnet werden (Mapping). Diese Zuordnung wird nachfolgend beschrieben.

Basisdatentypen

Die Zuordnung der Basisdatentypen wird in PLCopen OPC UA Information Model for IEC 61131-3 beschrieben.

OPC-UA-Datentyp	SPS-Datentyp
Boolean	BOOL
SByte	SINT
Byte	USINT
Int16	INT
Int32	DINT
String	STRING
USint	BYTE
Float	REAL
Double	LREAL
UInt16	UINT
UInt32	UDINT
Int64	LINT
UInt64	ULINT
DateTime	DT

Das Mapping können Sie sowohl auf die OPC UA Client PLCopen-Funktionsbausteine als auch auf den OPC UA I/O Client anwenden.

Abgeleitete Datentypen

OPC UA definiert Basisdatentypen. Andere Datentypen sind davon abgeleitet.

Auf Client-Seite ist nur mit SPS-Datentypen Zugriff auf alle UA-Datentypen möglich. Datentypen, die von den Basisdatentypen abgeleitet sind, werden ab TcOpcUaClient Version 2.1.0.36 ebenfalls unterstützt.

Derzeit unterstützte Nicht-Basisdatentypen auf Client-Seite sind:

- Counter (UDINT in SPS nutzen)

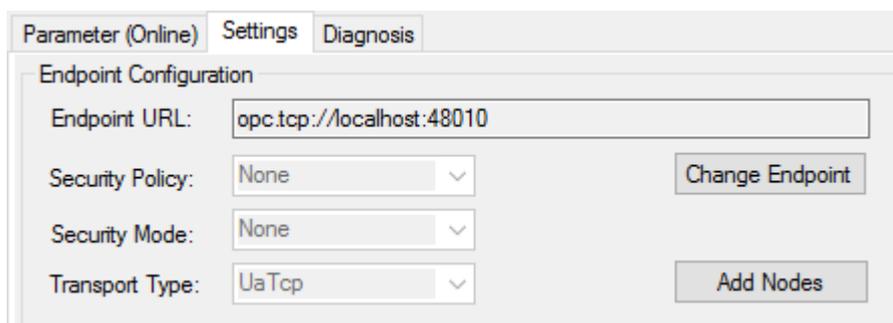
Zudem unterstützt der OPC UA I/O Client strukturierte Datentypen (StructuredTypes), wenn der strukturierte Datentyp keinen nicht-unterstützten Datentyp umfasst (z. B. in einer Membervariablen). Die folgende Liste bietet einen Überblick über die derzeit nicht unterstützten Datentypen:

- ByteString
- NodeID
- LocalizedText (und somit AnalogItemTypen, DiscretelItemTypen)

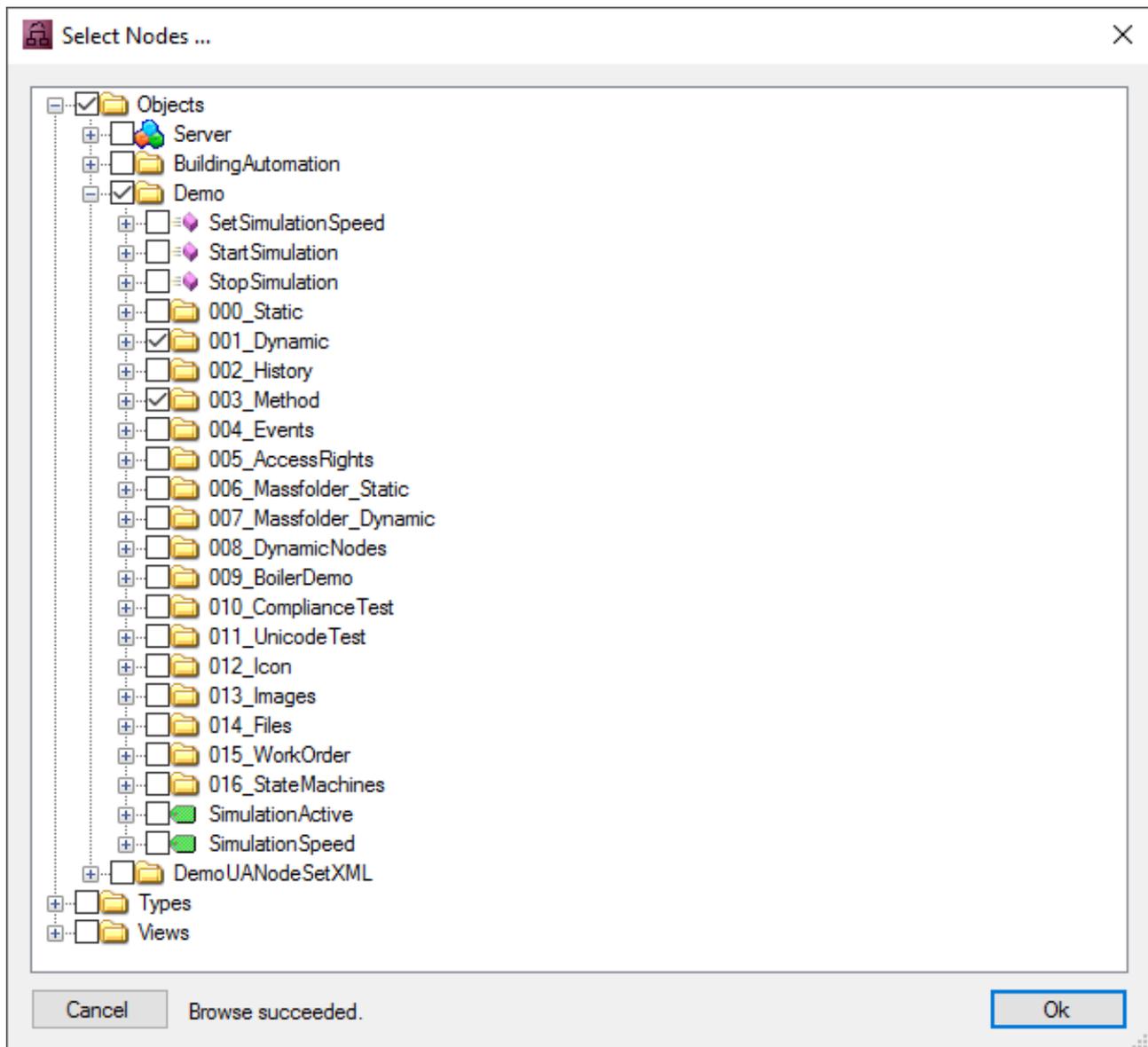
4.3.4 Knoten vom Server hinzufügen

Zum Hinzufügen von Knoten eines OPC UA Target Servers zum Prozessabbild des OPC UA I/O Clients können Sie den Namensraum-Browser verwenden, der in dem I/O Client integriert ist.

Öffnen Sie die Registerkarte **Settings** der Client-Konfiguration und klicken Sie auf die Schaltfläche **Add Nodes**, um den Namensraum-Browser zu öffnen.



Der Namensraum-Browser-Dialog stellt automatisch eine Verbindung zum Target Server her, indem die spezifizierten Verbindungsparameter genutzt werden.



Innerhalb des Namensraum-Browsers können Sie die Auswählkästchen aktivieren und deaktivieren, um Knoten zum Prozessabbild hinzuzufügen oder von dort zu entfernen. Jeder ausgewählte Knoten wird im Prozessabbild als Eingangs- oder Ausgangsvariable dargestellt.

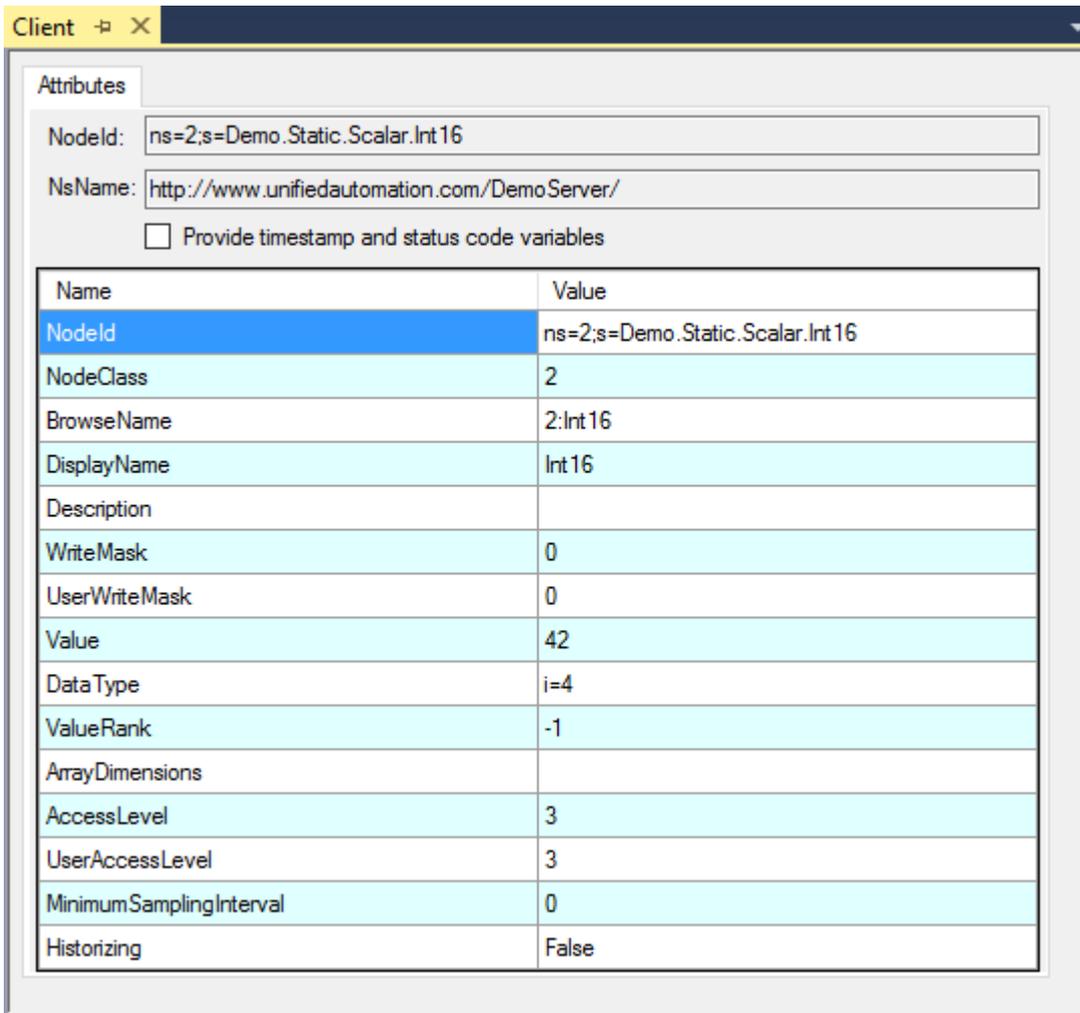
■ Int16 (OPC UA Variable)

- ▲ Inputs
 - ▶ Value
- ▲ Outputs
 - ▶ Value

Die Eingangsvariable liest Daten vom Knoten und enthält den letzten vom Knoten erhaltenen Wert. Die Ausgangsvariable schreibt Daten zum Knoten, wenn ein Wert gesetzt wurde.

4.3.5 Knotenattribute

Wenn Sie auf einen Knoten im Prozessabbild doppelklicken, sehen Sie die UA-Attribute mit ihren aktuellen Werten zu dem Zeitpunkt, zu dem das Fenster geöffnet wurde.



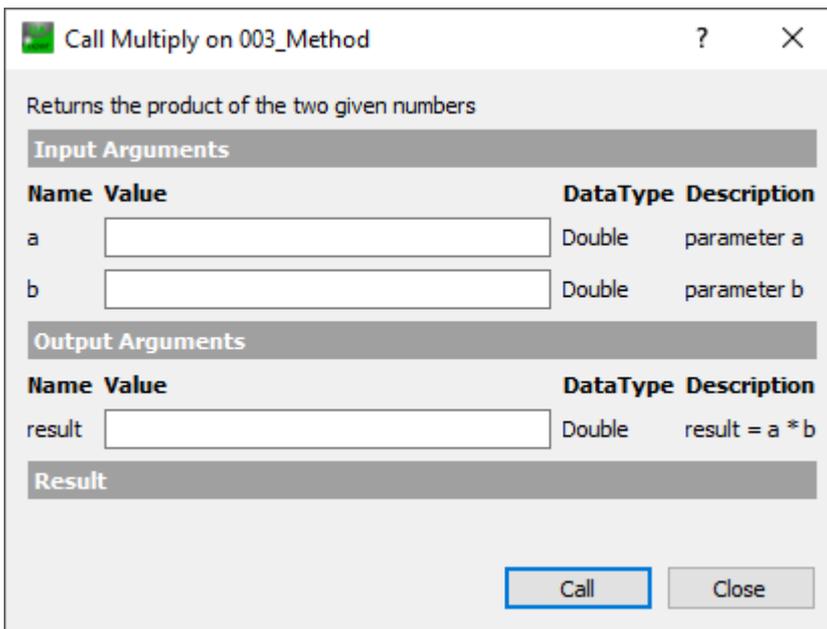
Über das Auswahlkästchen **Provide timestamp and status code variables** fügen Sie dem Prozessabbild weitere Variablen hinzu, die zu Diagnosezwecken verwendet werden können.

- Int16 (OPC UA Variable)
 - ▲ Inputs
 - 📄 TimeStamp
 - 📄 ErrorCode
 - 📄 Value
 - ▲ Outputs
 - 📄 Value

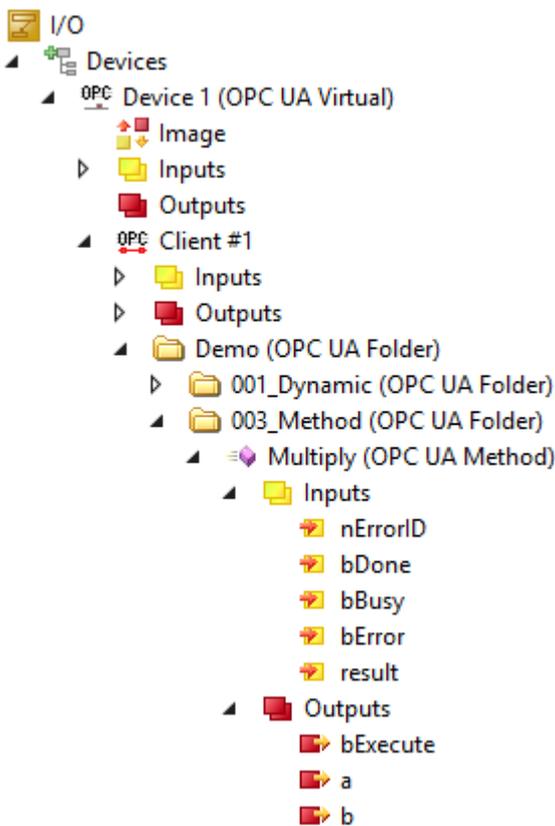
4.3.6 Methodenaufruf

Der OPC UA I/O Client unterstützt den Aufruf von Servermethoden. Sie können eine Methode zu dem Prozessabbild wie jede andere Variable hinzufügen. Die „Eingangsargumente“ der Methode sind dann als Ausgangsvariable im Prozessabbild verfügbar, wohingegen die „Ausgangsargumente“ als Eingangsvariablen hinzugefügt werden. Zusätzliche Eingangs- und Ausgangsvariablen, z. B. bExecute, bBusy, bError, werden zu dem Prozessabbild hinzugefügt, so dass die Methode aufgerufen werden kann.

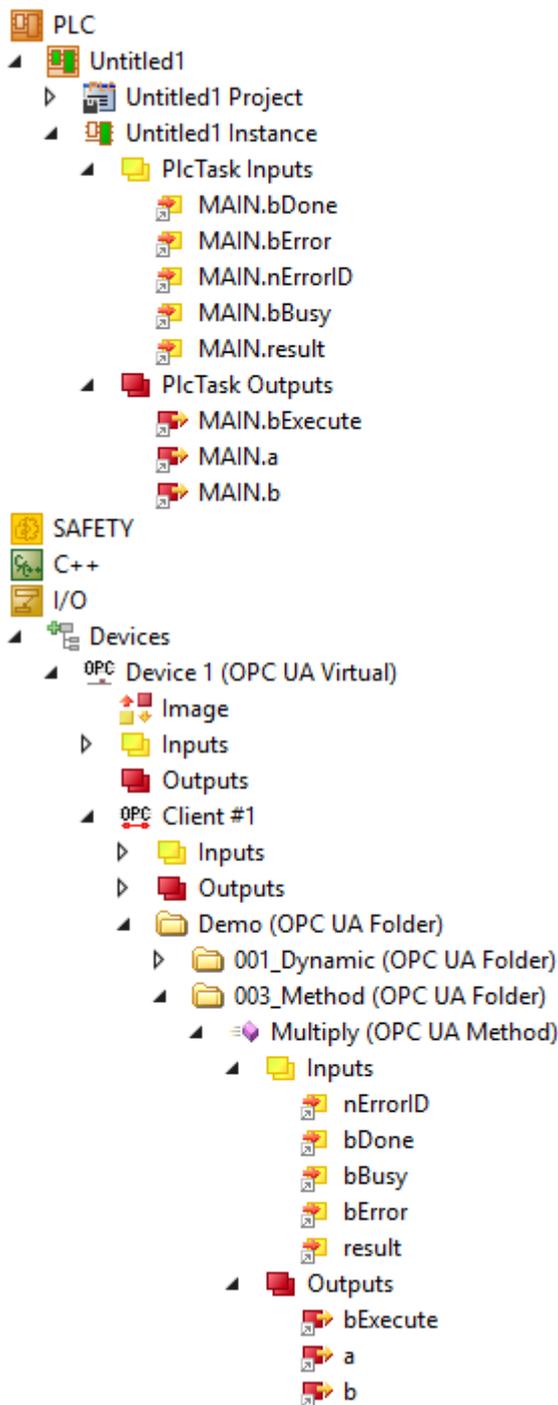
Beispiel: Methode auf Server



Beispiel: Methode nach dem Hinzufügen zum Prozessabbild



Sie können dann ein Mapping zwischen den Eingangs-/Ausgangsvariablen und den SPS-Variablen erstellen.



Aufruf einer Methode

Um eine Methode aufzurufen, setzen Sie die Ausgangsvariable bExecute auf TRUE. Über die Eingangsvariablen nErrorID, bDone, bBusy, bError können Sie prüfen, ob ein Methodenaufruf abgeschlossen wurde und erfolgreich war.

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

4.3.7 StructuredTypes

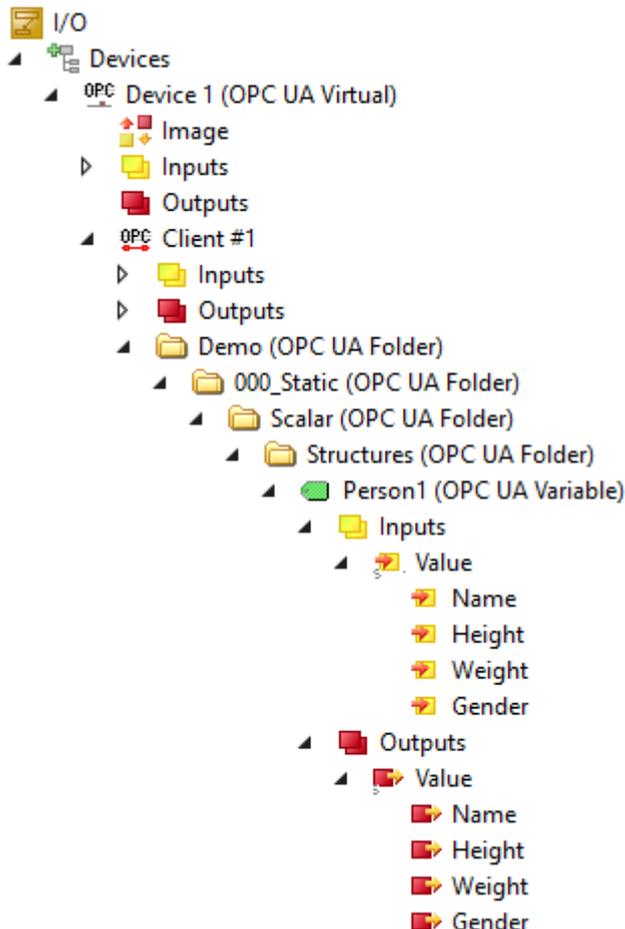
Der OPC UA I/O Client unterstützt Lese-/Schreibvorgänge bei strukturierten Datentypen (StructuredTypes). StructuredTypes können Sie dem Prozessabbild wie jede andere Variable auch hinzufügen. Beim Hinzufügen eines StructuredTypes zum Prozessabbild wird der zu parsende Typ dem Typsystem von TwinCAT hinzugefügt, um z. B. einfach von einer SPS-Anwendung verwendet zu werden.

Beispiel: StructuredType auf dem 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 diesem Beispiel enthält der Server einen Knoten des strukturierten Datentyps „Person“, der verschiedene Membervariablen enthält (Name, Height, Weight, Gender).

Beispiel: StructuredTypes im Prozessabbild



Nachdem Sie einen Knoten dem Prozessabbild hinzugefügt haben, enthält das Prozessabbild den Knoten und die strukturellen Informationen des Typs, z. B. ob eine einzige Membervariable des Knotens gelesen oder geschrieben werden soll.

StructuredTypes im Typsystem von TwinCAT

Der Datentyp wird dem Typsystem von TwinCAT hinzugefügt. Die „Value“ Tree Items haben dann diesen Datentyp.

Variable	Flags	Online	
Name:	Value		
Type:	Person ({3DC9DB7C-DA21-4069-A16A-EC995789785E})		
Group:	Inputs	Size:	91.0
Address:	10 (0xA)	User ID:	0

Sie können den Datentyp auch im Typsystem von TwinCAT unter SYSTEM > Type System einsehen.

Data Types	Interfaces	Functions	Event Classes	
Name	Namespace	GUID	Size	Type
Person		3DC9DB7...	91	Struct

Zur Unterscheidung des Datentyps von anderen Datentypen können Sie in den Einstellungen des OPC UA Clients ein Präfix hinzufügen.

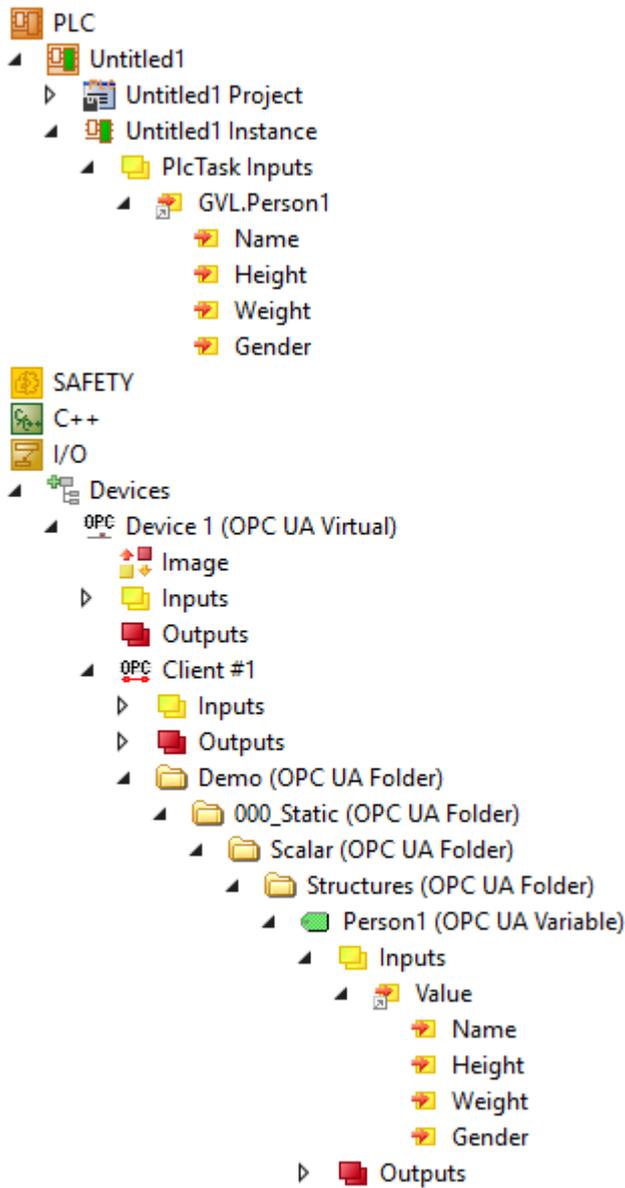
DataType Settings	
Name Prefix:	OPC_
String Size:	80
	Update

Mapping eines StructuredType

Da jeder StructuredType dem Typsystem von TwinCAT hinzugefügt wird, ist das Mapping der Variablen einfach. Erstellen Sie eine Eingangs-/Ausgangsvariable dieses Datentyps und anschließend ein Mapping.

```

GVL  ▸ ×
1   {attribute 'qualified_only'}
2   VAR_GLOBAL
3     Person1 AT%I*   : Person;
4   END_VAR
    
```



MAIN [Online] ✕		
TwinCAT_Project5.Untitled1.MAIN		
Expression	Type	Value
Person1	OpcUa_Person	
Name	STRING	'John Wayne'
Height	UINT	193
Weight	REAL	77
Gender	OPCUA_GENDER	Male

4.3.8 Datenaufnahme

Die Datenaufnahme wird jeweils auf Gerätebasis konfiguriert. Die Konfiguration wird mit einem Doppelklick auf das OPC UA-Client-Gerät geöffnet. Der Bereich **Process Data Configuration** zeigt verschiedene Parameter, um die verschiedenen Modi der Datenerfassung von einem Server zu beeinflussen.

Das OPC UA-Client-Gerät bietet drei verschiedene Modi zur Datenaufnahme an. Dazu gehören neben dem zyklischen Auslesen der Daten das Auslesen über eine Trigger-Variable und das Nutzen von Subscriptions.

Zyklisches Lesen/Schreiben

Eine der möglichen Arten der Datenaufnahme ist das zyklische Lesen und Schreiben. Dabei werden sowohl für das Lesen als auch für das Schreiben Zeitintervalle festgelegt. Außerdem kann festgelegt werden, wie viele Variablen in einem Lesebefehl gelesen werden sollen.

● Schreiben von Variablen im Polling- und Subscription-Modus

I Beim Schreiben ist zu beachten, dass nur bei Werteänderung geschrieben wird. Wenn nach Ablauf eines Zyklus keine Werteänderung in den konfigurierten Variablen stattgefunden hat, wird kein neuer Wert geschrieben.

Parameter	Beschreibung
Read Cycle Time	Spezifiziert, wie schnell Variablen zyklisch gelesen werden.
Write Cycle Time	Legt fest, wie häufig ein Schreibbefehl auf dem OPC UA-Kanal ausgelöst wird. Wenn sich ein Variablenwert innerhalb einer spezifizierten Zykluszeit mehrfach ändert, wird nur der letzte Wert auf den OPC UA-Kanal geschrieben. Wenn sich kein konfigurierter Wert in der Zykluszeit geändert hat, wird kein Schreibbefehl ausgelöst.
ReadList	Lesebefehle auf dem OPC UA-Kanal werden gebündelt, um Bandbreite einzusparen. Dieser Parameter spezifiziert, wie viele Variablen in einen einzigen Lesebefehl auf dem OPC UA-Kanal aufgenommen werden. Die Beschriftung dahinter gibt an, wie viele Lesebefehle sich aus der aktuellen Konfiguration ergeben.
Array Single Write	Bei Aktivierung wird bei Änderungen eines Wertes in einem Array ein Schreibvorgang nur für diesen Wert auf dem OPC UA-Kanal ausgeführt. Bei Nicht-Aktivierung wird immer das gesamte Array geschrieben.

Lesen und Schreiben über Trigger-Variablen

Außerdem gibt es die Möglichkeit, das Lesen und Schreiben über Trigger-Variablen auszulösen. Für jedes OPC UA-Client-Gerät gibt es eine Trigger-Variable (zu finden unter Outputs/Control/Execute), die mit einer Variablen aus der SPS verbunden und bei Bedarf gesetzt werden kann. Diese Möglichkeit eignet sich zum Beispiel, wenn das Lesen von Daten von einem OPC UA-Server erst bei einem bestimmten Ereignis in der SPS erfolgen soll. Bleibt die Trigger-Variable dauerhaft gesetzt, verhält sich die Art der Datenaufnahme so wie die zyklische Konfiguration.

Beim Schreiben hingegen wird bei gesetzter Trigger-Variable in jedem Zyklus ein Wert geschrieben. Hierbei wird keine Werteänderung betrachtet.

Lesen und Schreiben mithilfe von Subscriptions

Die dritte und letzte Möglichkeit der Datenaufnahme ist das Nutzen von Subscriptions. Dabei meldet der I/O-Client beim verbundenen OPC UA-Server eine Subscription an. Spezifizieren lassen sich die unten beschriebenen Parameter für Publish Interval, Lifetime Count und Keepalive Count.

Der Subscription-Modus ist vor allem für das Lesen von Variablen gedacht. Wenn man in diesem Modus Werte schreibt, gilt das gleiche Verhalten wie für das zyklische Schreiben (siehe oben).

Process Data Configuration

Data Collection: Subscriptions

Publish Interval: 1000 ms

Lifetime Count: 1200 (20m)

Keepalive Count: 1280

Parameter	Beschreibung
Publish Interval	Nach der angegebenen Zeit überprüft der verbundene OPC UA-Server, ob neue Benachrichtigungs-Pakete für den Client vorliegen. Sollten in einem Publishing Interval mehrere Wertänderungen auftreten, wird trotzdem nur der letzte Wert übertragen.
Lifetime Count	Der OPC UA-Client ist dafür verantwortlich, einen PublishRequest an den Server zu schicken. In der PublishResponse schickt der Server die jeweiligen Benachrichtigungspakete zurück. Der Lifetime Count gibt an, nach wie vielen nicht erhaltenen PublishRequests des Clients der Server die Subscription löscht. In Klammern steht die berechnete Zeitdauer (im Beispiel 1200 multipliziert mit 1000ms = 20 Minuten).
Keepalive Count	Wenn der Server keine neuen Benachrichtigungs-Pakete für den Client hat, schickt er keine Daten zurück. Der Keepalive Count gibt an, nach wie vielen ausgelassenen Nachrichten der Server eine leere Nachricht an den Client schicken würde, um mitzuteilen, dass er noch aktiv ist und die Subscription noch besteht.

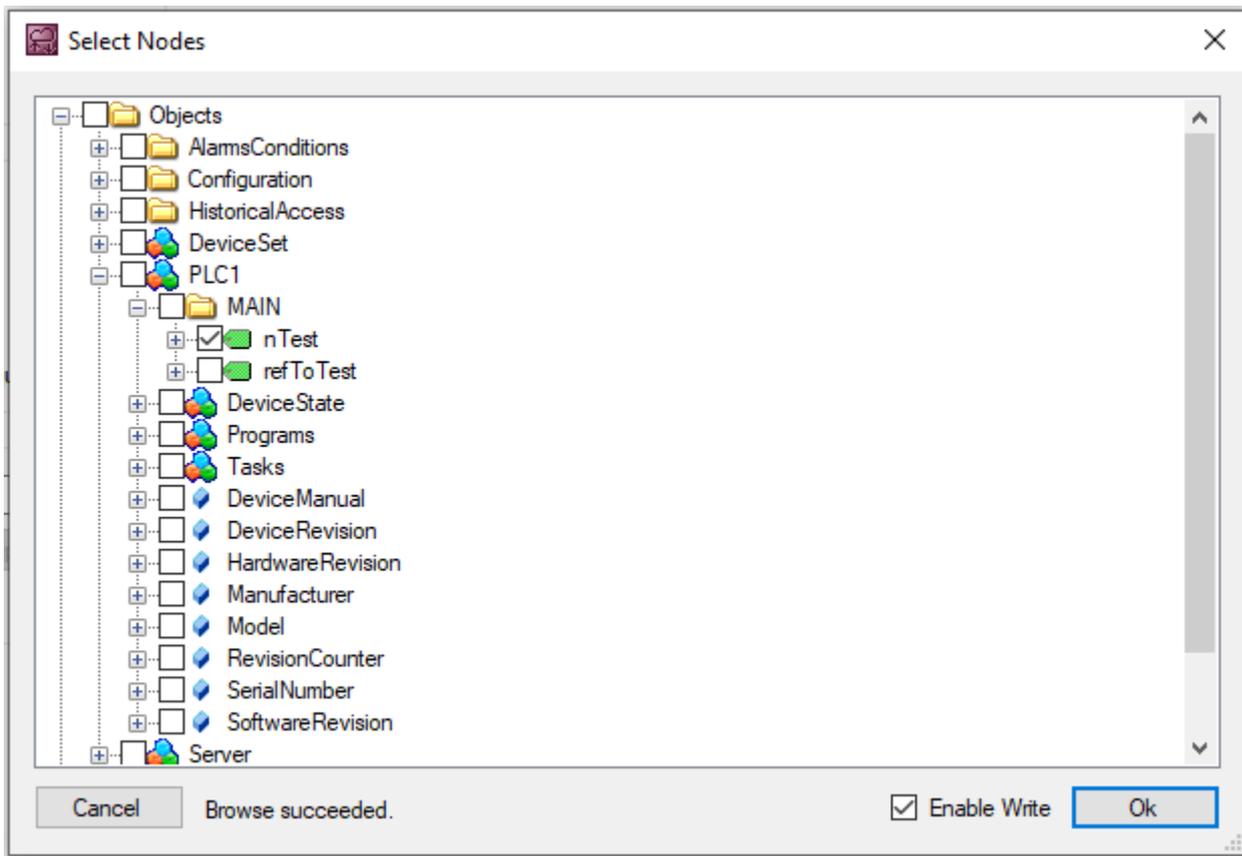
4.3.9 Schreiben von Variablen

Um das Schreiben von Variablen zu aktivieren, müssen mehrere Voraussetzungen erfüllt sein:

1. An der Variablen muss das Flag "Enable Write" gesetzt sein. Dies kann entweder während des Hinzufügens über den Button **Add Nodes** erfolgen oder nachträglich in den Parametereinstellungen der Variablen.
2. Vor einem Schreibkommando muss der Ausgang "Write Enable" am I/O Client global aktiviert werden. Nur dann werden die Schreibkommandos erzeugt.
3. In den Modi „Polling“ und „Subscriptions“ wird nur nach Werteänderung innerhalb des I/O-Clients geschrieben. Das ist vor allem bei Server-Neustarts zu beachten. Nach einem Server-Neustart werden einmal geschriebene Werte in diesen Modi nicht automatisch nochmal geschrieben, da in der Zwischenzeit ein anderer OPC UA-Client einen neuen Wert geschrieben haben könnte und dieser dann von einem „alten“ Wert überschrieben wird.

Setzen von Enable Write an einer Variablen

Damit für eine Variable nicht nur ein Input (Read), sondern auch ein Output (Write)-Element im Prozessabbild hinzugefügt wird, muss dieses explizit aktiviert werden. Dies kann zum Beispiel über den **Add Nodes**-Dialog schon während des Hinzufügens der Variablen erfolgen:

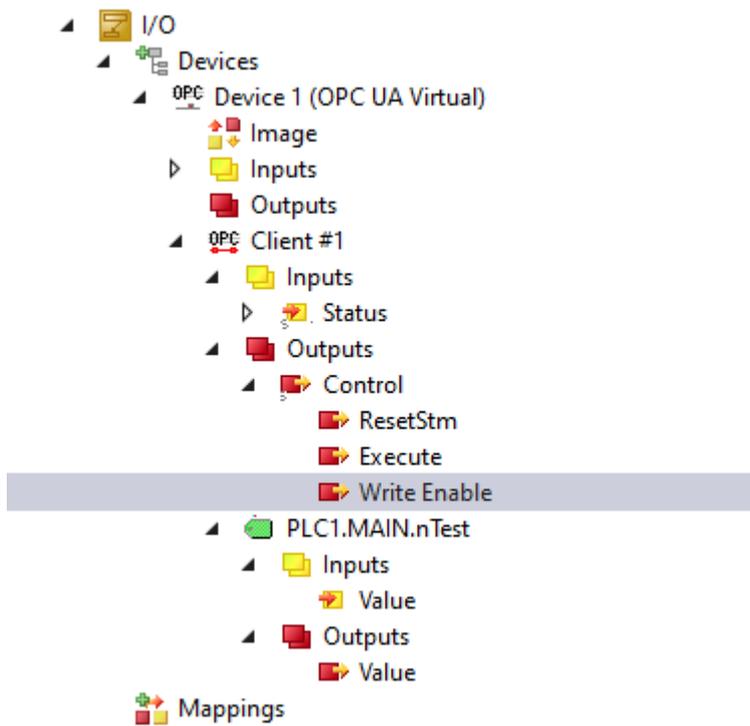


Alternativ kann diese Einstellung auch nachträglich noch über die Konfigurationsparameter der Variablen im Prozessabbild aktiviert/deaktiviert werden.

Attributes	
NodeID:	ns=4;s=MAIN.nTest
NsName:	um:BeckhoffAutomation:Ua:PLC1
<input checked="" type="checkbox"/> Enable Write <input type="checkbox"/> Provide timestamp and status code variables	
Name	Value
NodeID	ns=4;s=MAIN.nTest
NodeClass	2
BrowseName	4.nTest
DisplayName	nTest
Description	
WriteMask	0
UserWriteMask	0
Value	0
Data Type	i=4
ValueRank	-1
ArrayDimensions	
AccessLevel	3
UserAccessLevel	3
MinimumSamplingInterval	0
Historizing	False

Globales Aktivieren des Schreibzugriffs

Vor dem Absenden von Schreibkommandos müssen diese global freigeschaltet werden. Dies erfolgt durch das Setzen der Ausgangsvariablen "Write Enable" am I/O Client:



4.3.10 Sicherheit

4.3.10.1 Übersicht

Einer der Gründe für den Erfolg von OPC UA als Kommunikationstechnologie sind die verschiedenen, integrierten Sicherheitsmechanismen. Eine auf OPC UA basierte Datenkommunikation lässt sich auf zwei Ebenen absichern:

1. Transportebene
2. Applikationsebene

Endpunkte

Ein Server bietet dem Client eine Liste mit verschiedenen Endpunkten [► 105] an mit denen sich der Client verbinden kann. Ein Endpunkt beschreibt hierbei unter Anderem welche Sicherheitsfunktionen (z. B. Message Security Mode, Security Policy und zur Verfügung stehende Identity Token) die Kommunikationsverbindung über diesen Endpunkt erfüllen soll. So kann ein Endpunkt z. B. eine Signierung und Verschlüsselung der Datenpakete erfordern (Transportebene), sowie eine zusätzliche Authentifizierung des Clients auf Basis von Benutzername/Password (Applikationsebene).

Transportebene

Eine auf OPC UA basierte Kommunikationsverbindung kann auf Transportebene abgesichert werden. Dies geschieht durch die Verwendung von Client/Server Zertifikaten und eine gegenseitige Vertrauensstellung zwischen Client- und Serverapplikation. Hierbei muss der Client dem Server-Zertifikat vertrauen und umgekehrt, damit eine Kommunikationsverbindung hergestellt werden kann. Hierfür ist ein gegenseitiger Zertifikatsaustausch [► 177] notwendig.

Applikationsebene

Zusätzlich zur Transportebene lässt sich eine Kommunikationsverbindung auch auf Applikationsebene absichern. Hierfür stehen verschiedene Authentifizierungsmechanismen [► 106] zur Verfügung, die vom Serverendpunkt angeboten werden.

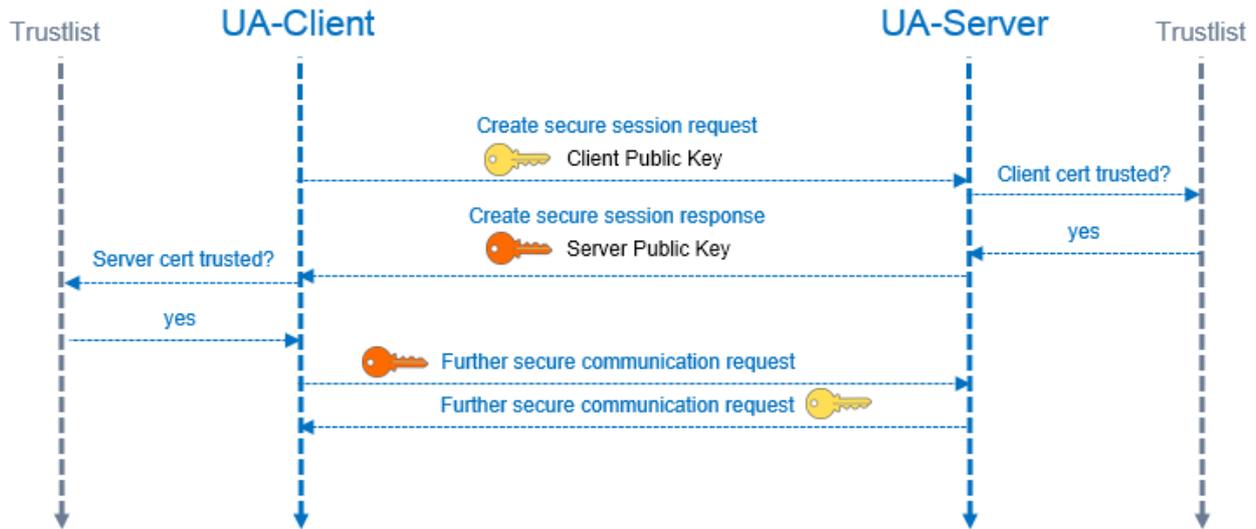
Sehen Sie dazu auch

Zugriffsrechte [▶ 109]

4.3.10.2 Zertifikatsaustausch

Für eine Absicherung der Kommunikationsverbindung auf Transportebene über einen sicheren Endpunkt [▶ 105] ist die Herstellung einer gegenseitigen Vertrauensstellung zwischen Client und Server notwendig.

Standardmäßig generieren sowohl der TwinCAT OPC UA Server als auch der TwinCAT OPC UA Client beim ersten Start ein maschinenspezifisches, selbstsigniertes Zertifikat zur Authentifizierung der jeweiligen Applikation.



Vertrauensstellung auf dem Server einrichten

Zur Einrichtung einer Vertrauensstellung zwischen einem beliebigen OPC UA Client und dem TwinCAT OPC UA Server, benötigen Sie den öffentlichen Schlüssel des Clientzertifikats. Der Server muss diesem vertrauen. Dies kann zum Beispiel über das Dateisystem erfolgen. Der Server verwaltet die Vertrauenseinstellungen für Client-Zertifikate im Unterverzeichnis PKI.

- Vertrauenswürdige Zertifikate: %InstallDir%\Server\PKI\CA\trusted\certs
- Nich-Vertrauenswürdige Zertifikate: %InstallDir%\Server\PKI\CA\rejected\certs

Durch Verschieben von Client-Zertifikaten zwischen diesen Verzeichnissen können die Vertrauenseinstellungen entsprechend angepasst werden. Der öffentliche Schlüssel eines Clientzertifikats wird beim ersten Verbindungsversuch des Clients mit einem sicheren Endpunkt automatisch im oben genannten Verzeichnis für nicht-Vertrauenswürdige Zertifikate abgelegt. Durch das anschließende Verschieben des öffentlichen Schlüssels in das Verzeichnis für vertrauenswürdige Zertifikate, wird dem Client beim nächsten Verbindungsversuch vertraut.

i AutomaticallyTrustAllClientCertificates

Ist diese Option in der TcUaServerConfig.xml aktiviert, so vertraut der Server automatisch allen Clientzertifikaten. Diese werden in diesem Fall nicht in einem der oben genannten Verzeichnisse aufgelistet.

Vertrauensstellung auf dem Client einrichten

Abhängig vom verwendeten OPC UA Client müssen eventuell unterschiedliche Schritte vorgenommen werden, damit der OPC UA Client dem OPC UA Server vertraut. Typischerweise erfolgt bei Client-Applikationen mit grafischer Benutzeroberfläche ein Warnhinweis bei der ersten Verbindung mit dem Server, wobei das Server-Zertifikat dann als vertrauenswürdige eingestuft werden kann.

Die folgende Anweisung ist daher nur für den TwinCAT OPC UA Client gültig.

Der öffentliche Schlüssel des OPC UA Servers befindet sich als DER-Datei in dem folgenden Verzeichnis:
 %InstallDir%\Server\PKI\CA\own\certs

Kopieren Sie die Datei beim TwinCAT OPC UA Client in das entsprechende „Trusted“-Verzeichnis:
 %InstallDir%\Client\PKI\CA\trusted\certs

4.4 Client PLCopen

4.4.1 Übersicht

Der TwinCAT OPC UA Client bietet mehrere Möglichkeiten direkt aus der Steuerungslogik heraus mit einem oder mehreren OPC UA Servern zu kommunizieren. Zum einen steht eine OPC-UA-Schnittstelle direkt aus der SPS zur Verfügung, bei der über PLCopen genormte Funktionsbausteine eine Verbindung mit einem OPC UA Server initiiert werden kann. Zum anderen gibt es ein OPC-UA-Client-I/O-Gerät, das eine einfache, Mapping-basierte Schnittstelle bietet.

SPS-Funktionsbausteine

Die folgende Tabelle gibt einen Überblick über die angebotenen Funktionalitäten:

Funktionalität	Beschreibung	Relevante Funktionsbausteine
Connect/Disconnect	Herstellen und Trennen von Verbindungen zu OPC UA Servern.	UA_Connect UA_Disconnect
Polling (Read/Write)	Lesen und Schreiben von Variablen im UA-Namensraum im Polling-Modus. Beinhaltet keine Subscriptions.	UA_Connect UA_Disconnect UA_Read UA_Write UA_GetNamespaceIndex UA_NodeGetHandle UA_NodeReleaseHandle
Method-Call	Ausführen von Methoden im UA-Namensraum.	UA_Connect UA_Disconnect UA_MethodGetHandle UA_MethodReleaseHandle UA_MethodCall
Security	Herstellen einer verschlüsselten Verbindung zu einem OPC UA Server.	UA_Connect UA_Disconnect

Die Schnittstellen jedes Funktionsbausteins werden im Abschnitt [SPS API](#) [[▶ 216](#)] beschrieben.

I/O-Gerät

Weitere Informationen zum TwinCAT OPC UA Client I/O-Gerät finden Sie im Abschnitt [Client I/O > Schnelleinstieg](#) [[▶ 161](#)].

4.4.2 Unterstützte Datentypen

Der OPC UA Client ermöglicht einen Zugriff auf OPC UA Server direkt aus der Echtzeitlogik heraus. Um Daten zu lesen und zu schreiben, müssen die Datentypen beider Umgebungen zugeordnet werden (Mapping). Nachfolgend wird diese Zuordnung beschrieben.

Basisdatentypen

Die Zuordnung der Basisdatentypen wird in PLCopen OPC UA Information Model for IEC 61131-3 beschrieben.

OPC-UA-Datentyp	SPS-Datentyp
Boolean	BOOL
SByte	SINT
Byte	USINT
Int16	INT
Int32	DINT
String	STRING
USint	BYTE
Float	REAL
Double	LREAL
UInt16	UINT
UInt32	UDINT
Int64	LINT
UInt64	ULINT
DateTime	DT

Abgeleitete Datentypen

OPC UA definiert Basisdatentypen. Andere Datentypen sind davon abgeleitet.

Auf Client-Seite ist nur mit SPS-Datentypen Zugriff auf alle UA-Datentypen möglich. Datentypen, die von den Basisdatentypen abgeleitet sind, werden ab TcOpcUaClient Version 2.1.0.36 ebenfalls unterstützt.

Derzeit unterstützte Nicht-Basisdatentypen auf Client-Seite sind:

- Counter (UDINT in SPS nutzen)

Zugriff auf Arrays

Beim Anlegen eines Knotenhandle prüft das System die Zugriffsmöglichkeiten auf den Knoten. Weitere Prüfungen erfolgen beim Lesen und Schreiben.

Für den Zugriff auf Array-Knoten sowie Ein- und Ausgabeparameter von Methodenaufrufen müssen bestimmte Bedingungen erfüllt sein.

- Für den Zugriff auf Knoten: Array-Dimension und Datenlänge müssen beim Lesen und Schreiben zu den bereitgestellten Daten passen.
- Lesen von Zeichenketten: Sobald eine Zeichenkette die vorgegebene Länge für SPS-Zeichenketten überschreitet, schlägt der Lesevorgang fehl.
- Es werden nur Array-Dimensionen bis zu drei Ebenen unterstützt.

4.4.3 Best Practice

4.4.3.1 Wie Kommunikationsparameter zu bestimmen sind

Im Allgemeinen wird ein grafischen OPC UA Client verwendet, um die Attribute eines Knotens oder Methoden zu bestimmen, die zusammen mit den SPS-Funktionsbausteinen verwendet werden müssen, z. B.:

- [Node Identifier \[► 180\]](#)
- [Node Namespace Index und entsprechender Namespace URI \[► 180\]](#)
- [Node Data Type \[► 181\]](#)
- [Method Node ID und Object Node ID \[► 182\]](#)

Die folgende Dokumentation verwendet den generischen OPC UA Client UA Expert als Beispiel. Dieser Client kann von den Websites von Unified Automation erworben werden: www.unified-automation.com.

Node Identifier

Eine allgemeine Aufgabe besteht aus dem Lesen oder Schreiben von Variablen, die im Hinblick auf OPC UA im Allgemeinen als Knoten bezeichnet werden.

Knoten sind durch die folgenden drei Attribute gekennzeichnet:

- NamespaceIndex: Namensraum, in dem sich der Knoten befindet, z. B. die SPS-Laufzeit
- Identifier: Eindeutiger Bezeichner des Knotens innerhalb seines Namensraums
- IdentifierType: Typ des Knotens: String, Guid und Numeric

Diese Attribute stellen die sogenannte NodeID dar – die Darstellung eines Knotens auf einem OPC UA Server – und werden von manchen Funktionsbausteinen benötigt.

Mithilfe von UA Expert können Sie die Attribute eines Knotens einfach bestimmen, indem Sie eine Verbindung zum OPC UA Server aufbauen und zum gewünschten Knoten browsen. Die Attribute sind dann im Attributes-Panel sichtbar.

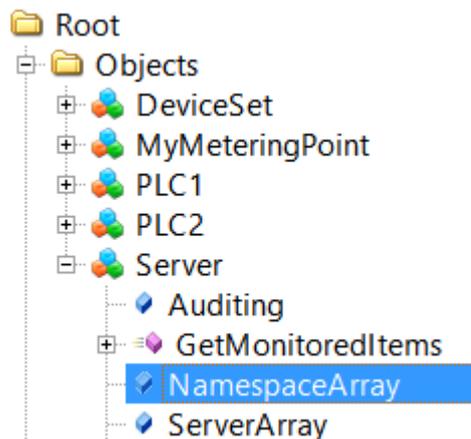
Beispiel:

NodellD	NodellD
NamespaceIndex	5
IdentifierType	String
Identifier	MAIN.nCounter

Node NamespaceIndex und entsprechender NamespaceURI

Nach der OPC-UA-Spezifikation kann der Namespace Index (wie vorstehend wiedergegeben) ein dynamisch generierter Wert sein. Daher müssen OPC UA Clients immer den entsprechenden NamespaceURI zur Auflösung des NamespaceIndex verwenden, bevor ein Knotenhandle erfasst wird.

Um den NamespaceIndex für einen NamespaceURI zu erfassen, verwenden Sie den Funktionsbaustein [UA_GetNamespaceIndex \[► 238\]](#). Den hierfür notwendigen NamespaceURI können Sie mithilfe von UA Expert bestimmen, indem Sie eine Verbindung zum OPC UA Server aufbauen und zum Knoten NamespaceArray browsen.



Dieser Knoten enthält Informationen über alle eingetragenen Namespaces auf dem OPC UA Server. Die entsprechenden NamespaceURIs sind im Attributes-Panel sichtbar.

Beispiel:

Value	
SourceTimestamp	16.02.2015 08:56:06.350
ServerTimestamp	16.02.2015 09:31:01.945
SourcePicoseconds	0
ServerPicoseconds	0
Value	String Array[9]
[0]	http://opcfoundation.org/UA/
[1]	urn:SvenG-NB04:BeckhoffAutomation:TcOpcUaServer:1
[2]	http://opcfoundation.org/UA/DI/
[3]	http://PLCopen.org/OpcUa/IEC61131-3/
[4]	urn://SVENG-NB04/BeckhoffAutomation/Ua/Typesystem
[5]	urn://SVENG-NB04/BeckhoffAutomation/Ua/PLC1
[6]	urn://SVENG-NB04/BeckhoffAutomation/Ua/PLC2
[7]	http://www.opcfoundation.org/Energy/DataAcquisition/
[8]	http://Beckhoff.com/TwinCAT/TF6100/Server/Configuration

Im Abschnitt [Node Identifier](#) [▶ 180] wird im Beispiel eine NodeID gezeigt, in der der NamespaceIndex 5 ist. Nach dem in der Abbildung gezeigten NamespaceArray ist der entsprechende NamespaceURI urn://SVENG-NB04/BeckhoffAutomation/Ua/PLC1. Dieser URI kann nun für den Funktionsbaustein UA_GetNamespaceIndex verwendet werden. Der OPC UA Server stellt sicher, dass der URI immer derselbe bleibt, auch nach einem Neustart. Da sich der gezeigte NamespaceIndex jedoch verändern kann, sollten für die spätere Nutzung mit anderen Funktionsbausteinen, z. B. [UA Read](#) [▶ 250], [UA Write](#) [▶ 253], zur Auflösung des korrekten NamespaceIndex immer der NamespaceURI in Kombination mit dem Funktionsbaustein UA_GetNamespaceIndex verwendet werden.

Node DataType

Der Datentyp eines Knotens ist erforderlich, um zu sehen, welcher SPS-Datentyp verwendet werden muss, um einen ausgelesenen Wert zuzuordnen oder um in einen Knoten zu schreiben. Mithilfe von UA Expert können Sie den Datentyp eines Knotens einfach bestimmen, indem Sie eine Verbindung zum OPC UA Server aufbauen und zum gewünschten Knoten browsen. Der Datentyp ist dann im Attributes-Panel sichtbar.

Beispiel:

DataType	Int16
NamespaceIndex	0
IdentifierType	Numeric
Identifier	4

In diesem Fall ist der Datentyp (DataType) „Int16“. Dieser muss einem äquivalenten Datentyp in der SPS zugeordnet werden, z. B. „INT“.

Die PLCopen IEC61131-zu-OPC-UA-Spezifikation beschreibt das definierte Datentyp-Mapping. Die folgende Tabelle ist ein Auszug aus dieser Spezifikation:

IEC61131 elementare Datentypen	OPC UAeingebaute Datentypen
BOOL	Boolean
SINT	SByte
USINT	Byte
INT	Int16
UINT	UInt16
DINT	Int32
UDINT	UInt32
LINT	Int64
ULINT	UInt64
BYTE	Byte
WORD	UInt16
DWORD	UInt32
LWORD	UInt64
REAL	Float
LREAL	Double
STRING	String
CHAR	Byte
WSTRING	String
WCHAR	UInt16
DT	DateTime
DATE_AND_TIME	
DATE	DateTime
TOD	DateTime
TIME_OF_DAY	
TIME	Double

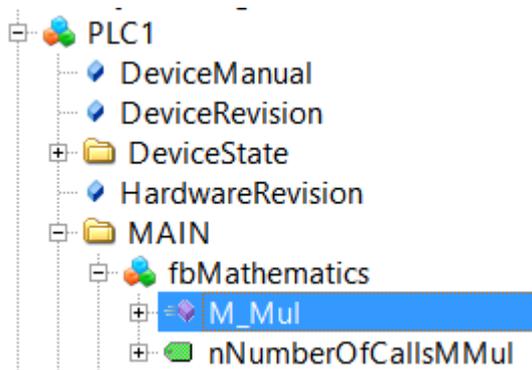
Method NodeID und Object NodeID

Beim Aufruf von Methoden aus dem OPC-UA-Namensraum sind zwei Identifier erforderlich, wenn der Methodenhandle unter Verwendung des Funktionsbausteins [UA_MethodGetHandle](#) [► 244] erfasst wird:

- ObjectNodeID: Identifiziert das UA-Objekt, das die Methode enthält
- MethodNodeID: Identifiziert die Methode selbst

Mithilfe von UA Expert können Sie beide NodeIDs einfach bestimmen, indem Sie eine Verbindung zum OPC UA Server aufbauen und zu der gewünschten Methode bzw. dem gewünschten UA-Objekt, das die Methode enthält, browsen.

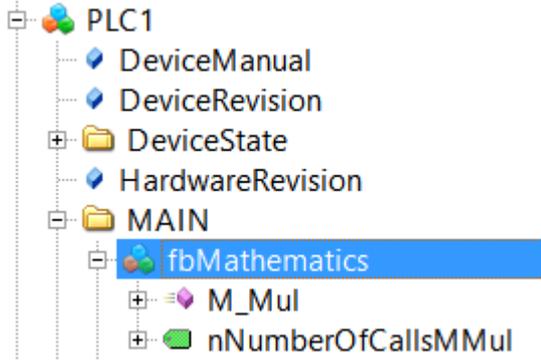
Beispiel Methode M_Mul:



Der Method Identifier ist dann im Attributes-Panel sichtbar.

NodId	NodId
NamespaceIndex	5
IdentifierType	String
Identifier	MAIN.fbMathematics#M_Mul

Beispiel Objekt fbMathematics:



Der Object Identifier ist dann im Attributes-Panel sichtbar.

NodId	NodId
NamespaceIndex	5
IdentifierType	String
Identifier	MAIN.fbMathematics

4.4.3.2 Wie eine Verbindung hergestellt wird

Im nachfolgenden Abschnitt wird beschrieben, wie Sie die Funktionsbausteine TcX_PLCOpen_OpcUa verwenden, um eine Verbindung zu einem lokalen oder remote OPC UA Server herzustellen. Diese Verbindung kann dann verwendet werden, um weitere Funktionalitäten aufzurufen, z. B. Knoten auslesen oder schreiben, oder Methoden aufrufen.

Der Abschnitt beinhaltet folgende Themen:

- [Übersicht \[► 183\]](#)
- [Schematischer Arbeitsablauf \[► 183\]](#)
- [Allgemeine Hinweise \[► 184\]](#)
- [Code-Ausschnitt \[► 184\]](#)

Übersicht

Die folgenden Funktionsbausteine sind erforderlich, um eine Verbindung zu einem OPC UA Server herzustellen und später die Sitzung zu unterbrechen: [UA_Connect \[► 235\]](#), [UA_Disconnect \[► 237\]](#).

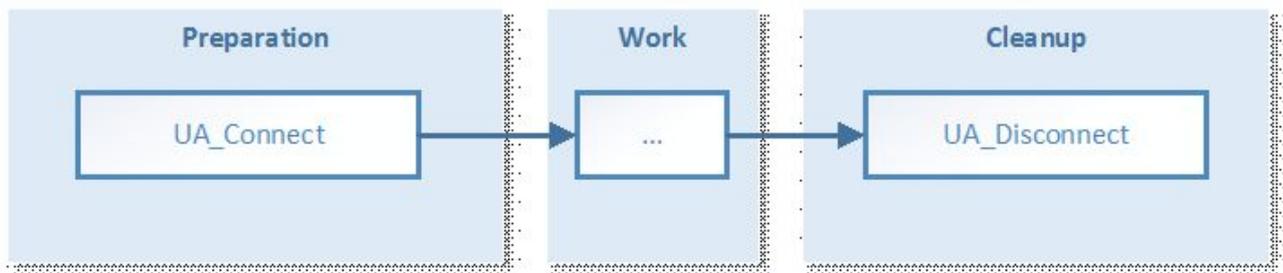


Lesen Sie zunächst den Abschnitt [Wie Kommunikationsparameter zu bestimmen sind \[► 179\]](#), um bestimmte UA-Funktionalitäten besser verstehen zu können (z. B. wie NodIdentifier bestimmt werden können).

Schematischer Arbeitsablauf

Der schematische Arbeitsablauf jedes TwinCAT OPC UA Client kann in drei verschiedene Phasen kategorisiert werden: Preparation, Work und Cleanup.

Der in diesem Abschnitt beschriebene Verwendungsfall kann wie folgt visualisiert werden:



Allgemeine Hinweise

Der Funktionsbaustein UA_Connect erfordert die folgenden Informationen, um eine Verbindung zu einem lokalen oder remote OPC UA Server herstellen zu können:

- Server URL
- Session Connect Information

Die Server URL besteht grundsätzlich aus einem Präfix, einem Hostnamen und einem Port. Das Präfix beschreibt das OPC-UA-Transportprotokoll, das für die Verbindung verwendet werden sollte, z. B. „opc.tcp://“ für eine binäre TCP-Verbindung (Standard). Der Hostname bzw. IP-Adressenteil beschreibt die Adressinformationen des OPC-UA-Zielservers, z. B. „192.168.1.1“ oder „CX-12345“. Die Portnummer ist der Zielport des OPC UA Servers, z. B. „4840“. Insgesamt kann die Server URL dann wie folgt aussehen: opc.tcp://CX-12345:4840.

Code-Ausschnitt

Deklaration:

```
(* 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;
```

Implementierung:

```
CASE iState OF
0:
  bError := FALSE;
  nErrorID := 0;
  SessionConnectInfo.tConnectTimeout := T#1M;
  SessionConnectInfo.tSessionTimeout := T#1M;
  SessionConnectInfo.sApplicationName := '';
  SessionConnectInfo.sApplicationUri := '';
  SessionConnectInfo.eSecurityMode := eUASecurityMsgMode_None;
  SessionConnectInfo.eSecurityPolicyUri := eUASecurityPolicy_None;
  SessionConnectInfo.eTransportProfileUri := eUATransportProfileUri_UATcp;
  stNodeAddInfo.nIndexRangeCount := nIndexRangeCount;
  stNodeAddInfo.stIndexRange := stIndexRange;
  iState := iState + 1;

1:
  fbUA_Connect (
    Execute := TRUE,
    ServerURL := 'opc.tcp://192.168.1.1:4840',
    SessionConnectInfo := SessionConnectInfo,
    Timeout := T#5S,
    ConnectionHdl => nConnectionHdl);
  IF NOT fbUA_Connect.Busy THEN
    fbUA_Connect(Execute := FALSE);
  IF NOT fbUA_Connect.Error THEN
    iState := iState + 1;
```

```

ELSE
  bError := TRUE;
  nErrorID := fbUA_Connect.ErrorID;
  nConnectionHdl := 0;
  iState := 0;
END_IF
END_IF

2:
fbUA_Disconnect(
  Execute := TRUE,
  ConnectionHdl := nConnectionHdl);

IF NOT fbUA_Disconnect.Busy THEN
  fbUA_Disconnect(Execute := FALSE);
IF NOT fbUA_Disconnect.Error THEN
  iState := 0;
ELSE
  bError := TRUE;
  nErrorID := fbUA_Disconnect.ErrorID;
  iState := 0;
  nConnectionHdl := 0;
END_IF
END_IF

END_CASE

```

4.4.3.3 Wie die Knoten auszulesen sind

Im nachfolgenden Abschnitt wird beschrieben, wie Sie die Funktionsbaustein TcX_PLCOpen_OpcUa verwenden, um einen OPC-UA-Knoten von einem lokalen oder remote OPC UA Server auszulesen.

Dieser Abschnitt beinhaltet folgende Themen:

- [Übersicht \[► 185\]](#)
- [Schematischer Arbeitsablauf \[► 185\]](#)
- [Allgemeine Hinweise \[► 186\]](#)
- [Code-Ausschnitt \[► 186\]](#)

Übersicht

Die folgenden Funktionsbausteine sind erforderlich, um eine Verbindung zu einem OPC UA Server herzustellen, UA-Knoten auszulesen und später die Sitzung zu unterbrechen: [UA_Connect \[► 235\]](#), [UA_GetNamespaceIndex \[► 238\]](#), [UA_NodeGetHandle \[► 246\]](#), [UA_Read \[► 250\]](#), [UA_NodeReleaseHandle \[► 248\]](#), [UA_Disconnect \[► 237\]](#).

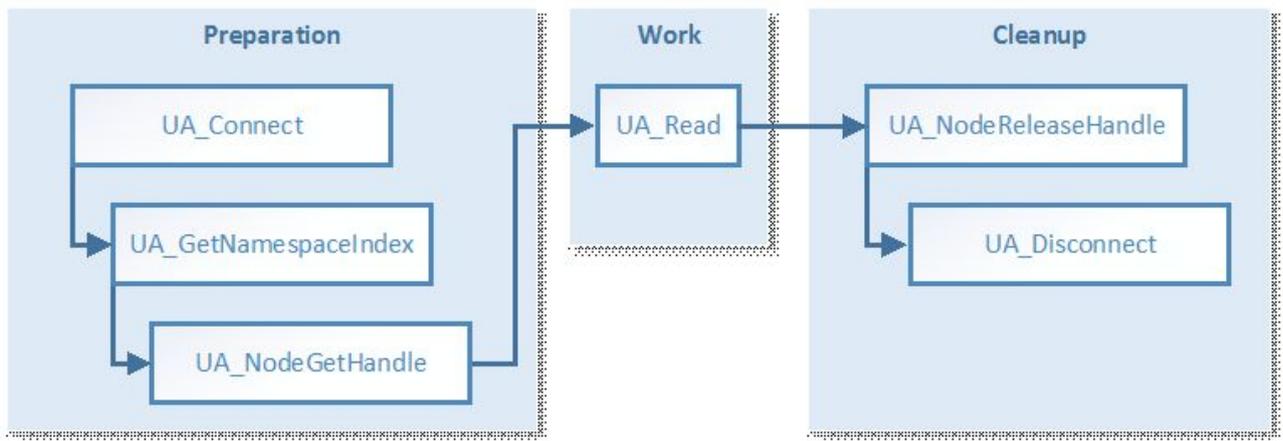


Lesen Sie zunächst den Abschnitt [Wie Kommunikationsparameter zu bestimmen sind \[► 179\]](#), um bestimmte UA-Funktionalitäten besser verstehen zu können (z. B. wie NodeIdentifier bestimmt werden können) sowie den Abschnitt [Wie eine Verbindung hergestellt wird \[► 183\]](#).

Schematischer Arbeitsablauf

Der schematische Arbeitsablauf jedes TwinCAT OPC UA Client kann in drei verschiedene Phasen kategorisiert werden: Preparation, Work und Cleanup.

Der in diesem Abschnitt beschriebene Verwendungsfall kann wie folgt visualisiert werden:



Allgemeine Hinweise

- Der Funktionsbaustein UA_Connect erfordert die folgenden Informationen, um eine Verbindung zu einem lokalen oder entfernten OPC UA Server herstellen zu können (siehe auch [Wie eine Verbindung hergestellt wird](#) [► 183]):
 - Server URL
 - Session Connect Information
- Der Funktionsbaustein UA_GetNamespaceIndex erfordert einen Connection Handle (von UA_Connect) und einen NamespaceURI zur Auflösung in einen NamespaceIndex, der später von UA_NodeGetHandle verwendet wird, um einen Knotenhandle zu erfassen (siehe auch [Wie Kommunikationsparameter zu bestimmen sind](#) [► 179]).
- Der Funktionsbaustein UA_NodeGetHandle erfordert einen Connection Handle (von UA_Connect) und die NodeID (von ST_UANodeID), um einen Knotenhandle zu erfassen (siehe auch [Wie Kommunikationsparameter zu bestimmen sind](#) [► 179]).
- Der Funktionsbaustein UA_Read erfordert einen Connection Handle (von UA_Connect), einen Knotenhandle (von UA_NodeGetHandle) und einen Zeiger zur Zielvariablen (wo der ausgelesene Wert gespeichert werden sollte). Stellen Sie dabei sicher, dass die Zielvariable den korrekten Datentyp aufweist (siehe auch [Wie Kommunikationsparameter zu bestimmen sind](#) [► 179]).
- Der Funktionsbaustein UA_NodeReleaseHandle erfordert einen Connection Handle (von UA_Connect) und einen Knotenhandle (von UA_NodeGetHandle).

Code-Ausschnitt

Deklaration:

```

(* 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;
  
```

Implementierung:

```

CASE iState OF
0:
  [...]

2: (* GetNS Index *)
  fbUA_GetNamespaceIndex(
    Execute := TRUE,
    ConnectionHdl := nConnectionHdl,
    NamespaceUri := sNamespaceUri,
    NamespaceIndex => nNamespaceIndex
  );
  IF NOT fbUA_GetNamespaceIndex.Busy THEN
    fbUA_GetNamespaceIndex(Execute := FALSE);
    IF NOT fbUA_GetNamespaceIndex.Error THEN
      iState := iState + 1;
    ELSE
      bError := TRUE;
      nErrorID := fbUA_GetNamespaceIndex.ErrorID;
      iState := 6;
    END_IF
  END_IF

3: (* UA_NodeGetHandle *)
  NodeID.eIdentifierType := eUAIdentifierType_String;
  NodeID.nNamespaceIndex := nNamespaceIndex;
  NodeID.sIdentifier := sNodeIdentifier;
  fbUA_NodeGetHandle(
    Execute := TRUE,
    ConnectionHdl := nConnectionHdl,
    NodeID := NodeID,
    NodeHdl => nNodeHdl);
  IF NOT fbUA_NodeGetHandle.Busy THEN
    fbUA_NodeGetHandle(Execute := FALSE);
    IF NOT fbUA_NodeGetHandle.Error THEN
      iState := iState + 1;
    ELSE
      bError := TRUE;
      nErrorID := fbUA_NodeGetHandle.ErrorID;
      iState := 6;
    END_IF
  END_IF

4: (* UA_Read *)
  fbUA_Read(
    Execute := TRUE,
    ConnectionHdl := nConnectionHdl,
    NodeHdl := nNodeHdl,
    cbData := SIZEOF(nReadData),
    stNodeAddInfo := stNodeAddInfo,
    pVariable := ADR(nReadData));
  IF NOT fbUA_Read.Busy THEN
    fbUA_Read(Execute := FALSE, cbData_R => cbDataRead);
    IF NOT fbUA_Read.Error THEN
      iState := iState + 1;
    ELSE
      bError := TRUE;
      nErrorID := fbUA_Read.ErrorID;
      iState := 6;
    END_IF
  END_IF

5: (* Release Node Handle *)
  fbUA_NodeReleaseHandle(
    Execute := TRUE,
    ConnectionHdl := nConnectionHdl,
    NodeHdl := nNodeHdl);
  IF NOT fbUA_NodeReleaseHandle.Busy THEN
    fbUA_NodeReleaseHandle(Execute := FALSE);
    IF NOT fbUA_NodeReleaseHandle.Error THEN
      iState := iState + 1;
    ELSE
      bError := TRUE;
      nErrorID := fbUA_NodeReleaseHandle.ErrorID;
      iState := 6;
    END_IF
  END_IF

6:

```

[...]

END_CASE

4.4.3.4 Wie Knoten zu schreiben sind

Im nachfolgenden Abschnitt wird beschrieben, wie Sie die Funktionsbausteine TcX_PLCOpen_OpcUa verwenden, um Werte in einem OPC-UA-Knoten von einem lokalen oder remote OPC UA Server zu schreiben.

Dieser Abschnitt beinhaltet folgende Themen:

- [Übersicht \[► 188\]](#)
- [Schematischer Arbeitsablauf \[► 188\]](#)
- [Allgemeine Hinweise \[► 188\]](#)
- [Code-Ausschnitt \[► 189\]](#)

Übersicht

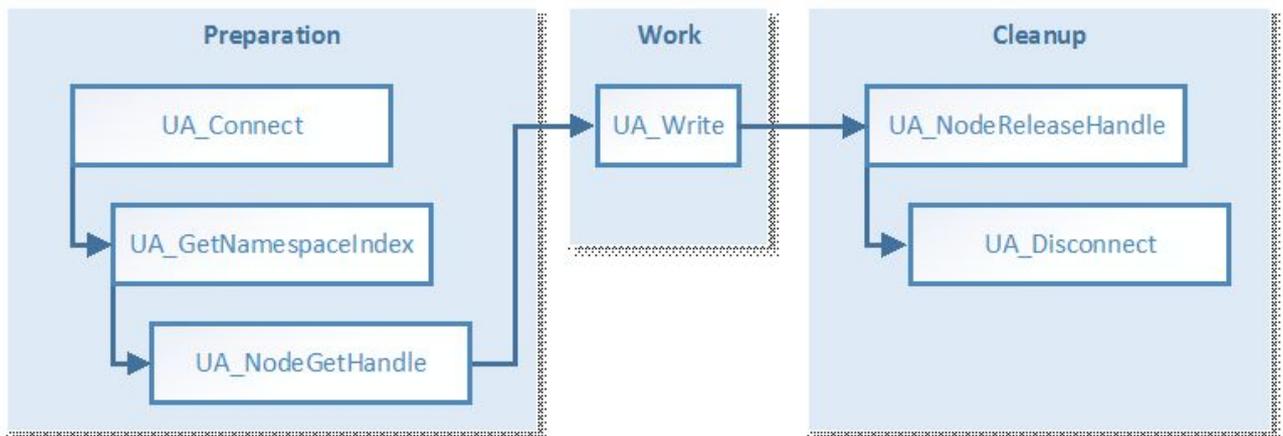
Die folgenden Funktionsbausteine sind erforderlich, um eine Verbindung zu einem OPC UA Server herzustellen, UA-Knoten zu schreiben und später die Sitzung zu unterbrechen: [UA_Connect \[► 235\]](#), [UA_GetNamespaceIndex \[► 238\]](#), [UA_NodeGetHandle \[► 246\]](#), [UA_Write \[► 253\]](#), [UA_NodeReleaseHandle \[► 248\]](#), [UA_Disconnect \[► 237\]](#).

i Lesen Sie zunächst den Abschnitt [Wie Kommunikationsparameter zu bestimmen sind \[► 179\]](#), um bestimmte UA-Funktionalitäten besser verstehen zu können (z. B. wie NodeIdentifier bestimmt werden können) sowie den Abschnitt [Wie eine Verbindung hergestellt wird \[► 183\]](#).

Schematischer Arbeitsablauf

Der schematische Arbeitsablauf jedes TwinCAT OPC UA Client kann in drei verschiedene Phasen kategorisiert werden: Preparation, Work und Cleanup.

Der in diesem Abschnitt beschriebene Verwendungsfall kann wie folgt visualisiert werden:



Allgemeine Hinweise

- Der Funktionsbaustein UA_Connect erfordert die folgenden Informationen, um eine Verbindung zu einem lokalen oder remote OPC UA Server herstellen zu können (siehe auch [Wie eine Verbindung hergestellt wird \[► 183\]](#)):
 - Server URL
 - Session Connect Information

- Der Funktionsbaustein UA_GetNamespaceIndex erfordert einen Connection Handle (von UA_Connect) und einen NamespaceURI zur Auflösung in einen NamespaceIndex, der später von UA_NodeGetHandle verwendet wird, um einen Knotenhandle zu erfassen (siehe auch Wie Kommunikationsparameter zu bestimmen sind [► 179]).
- Der Funktionsbaustein UA_NodeGetHandle erfordert einen Connection Handle (von UA_Connect) und die NodeID (von ST_UANodeID), um einen Knotenhandle zu erfassen (siehe auch Wie Kommunikationsparameter zu bestimmen sind [► 179]).
- Der Funktionsbaustein UA_Write erfordert einen Connection Handle (von UA_Connect), einen Knotenhandle (von UA_NodeGetHandle) und einen Zeiger zu einer Variablen, die den Wert enthält, der geschrieben werden soll. Stellen Sie dabei sicher, dass die Zielvariable den korrekten Datentyp aufweist (siehe auch Wie Kommunikationsparameter zu bestimmen sind [► 179]).
- Der Funktionsbaustein UA_NodeReleaseHandle erfordert einen Connection Handle (von UA_Connect) und einen Knotenhandle (von UA_NodeGetHandle).

Code-Ausschnitt

Deklaration:

```
(* 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;
```

Implementierung:

```
CASE iState OF
0:
  [...]

2: (* GetNS Index *)
  fbUA_GetNamespaceIndex(
    Execute := TRUE,
    ConnectionHdl := nConnectionHdl,
    NamespaceUri := sNamespaceUri,
    NamespaceIndex => nNamespaceIndex
  );
  IF NOT fbUA_GetNamespaceIndex.Busy THEN
    fbUA_GetNamespaceIndex(Execute := FALSE);
    IF NOT fbUA_GetNamespaceIndex.Error THEN
      iState := iState + 1;
    ELSE
      bError := TRUE;
      nErrorID := fbUA_GetNamespaceIndex.ErrorID;
      iState := 6;
    END_IF
  END_IF

3: (* UA_NodeGetHandle *)
  NodeID.eIdentifierType := eUAIdentifierType_String;
  NodeID.nNamespaceIndex := nNamespaceIndex;
  NodeID.sIdentifier := sNodeIdentifier;
  fbUA_NodeGetHandle(
    Execute := TRUE,
    ConnectionHdl := nConnectionHdl,
    NodeID := NodeID,
    NodeHdl => nNodeHdl);
  IF NOT fbUA_NodeGetHandle.Busy THEN
    fbUA_NodeGetHandle(Execute := FALSE);
```

```

    IF NOT fbUA_NodeGetHandle.Error THEN
        iState := iState + 1;
    ELSE
        bError := TRUE;
        nErrorID := fbUA_NodeGetHandle.ErrorID;
        iState := 6;
    END_IF
END_IF

4: (* UA Write *)
fbUA_Write(
    Execute := TRUE,
    ConnectionHdl := nConnectionHdl,
    NodeHdl := nNodeHdl,
    stNodeAddInfo := stNodeAddInfo,
    cbData := SIZEOF(nWriteData),
    pVariable := ADR(nWriteData));
IF NOT fbUA_Write.Busy THEN
    fbUA_Write(
        Execute := FALSE,
        pVariable := ADR(nWriteData));
    IF NOT fbUA_Write.Error THEN
        iState := iState + 1;
    ELSE
        bError := TRUE;
        nErrorID := fbUA_Write.ErrorID;
        iState := 6;
    END_IF
END_IF

5: (* Release Node Handle *)
fbUA_NodeReleaseHandle(
    Execute := TRUE,
    ConnectionHdl := nConnectionHdl,
    NodeHdl := nNodeHdl);
IF NOT fbUA_NodeReleaseHandle.Busy THEN
    fbUA_NodeReleaseHandle(Execute := FALSE);
    IF NOT fbUA_NodeReleaseHandle.Error THEN
        iState := iState + 1;
    ELSE
        bError := TRUE;
        nErrorID := fbUA_NodeReleaseHandle.ErrorID;
        iState := 6;
    END_IF
END_IF

6:
[... ]

END_CASE

```

4.4.3.5 Wie Methoden aufzurufen sind

Im nachfolgenden Abschnitt wird beschrieben, wie Sie die Funktionsbausteine TcX_PLCOpen_OpcUa verwenden, um Methoden auf einem lokalen oder remote OPC UA Server aufzurufen.

Dieser Abschnitt beinhaltet folgende Themen:

- [Übersicht \[► 190\]](#)
- [Schematischer Arbeitsablauf \[► 191\]](#)
- [Allgemeine Hinweise \[► 191\]](#)
- [Code-Ausschnitt \[► 191\]](#)

Übersicht

Die folgenden Funktionsbausteine sind erforderlich, um eine Verbindung zu einem OPC UA Server herzustellen, UA-Methoden aufzurufen und später die Sitzung zu unterbrechen: [UA Connect \[► 235\]](#), [UA GetNamespaceIndex \[► 238\]](#), [UA MethodGetHandle \[► 244\]](#), [UA MethodCall \[► 242\]](#), [UA MethodReleaseHandle \[► 245\]](#), [UA Disconnect \[► 237\]](#).

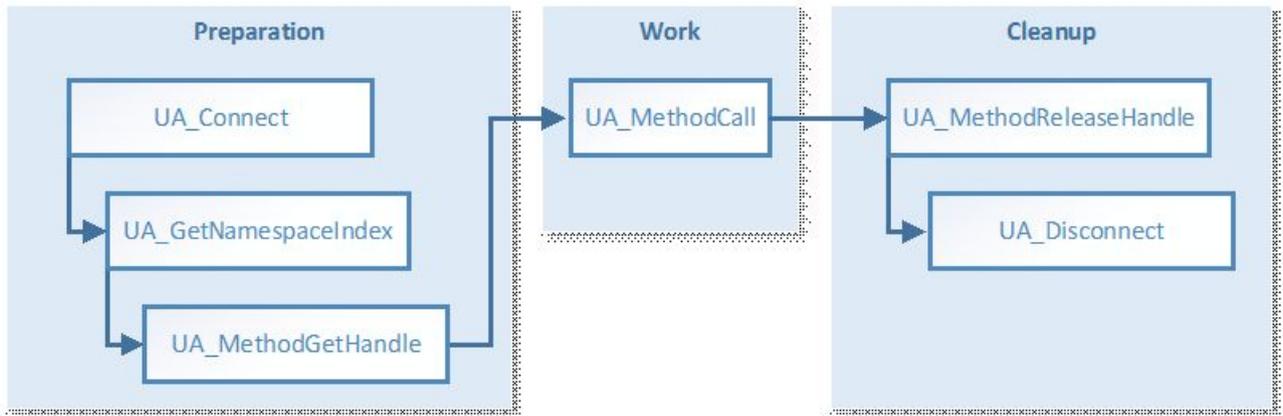


Lesen Sie zunächst den Abschnitt [Wie Kommunikationsparameter zu bestimmen sind](#) [► 179], um bestimmte UA-Funktionalitäten besser verstehen zu können (z. B. wie MethodIdentifier bestimmt werden können) sowie den Abschnitt [Wie eine Verbindung hergestellt wird](#) [► 183].

Schematischer Arbeitsablauf

Der schematische Arbeitsablauf jedes TwinCAT OPC UA Client kann in drei verschiedene Phasen kategorisiert werden: Preparation, Work und Cleanup.

Der in diesem Abschnitt beschriebene Verwendungsfall kann wie folgt visualisiert werden:



Allgemeine Hinweise

- Der Funktionsbaustein UA_Connect erfordert die folgenden Informationen, um eine Verbindung zu einem lokalen oder remote OPC UA Server herstellen zu können (siehe auch [Wie eine Verbindung hergestellt wird](#) [► 183]):
 - Server URL
 - Session Connect Information
- Der Funktionsbaustein UA_GetNamespaceIndex erfordert einen Connection Handle (von UA_Connect) und einen NamespaceURI zur Auflösung in einen NamespaceIndex, der später von UA_NodeGetHandle verwendet wird, um einen Knotenhandle zu erfassen (siehe auch [Wie Kommunikationsparameter zu bestimmen sind](#) [► 179]).
- Der Funktionsbaustein UA_MethodGetHandle erfordert einen Connection Handle (von UA_Connect), eine ObjectNodeID und eine MethodNodeID, um einen Methodenhandle zu erfassen (siehe auch [Wie Kommunikationsparameter zu bestimmen sind](#) [► 179]).
- Der Funktionsbaustein UA_MethodCall erfordert einen Connection Handle (von UA_Connect), einen Methodenhandle (von UA_MethodGetHandle) und Informationen über die Eingangs- und Ausgangsargumente der Methode, die aufgerufen werden soll. Informationen über die Eingangsargumente werden durch die Eingangsparameter pInputArgInfo und pInputArgData von UA_MethodCall repräsentiert. Information über die Ausgangsparameter werden durch die pOutputArgInfo und pOutputArgData Eingangsparameter von UA_MethodCall repräsentiert. Der Eingangsparameter pOutputArgInfoAndData stellt dann einen Zeiger zu einer Struktur dar, die die Ergebnisse des Methodenaufrufs enthält, einschließlich aller Ausgangsparameter. In dem nachfolgenden Code-Ausschnitt werden die pInputArgInfo und pInputArgData Parameter in der M_Init-Methode berechnet und erstellt.
- Der Funktionsbaustein UA_NodeReleaseHandle erfordert einen Connection Handle (von UA_Connect) und einen Methodenhandle (von UA_MethodGetHandle).

Code-Ausschnitt

Initialisierungsmethode M_Init des Funktionsbausteins, der den UA-Methodenaufruf enthält

```

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;

```

Deklaration:

```

(* 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;

```

Implementierung:

```

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
END_IF

```

```

3: (* Get Method Handle *)
ObjectNodeID.eIdentifierType := eUAIdentifierType_String;
ObjectNodeID.nNamespaceIndex := nNamespaceIndex;
ObjectNodeID.sIdentifier := sObjectNodeIdIdentifier;
MethodNodeID.eIdentifierType := eUAIdentifierType_String;
MethodNodeID.nNamespaceIndex := nNamespaceIndex;
MethodNodeID.sIdentifier := sMethodNodeIdIdentifier;

M_Init();

IF bInputDataError = FALSE THEN
  iState := iState + 1;
ELSE
  bBusy := FALSE;
  bError := TRUE;
  nErrorID := 16#70A; //out of memory
END_IF

4: (* Method Get Handle *)
fbUA_MethodGetHandle(
  Execute := TRUE,
  ConnectionHdl := nConnectionHdl,
  ObjectNodeID := ObjectNodeID,
  MethodNodeID := MethodNodeID,
  MethodHdl => nMethodHdl);
IF NOT fbUA_MethodGetHandle.Busy THEN
  fbUA_MethodGetHandle(Execute := FALSE);
  IF NOT fbUA_MethodGetHandle.Error THEN
    iState := iState + 1;
  ELSE
    bError := TRUE;
    nErrorID := fbUA_MethodGetHandle.ErrorID;
    iState := 6;
  END_IF
END_IF

5: (* Method Call *)
stOutputArgInfo[1].nLenData := SIZEOF(stOutputArgInfoAndData.pro);
fbUA_MethodCall(
  Execute := TRUE,
  ConnectionHdl := nConnectionHdl,
  MethodHdl := nMethodHdl,
  nNumberOfInputArguments := nNumberOfInputArguments,
  pInputArgInfo := ADR(InputArguments),
  cbInputArgInfo := SIZEOF(InputArguments),
  pInputArgData := ADR(nInputData),
  cbInputArgData := cbWriteData,
  pInputWriteData := 0,
  cbInputWriteData := 0,
  nNumberOfOutputArguments := nNumberOfOutputArguments,
  pOutputArgInfo := ADR(stOutputArgInfo),
  cbOutputArgInfo := SIZEOF(stOutputArgInfo),
  pOutputArgInfoAndData := ADR(stOutputArgInfoAndData),
  cbOutputArgInfoAndData := SIZEOF(stOutputArgInfoAndData));
IF NOT fbUA_MethodCall.Busy THEN
  fbUA_MethodCall(Execute := FALSE);
  IF NOT fbUA_MethodCall.Error THEN
    iState := iState + 1;
    numberOutPro := stOutputArgInfoAndData.pro;
  ELSE
    bError := TRUE;
    nErrorID := fbUA_MethodCall.ErrorID;
    iState := 6;
  END_IF
END_IF

6: (* Release Method Handle *)
fbUA_MethodReleaseHandle(
  Execute := TRUE,
  ConnectionHdl := nConnectionHdl,
  MethodHdl := nMethodHdl);
IF NOT fbUA_MethodReleaseHandle.Busy THEN
  fbUA_MethodReleaseHandle(Execute := FALSE);
  bBusy := FALSE;
  IF NOT fbUA_MethodReleaseHandle.Error THEN
    iState := 7;
  ELSE
    bError := TRUE;
    nErrorID := fbUA_MethodReleaseHandle.ErrorID;

```

```
        iState := 7;
    END_IF
END_IF

7:
    [...]

END_CASE
```

4.4.4 Sicherheit

4.4.4.1 Übersicht

Einer der Gründe für den Erfolg von OPC UA als Kommunikationstechnologie sind die verschiedenen, integrierten Sicherheitsmechanismen. Eine auf OPC UA basierte Datenkommunikation lässt sich auf zwei Ebenen absichern:

1. Transportebene
2. Applikationsebene

Endpunkte

Ein Server bietet dem Client eine Liste mit verschiedenen Endpunkten [► 105] an mit denen sich der Client verbinden kann. Ein Endpunkt beschreibt hierbei unter Anderem welche Sicherheitsfunktionen (z. B. Message Security Mode, Security Policy und zur Verfügung stehende Identity Token) die Kommunikationsverbindung über diesen Endpunkt erfüllen soll. So kann ein Endpunkt z. B. eine Signierung und Verschlüsselung der Datenpakete erfordern (Transportebene), sowie eine zusätzliche Authentifizierung des Clients auf Basis von Benutzername/Password (Applikationsebene).

Transportebene

Eine auf OPC UA basierte Kommunikationsverbindung kann auf Transportebene abgesichert werden. Dies geschieht durch die Verwendung von Client/Server Zertifikaten und eine gegenseitige Vertrauensstellung zwischen Client- und Serverapplikation. Hierbei muss der Client dem Server-Zertifikat vertrauen und umgekehrt, damit eine Kommunikationsverbindung hergestellt werden kann. Hierfür ist ein gegenseitiger Zertifikatsaustausch [► 194] notwendig.

Applikationsebene

Zusätzlich zur Transportebene lässt sich eine Kommunikationsverbindung auch auf Applikationsebene absichern. Hierfür stehen verschiedene Authentifizierungsmechanismen [► 106] zur Verfügung, die vom Serverendpunkt angeboten werden.

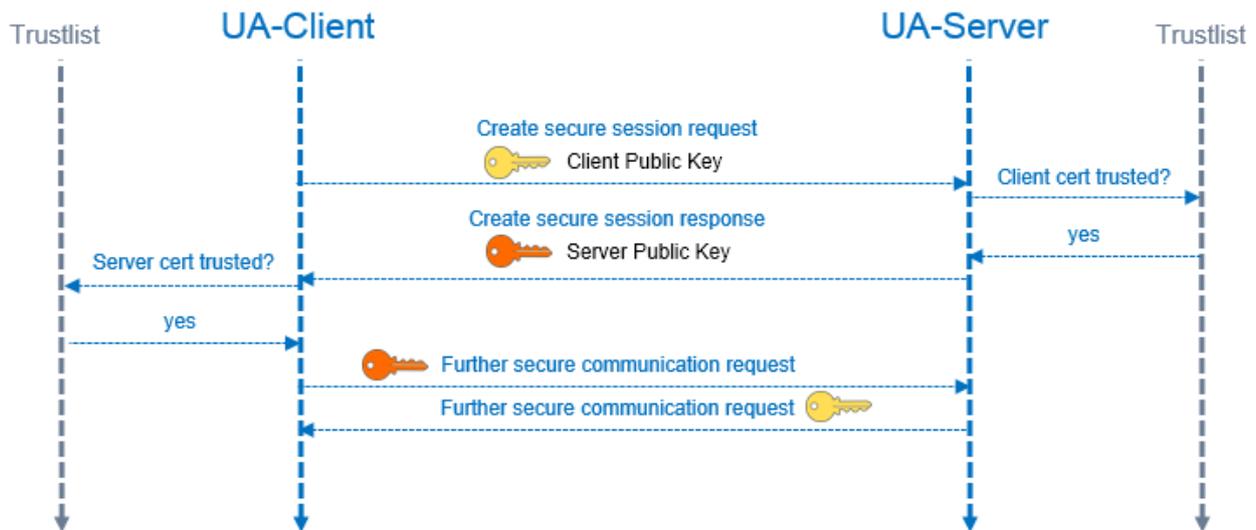
Sehen Sie dazu auch

- 📖 Zugriffsrechte [► 109]

4.4.4.2 Zertifikatsaustausch

Für eine Absicherung der Kommunikationsverbindung auf Transportebene über einen sicheren Endpunkt [► 105] ist die Herstellung einer gegenseitigen Vertrauensstellung zwischen Client und Server notwendig.

Standardmäßig generieren sowohl der TwinCAT OPC UA Server als auch der TwinCAT OPC UA Client beim ersten Start ein maschinenspezifisches, selbstsigniertes Zertifikat zur Authentifizierung der jeweiligen Applikation.



Vertrauensstellung auf dem Server einrichten

Zur Einrichtung einer Vertrauensstellung zwischen einem beliebigen OPC UA Client und dem TwinCAT OPC UA Server, benötigen Sie den öffentlichen Schlüssel des Clientzertifikats. Der Server muss diesem vertrauen. Dies kann zum Beispiel über das Dateisystem erfolgen. Der Server verwaltet die Vertrauenseinstellungen für Client-Zertifikate im Unterverzeichnis PKI.

- Vertrauenswürdige Zertifikate: %InstallDir%\Server\PKI\CA\trusted\certs
- Nicht-Vertrauenswürdige Zertifikate: %InstallDir%\Server\PKI\CA\rejected\certs

Durch Verschieben von Client-Zertifikaten zwischen diesen Verzeichnissen können die Vertrauenseinstellungen entsprechend angepasst werden. Der öffentliche Schlüssel eines Clientzertifikats wird beim ersten Verbindungsversuch des Clients mit einem sicheren Endpunkt automatisch im oben genannten Verzeichnis für nicht-Vertrauenswürdige Zertifikate abgelegt. Durch das anschließende Verschieben des öffentlichen Schlüssels in das Verzeichnis für vertrauenswürdige Zertifikate, wird dem Client beim nächsten Verbindungsversuch vertraut.

AutomaticallyTrustAllClientCertificates

Ist diese Option in der TcUaServerConfig.xml aktiviert, so vertraut der Server automatisch allen Clientzertifikaten. Diese werden in diesem Fall nicht in einem der oben genannten Verzeichnisse aufgelistet.

Vertrauensstellung auf dem Client einrichten

Abhängig vom verwendeten OPC UA Client müssen eventuell unterschiedliche Schritte vorgenommen werden, damit der OPC UA Client dem OPC UA Server vertraut. Typischerweise erfolgt bei Client-Applikationen mit grafischer Benutzeroberfläche ein Warnhinweis bei der ersten Verbindung mit dem Server, wobei das Server-Zertifikat dann als vertrauenswürdige eingestuft werden kann.

Die folgende Anweisung ist daher nur für den TwinCAT OPC UA Client gültig.

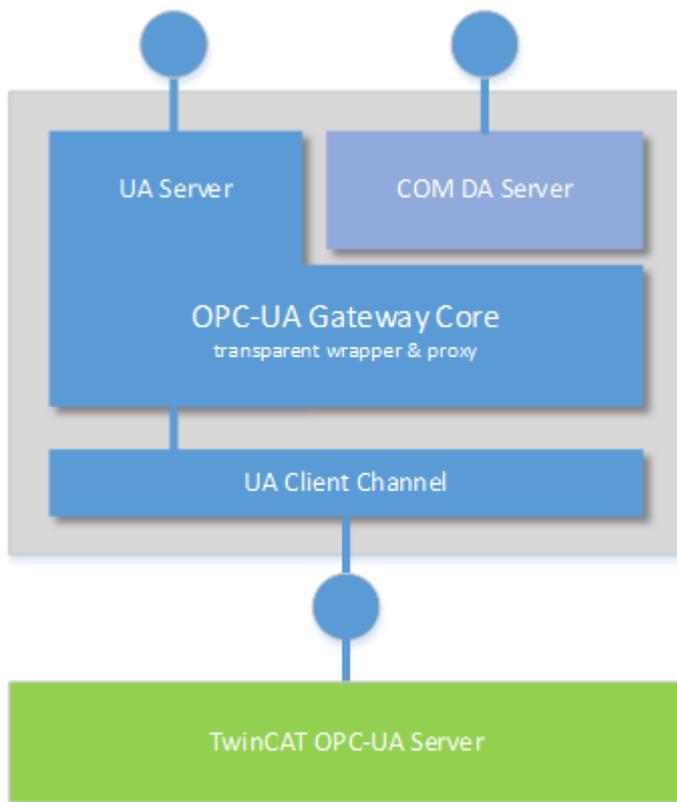
Der öffentliche Schlüssel des OPC UA Servers befindet sich als DER-Datei in dem folgenden Verzeichnis:
 %InstallDir%\Server\PKI\CA\own\certs

Kopieren Sie die Datei beim TwinCAT OPC UA Client in das entsprechende „Trusted“-Verzeichnis:
 %InstallDir%\Client\PKI\CA\trusted\certs

4.5 Gateway

4.5.1 Übersicht

Das TwinCAT OPC UA Gateway ist die neueste Erweiterung des TS6100/TF6100-Softwareprodukts. Es beinhaltet nicht nur eine herkömmliche OPC-DA-Schnittstelle, um ältere OPC-COM-DA-Anwendungen mit dem TwinCAT OPC UA Server zu verbinden und kann demzufolge als der Nachfolger des alten TwinCAT OPC DA Servers (TS6120/TF6120) betrachtet werden, sondern es bietet auch eine OPC-UA-Schnittstelle, um mehrere grundlegende TwinCAT OPC UA Server in einen zentralen OPC-UA-Serverkanal zu bündeln.



4.5.2 Schnelleinstieg

Das TwinCAT OPC UA Gateway steht als separates Setup zum Download zur Verfügung. Das Setup konfiguriert automatisch den Zugang zu einem TwinCAT OPC UA Server, der auf demselben Computer wie das Gateway läuft.

Wenn dem Gateway mehr als ein OPC UA Server hinzugefügt werden oder der Server auf einem anderen Computer läuft, müssen Änderungen an der Standardkonfiguration vorgenommen werden. Verwenden Sie den [Konfigurator \[► 200\]](#), um diese Einstellungen zu konfigurieren.

i Konfiguration des TwinCAT OPC UA Servers

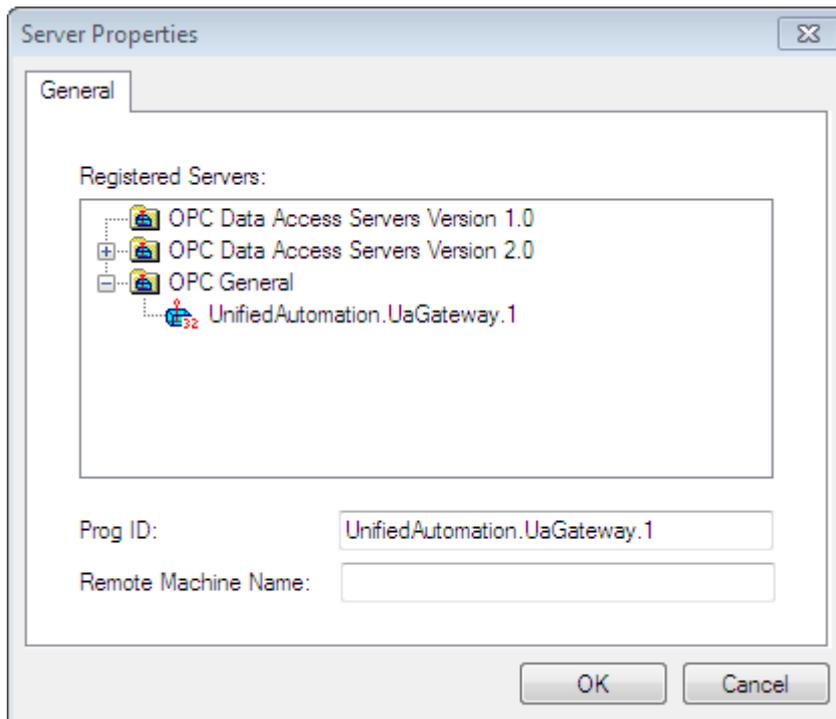
Prüfen Sie die Konfiguration des OPC UA Servers und vergewissern Sie sich, dass er wie erwartet arbeitet, bevor Sie fortfahren.

Für weitere Informationen bezüglich der Konfiguration des OPC UA Servers lesen Sie den [Schnelleinstieg \[► 35\]](#) im Abschnitt „OPC UA Server“.

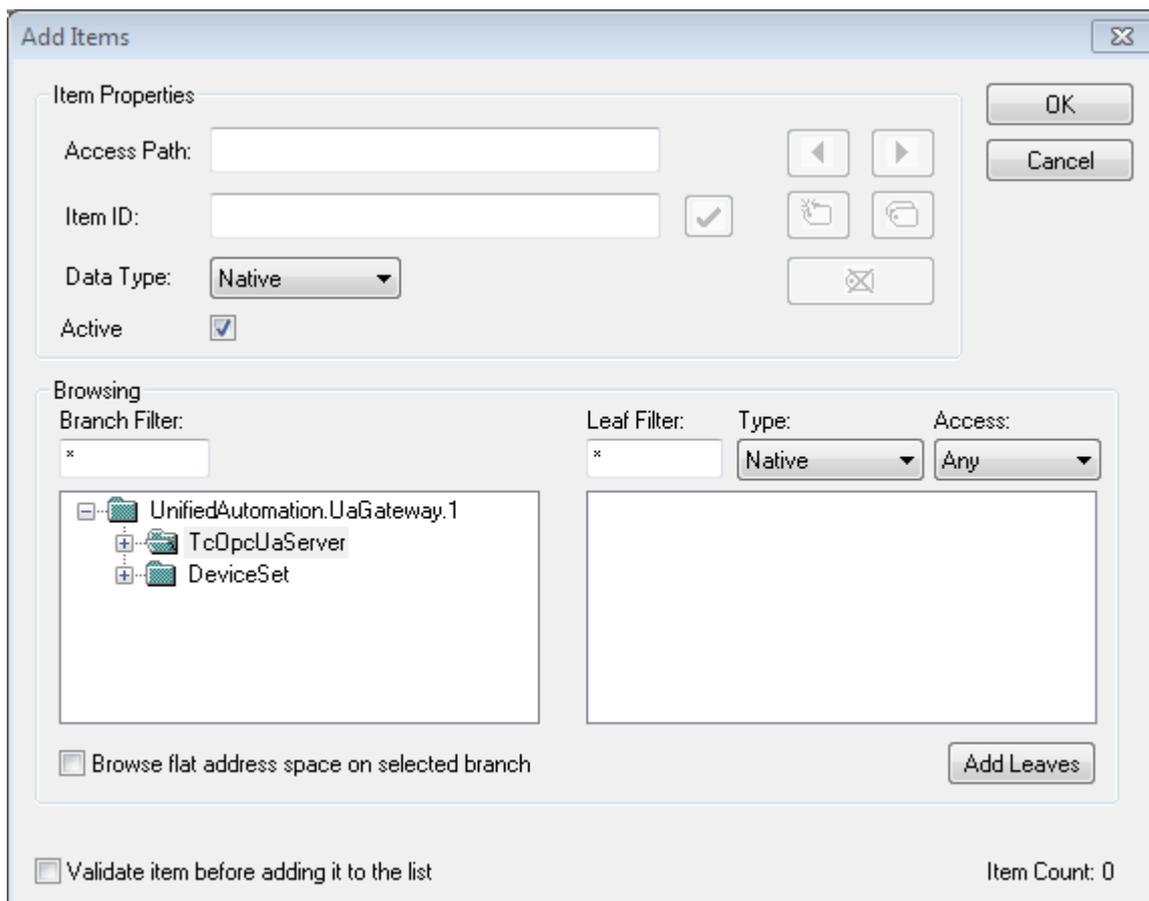
Schnelleinstieg OPC COM DA

Um einen OPC COM DA Client mit dem Gateway zu verbinden, starten Sie den Client und stellen Sie eine Verbindung zu der folgenden ProgId her:

UnifiedAutomation.UaGateway.1



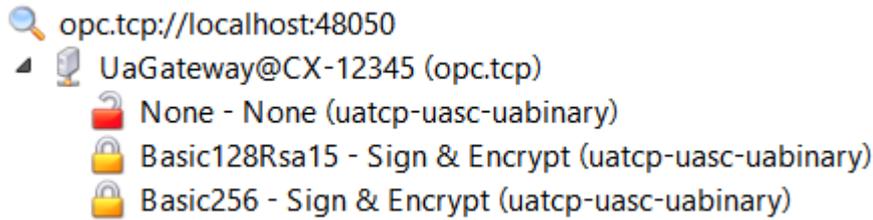
Beim Durchsuchen des Gateway werden ein oder mehrere OPC UA Server im Namensraum des Gateway sichtbar.



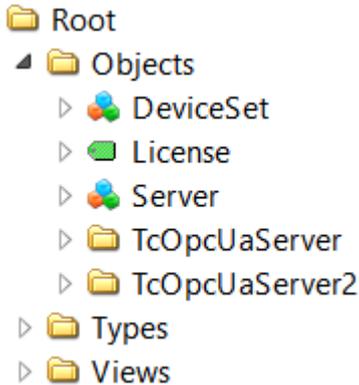
Schnelleinstieg OPC UA

Das Gateway bietet nicht nur eine OPC-COM-DA-Schnittstelle, sondern erlaubt die Aggregation von einem oder mehreren OPC UA Servern. Hierzu öffnet das Gateway ebenfalls eine OPC-UA-Schnittstelle. Das Gateway ist über folgende OPC UA Server URL erreichbar:

opc.tcp://[HostnameOrIpAddressOrLocalhost]:48050



Der Namensraum des Gateway beinhaltet dann alle zugrunde liegenden TwinCAT OPC UA Server.



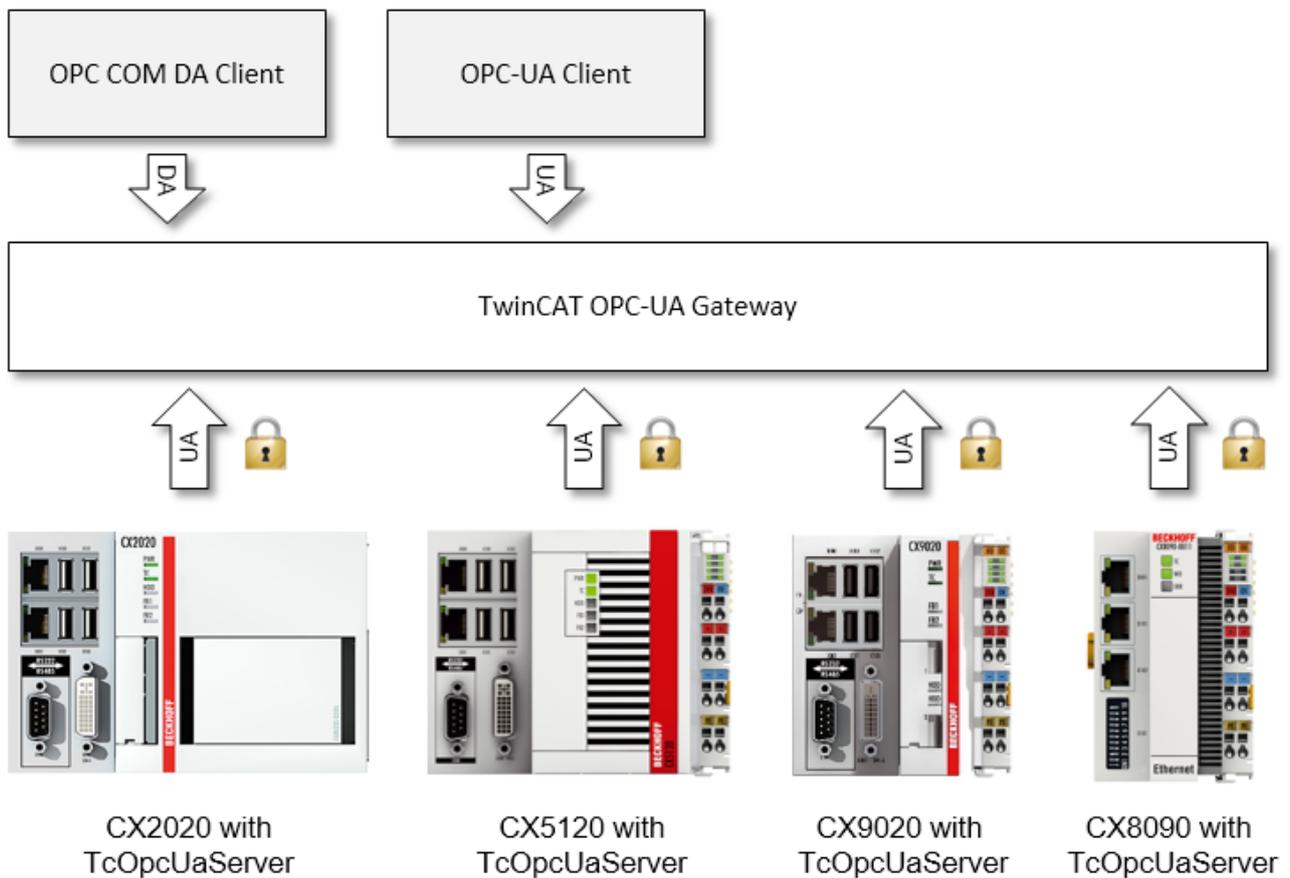
4.5.3 Lizenzierung

Das TwinCAT OPC UA Gateway wird kostenfrei geliefert. Es ist kein weiterer Lizenzerwerb erforderlich.

Beachten Sie, dass die Gateway-Komponente ausschließlich für die Verbindung mit TwinCAT OPC UA Servern verwendet werden kann. Die Verbindung mit UA Servern von Drittanbietern wird programmierungstechnisch verhindert. Wenn die Umgebung die Verbindung mit einem UA Server eines Drittanbieters erforderlich macht, wird das Unified Automation UA Gateway empfohlen. Diese kann bei <http://www.unified-automation.com> erworben werden.

4.5.4 Szenarien

Aufgrund der offenen und flexiblen PC-basierten Automatisierungstechnik von Beckhoff kann das OPC UA Gateway auf verschiedene Arten betrieben und installiert werden. Im folgenden Abschnitt werden die verschiedenen Setup-Szenarien beschrieben und die Vor- und Nachteile einer jeden Einrichtung erläutert.



Gateway und UA Server auf demselben Computer

Bei diesem Szenario sind das Gateway und der UA Server auf demselben Computer installiert. Das Gateway ist mit den Standardeinstellungen konfiguriert, um eine Verbindung mit dem lokalen OPC UA Server mit der folgenden Server-URL herzustellen: `opc.tcp://localhost:4840`.

Dieses Szenario funktioniert lediglich auf Nicht-Windows-CE-Geräten.

Der TwinCAT OPC UA Server ist auch für Windows CE erhältlich, aber das Gateway ist nur für Big-Windows-Plattformen erhältlich.

Gateway und UA Server auf verschiedenen Computern

Bei diesem Szenario sind das Gateway und der UA Server auf verschiedenen Computern installiert. Das Gateway ist für die Herstellung einer Verbindung mit dem remote OPC UA Server konfiguriert, indem dessen entsprechende Server-URL, z. B. `opc.tcp://192.168.1.1:4840`, festgelegt wird.

Der OPC UA Server ist möglicherweise auf einem kleinen Embedded-Gerät (z. B. einem CX8090 mit Windows CE) installiert, während die Gateway-Komponente auf einer getrennten Big-Windows-Plattform, z. B. einem zentralen Server-Gerät, installiert ist.

Gateway mit mehreren UA-Servergeräten verbunden

Beachten Sie, dass es, ungeachtet der oben beschriebenen Szenarien, möglich ist, andere TwinCAT OPC UA Server ebenfalls an das Gateway anzuschließen. So kann z. B. das Gateway konfiguriert werden, um auf folgende Server zuzugreifen:

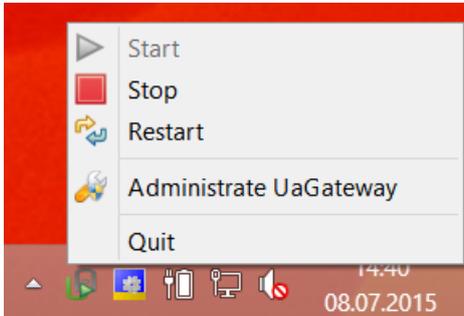
- den lokalen TwinCAT OPC UA Server (z. B. `opc.tcp://localhost:4840`)
- einen remote TwinCAT OPC UA Server (z. B. `opc.tcp://192.168.1.1:4840`)
- einen anderen remote TwinCAT OPC UA Server (z. B. `opc.tcp://192.168.1.21:4841`)
- ...

4.5.5 Konfigurator

4.5.5.1 Übersicht

Das UA Gateway Administration Tool ist eine grafische Benutzerschnittstelle für die Konfiguration des Gateway.

Sie öffnen das Tool über das Kontextmenü des Gateway-Symbols in der Windows-Taskleiste. Nach dem Starten des Verwaltungswerkzeugs über den Befehl **Administrate UaGateway** wird die grafische Benutzerschnittstelle eingeblendet.

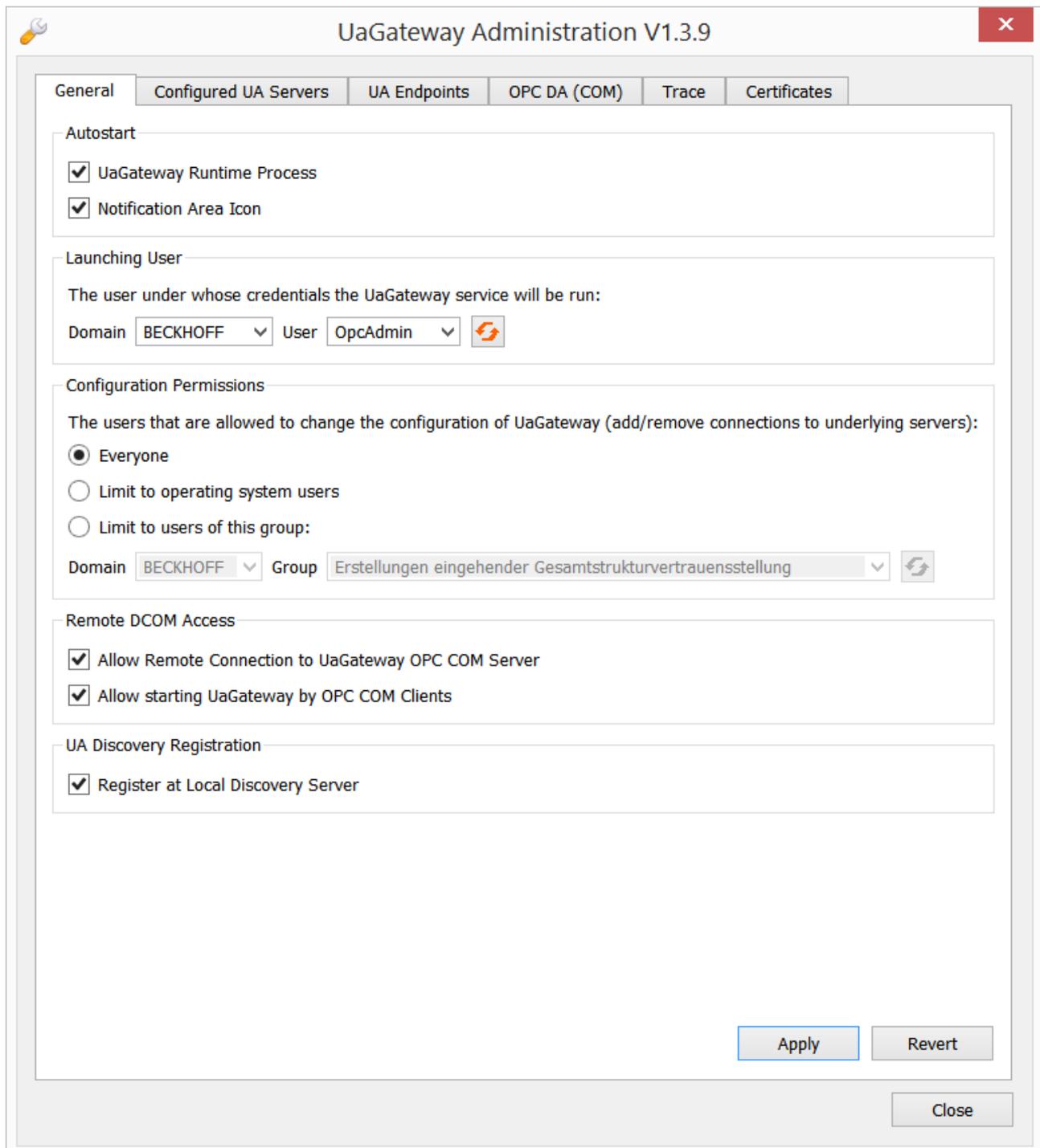


Die Schnittstelle bietet mehrere Konfigurationsoptionen:

- [Allgemeine Einstellungen](#) [► 200]
- [Zusätzliche UA Server](#) [► 202]
- [Zusätzliche Endpunkte](#) [► 203]
- [OPC-COM-DA-Einstellungen](#) [► 204]

4.5.5.2 Allgemeine Einstellungen

Die Registerkarte General zeigt allgemeine Einstellungen des UA Gateway.



Autostart

In diesem Bereich können Sie das Autostart-Verhalten des UA Gateway konfigurieren.

Aktivieren **UaGateway Runtime Process**, um den UA Gateway Service automatisch beim Einschalten des Computers zu starten.

Aktivieren Sie **Notification Area Icon**, um das Symbol des Benachrichtigungsbereichs bei der Anmeldung eines Benutzers zu starten.

Startender User (Launching User)

Das UA Gateway wird als Windows NT Service ausgeführt. Diesem Service wird ein spezifischer Userkontext zugewiesen, damit COM/DCOM ordnungsgemäß konfiguriert werden kann. Der User, den Sie auswählen, wird dem UA Gateway Service zugeordnet. Darüber hinaus wird dem User ein LogOnAsService-Recht eingeräumt (so kann er/sie den Service starten) und er wird einer lokalen Nutzergruppe hinzugefügt

(„UaGatewayUsers“). Diese Gruppe wird der Zugriffskontrollliste (ACL, Access Control List) der lokalen Maschine hinzugefügt. Für die ordnungsgemäße COM/DCOM-Konfiguration müssen Sie dieser Gruppe alle User hinzufügen, die das UA Gateway starten und auf dieses zuzugreifen dürfen.

Konfigurationsberechtigungen (Configuration Permissions)

Es besteht die Möglichkeit, nur bestimmten Benutzern zu erlauben, die Konfiguration des UA Gateway zu verändern, sprich Verbindungen zu grundlegenden Servern hinzuzufügen oder zu entfernen. Sie können aus folgenden Einstellungen auswählen:

Jeder (Everyone)	Jeder (auch die anonym bei UA angemeldeten User), der mit dem UA Gateway in Verbindung treten kann, kann die Konfiguration verändern.
Beschränkt auf Betriebssystemnutzer (Limit to operating system users)	Nur lokale User und User von der gleichen Domain können die Konfiguration ändern.
Beschränkt auf Nutzer dieser Gruppe (Limit to users of this group)	Die Berechtigung, die Konfiguration zu ändern, nur den Usern einer bestimmten Gruppe zugestehen. Wenn nicht alle verfügbaren Gruppen in der Drop-down-Liste Group angezeigt werden (oder eine neu erstellt Gruppe fehlt), kann mittels Drücken der Schaltfläche Refresh diese Gruppe erneut eingelesen werden.

Remote-DCOM-Zugriff (Remote DCOM Access)

Bei der Aktivierung von **Allow Remote Connection to UaGateway OPC COM Server**, werden DCOM Port 135 und das ausführbare UA Gateway der Firewall-Ausnahmeliste hinzugefügt.

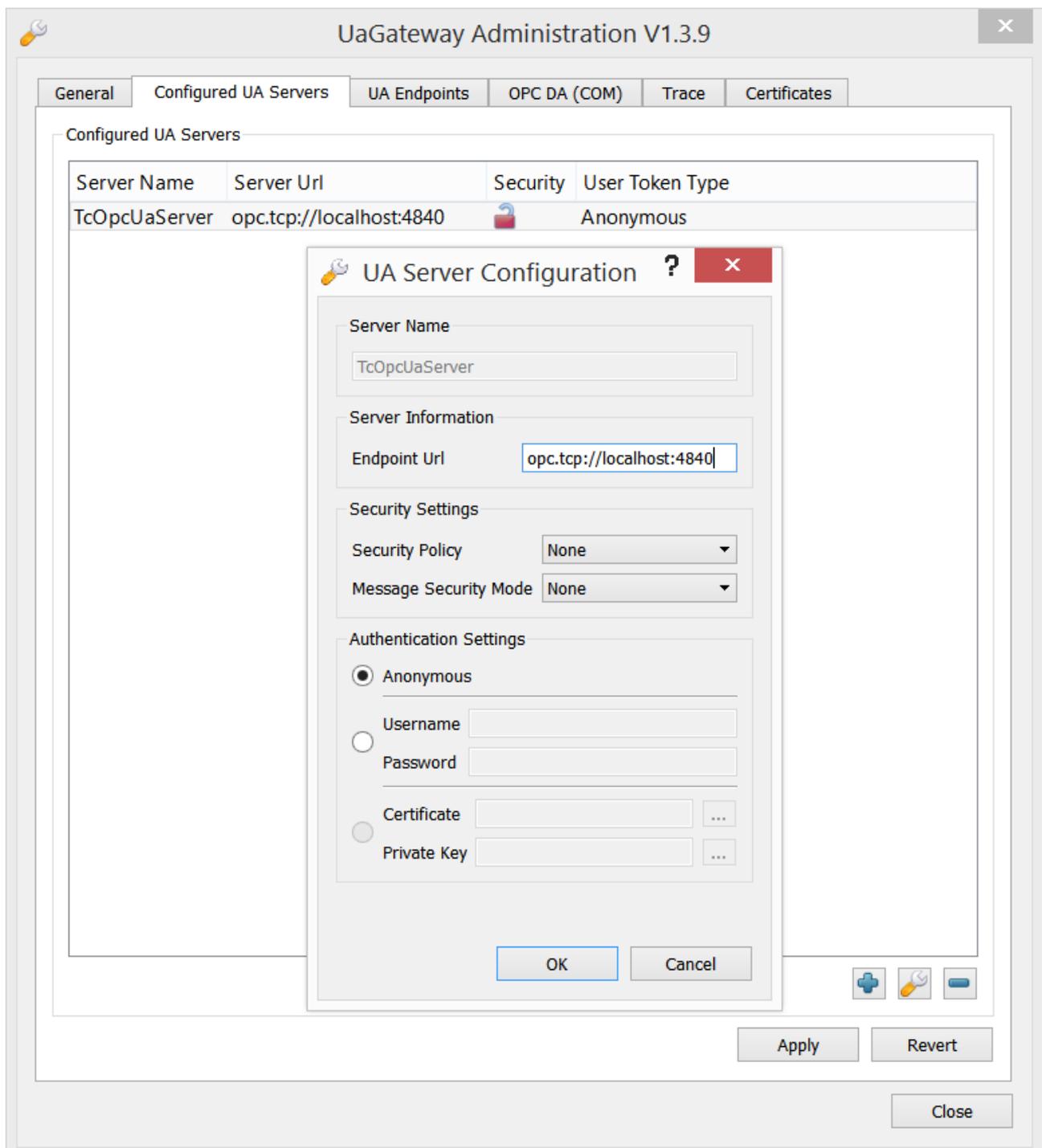
Wird **Allow starting UaGateway by DCOM Clients** deaktiviert, dann können DCOM-Clients das UA Gateway nicht starten. In diesem Fall kann UA Gateway weiterhin mithilfe des Notification Area Icon oder den Startmenüeinträgen gestartet oder gestoppt werden.

UA-Discovery-Anmeldung (UA Local Discovery Server)

Aktivieren Sie **Register at Local Discovery Server**, wenn das UA Gateway beim OPC UA LDS (Local Discovery Server), falls einer installiert ist, angemeldet werden soll.

4.5.5.3 Zusätzliche UA Server

Die Registerkarte **Configured UA Servers** bietet Optionen für die Konfiguration der zugrunde liegenden OPC UA Server. Standardmäßig stellt das Gateway bereits eine Verbindung mit dem lokalen OPC UA Server (der auf demselben Computer läuft) her.



Um weitere OPC UA Server zu konfigurieren oder aus der Konfiguration zu entfernen, klicken Sie auf die Plus- und Minus-Tasten unten rechts und anschließend auf **Übernehmen (Apply)**, um die Änderungen zu speichern.

4.5.5.4 Zusätzliche Endpunkte

Die Registerkarte **UA Endpoints** zeigt die Einstellungen zur UA-Endpunktkonfiguration.

Der UA Endpunkt (Endpoint) ist die Verbindungsinformation, die ein UA Client benötigt, um eine Verbindung mit dem Gateway herzustellen.

Allgemeines

Spezifizieren Sie mithilfe der Kontrollkästchen die Anmeldemethoden, die ein Client für die Herstellung einer Verbindung mit Ihrem UA Gateway verwenden kann.

Endpunkte

An dieser Stelle können Sie alle notwendigen Einstellungen für verschiedene UA Endpunkte festlegen. Der Endpunkt wird standardmäßig mit Voreinstellungen konfiguriert. Diese stellen einen einzelnen UA Endpunkt dar, der zwei Sicherheitsoptionen anbietet: Keine und Basic128RSsa15.

Die Sicherheitsoption „Keine“ erlaubt jedem UA Client die Herstellung einer Verbindung mit dem UA Gateway und wird lediglich für die Inbetriebnahme und Prüfung empfohlen, während diese Konfiguration in der Produktionsumgebung ausgeschaltet werden sollte.

Die verschiedenen Konfigurationselemente werden in den folgenden Abschnitten beschrieben.

Netzwerkconfiguration

Endpunkt URL	Dies ist die Endpunkt-URL des UA Gateway, so wie sie in FindServers und GetEndpoint Aufrufen zu sehen ist.
Protokoll	Dies ist das für diesen Endpunkt verwendete Protokoll.
Hostname/IP	Dies ist der Hostname des UA Gateway (es kann sich auch um die IP-Adresse des PC handeln, auf dem das UA Gateway ausgeführt wird).
Netzwerkadapter	Dies ist der Netzwerkadapter, mit dem eine Bindung herzustellen ist. Zur Auswahl stehen:
Alle	Auswahl, dass eine Bindung mit allen IP-Adressen des Computers herzustellen ist. Der Endpunkt wird über den gegebenen Anschluss auf allen IP-Adressen erreichbar sein.
Netzwerkadapter	Wählen Sie einen Netzwerkadapter und eine IP-Adresse (unten), um lediglich eine Bindung mit dieser Adresse herzustellen. Der Endpunkt wird nur für Clients erreichbar sein, die mit der ausgewählten IP-Adresse eine Verbindung herstellen.
Nur lokal	Bei dieser Auswahl stellt das UA Gateway lediglich eine Bindung mit dem Loopback-Adapter her. Der Endpunkt ist lediglich für die Clients erreichbar, die auf derselben Maschine wie das UA Gateway laufen.
Port	Dies ist das TCP-Port des Endpunkts (normalerweise 48050).

Sicherheit

In diesem Bereich können Sie die unterstützten Sicherheitseinstellungen des Endpunkts konfigurieren. Aktivieren Sie die Kontrollkästchen vor den Sicherheitsoptionen, die für einen bestimmten Endpunkt gelten sollen. Für die Optionen ungleich „Keine“, muss der (müssen die) verfügbare(n) Nachrichtensicherheitsmodus (modi) angegeben werden. Die Signierung sorgt dafür, dass Nachrichten nicht verändert werden können und dass sie zwischen den Anwendungen, die eine Verbindung hergestellt haben, ausgetauscht werden. Die Verschlüsselung garantiert, dass niemand die Nachrichten lesen kann.

4.5.5.5 OPC-COM-DA-Einstellungen

Die Registerkarte **OPC DA (COM)** zeigt die Einstellungen zur Konfiguration des COM DA Servers des UA Gateway.

Im folgenden Abschnitt wird die Konfiguration des COM DA Servers des UA Gateway unter Verwendung des Administrationstools beschrieben.

Allgemeines

ItemIDs des COM DA Server werden aus dem URI-Namensraum und dem Bezeichner des Variablenknotens im OPC-UA-Adressenraum gebildet. Der Namensraumteil kann im Fall eines einzigen Namensraums weggelassen werden.

Im Drop-down-Feld **Default Name Space** kann der standardmäßige Namensraum mit dem Namensraum eines zugrundeliegenden OPC-Servers festgelegt werden. Die ItemIDs dieses bestimmten Namensraums können dann mittels ausschließlicher Angabe des Bezeichners erreicht werden, weil beim Zugriff auf ein Element der standardmäßige Namensraum intern automatisch hinzugefügt wird. Dieses Merkmal kann dazu

verwendet werden, alle ItemIDs im Client, der auf den UA Gateway Server zugreift, neu zu konfigurieren, wenn Letzterer als Tunnellösung für einen zugrundeliegenden COM DA Server fungiert, und um dabei die ItemIDs des ursprünglichen COM DA Servers beizubehalten.

Im zweiten Drop-down-Feld kann die **Timestamp Source** (Zeitstempelquelle) festgelegt werden. Folgende Optionen stehen zur Auswahl:

Interner	Die Zeitstempel werden vom OPC COM DA Server erzeugt.
SourceTimestamp	Die SourceTimestamps werden als Zeitstempel, die vom OPC COM DA Server bereitgestellt werden, verwendet.
ServerTimestamp	Die ServerTimestamps werden als Zeitstempel, die vom OPC COM DA Server bereitgestellt werden, verwendet.

Eigenschaften-Mapping vom UA zum COM DA

Bei der Herstellung der Verbindung mit dem OPC-COM-DA-Server des UA Gateway werden alle sechs Standardeigenschaften (DataType, Value, Quality, TimeStamp, AccessRights und ScanRate) automatisch zugeordnet. Zugrunde liegende OPC-Server können weitere Eigenschaften (z. B. benutzerdefinierte Eigenschaften, DI-Eigenschaften, usw.) bereitstellen. Diese Eigenschaften können herstellerspezifischen Eigenschaften (PropertyID \geq 5000) im COM DA Server des UA Gateway zugeordnet werden.

Diese herstellerspezifischen PropertyIDs werden automatisch zugeordnet, wenn die Eigenschaften das erste Mal angefordert werden. Mithilfe dieses Dialogs können Sie die zugeordneten PropertyIDs ändern oder konfigurieren, wie die OPC-UA-Eigenschaften im UA-Gateway-Adressenraum den herstellerspezifischen COM-DA-Eigenschaften zuzuordnen sind. Sie müssen den UA-seitigen Eigenschaftennamen und den Namensraum der Eigenschaft im UA Gateway festlegen und diese den COM DA PropertyIDs zuordnen. Bei der Herstellung der Verbindung mit dem COM-DA-Server des UA Gateway können Sie die verfügbaren Eigenschaften (QueryAvailableProperties) eines einzelnen OPCItems durchsuchen und dann werden Sie die zugeordneten Eigenschaften, so wie diese (im Bereich der herstellerspezifischen PropertyIDs oberhalb 5000) konfiguriert wurden, sehen können.

Drücken Sie die Tasten **[+]** oder **[-]**, um eine bestimmte Eigenschaft hinzuzufügen bzw. zu entfernen. Um den Inhalt eines bestimmten Feldes zu ändern, doppelklicken Sie auf dieses und geben die gewünschten Werte ein. Wenn Sie auf einen Wert in der Spalte **UA Property NameSpace URI** doppelklicken, wird ein Drop-down-Menü eingeblendet, in dem Sie eine Auswahl treffen können.

Wenn Sie durch Drücken auf **[+]** eine neue Eigenschaft hinzufügen, werden die Werte des letzten Eintrags in die neue Zeile kopiert und die PropertyID automatisch inkrementiert.

4.5.6 Migration von Tx6120

Einer der vorrangigen Zwecke des UA Gateway ist die Bereitstellung einer zukunftsfähigen Konnektivität, um das Supplement / die Function Tx6120 OPC DA zu ersetzen. Wenn Sie Tx6120 OPC DA nach UA Gateway migrieren möchten, beachten Sie die nachfolgenden Hinweise.

Standardkonfiguration

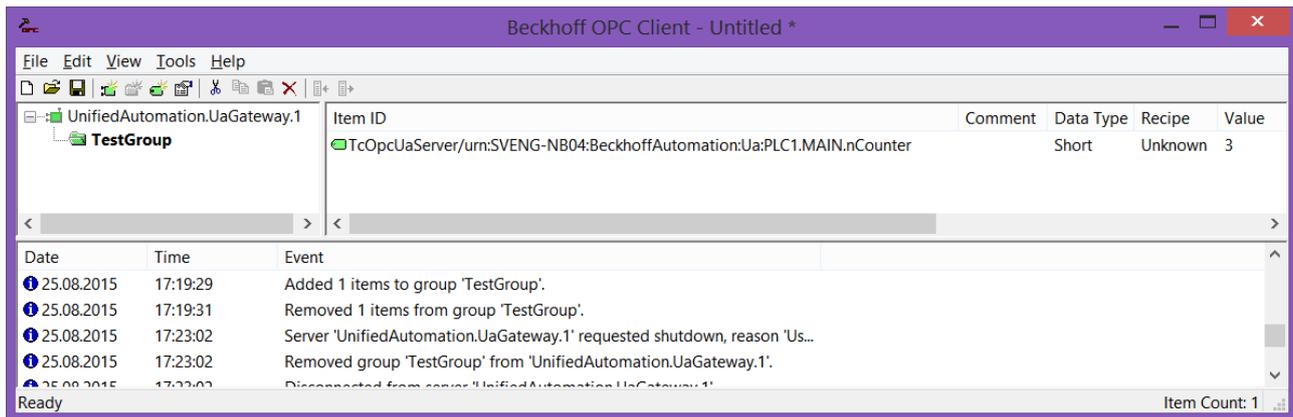
Die Standardkonfiguration des UA Gateway stellt automatisch eine Verbindung mit dem lokalen OPC UA Server her und bietet den OPC DA Clients eine OPC-DA-Schnittstelle. Bei einer Verbindung auf der Grundlage dieser Standardkonfiguration müssen die OPC-DA-Clients Folgendes berücksichtigen:

- Die standardmäßige ProgID des UA Gateway lautet „UnifiedAutomation.Gateway.1“. Der TwinCAT OPC DA Server verwendet eine andere ProgID („Beckhoff.TwinCATOpcServerDA“).
- Das UA Gateway verwendet stets eine ProgID anstelle von mehreren Klonen.
- Der ItemIdentifier eines OPC-Symbols wird im UA Gateway anders erzeugt. Dieses Verhalten kann geändert werden.

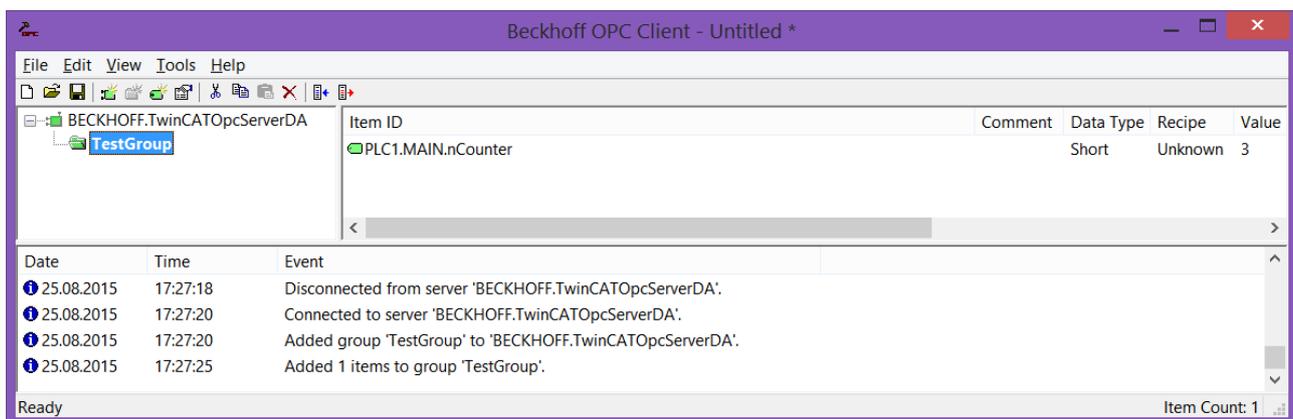
Syntax eines ItemIdentifiers verändern

Die Syntax, die das UA Gateway für ItemIdentifier verwendet, kann verändert werden, damit letztere eher der Art des TwinCAT OPC DA Servers entsprechen. Standardmäßig verwendet das UA Gateway bei der Bildung seiner Identifier eine andere Syntax als der TwinCAT OPC DA Server.

Beispiel UA Gateway:



Beispiel TwinCAT OPC DA Server:



Das UA Gateway verwendet ein Präfix, sodass der zugrunde liegende OPC UA Client, von dem die Variable stammt, eindeutig identifiziert werden kann.

Um das UA Gateway so zu konfigurieren, dass es seine Identifier in etwa so bildet, wie der TwinCAT OPC DA Server, sind die nachfolgenden Schritte erforderlich. Die Funktionalität wurde implementiert, um den Migrationsprozess zu vereinfachen.

1. Öffnen Sie die UA-Gateway-Konfigurationsdatei
C:\Program Files (x86)\UnifiedAutomation\UaGateway\bin\uagateway.config.xml
2. Suchen Sie nach den folgenden XML-Tags in der XML-Datei:

```
<OpcServerConfig>
  <ComDaServerConfig>
    <ComDaNamespaceUseAlias>>false</ComDaNamespaceUseAlias>
  </ComDaServerConfig>
</OpcServerConfig>
```

1. Wenn das XML-Tag ComDaNamespaceUseAlias auf „true“ gesetzt wird, können benutzerdefinierte Präfixes bestimmt werden. Suchen Sie hierfür nach dem folgenden XML-Tag in derselben XML-Datei:

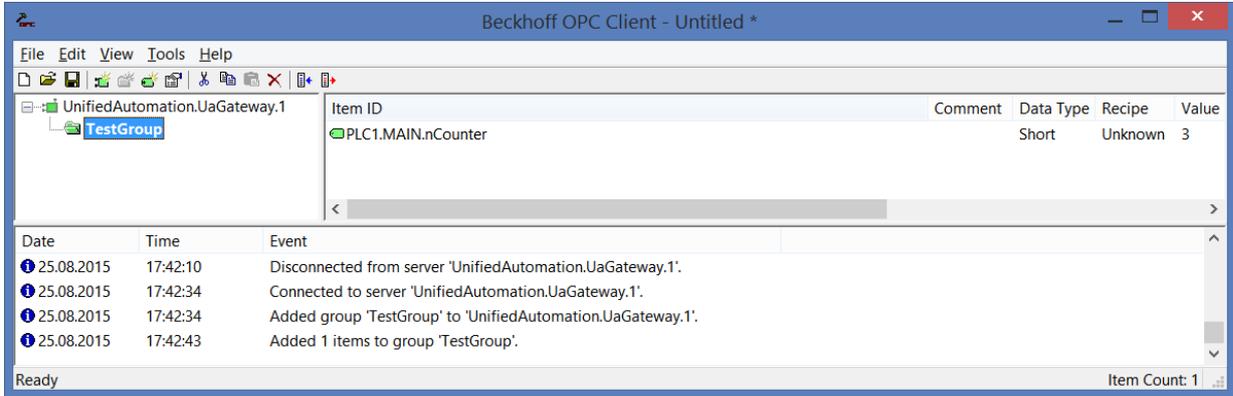
```
<OpcServerConfig>
  <UaServerConfig>
    <ConfiguredNamespaces>
      ...
    </ConfiguredNamespaces>
  </UaServerConfig>
</OpcServerConfig>
```

1. Identifizieren Sie in dieser XML-Struktur den TwinCAT-OPC-UA-Server-Namensraum. Standardmäßig sollte dieser folgendermaßen lauten:

```
<OpcServerConfig>
  <UaServerConfig>
    <ConfiguredNamespaces>
      ...
      <Namespace>
        <Index>...</Index>
        <Uri>TcOpcUaServer/urn:Hostname:BeckhoffAutomation:Ua:PLC1</Uri>
        <AllowRenameUri>>false</AllowRenameUri>
      </Namespace>
    </ConfiguredNamespaces>
  </UaServerConfig>
</OpcServerConfig>
```

```
<UniqueId>TcOpcUaServer#TcOpcUaServer/urn:Hostname:BeckhoffAutomation:Ua:PLC1</UniqueId>
<ComAlias>...</ComAlias>
</Namespace>
...
</ConfiguredNamespaces
</UaServerConfig>
</OpcServerConfig>
```

1. Auf Ihrem Computer kann der Platzhalter „...“ anders aussehen. Setzen Sie <ComAlias> auf das von Ihnen bevorzugte Präfix, zum Beispiel „PLC1“. Daraufhin werden die Bezeichner mit dem Präfix „PLC1“ gebildet.



4.5.7 Sicherheit

4.5.7.1 Übersicht

Einer der Gründe für den Erfolg von OPC UA als Kommunikationstechnologie sind die verschiedenen, integrierten Sicherheitsmechanismen. Eine auf OPC UA basierte Datenkommunikation lässt sich auf zwei Ebenen absichern:

1. Transportebene
2. Applikationsebene

Endpunkte

Ein Server bietet dem Client eine Liste mit verschiedenen Endpunkten [▶ 105] an mit denen sich der Client verbinden kann. Ein Endpunkt beschreibt hierbei unter Anderem welche Sicherheitsfunktionen (z. B. Message Security Mode, Security Policy und zur Verfügung stehende Identity Token) die Kommunikationsverbindung über diesen Endpunkt erfüllen soll. So kann ein Endpunkt z. B. eine Signierung und Verschlüsselung der Datenpakete erfordern (Transportebene), sowie eine zusätzliche Authentifizierung des Clients auf Basis von Benutzername/Password (Applikationsebene).

Transportebene

Eine auf OPC UA basierte Kommunikationsverbindung kann auf Transportebene abgesichert werden. Dies geschieht durch die Verwendung von Client/Server Zertifikaten und eine gegenseitige Vertrauensstellung zwischen Client- und Serverapplikation. Hierbei muss der Client dem Server-Zertifikat vertrauen und umgekehrt, damit eine Kommunikationsverbindung hergestellt werden kann. Hierfür ist ein gegenseitiger Zertifikatsaustausch [▶ 208] notwendig.

Applikationsebene

Zusätzlich zur Transportebene lässt sich eine Kommunikationsverbindung auch auf Applikationsebene absichern. Hierfür stehen verschiedene Authentifizierungsmechanismen [▶ 106] zur Verfügung, die vom Serverendpunkt angeboten werden.

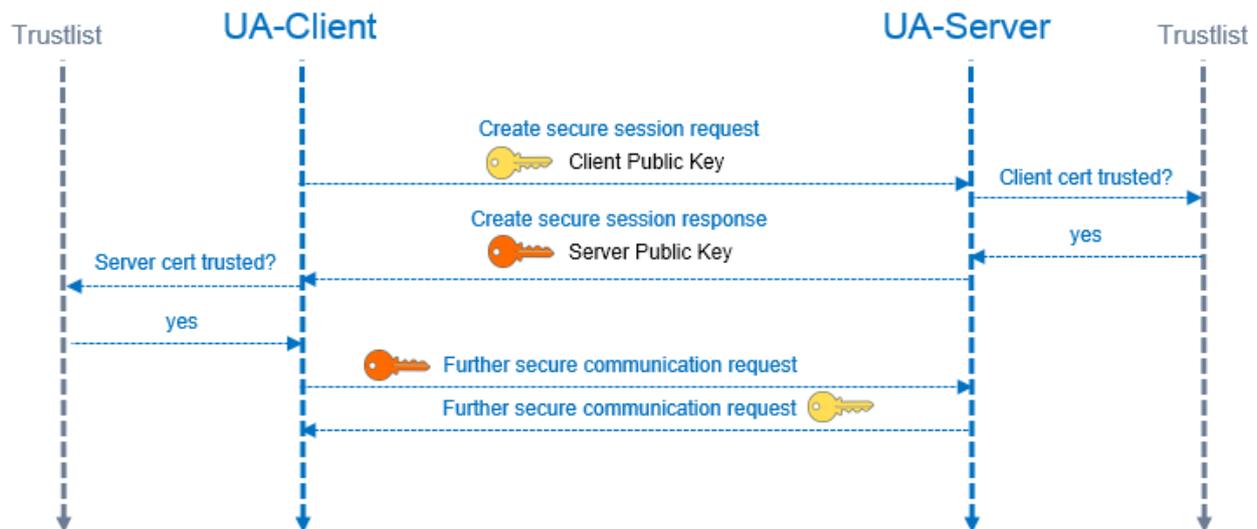
Sehen Sie dazu auch

- 📄 Zugriffsrechte [▶ 109]

4.5.7.2 Zertifikatsaustausch

Für eine Absicherung der Kommunikationsverbindung auf Transportebene über einen sicheren Endpunkt [► 105] ist die Herstellung einer gegenseitigen Vertrauensstellung zwischen Client und Server notwendig.

Standardmäßig generieren sowohl der TwinCAT OPC UA Server als auch der TwinCAT OPC UA Client beim ersten Start ein maschinenspezifisches, selbstsigniertes Zertifikat zur Authentifizierung der jeweiligen Applikation.



Vertrauensstellung auf dem Server einrichten

Zur Einrichtung einer Vertrauensstellung zwischen einem beliebigen OPC UA Client und dem TwinCAT OPC UA Server, benötigen Sie den öffentlichen Schlüssel des Clientzertifikats. Der Server muss diesem vertrauen. Dies kann zum Beispiel über das Dateisystem erfolgen. Der Server verwaltet die Vertrauenseinstellungen für Client-Zertifikate im Unterverzeichnis PKI.

- Vertrauenswürdige Zertifikate: %InstallDir%\Server\PKI\CA\trusted\certs
- Nicht-Vertrauenswürdige Zertifikate: %InstallDir%\Server\PKI\CA\rejected\certs

Durch Verschieben von Client-Zertifikaten zwischen diesen Verzeichnissen können die Vertrauenseinstellungen entsprechend angepasst werden. Der öffentliche Schlüssel eines Clientzertifikats wird beim ersten Verbindungsversuch des Clients mit einem sicheren Endpunkt automatisch im oben genannten Verzeichnis für nicht-Vertrauenswürdige Zertifikate abgelegt. Durch das anschließende Verschieben des öffentlichen Schlüssels in das Verzeichnis für vertrauenswürdige Zertifikate, wird dem Client beim nächsten Verbindungsversuch vertraut.

● AutomaticallyTrustAllClientCertificates

I Ist diese Option in der TcUaServerConfig.xml aktiviert, so vertraut der Server automatisch allen Clientzertifikaten. Diese werden in diesem Fall nicht in einem der oben genannten Verzeichnisse aufgelistet.

Vertrauensstellung auf dem Client einrichten

Abhängig vom verwendeten OPC UA Client müssen eventuell unterschiedliche Schritte vorgenommen werden, damit der OPC UA Client dem OPC UA Server vertraut. Typischerweise erfolgt bei Client-Applikationen mit grafischer Benutzeroberfläche ein Warnhinweis bei der ersten Verbindung mit dem Server, wobei das Server-Zertifikat dann als vertrauenswürdige eingestuft werden kann.

Die folgende Anweisung ist daher nur für den TwinCAT OPC UA Client gültig.

Der öffentliche Schlüssel des OPC UA Servers befindet sich als DER-Datei in dem folgenden Verzeichnis:
%InstallDir%\Server\PKI\CA\own\certs

Kopieren Sie die Datei beim TwinCAT OPC UA Client in das entsprechende „Trusted“-Verzeichnis:
%InstallDir%\Client\PKI\CA\trusted\certs

4.6 Sample Client

4.6.1 Übersicht

Ab Version 1.6.80 des TwinCAT OPC UA Servers wird ein kleines Programm „UA Sample Client“ automatisch installiert. Das Programm ermöglicht Ihnen, den OPC-UA-Namensraum zu durchsuchen und die UA-Server-Installation zu testen. Es befindet sich im Windows Startmenü und im Installationsverzeichnis des/der Supplements/Function. Sie können das Programm sowohl direkt auf dem UA Server als auch auf einem Computer in Ihrem Netzwerk ausführen.

Derzeit bietet der UA Sample Client die folgenden Funktionalitäten:

- Verbindung mit OPC UA Server herstellen
- Sichere Verbindung mit OPC UA Server herstellen (siehe [Sichere Verbindung mit OPC UA Server herstellen \[► 210\]](#))
- Durchsuchen des UA-Namensraums eines OPC UA Servers (siehe [UA-Namensraum durchsuchen \[► 213\]](#))
- Hinzufügen eines UA Nodes aus dem Namensraum in die Watchliste, die den Wert des Knotens regelmäßig liest (siehe [Watchliste verwenden \[► 214\]](#))

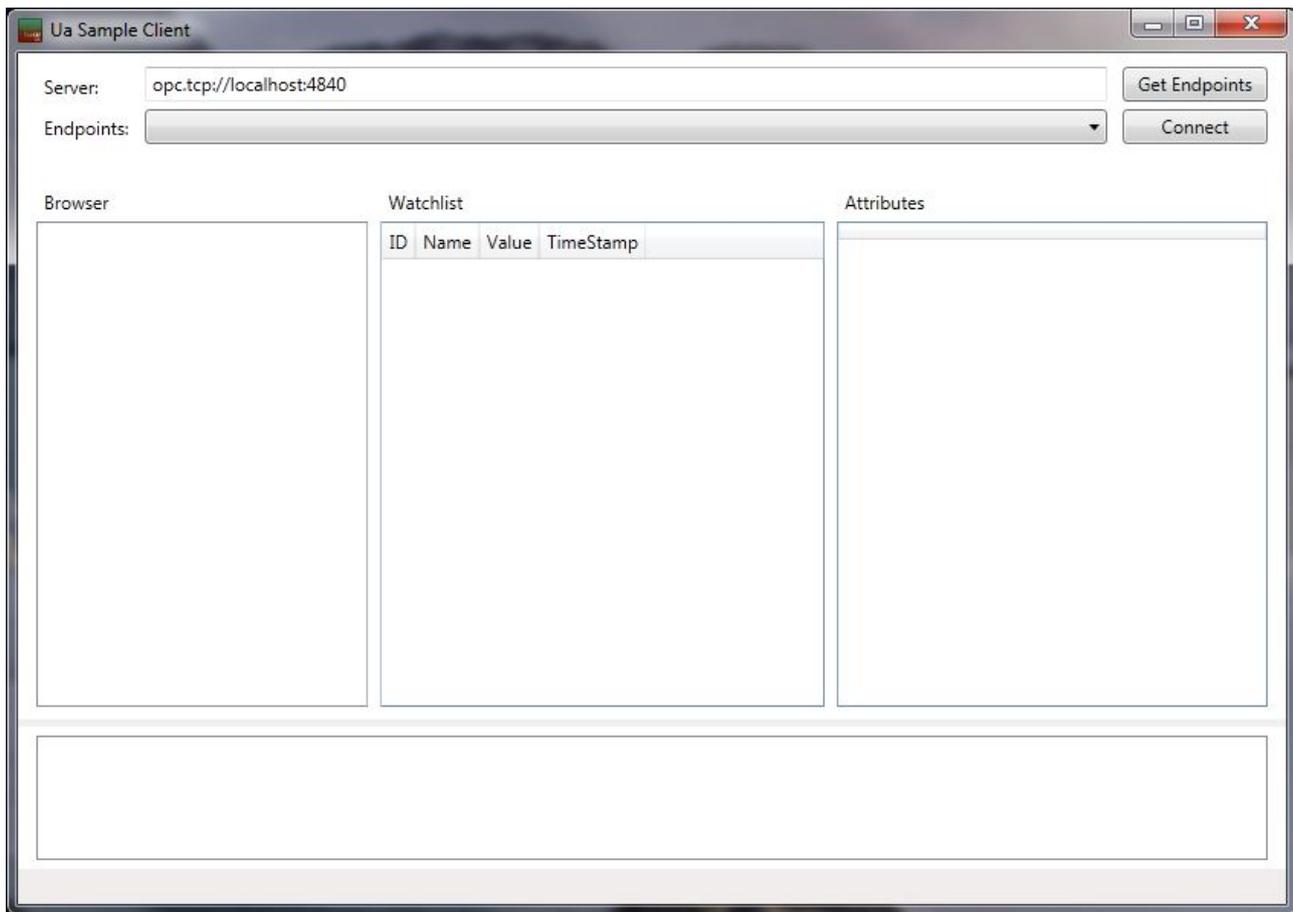
Diese Anwendung ist lediglich ein OPC-UA-Beispiel-Client. Sie bietet keine anspruchsvollen Funktionalitäten, sondern wurde entwickelt, um den Benutzern eine einfach zu bedienende Schnittstelle zur Verfügung zu stellen, um erste Tests beim OPC UA Server durchzuführen

Um die Anwendung zu starten, führen Sie die Datei *UA SampleClient.exe* mit dem Befehl **Als Administrator ausführen** aus.

Endpunkte des OPC UA Servers

Der UA Sample Client verbindet sich zuerst mit einer spezifizierten Server-URL. Der Client akquiriert alle Endpunkte des OPC UA Servers (siehe Drop-Down-Liste **Endpoints**). Die an den Server zurückgesandte Liste enthält dann mehr Informationen über alle verfügbaren Endpunkte, mit denen sich der Client verbinden kann. Jeder Endpunkt kann den Hostnamen des OPC UA Servers anstatt der IP-Adresse enthalten. Der Client verwendet dann die Informationen aus dem Endpunkt zur Verbindung mit dem Server.

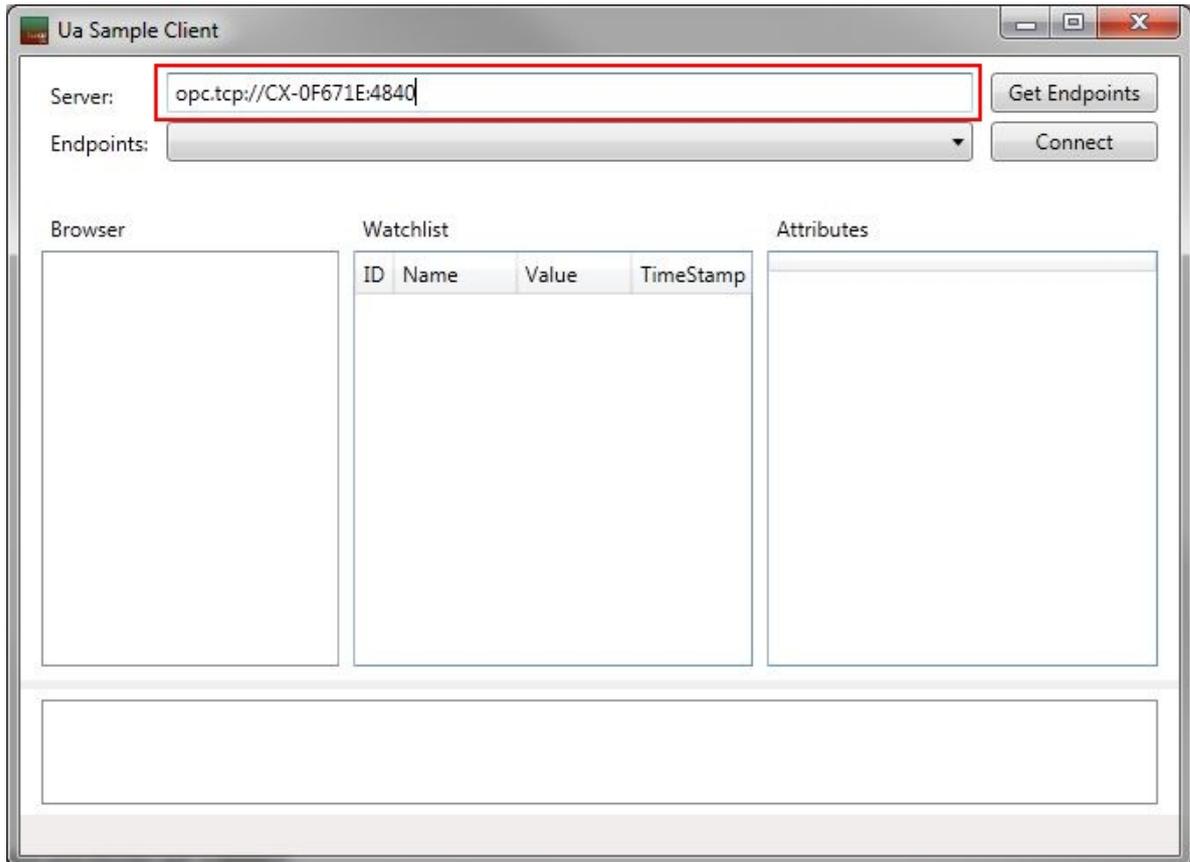
Wenn die Namenslösung nicht im Netzwerk des Benutzers funktioniert, kann der Client die Verbindung nicht herstellen. Wenn der Endpunkt, mit dem sich der Client verbinden soll, den Hostnamen des Servers enthält, stellen Sie sicher, dass die Namenslösung in Ihrem Netzwerk funktioniert und dass der Hostname auf dem Server erreichbar ist.



4.6.2 Sichere Verbindung mit OPC UA Server herstellen

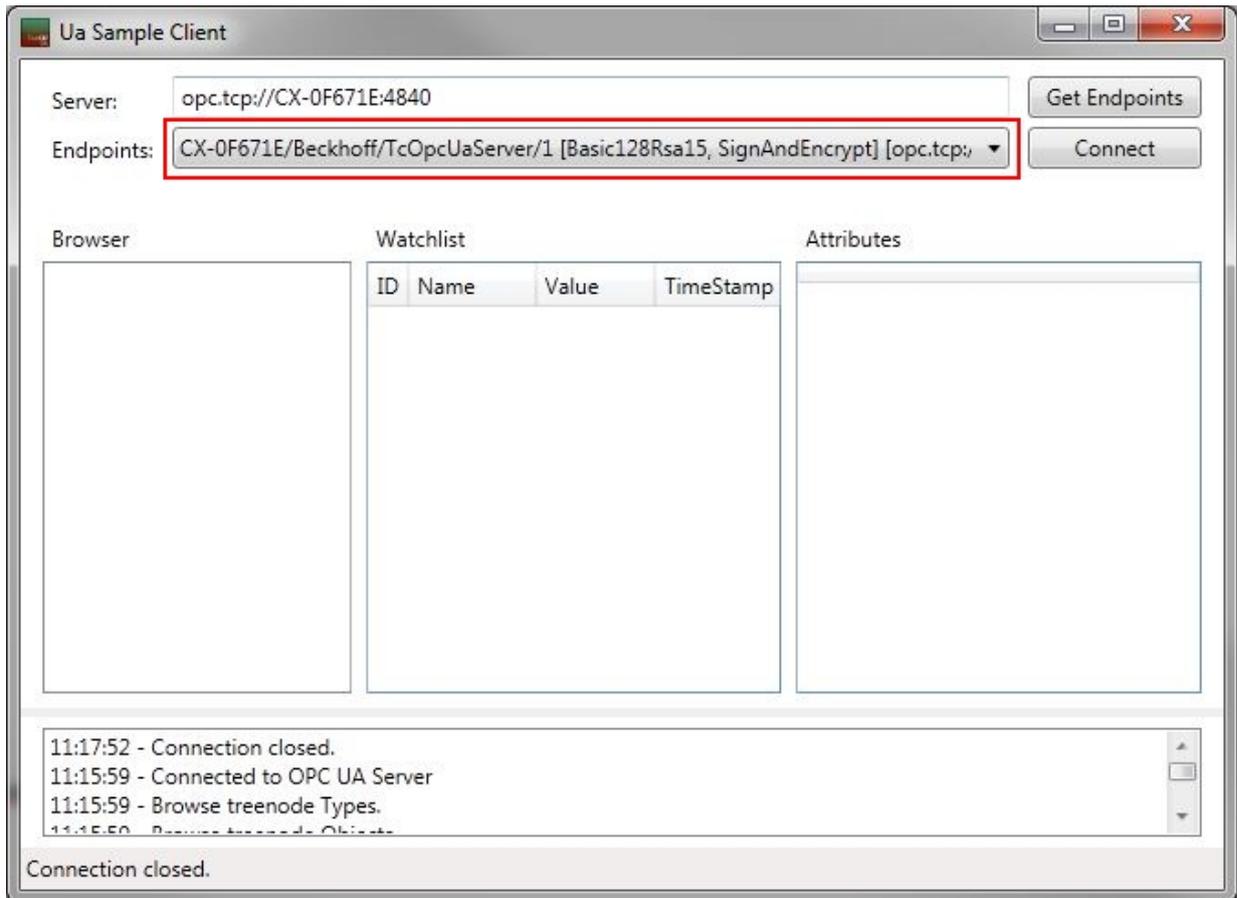
1. Geben Sie im oberen Textfeld des UA Sample Client die URL eines OPC UA Servers ein.
2. Klicken Sie auf die Schaltfläche **Get Endpoints**.

⇒ Die vom UA Server bereitgestellten Endpunkte werden daraufhin in der Drop-Down-Liste **Endpoints** angezeigt.

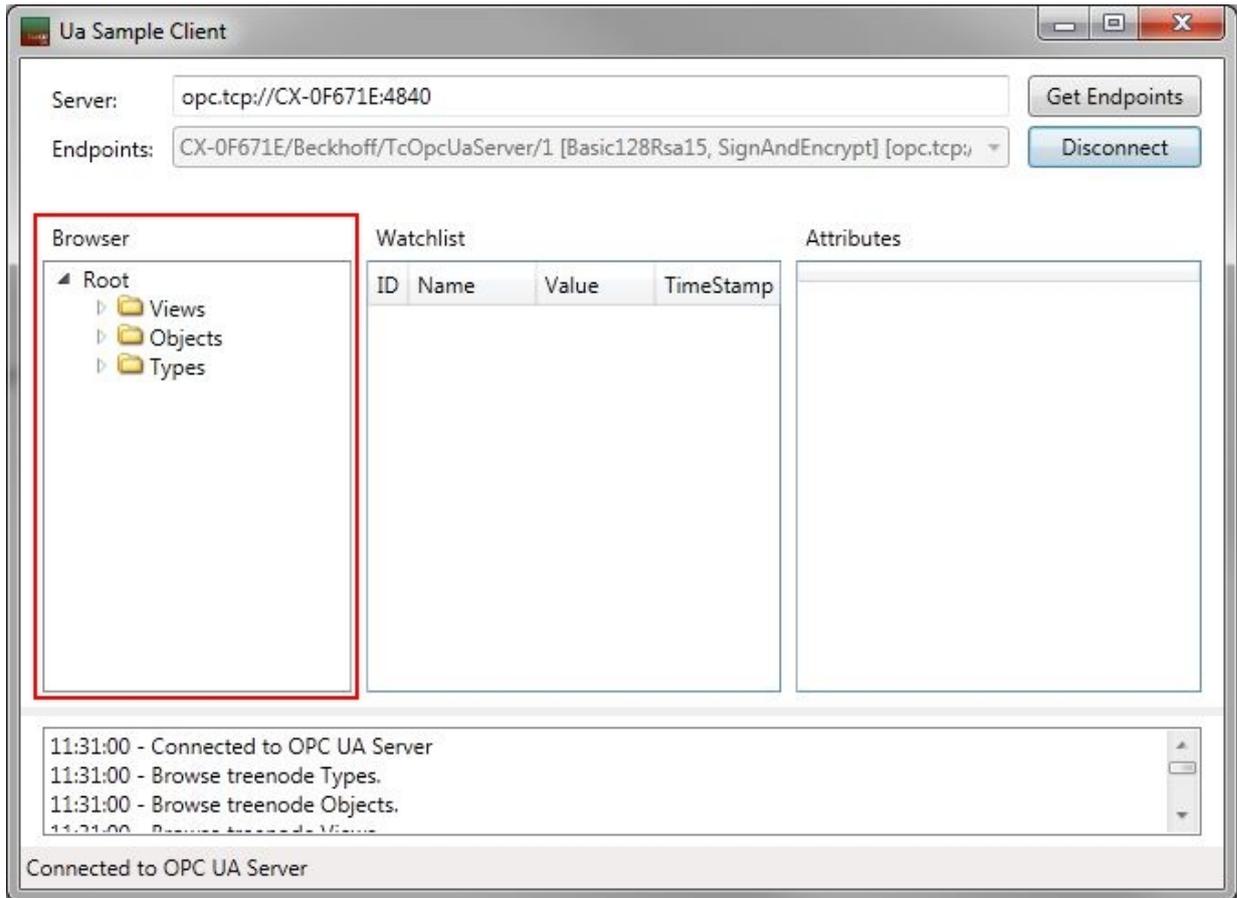


3. Wählen Sie in diesem Beispiel den Eintrag „<SomeName>/Beckhoff/TcOpcUaServer/1 [Basic128Rsa15, SignAndEncrypt] [opc.tcp://<SomeName>:4840]“ und klicken Sie auf **Connect**. Sie müssen den Public Key vom Zertifikat des UA Sample Clients auf den UA Server kopieren, damit dieser dem Sample Client „vertraut“. Anderenfalls wird der Verbindungsversuch über den sicheren Kanal

vom UA Server abgewiesen („BadSecureChannelClosed“). Weitere Informationen über das Zertifikatsmanagement beim OPC UA Server finden Sie im Abschnitt [Zertifikatsaustausch \[► 208\]](#).

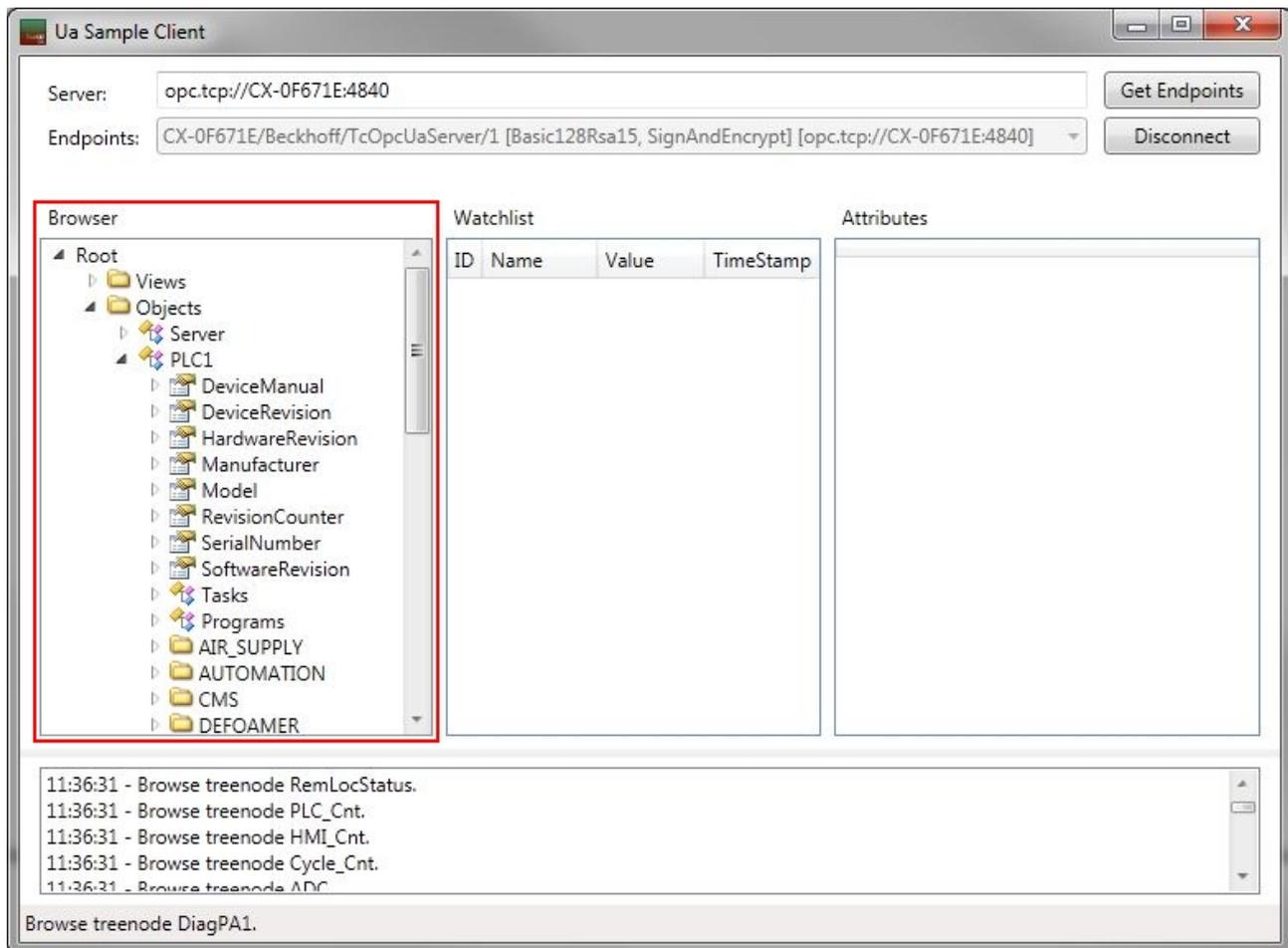


⇒ Über den **Browser** in der linken Hälfte des Fensters können Sie nun durch den UA-Namensraum navigieren.



4.6.3 UA-Namensraum durchsuchen

Über den **Browser** in der linken Hälfte des UA Sample Client können Sie nach erfolgreichem Aufbau einer Verbindung durch den UA-Namensraum navigieren. Unterhalb des Knotens **PLC1** finden Sie das momentan laufende SPS-Programm und können sich die dort deklarierten und für UA freigegebenen Variablen anzeigen lassen.



4.6.4 Watchliste verwenden

Sie können SPS-Variablen aus dem UA-Namensraum in eine Watchlist einfügen, zum Beispiel um deren Werte zyklisch von dem UA Sample Client lesen zu lassen. Öffnen Sie dazu das Kontextmenü einer Variablen und wählen Sie **Add to Watchlist**. Die Variable wird daraufhin in die Watchlist übernommen und deren Werte werden automatisch zyklisch aus der SPS ausgelesen.

The screenshot shows the 'Ua Sample Client' application window. At the top, the 'Server' field is set to 'opc.tcp://CX-0F671E:4840' and the 'Endpoints' dropdown shows 'CX-0F671E/Beckhoff/TcOpcUaServer/1 [Basic128Rsa15, SignAndEncrypt] [opc.tcp://CX-0F671E:4840]'. Buttons for 'Get Endpoints' and 'Disconnect' are visible.

The main interface is divided into three panes:

- Browser:** A tree view showing the OPC UA hierarchy. The path is expanded to 'AIR_SUPPLY' > 'PT100' > 'AH'. The 'AH' node is highlighted with a red box.
- Watchlist:** A table with columns 'ID', 'Name', 'Value', and 'TimeStamp'. It contains one entry: 'ns=4 AH False 3/1/2012 10:46:02 AM'. This table is also highlighted with a red box.
- Attributes:** A table showing the properties of the selected 'AH' node. The 'Value' attribute is set to 'False'.

At the bottom, a log window shows several messages: '11:38:19 - Browse treenode YF.', '11:38:19 - Browse treenode Y.', '11:38:19 - Browse treenode Units.', '11:38:19 - Browse treenode Act_LL.', and '11:38:19 - Browse treenode Act I.'. Below the log, a status bar displays 'Received data: Variable - AH.'

5 SPS API

5.1 Tc2_OpcUa

5.1.1 Datentypen

5.1.1.1 ST_OpcUAServerInfo

ST_OpcUAServerInfo beinhaltet Sessioninformationen eines TwinCAT OPC UA Servers.

Syntax

```

TYPE ST_OpcUAServerInfo :
STRUCT
  nReserved : UDINT;
  nCumulatedSessionCount      : UDINT;
  nCurrentSessionCount        : UDINT;
  nRejectedSessionCount       : UDINT;
  nSecurityRejectedSessionCount : UDINT;
  nSessionTimeoutCount        : UDINT;
  nCurrentSubscriptionCount    : UDINT;
  nRejectedRequestCount        : UDINT;
  nSecurityRejectedRequestCount : UDINT;
END_STRUCT
END_TYPE

```

Parameter

Name	Typ	Beschreibung
nReserved	UDINT	Platzhalter.
nCumulatedSessionCount	UDINT	Gesamtanzahl der Client-Sessions seit Start des Servers.
nCurrentSessionCount	UDINT	Gesamtzahl der aktuellen Client-Sessions.
nRejectedSessionCount	UDINT	Gesamtzahl der vom Server abgelehnten Sessions.
nSecurityRejectedSessionCount	UDINT	Gesamtzahl der aus Security-Gründen vom Server abgelehnten Sessions (Beispiel: Falsche Kombination aus Benutzername und Passwort).
nSessionTimeoutCount	UDINT	Gesamtzahl der Sessions, die einen Timeout hatten.
nCurrentSubscriptionCount	UDINT	Gesamtzahl der aktuellen Subscriptions im Server.
nRejectedRequestCount	UDINT	Gesamtzahl der fehlgeschlagenen Requests.
nSecurityRejectedRequestCount	UDINT	Gesamtzahl der aus Security-Gründen fehlgeschlagenen Requests.

5.1.1.2 E_OpcUAServerOption

E_OpcUAServerOption legt fest welches Kommando an den TwinCAT OPC UA Server geschickt werden soll.

Syntax

```

TYPE E_OpcUAServerOption
(
  eOPCUAServerOption_None,
  eOPCUAServerOption_Restart,
  eOPCUAServerOption_Shutdown,
  eOPCUAServerOption_RefreshCfg,
  eOPCUAServerOption_ServerInfo
);
END_TYPE

```

Parameter

Name	Beschreibung
eOPCUAServerOption_None	Ausgangszustand der Aufzählung.
eOPCUAServerOption_Restart	Diese Option triggert einen Neustart des OPC UA-Interfaces des Servers.
eOPCUAServerOption_Shutdown	Diese Option triggert das Herunterfahren des OPC UA-Interfaces des Servers. Da die voranstehende Restart-Option über OPC UA funktioniert, ist diese nach Nutzen dieser Option bis zu einem kompletten Server-Neustart nicht mehr verfügbar.
eOPCUAServerOption_RefreshCfg	Diese Option hat aktuell keine Funktion.
eOPCUAServerOption_ServerInfo	Diese Option fragt die in <u>ST_OpcUAServerInfo</u> [▶ 216] enthaltenen Server-Informationen ab.

5.1.1.3 E_OpcUAServerStatus

E_OpcUAServerStatus repräsentiert den Laufzeitstatus eines TwinCAT OPC UA Servers.

Syntax

```

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

Parameter

Name	Beschreibung
eOPCUAServerStatus_None	Ausgangszustand der Aufzählung.
eOPCUAServerStatus_Alive	Das ADS-Interface des TwinCAT OPC UA Servers ist erreichbar.
eOPCUAServerStatus_NotResponding	Das ADS-Interface des TwinCAT OPC UA Servers ist nicht erreichbar.

5.1.2 Funktionsbausteine

5.1.2.1 FB_OpcUAServer



Der Funktionsbaustein ermöglicht das Auslesen von Statusinformationen und Neustarten eines TwinCAT OPC UA Servers.

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

```

Eingänge

Name	Typ	Beschreibung
sNetId	T_AmsNetId	AmsNetId des Systems auf dem der TwinCAT OPC UA Server läuft.
bExecute	BOOL	Eine steigende Flanke startet die Abarbeitung des Funktionsbausteins.
eOpcUAServerOption	<u>E_OpcUAServerOption</u> [▶ 216]	Gibt die auszuführende Operation an.
tTimeout	TIME	ADS Timeout

Ausgänge

Name	Typ	Beschreibung
stOpcUAServerInfo	<u>ST_OpcUAServerInfo</u> [▶ 216]	Enthält Statusinformationen vom Server wenn beim Eingang eOpcUAServerOption "ServerInfo" ausgewählt wurde.
bBusy	BOOL	TRUE, solange die Abarbeitung des Funktionsbausteins nicht beendet ist.
bError	BOOL	Wird TRUE, sobald eine Fehlersituation eintritt.
nErrorId	UDINT	Enthält bei Auftreten eines Fehlers (bError) den Fehlercode.

5.1.2.2 FB_OpcUAServerGetStatus



Der Funktionsbaustein ermöglicht das Auslesen des aktuellen Status (Alive, NotResponding) eines TwinCAT OPC UA Servers. An dieser Stelle ist anzumerken, dass sich dieser Funktionsbaustein mit dem ADS-Interface des OPC UA Servers beschäftigt. Wenn der OPC UA-Server neugestartet oder heruntergefahren wird, bleibt das ADS-Interface des Servers erreichbar. Das ADS-Interface lässt sich nur durch Beenden des Server-Prozesses beenden.

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

```

 **Eingänge**

Name	Typ	Beschreibung
sNetId	T_AmsNetId	AmsNetId des Systems auf dem der TwinCAT OPC UA Server läuft.
bGetStatus	BOOL	Eine steigende Flanke startet die Abarbeitung des Funktionsbausteins.
tTimeout	TIME	ADS Timeout

 **Ausgänge**

Name	Typ	Beschreibung
eOPCUAServerStatus	E_OpcUAServerStatus [▶ 217]	Enthält Statusinformationen des Servers.
bDone	BOOL	TRUE, wenn die Abarbeitung des Funktionsbausteins beendet ist.
bBusy	BOOL	TRUE, solange die Abarbeitung des Funktionsbausteins nicht beendet ist.
bError	BOOL	Wird TRUE, sobald eine Fehlersituation eintritt.
nErrorId	UDINT	Enthält bei Auftreten eines Fehlers (bError) den Fehlercode.

5.2 Tc3_PLCopen_OpcUa

5.2.1 Datentypen

5.2.1.1 E_UAAttributeID

Syntax

```

TYPE E_UAAttributeID:
(
    eUAAI_NodeID          := 1,
    eUAAI_NodeClass       := 2,
    eUAAI_BrowseName      := 3,
    eUAAI_DisplayName     := 4,
    eUAAI_Description     := 5,
    eUAAI_WriteMask       := 6,
    eUAAI_UserWriteMask   := 7,
    eUAAI_IsAbstract      := 8,
    eUAAI_Symmetric       := 9,
    eUAAI_InverseName     := 10,
    eUAAI_ContainsNoLoops := 11,
    eUAAI_EventNotifier   := 12,
    eUAAI_Value           := 13,
    eUAAI_DataType        := 14,
    eUAAI_ValueRank       := 15,
    eUAAI_ArrayDimensions := 16
) DINT;
END_TYPE

```

Werte

Name	Beschreibung
NodeID	OPC UA NodeID
NodeClass	OPC UA NodeClass
BrowseName	OPC UA BrowseName
DisplayName	OPC UA DisplayName
Description	OPC UA Description
WriteMask	OPC UA WriteMask
UserWriteMask	OPC UA UserWriteMask
IsAbstract	OPC UA IsAbstract
Symmetric	OPC UA Symmetric
InverseName	OPC UA InverseName
ContainsNoLoops	OPC UA ContainsNoLoops
EventNotifier	OPC UA EventNotifier
Value	OPC UA Value
Data Type	OPC UA Data Type
ValueRank	OPC UA ValueRank
ArrayDimensions	OPC UA ArrayDimensions

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.2 E_UABrowseDirection**Syntax**

```

TYPE E_UABrowseDirection:
(
  eUABD_Forward := 0,
  eUABD_Inverse := 1,
  eUABD_Both := 2
) DINT;
END_TYPE

```

Werte

Name	Beschreibung
eUABD_Forward	Vorwärts-Referenzen
eUABD_Inverse	Rückwärts-Referenzen
eUABD_Both	Vorwärts- und Rückwärtsreferenzen

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.3 E_UABrowseResultMask**Syntax**

```

TYPE E_UABrowseResultMask:
(
  eUABRM_ReferenceTypeId := 1,
  eUABRM_IsForward := 2,
  eUABRM_ReferenceTypeInfo := 3,

```

```
eUABRM_NodeClass      := 4,
eUABRM_BrowseName     := 8,
eUABRM_DisplayName    := 16,
eUABRM_TypeDefinition := 32,
eUABRM_TargetInfo     := 60,
eUABRM_All             := 63
) DINT;
END_TYPE
```

Werte

Name	Beschreibung
eUABRM_ReferenceTypeld	ReferenceTypeld
eUABRM_IsForward	IsForward
eUABRM_ReferenceTypeInfo	ReferenceTypeInfo
eUABRM_NodeClass	NodeClass
eUABRM_BrowseName	BrowseName
eUABRM_DisplayName	DisplayName
eUABRM_TypeDefinition	TypeDefinition
eUABRM_TargetInfo	TargetInfo
eUABRM_All	All

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.4 E_UAConnectionStatus

Syntax

```
TYPE E_UAConnectionStatus:
(
    Connected      := 0,
    ConnectionError := 1,
    Shutdown       := 2
) DINT;
END_TYPE
```

Werte

Name	Beschreibung
Connected	Verbindung wurde hergestellt.
ConnectionError	Ein Fehler beim Herstellen der Verbindung ist aufgetreten.
Shutdown	Die Verbindung wurde getrennt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

SPS-Bibliothek	Benötigte Version
Tc3_PLCOpen_OpcUa	>= 3.2.11.0

5.2.1.5 E_UADataType

Syntax

```
TYPE E_UADataType:
(
    eUAType_Undefined := -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

```

Werte

Name	Beschreibung
eUAType_Undefined	Undefined
eUAType_Null	Null
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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.6 E_UAIdentifierType

Syntax

```

TYPE E_UAIdentifierType:
(
    eUAIdentifierType_String := 1,
    eUAIdentifierType_Numeric := 2,
    eUAIdentifierType_GUID := 3,
    eUAIdentifierType_Opaque := 4
) DINT;
END_TYPE
    
```

Werte

Name	Beschreibung
eUAIdentifierType_String	String
eUAIdentifierType_Numeric	Numeric
eUAIdentifierType_GUID	GUID
eUAIdentifierType_Opaque	Opaque

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.7 E_UANodeClassMask

Syntax

```

TYPE E_UANodeClassMask:
(
    eUANCM_Unspecified := 0,
    eUANCM_Object := 1,
    eUANCM_Variable := 2,
    eUANCM_Method := 4,
    eUANCM_ObjectType := 8,
    eUANCM_VariableType := 16,
    eUANCM_ReferenceType := 32,
    eUANCM_DataType := 64,
    eUANCM_View := 128,
    eUANCM_All := 255
) DINT;
END_TYPE
    
```

Werte

Name	Beschreibung
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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.8 E_UASecurityMsgMode**Syntax**

```

TYPE E_UASecurityMsgMode:
(
  eUASecurityMsgMode_BestAvailable := 0,
  eUASecurityMsgMode_None         := 1,
  eUASecurityMsgMode_Sign         := 2,
  eUASecurityMsgMode_Sign_Encrypt := 3
) DINT;
END_TYPE

```

Werte

Name	Beschreibung
eUASecurityMsgMode_BestAvailable	Höchste verfügbare Security
eUASecurityMsgMode_None	Keine Security
eUASecurityMsgMode_Sign	Signierung
eUASecurityMsgMode_Sign_Encrypt	Signierung und Verschlüsselung

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.9 E_UASecurityPolicy**Syntax**

```

TYPE E_UASecurityPolicy:
(
  eUASecurityPolicy_BestAvailable := 0
  eUASecurityPolicy_None         := 1,
  eUASecurityPolicy_Basic128     := 2,
  eUASecurityPolicy_Basic128Rsa15 := 3,
  eUASecurityPolicy_Basic256     := 4
) DINT;
END_TYPE

```

Werte

Name	Beschreibung
BestAvailable	Höchste verfügbare Security.
None	Richtlinie für Konfigurationen mit geringsten Sicherheitsanforderungen.
Basic128	Richtlinie für Konfigurationen mit geringen bis mittleren Sicherheitsanforderungen.
Basic128Rsa15	Definiert eine Sicherheitsrichtlinie für Konfigurationen mit mittleren bis hohen Sicherheitsanforderungen.
Basic256	Definiert eine Sicherheitsrichtlinie für Konfigurationen mit hohen Sicherheitsanforderungen.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.10 E_UAServerState

Syntax

```

TYPE E_UAServerState:
(
    Running           := 0
    Failed            := 1,
    NoConfiguration   := 2,
    Suspended         := 3,
    Shutdown          := 4,
    Test              := 5,
    CommunicationFault := 6,
    Unknown           := 7
) DINT;
END_TYPE
    
```

Werte

Name	Beschreibung
Running	Running
Failed	Failed
NoConfiguration	NoConfiguration
Suspended	Suspended
Shutdown	Shutdown
Test	Test
CommunicationFault	CommunicationFault
Unknown	Unknown

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

SPS-Bibliothek	Benötigte Version
Tc3_PLCOpen_OpcUa	>= 3.2.11.0

5.2.1.11 E_UATransportProfile

Syntax

```

TYPE E_UATransportProfile:
(
    eUATransportProfileUri_UATcp := 1,
    
```

```

    eUATransportProfileUri_WSHttpBinary      := 2,
    eUATransportProfileUri_WSHttpXmlOrBinary := 3,
    eUATransportProfileUri_WSHttpXml        := 4
) DINT;
END_TYPE

```

Werte

Name	Beschreibung
eUATransportProfileUri_UATcp	UATcp
eUATransportProfileUri_WSHttpBinary	WSHttpBinary
eUATransportProfileUri_WSHttpXmlOrBinary	WSHttpXmlOrBinary
eUATransportProfileUri_WSHttpXml	WSHttpXml

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.12 E_UAUserIdentityTokenType

Syntax

```

TYPE E_UAUserIdentityTokenType:
(
    eUAUITT_Anonymous      := 0,
    eUAUITT_Username       := 1,
    eUAUITT_x509           := 2,
    eUAUITT_IssuedToeken   := 3
) DINT;
END_TYPE

```

Werte

Name	Beschreibung
eUAUITT_Anonymous	Anonymous-User.
eUAUITT_Username	Einloggen per Username.
eUAUITT_x509	Zertifikatsdatei zum Einloggen.
eUAUITT_IssuedToeken	Einloggen per Token.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.13 ST_UABrowseDescription

Syntax

```

TYPE ST_UABrowseDescription:
STRUCT
    stStartingNodeId : ST_UANodeId;
    eDirection       : E_UABrowseDirection;
    stReferenceTypeId : ST_UANodeId;
    bIncludeSubtypes : BOOL;
    eNodeClass       : E_UANodeClassMask;
    eResultMask      : E_UABrowseResultMask;
END_STRUCT
END_TYPE

```

Werte

Name	Beschreibung
stStartingNodeID	Default Starting Node: ObjectRoot
eDirection	Default Browse Direction: Forward
stReferenceTypeID	Default ReferenceType: Hierarchical
blIncludeSubtypes	Default IncludeSubtypes: TRUE
eNodeClass	Default NodeClassMask: All
eResultMask	Default BrowseResultMask: All

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.14 ST_UAExpandedNodeID

Syntax

```

TYPE ST_UAExpandedNodeID:
STRUCT
    nServerIndex : UDINT;
    sNamespaceURI : STRING(MAX_STRING_LENGTH);
    stNodeID : ST_UANodeID;
END_STRUCT
END_TYPE
    
```

Werte

Name	Beschreibung
nServerIndex	ServerIndex
sNamespaceURI	NamespaceName
stNodeID	NodeID (ST_UANodeID [▶ 229])

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.15 ST_UASessionConnectInfo

Syntax

```

TYPE ST_UASessionConnectInfo:
STRUCT
    sApplicationName : STRING(MAX_STRING_LENGTH);
    eSecurityMode : E_UASecurityMsgMode;
    eSecurityPolicyUri : E_UASecurityPolicy;
    eTransportProfileUri : E_UATransportProfile;
    tSessionTimeout : TIME;
    tConnectTimeout : TIME;
END_STRUCT
END_TYPE
    
```

Werte

Name	Beschreibung
sApplicationUri (veraltet)	Anwendungs-Uri maximale Zeichenkettenlänge 255. Ab TcUAClient 2.0.0.14 wird diese automatisch vom Zertifikat vorgegeben, wie in der PLCOpen Spezifikation definiert. Daher in aktuellen Bibliotheksversionen nicht mehr verwendet.
sApplicationName	Anwendungsname mit maximaler Zeichenkettenlänge von 255.
eSecurityMode	Sicherheitsmeldungsmodus. Verfügbare Modi siehe E_UASecurityMsgMode [▶ 224].
eSecurityPolicyUri	Sicherheitsrichtlinien-Uri. Verfügbare Sicherheitsrichtlinien-Uri siehe E_UASecurityPolicy [▶ 224].
eTransportProfileUri	Transportprofil-Uri. Verfügbare Transportprofil-Uri siehe E_UATransportProfile [▶ 225];
stUserIdentTokenType	Struktur mit Authentifizierungsdaten für die Anmeldung am OPC UA-Server. Vollständige Beschreibung unter ST_UAUserIdentityTokenType [▶ 231].
tSessionTimeout	Wert Sitzungstimeout.
tConnectTimeout	Wert für den Verbindungstimeout. Dieser muss passend zum ADS Timeout am UA_Connect Baustein gesetzt werden. Hierbei gilt die Faustregel: ADS Timeout > 2 * ConnectionTimeout.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.16 ST_UAIndexRange**Syntax**

```

TYPE ST_UAIndexRange :
STRUCT
    nStartIndex : UDINT;
    nEndIndex   : UDINT;
END_STRUCT
END_TYPE

```

Werte

Name	Beschreibung
nStartIndex	Startindex der Daten.
nEndIndex	Endindex der Daten.

Für alle Dimensionen:

- StartIndex und EndIndex müssen zugewiesen werden.
- StartIndex muss kleiner als EndIndex sein.
- Um auf alle Elemente in einer Dimension zugreifen zu können, müssen StartIndex und EndIndex abhängig von der Gesamtzahl Elemente in der Dimension zugewiesen werden.
- Einzelne Elemente einer Dimension können ausgewählt werden, indem der gleiche StartIndex und EndIndex angegeben wird.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.17 ST_UALocalizedText

Syntax

```

TYPE ST_UALocalizedText:
STRUCT
  sLocale : STRING(6);
  sText   : STRING(MAX_STRING_LENGTH);
END_STRUCT
END_TYPE
    
```

Werte

Name	Beschreibung
sLocale	Sprachkennung des LocalizedText
sText	Text

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.18 ST_UAMethodArgInfo

Syntax

```

TYPE ST_UAMethodArgInfo:
STRUCT
  DataType      : E_UADatatype := -1;
  ValueRank     : DINT := 2147483647;
  ArrayDimensions : ARRAY[1..3] OF UDINT := [0,0,0];
  nLenData      : DINT;
END_STRUCT
END_TYPE
    
```

Werte

Name	Beschreibung
DataType	Legt den UA-Datentyp für den Methodenparameter fest. (Typ: E_UADatatype [▶ 2211])
ValueRank	Legt fest, ob der Parameter Skalar (-1) oder Array ist.
ArrayDimensions	Wenn der Parameter ein Array ist, spezifiziert dieser die Dimensionen des Arrays. Jedes Element bestimmt die Länge pro Dimension.
nLenData	Spezifiziert die Länge des Arguments. Bei Ausgabeinformationen wird von STRUCT nur dieses Element gefordert.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.19 ST_UANodeID

Syntax

```

TYPE ST_UANodeID:
STRUCT
  nNamespaceIndex : UINT;
  nReserved       : ARRAY [1..2] OF BYTE; //fill bytes
  sIdentifier     : STRING(MAX_STRING_LENGTH);
  eIdentifierType : E_UAIdentifierType;
END_STRUCT
END_TYPE
    
```

Werte

Name	Beschreibung
nNamespaceIndex	Namensraum-Index unter dem der Knoten verfügbar ist. Kann mit dem Funktionsbaustein UA_GetNamespaceIndex [► 238] bestimmt werden.
nReserved	Platzhalter
sIdentifier	Bezeichner wie im UA Namensraum gezeigt (Attribut 'Identifier').
eIdentifierType	Typ der Variablen, beschrieben mittels E_UAIdentifierType [► 223] .

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.20 ST_UANodeAdditionalInfo**Syntax**

```

TYPE ST_UANodeAdditionalInfo:
STRUCT
    eAttributeID      : E_UAAttributeID;
    nIndexRangeCount : UINT;
    nReserved         : ARRAY[1..2] OF BYTE; // fill bytes
    stIndexRange     : ARRAY[1..nMaxIndexRange] OF ST_UAIndexRange;
END_STRUCT
END_TYPE

```

Werte

Name	Beschreibung
eAttributeID	Spezifiziert die ID des OPC-UA-Attributs. Standardmäßig wird eUAAI_Value verwendet. (Typ: E_UAAttributeID [► 219]).
nIndexRangeCount	Legt fest, wie viele Indexbereiche in stIndexRange verwendet werden.
nReserved	Platzhalter
stIndexRange	Spezifiziert einen Indexbereich für das Lesen von Werten aus einem Array. (Typ: ST_UAIndexRange [► 228]).

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.21 ST_UAReferenceDescription**Syntax**

```

TYPE ST_UAReferenceDescription:
STRUCT
    stReferenceTypeId : ST_UANodeId;
    bIsForward        : BOOL;
    stNodeId          : ST_UAExpandedNodeId;
    stBrowseName      : STRING(MAX_STRING_LENGTH);
    stDisplayName     : ST_UALocalizedText;
    eNodeClass        : E_UANodeClassMask;
    stTypeDefinition  : ST_UAExpandedNodeId;
END_STRUCT
END_TYPE

```

Werte

Name	Beschreibung
stReferenceTypeld	Nodeld des Referenztyps (z.B. Organizes, HasChild, HasTypeDefinition, ...) als Datentyp ST_UANodeld [▶ 229] .
blsForward	Gibt an ob es sich bei der Referenz um eine Forward oder Backward Referenz handelt.
stNodeld	Nodeld als Datentyp ST_UAExpandedNodeld [▶ 227] .
stBrowseName	BrowseName der Referenz.
stDisplayName	DisplayName der Referenz (ST_UALocalizedText [▶ 229]).
eNodeClass	NodeClass der Referenz (E_UANodeClassMask [▶ 223]).
stTypeDefinition	Typdefinition (HasTypeDefinition) (ST_UAExpandedNodeld [▶ 227]).

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.22 ST-UAUserIdentityTokenType

Syntax

```

TYPE ST-UAUserIdentityTokenType:
STRUCT
    eUserIdentTokenType : E-UAUserIdentityTokenType;
    sTokenParam1        : STRING(MAX_STRING_LENGTH);
    sTokenParam2        : STRING(MAX_STRING_LENGTH);
END_STRUCT
END_TYPE
    
```

Werte

Name	Beschreibung
eUserIdentTokenType	Typ des Users, beschrieben mittels E-UAUserIdentityTokenType [▶ 226] ..
sTokenParam1	Username zur Anmeldung am OPC UA-Server.
sTokenParam2	Passwort zur Anmeldung am OPC UA-Server.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.1.23 UAHADataValue

Dieser Funktionsbaustein agiert als Datenobjekt. Eine Instanz repräsentiert einen Wert für die OPC-UA-Historical-Access-Funktionalität. Dem Funktionsbaustein [UA_HistoryUpdate \[▶ 239\]](#) wird ein ganzes Feld dieser Werte beim Aufruf übergeben.

Syntax

```

aDataValues : ARRAY [1..50] OF UAHADataValue(ValueSize:=SIZEOF(LREAL));
    
```

Jedes Datenobjekt wird mit der zu erwartenden Größe (in Bytes) des Wertes initialisiert.

 **Eigenschaften**

Name	Typ	Zugriff	Initialwert	Beschreibung
Value	PVOID	Set	-	Gibt die Adresse einer Variablen an, welche den gewünschten Wert beinhaltet. Typischerweise wird diese mithilfe des Operators ADR() zugewiesen. Hiermit wird zugleich intern der Wert selbst zugewiesen und in das Datenobjekt kopiert.
StatusCode	UAHAUpdateStatusCode [▶ 232]	Get, Set	UAHAUpdateStatusCode.HistorianRaw	Gibt den Statuscode des Wertes an.
SourceTimeStamp	ULINT	Get, Set	0	Gibt den Zeitstempel der Quelle im UTC-Format an. Dieser kann mit der Funktion F_GetSystemTime (Tc2_System SPS Bibliothek) ermittelt werden.
ServerTimeStamp	ULINT	Get, Set	0	Gibt den Zeitstempel des OPC UA Servers im UTC-Format an. Diese Funktionalität wird aktuell nicht unterstützt.

● Datentypgröße des Wertes

I Die Größe des verwendeten Datentyps wird bereits bei der Deklaration des Datenobjektes angegeben und damit festgelegt. Bei späterer Zuweisung eines Wertes wird diese Größe zugrunde gelegt.

Werte vom Typ STRING werden demnach ebenso mit fest initialisierter Größe abgespeichert und übertragen. Es kann keine Angabe über die aktuelle Textlänge gemacht werden.

Beispiel

```
{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 diesem Beispiel wird ein Feld von 50 Werten definiert, welche jeweils durch ein Datenobjekt repräsentiert werden. Dem ersten Wert wird der aktuelle Inhalt der Variablen fMyValue (= 27.75) zugewiesen.

Das Feld kann nun mittels weiterer Zuweisungen in späteren SPS-Zyklen gefüllt werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT 3.1 >= 4024.1	Win32, Win64, WinCE-x86	Tc3_PLCopen_OpcUa >= v3.1.9.0

5.2.1.24 UAHAUpdateStatusCode

Jedem mit der OPC-UA-Historical-Access-Funktionalität übertragenen Datenwert wird ein Statuscode zugeordnet. Dies ist eine Eigenschaft des Objektes [UAHADataValue](#) [▶ 231].

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 interval.
  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 minimum values at different timestamps within the same interval).
) UDINT;
END_TYPE
```

Werte

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT 3.1 >= 4024.1	Win32, Win64, WinCE-x86	Tc3_PLCopen_OpcUa >= v3.1.9.0

5.2.2 Funktionsbausteine

5.2.2.1 UA_Browse



Dieser Funktionsbaustein ermöglicht das Browsen durch den Namensraum eines Servers. Ausgehend von einem Startknoten werden dessen Referenzen ausgelesen und entsprechend zurückgegeben.

Eingänge

```
VAR_INPUT
  Execute           : BOOL;
  ConnectionHdl     : DWORD;
  BrowseDescription : ST_UABrowseDescription;
  ContinuationPointIn : DWORD;
  Timeout           : TIME;
END_VAR
```

Name	Typ	Beschreibung
Execute	BOOL	Der Befehl wird durch eine steigende Flanke an diesem Eingang ausgelöst.
ConnectionHdl	DWORD	Verbindungshandle, das vorher vom Funktionsbaustein UA_Connect ausgegeben wurde.
BrowseDescription	ST_UABrowseDescription [► 226]	Hier werden die Adressinformationen zum auszulesenen Knoten angegeben.
ContinuationPointIn	DWORD	Falls ein vorheriger Aufruf des Funktionsbausteins einen Wert als ContinuationPointOut zurückgeliefert hat, kann dieser Wert hier angelegt werden um weitere Daten vom Server abzufragen.
Timeout	TIME	Zeit bis zum Abbruch der Funktion.

Ein-/Ausgänge

```
VAR_IN_OUT
  ReferenceDescriptions : POINTER TO ST_UAReferenceDescriptions;
END_VAR
```

Name	Typ	Beschreibung
ReferenceDescriptions	POINTER TO ST_UAReferenceDescriptions	Enthält die Liste der vom Server zurückgegebenen ReferenceDescriptions, also das Ergebnis vom UA_Browse Aufruf. Die enthaltenen ReferenceDescriptions können dann für weitere UA_Browse Aufrufe in der BrowseDescription verwendet werden, z.B. um tiefer in den Namensraum zu navigieren.

Ausgänge

```
VAR_OUTPUT
  Done           : BOOL;
  Busy           : BOOL;
  Error          : BOOL;
  ErrorID        : DWORD;
  ContinuationPointOut : DWORD;
  cbBrowseResultCnt : UDINT;
END_VAR
```

Name	Typ	Beschreibung
Done	BOOL	Schaltet auf TRUE, wenn der Funktionsbaustein erfolgreich ausgeführt wurde.
Busy	BOOL	TRUE, bis der Baustein einen Befehl ausführt hat, maximal für die Dauer des „Timeout“ am Eingang. Solange Busy = TRUE ist, akzeptieren die Eingänge keinen neuen Befehl. Es wird nicht die Zeit der Verbindung, sondern die Empfangszeit überwacht.
Error	BOOL	Schaltet auf TRUE, wenn bei der Ausführung eines Befehls ein Fehler auftritt. Der befehlspezifische Fehlercode ist in ErrorID enthalten.
ErrorID	DWORD	Enthält den befehlspezifischen Fehlercode des zuletzt ausgeführten Befehls.
ContinuationPointOut	DWORD	Wenn der Server Batch-weise Daten zurückliefert (ContinuationPointOut != 0), so kann der Wert von ContinuationPointOut beim nächsten Aufruf des Funktionsbausteins als ContinuationPointIn verwendet werden um die weiteren Daten abzurufen.
cbBrowseResultCnt	UDINT	Anzahl der ReferenceDescriptions.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.2.2 UA_Connect



Dieser Funktionsbaustein stellt eine OPC-UA-Remote-Verbindung zu einem anderen OPC UA Server her, der via ServerUrl und SessionConnectInfo spezifiziert wird. Der Funktionsbaustein gibt ein Verbindungshandle zurück, der für andere Funktionsbausteine, z. B. UA_Read, verwendet werden kann.

Eingänge

```
VAR_INPUT
    Execute          : BOOL;
    ServerUrl        : STRING(MAX_STRING_LENGTH);
    SessionConnectInfo : ST_UASessionConnectInfo;
    Timeout          : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Typ	Beschreibung
Execute	BOOL	Der Befehl wird durch eine steigende Flanke an diesem Eingang ausgelöst.
ServerUrl	STRING(MAX_STRING_LENGTH)	OPC UA Server URL, d. h. 'opc.tcp://172.16.3.207:4840' oder 'opc.tcp://CX_0193BF:4840'.
SessionConnectInfo	ST_UASessionConnectInfo	Verbindungsinformation (siehe ST_UASessionConnectInfo [▶ 227])
Timeout	TIME	Zeit bis zum Abbruch der Funktion. DEFAULT_ADS_TIMEOUT ist eine globale Konstante, gesetzt auf 5 Sekunden. Der Wert muss passend zum ST_UASessionConnectInfo.tConnectionTimeout gesetzt werden. Hierbei gilt die Faustregel: ADS Timeout > 2 * ConnectionTimeout.

Ausgänge

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

Name	Typ	Beschreibung
ConnectionHdl	DWORD	OPC UA Verbindungshandle.
Done	BOOL	Schaltet auf TRUE, wenn der Funktionsbaustein erfolgreich ausgeführt wurde.
Busy	BOOL	TRUE, bis der Baustein einen Befehl ausführt hat, maximal für die Dauer des „Timeout“ am Eingang. Solange Busy = TRUE ist, akzeptieren die Eingänge keinen neuen Befehl. Es wird nicht die Zeit der Verbindung, sondern die Empfangszeit überwacht.
Error	BOOL	Schaltet auf TRUE, wenn bei der Ausführung eines Befehls ein Fehler auftritt. Der befehlspezifische Fehlercode ist in ErrorID enthalten.
ErrorID	DWORD	Enthält den befehlspezifischen Fehlercode des zuletzt ausgeführten Befehls.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.2.3 UA_ConnectGetStatus

UA_ConnectGetStatus	
Execute <i>BOOL</i>	<i>BOOL</i> Done
ConnectionHdl <i>DWORD</i>	<i>BOOL</i> Busy
GetServiceLevel <i>BOOL</i>	<i>BOOL</i> Error
Timeout <i>TIME</i>	<i>DWORD</i> ErrorID
	<i>E_UAConnectionStatus</i> ConnectionStatus
	<i>E_UAServerState</i> ServerState
	<i>BYTE</i> ServiceLevel

Dieser Funktionsbaustein überprüft den Verbindungsstatus einer existierenden Verbindung zu einem anderen OPC UA Server. Die Verbindung wird hierbei über den jeweiligen Connection Handle referenziert. Der Status wird dann als `E_UAConnectionStatus` [► 221] zurückgegeben. Der Verbindungsstatus wird anhand der internen Session-Info bzw. des OPC UA Heartbeats ermittelt, es wird keine zusätzliche Kommunikation (Read o.ä.) durchgeführt.

Über den zusätzlichen Eingabeparameter `GetServiceLevel` lässt sich das `ServiceLevel` des OPC UA Servers auslesen. Hierfür wird im Hintergrund ein Lesebefehl an den Server abgesetzt, um diese Information zu ermitteln.

Eingänge

```
VAR_INPUT
  Execute          : BOOL;
  ConnectionHdl   : DWORD;
  GetServiceLevel : BOOL;
  Timeout         : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Typ	Beschreibung
Execute	BOOL	Der Befehl wird durch eine steigende Flanke an diesem Eingang ausgelöst.
ConnectionHdl	DWORD	Connection Handle einer existierenden Kommunikationsverbindung.
GetServiceLevel	BOOL	Liest das <code>ServiceLevel</code> des OPC UA Servers aus.
Timeout	TIME	Zeit bis zum Abbruch der Funktion. <code>DEFAULT_ADS_TIMEOUT</code> ist eine globale Konstante, gesetzt auf 5 Sekunden. Der Wert muss passend zum <code>ST_UASessionConnectInfo.tConnectionTimeout</code> gesetzt werden. Hierbei gilt die Faustregel: <code>ADS Timeout > 2 * ConnectionTimeout</code> .

Ausgänge

```
VAR_OUTPUT
  Done          : BOOL;
  Busy          : BOOL;
  Error         : BOOL;
  ErrorID       : DWORD;
  ConnectionStatus : E_UAConnectionStatus;
  ServerState   : E_UAServerState;
  ServiceLevel  : BYTE;
END_VAR
```

Name	Typ	Beschreibung
Done	BOOL	Schaltet auf TRUE, wenn der Funktionsbaustein erfolgreich ausgeführt wurde.
Busy	BOOL	TRUE, bis der Baustein einen Befehl ausführt hat, maximal für die Dauer des „Timeout“ am Eingang. Solange Busy = TRUE ist, akzeptieren die Eingänge keinen neuen Befehl. Es wird nicht die Zeit der Verbindung sondern die Empfangszeit überwacht.
Error	BOOL	Schaltet auf TRUE, wenn bei der Ausführung eines Befehls ein Fehler auftritt. Der befehlspezifische Fehlercode ist in ErrorID enthalten.
ErrorID	DWORD	Enthält den befehlspezifischen Fehlercode des zuletzt ausgeführten Befehls.
ConnectionStatus	E_UAConnectionStatus	Verbindungsstatus (siehe E_UAConnectionStatus [▶ 221]).
ServerState	E_UAServerState	Serverstatus (siehe E_UAServerState [▶ 225]).
ServerState	BYTE	ServiceLevel des OPC UA Servers.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

SPS-Bibliothek	Benötigte Version
Tc3_PLCOpen_OpcUa	>= 3.2.11.0

5.2.2.4 UA_Disconnect



Dieser Funktionsbaustein schließt eine OPC-UA-Remote-Verbindung zu einem anderen OPC UA Server. Die Verbindung wird über ihr Verbindungshandle spezifiziert.

● Trennen aller Verbindungen

i Wenn die UA-Disconnect-Methode aufgerufen wird und ein Connection Handle von 0 übergeben wird, trennt der OPC UA-Client alle bestehenden Verbindungen. Das gilt auch für Verbindungen, die über eine OPC UA I/O-Client-Konfiguration aufgebaut wurden.

📁 Eingänge

```
VAR_INPUT
    Execute      : BOOL;
    ConnectionHdl : DWORD;
    Timeout      : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Typ	Beschreibung
Execute	BOOL	Der Befehl wird durch eine steigende Flanke an diesem Eingang ausgelöst.
ConnectionHdl	DWORD	Verbindungshandle, das vorher vom Funktionsbaustein UA_Connect ausgegeben wurde.
Timeout	TIME	Zeit bis zum Abbruch der Funktion. DEFAULT_ADS_TIMEOUT ist eine globale Konstante, gesetzt auf 5 Sekunden.

Ausgänge

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

Name	Typ	Beschreibung
Done	BOOL	Schaltet auf TRUE, wenn der Funktionsbaustein erfolgreich ausgeführt wurde.
Busy	BOOL	TRUE, bis der Baustein einen Befehl ausführt hat, maximal für die Dauer des „Timeout“ am Eingang. Solange Busy = TRUE ist, akzeptieren die Eingänge keinen neuen Befehl. Es wird nicht die Zeit der Verbindung sondern die Empfangszeit überwacht.
Error	BOOL	Schaltet auf TRUE, wenn bei der Ausführung eines Befehls ein Fehler auftritt. Der befehlspezifische Fehlercode ist in nErrID enthalten.
ErrorID	DWORD	Enthält den befehlspezifischen Fehlercode des zuletzt ausgeführten Befehls.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.2.5 UA_GetNamespaceIndex



Dieser Funktionsbaustein erfasst den Namespace Index für einen Namespace URI. Der Namespace Index wird für die Identifizierung von Symbolen benötigt, z. B. wenn die Funktionsbausteine [UA_Read](#) [[▶ 250](#)] oder [UA_Write](#) [[▶ 253](#)] genutzt werden.

Eingänge

```
VAR_INPUT
  Execute      : BOOL;
  ConnectionHdl : DWORD;
  NamespaceUri : STRING(MAX_STRING_LENGTH);
  Timeout      : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Typ	Beschreibung
Execute	BOOL	Der Befehl wird durch eine steigende Flanke an diesem Eingang ausgelöst.
ConnectionHdl	DWORD	Verbindungshandle, das vorher vom Funktionsbaustein UA_Connect ausgegeben wurde.
NamespaceUri	STRING	Namensraum-URI, der aufgelöst werden soll. Beim TwinCAT OPC UA Server ist das für die erste SPS-Laufzeit „urn:BeckhoffAutomation:Ua:PLC1“.
Timeout	TIME	Zeit bis zum Abbruch der Funktion. DEFAULT_ADS_TIMEOUT ist eine globale Konstante, gesetzt auf 5 Sekunden.

Ausgänge

```
VAR_OUTPUT
  NamespaceIndex : UINT;
  Done          : BOOL;
```

```

    Busy      : BOOL;
    Error     : BOOL;
    ErrorID   : DWORD;
END_VAR

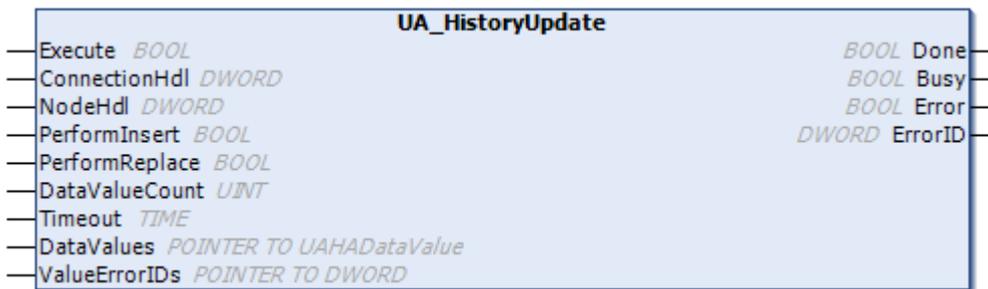
```

Name	Typ	Beschreibung
NamespaceIndex	UINT	Namespace Index des gegebenen Namensraum-URI. Dieser kann in anderen Funktionsbausteinen verwendet werden, z. B. UA_NodeGetHandle oder UA_MethodGetHandle.
Done	BOOL	Schaltet auf TRUE, wenn der Funktionsbaustein erfolgreich ausgeführt wurde.
Busy	BOOL	TRUE, bis der Baustein einen Befehl ausführt hat, maximal für die Dauer des „Timeout“ am Eingang. Solange Busy = TRUE ist, akzeptieren die Eingänge keinen neuen Befehl. Es wird nicht die Zeit der Verbindung sondern die Empfangszeit überwacht.
Error	BOOL	Schaltet auf TRUE, wenn bei der Ausführung eines Befehls ein Fehler auftritt. Der befehlspezifische Fehlercode ist in ErrorID enthalten.
ErrorID	DWORD	Enthält den befehlspezifischen Fehlercode des zuletzt ausgeführten Befehls.

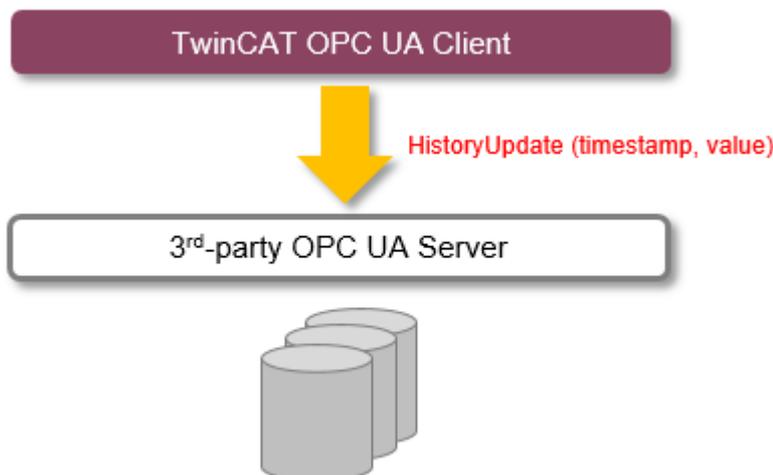
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.2.6 UA_HistoryUpdate



Dieser Funktionsbaustein schickt historische Daten über OPC UA zu einem Server, der die OPC-UA-HistoryUpdate-Funktionalität unterstützt, z. B. dem TwinCAT OPC UA Server. Mit einem Aufruf können Sie für ein Knotenhandle eine Vielzahl von Werten inklusive Zeitstempel an den Server übertragen. Dieser sorgt dafür, dass die übermittelten Werte in einem Datenspeicher gespeichert und über Historical Access verfügbar gemacht werden.



Wenn Werte von mehreren Knotenhandle (verschiedenen Variablen) übertragen werden sollen, kann der Funktionsbaustein mehrfach instanziiert werden.

Betrieb mit TwinCAT OPC UA Server

Der Funktionsbaustein eignet sich gut, wenn Sie im TwinCAT OPC UA Server Historical Access nutzen und Daten aus einem bestimmten Zeitintervall, in dem z. B. ein spezieller Maschinenzustand vorherrschte, zur Verfügung stellen wollen. Es lassen sich gezielt Werte für den gewünschten Zeitraum übertragen.

Wenn Werte hingegen zyklisch gesendet und über Historical Access im Server verfügbar gemacht werden sollen, so eignet sich die serverseitige Historical-Access-Funktionalität besser, da Sie hier lediglich die aufzuzeichnende Node im Konfigurator konfigurieren und die gewünschte Abtastrate einstellen müssen.

Siehe auch: [Programmbeispiel TF6100 OPCUA HASample \[▶ 255\]](#)

Eingänge

```
VAR_INPUT
  Execute      : BOOL;
  ConnectionHdl : DWORD;
  NodeHdl      : DWORD;
  PerformInsert : BOOL;
  PerformReplace : BOOL;
  DataValueCount : UINT;
  Timeout      : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Typ	Beschreibung
Execute	BOOL	Der Befehl wird durch eine steigende Flanke an diesem Eingang ausgelöst.
ConnectionHdl	DWORD	Verbindungshandle, das vorher vom Funktionsbaustein UA_Connect ausgegeben wurde.
NodeHdl	DWORD	Knotenhandle, der zuvor vom Funktionsbaustein UA_NodeGetHandle ausgegeben wurde.
PerformInsert	BOOL	Der Default ist TRUE.
PerformReplace	BOOL	Der Default ist FALSE. Sofern in der Historie bereits ein Wert für den gegebenen Zeitstempel existiert, soll dieser ersetzt werden, wenn die Option PerformReplace gesetzt ist (= TRUE). Diese Option kann aktuell nur für SQL Adapter ausgewählt werden. Andere Adapter unterstützen die Option nicht.
DataValueCount	UINT	Legt die Anzahl der übergebenen Werte fest. Eine Maximalanzahl von 1000 Werten wird unterstützt.
Timeout	TIME	Zeit bis zum Abbruch der Funktion. DEFAULT_ADS_TIMEOUT ist eine globale Konstante, gesetzt auf 5 Sekunden.

Ein-/Ausgänge

```
VAR_IN_OUT
  DataValues      : ARRAY[*] OF UAHADataValue;
  ValueErrorIDs   : ARRAY[*] OF DWORD;
END_VAR
```

Name	Typ	Beschreibung
DataValues (read-only)	ARRAY	Es werden alle gesammelten Werte in Form eines Feldes vom Typ UAHADataValue übergeben. Die Länge des Feldes ist nicht vorgeschrieben, muss jedoch mindestens der Angabe von DataValueCount entsprechen. Auf die Werte wird intern nur lesend zugegriffen.
ValueErrorIDs (write-only)	ARRAY	Nach Ausführung des Befehls beinhaltet dieses Feld für jeden Wert einen Fehlercode. Die Länge des Feldes muss mindestens der Angabe von DataValueCount entsprechen. Wenn ein oder mehrere Werte einen Fehler melden, so wird dies auch über die Ausgänge Error und ErrorID des Funktionsbausteins signalisiert. Mithilfe dieses Feldes kann daraufhin ermittelt werden, für welchen Wert welcher Fehler aufgetreten ist. Der Fehlercode 16#80000000 beispielsweise signalisiert eine fehlgeschlagene Operation, sodass der Wert nicht geschrieben werden konnte.

 **Ausgänge**

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

Name	Typ	Beschreibung
Done	BOOL	Schaltet auf TRUE, wenn der Funktionsbaustein erfolgreich ausgeführt wurde.
Busy	BOOL	TRUE, bis der Baustein einen Befehl ausführt hat, maximal für die Dauer des „Timeout“ am Eingang. Solange Busy = TRUE ist, akzeptieren die Eingänge keinen neuen Befehl. Es wird nicht die Zeit der Verbindung sondern die Empfangszeit überwacht.
Error	BOOL	Schaltet auf TRUE, wenn bei der Ausführung eines Befehls ein Fehler auftritt. Der befehlspezifische Fehlercode ist in ErrorID enthalten.
ErrorID	DWORD	Enthält den befehlspezifischen ADS Fehlercode des zuletzt ausgeführten Befehls.

 **Anzahl der übergebenen Werte**

i Je größer die Anzahl ist, umso größer ist auch der benötigte Rechenaufwand und damit die SPS-Ausführungsdauer bei Befehlsausführung.

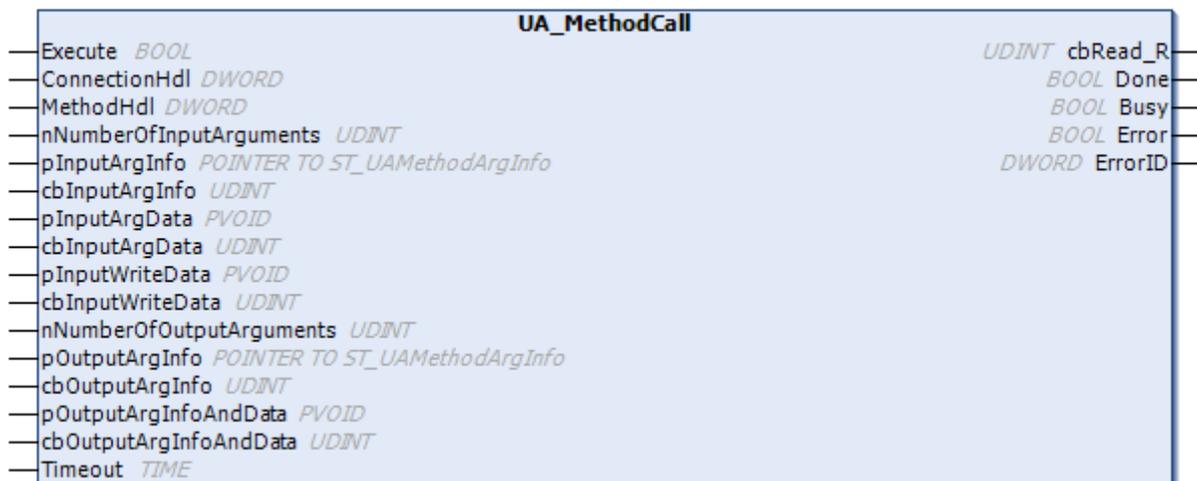
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT 3.1 >= 4024.1	Win32, Win64, WinCE-x86	Tc3_PLCOpen_OpcUa >= v3.1.9.0

Sehen Sie dazu auch

-  [UA_Connect \[▶ 235\]](#)
-  [UA_NodeGetHandle \[▶ 246\]](#)
-  [UAHADataValue \[▶ 231\]](#)

5.2.2.7 UA_MethodCall



Dieser Funktionsbaustein ruft eine Methode auf einem Remote-UA-Server auf. Die Methode wird durch eine Verbindung und ein Methodenhandle bestimmt. Erstere kann durch [UA_Connect](#) [► 235] und letzterer durch [UA_MethodGetHandle](#) [► 244] abgefragt werden.

Eingänge

```

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	Typ	Beschreibung
Execute	BOOL	Der Befehl wird durch eine steigende Flanke an diesem Eingang ausgelöst.
ConnectionHdl	DWORD	Verbindungshandle, das vorher vom Funktionsbaustein UA_Connect ausgegeben wurde.
MethodHdl	DWORD	Methodenhandle, zuvor vom Funktionsbaustein UA_MethodGetHandle ausgegeben.
nNumberOfInputArguments	UDINT	Anzahl Eingabeparameter.
pInputArgInfo	POINTER TO ST_UAMethodArgInfo	Zeigt auf die Pufferadresse, wo Eingabeparameterinformationen in Form eines Arrays ST_UAMethodArgInfo hinterlegt sind.
cbInputArgInfo	UDINT	Größe des Puffers, wo die Eingabeparameterinformation hinterlegt ist.
pInputArgData	PVOID	Zeigt auf die Pufferadresse, wo Eingabeparameter (konstanter Länge) hinterlegt sind.
cbInputArgData	UDINT	Größe des Eingabepuffers, wo Eingabeparameter (mit konstanter Länge) hinterlegt sind.
pInputWriteData	PVOID	Zeiger auf Pufferadresse, wo Eingabeparameter (dynamischer Länge) hinterlegt sind.
cbInputWriteData	UDINT	Größe des Eingabepuffers, wo Eingabeparameter (mit dynamischer Länge) hinterlegt sind.
nNumberOfOutputArguments	UDINT	Anzahl Ausgabeparameter.
pOutputArgInfo	POINTER TO ST_UAMethodArgInfo	Zeigt auf die Pufferadresse, wo Ausgabeparameterinformationen als Array ST_UAMethodArgInfo hinterlegt sind. Für die Bestimmung des Zielspeichers der einzelnen Ausgabeparameter ist nLenData erforderlich. Die anderen Elemente können so gesetzt werden, dass eine Typprüfung der zurückgegebenen Parameter erfolgt oder undefiniert bleiben.
cbOutputArgInfo	UDINT	Größe des Puffers, wo die Ausgabeparameterinformation hinterlegt ist.
pOutputArgInfoAndData	PVOID	Zeigt auf die Pufferadresse, wo die Ausgabeparameter als BYTE-Array gespeichert werden sollen. Das BYTE-Array enthält die Anzahl der Ausgabeparameter als DINT, 4 reservierte Bytes und Parameterinformationen als ARRAY OF ST_UAMethodArgInfo [► 229] (mit der Länge der Ausgabeparameter) gefolgt von reinen Daten. Beachten Sie, dass die Daten als 1-byte-Alignment gepackt sind.
cbOutputArgInfoAndData	UDINT	Größe des Puffers, in dem die Ausgabeparameter als BYTE-Array gespeichert werden sollen.

Name	Typ	Beschreibung
Timeout	TIME	Zeit bis zum Abbruch der Funktion. DEFAULT_ADS_TIMEOUT ist eine globale Konstante, gesetzt auf 5 Sekunden.

Ausgänge

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

Name	Typ	Beschreibung
cbRead_R	UDINT	Zählt alle empfangenen Bytes.
Done	BOOL	Schaltet auf TRUE, wenn der Funktionsbaustein erfolgreich ausgeführt wurde.
Busy	BOOL	TRUE, bis der Baustein einen Befehl ausführt hat, maximal für die Dauer des „Timeout“ am Eingang. Solange Busy = TRUE ist, akzeptieren die Eingänge keinen neuen Befehl. Es wird nicht die Zeit der Verbindung sondern die Empfangszeit überwacht.
Error	BOOL	Schaltet auf TRUE, wenn bei der Ausführung eines Befehls ein Fehler auftritt. Der befehlspezifische Fehlercode ist in nErrID enthalten.
ErrorID	UDINT	Enthält den befehlspezifischen Fehlercode des zuletzt ausgeführten Befehls.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.2.8 UA_MethodGetHandle



Dieser Funktionsbaustein erfasst ein Handle für eine UA-Methode, das dann für den Aufruf einer Methode über [UA_MethodCall](#) [► 242] verwendet werden kann.

Eingänge

```
VAR_INPUT
  Execute       : BOOL;
  ConnectionHdl : DWORD;
  ObjectNodeID  : ST_UANodeID;
  MethodNodeID  : ST_UANodeID;
  Timeout       : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Typ	Beschreibung
Execute	BOOL	Der Befehl wird durch eine steigende Flanke an diesem Eingang ausgelöst.
ConnectionHdl	DWORD	Verbindungshandle, das vorher vom Funktionsbaustein UA_Connect ausgegeben wurde.
ObjectNodeID	ST_UANodeID	Objektknoten-ID der aufzurufenden Methode. (Typ: ST_UANodeID [▶ 229]).
MethodNodeID	ST_UANodeID	Methoden-Knoten-ID der aufzurufenden Methode. Entspricht dem ID-Attribut im UA-Namensraum. (Typ: UA_Connect [▶ 235]).
Timeout	TIME	Zeit bis zum Abbruch der Funktion. DEFAULT_ADS_TIMEOUT ist eine globale Konstante, gesetzt auf 5 Sekunden.

Ausgänge

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

Name	Typ	Beschreibung
MethodHdl	DWORD	Gibt ein Methodenhandle zurück, das für den Aufruf einer Methode über UA MethodCall [▶ 242] verwendet werden kann.
Done	BOOL	Schaltet auf TRUE, wenn der Funktionsbaustein erfolgreich ausgeführt wurde.
Busy	BOOL	TRUE, bis der Baustein einen Befehl ausführt hat, maximal für die Dauer des „Timeout“ am Eingang. Solange Busy = TRUE ist, akzeptieren die Eingänge keinen neuen Befehl. Es wird nicht die Zeit der Verbindung sondern die Empfangszeit überwacht.
Error	BOOL	Schaltet auf TRUE, wenn bei der Ausführung eines Befehls ein Fehler auftritt. Der befehlspezifische Fehlercode ist in nErrID enthalten.
ErrorID	UDINT	Enthält den befehlspezifischen Fehlercode des zuletzt ausgeführten Befehls.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.2.9 UA_MethodReleaseHandle



Dieser Funktionsbaustein gibt das spezifizierte Methodenhandle frei.

Eingänge

```
VAR_INPUT
  Execute      : BOOL;
  ConnectionHdl : DWORD;
  MethodHdl    : DWORD;
  Timeout      : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Typ	Beschreibung
Execute	BOOL	Der Befehl wird durch eine steigende Flanke an diesem Eingang ausgelöst.
ConnectionHdl	DWORD	Verbindungshandle, das vorher vom Funktionsbaustein UA_Connect ausgegeben wurde.
MethodHdl	DWORD	Methodenhandle, das zuvor vom Funktionsbaustein UA_MethodGetHandle ausgegeben wurde.
Timeout	TIME	Zeit bis zum Abbruch der Funktion. DEFAULT_ADS_TIMEOUT ist eine globale Konstante, gesetzt auf 5 Sekunden.

Ausgänge

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

Name	Typ	Beschreibung
Done	BOOL	Schaltet auf TRUE, wenn der Funktionsbaustein erfolgreich ausgeführt wurde.
Busy	BOOL	TRUE, bis der Baustein einen Befehl ausführt hat, maximal für die Dauer des „Timeout“ am Eingang. Solange Busy = TRUE ist, akzeptieren die Eingänge keinen neuen Befehl. Es wird nicht die Zeit der Verbindung sondern die Empfangszeit überwacht.
Error	BOOL	Schaltet auf TRUE, wenn bei der Ausführung eines Befehls ein Fehler auftritt. Der befehlspezifische Fehlercode ist in nErrID enthalten.
ErrorID	UDINT	Enthält den befehlspezifischen ADS Fehlercode des zuletzt ausgeführten Befehls.

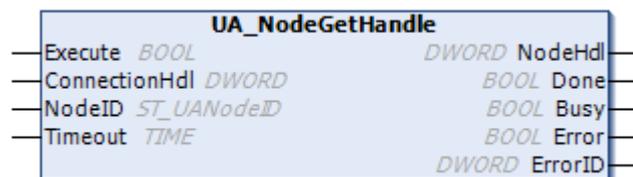
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

Sehen Sie dazu auch

-  [UA_Connect \[▶ 235\]](#)
-  [UA_MethodGetHandle \[▶ 244\]](#)

5.2.2.10 UA_NodeGetHandle



Dieser Funktionsbaustein fragt ein Knotenhandle für ein gegebenes Symbol im UA-Namensraum ab. Das Symbol wird durch ein Verbindungshandle und seine Knoten-ID spezifiziert.

Eingänge

```
VAR_INPUT
  Execute      : BOOL;
  ConnectionHdl : DWORD;
  NodeID       : ST_UANodeID;
  Timeout      : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Typ	Beschreibung
Execute	BOOL	Der Befehl wird durch eine steigende Flanke an diesem Eingang ausgelöst.
ConnectionHdl	DWORD	Verbindungshandle, das vorher vom Funktionsbaustein UA_Connect ausgegeben wurde.
Node ID	ST_UANodeID	Eindeutige Adressierung der UA Node, bestehend aus Identifier, IdentifierType und NamespaceIndex, welcher aus einem NamespaceName aufgelöst wird, z.B. mittels der Methode UA_GetNamespaceIndex [▶ 238].
Timeout	TIME	Zeit bis zum Abbruch der Funktion. DEFAULT_ADS_TIMEOUT ist eine globale Konstante, gesetzt auf 5 Sekunden.

Ausgänge

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

Name	Typ	Beschreibung
NodeHdl	DWORD	Knotenhandle, das für andere Funktionsbausteine verwendet werden kann, z. B. UA_Read oder UA_Write.
Done	BOOL	Schaltet auf TRUE, wenn der Funktionsbaustein erfolgreich ausgeführt wurde.
Busy	BOOL	TRUE, bis der Baustein einen Befehl ausführt hat, maximal für die Dauer des „Timeout“ am Eingang. Solange Busy = TRUE akzeptieren die Eingänge keinen neuen Befehl. Es wird nicht die Zeit der Verbindung sondern die Empfangszeit überwacht.
Error	BOOL	Schaltet auf TRUE, wenn bei der Ausführung eines Befehls ein Fehler auftritt. Der befehlspezifische Fehlercode ist in ErrorID enthalten.
ErrorID	DWORD	Enthält den befehlspezifischen ADS Fehlercode des zuletzt ausgeführten Befehls.

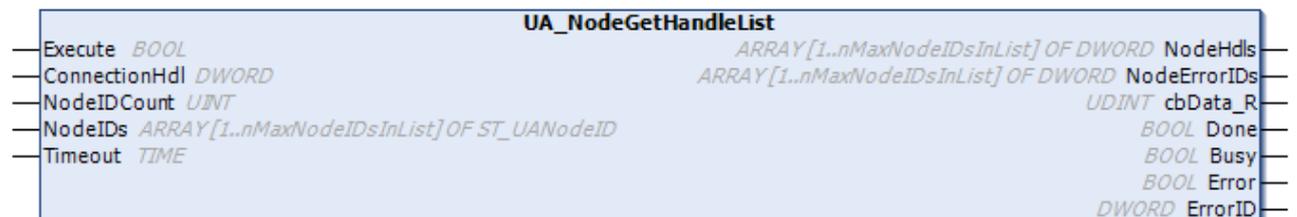
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

Sehen Sie dazu auch

UA_Connect [▶ 235]

5.2.2.11 UA_NodeGetHandleList



Dieser Funktionsbaustein fragt Knotenhandles für Knoten im UA-Namensraum ab.

Eingänge

```
VAR_INPUT
  Execute      : BOOL;
  ConnectionHdl : DWORD;
```

```

NodeIDCount      : UINT;
NodeIDs          : ARRAY[1..nMaxNodeIDsInList] OF ST_UANodeID;
Timeout         : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR

```

Name	Typ	Beschreibung
Execute	BOOL	Der Befehl wird durch eine steigende Flanke an diesem Eingang ausgelöst.
ConnectionHdl	DWORD	Verbindungshandle, das vorher vom Funktionsbaustein UA_Connect ausgegeben wurde.
NodeIDCount	UINT	Anzahl Knoten, für die ein Knotenhandle erforderlich ist.
NodeIDs	ARRAY	Array von NodeIDs, die mit der struct ST_UANodeID [► 229] erstellt wurden.
Timeout	TIME	Zeit bis zum Abbruch der Funktion. DEFAULT_ADS_TIMEOUT ist eine globale Konstante, gesetzt auf 5 Sekunden.

Ausgänge

```

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	Typ	Beschreibung
NodeHdls	ARRAY	Array angeforderter Knotenhandles.
NodeErrorIDs	ARRAY	Array von Fehler-IDs, falls keine Knotenhandles zur Verfügung stehen.
cbData_R	UDINT	Größe der gelesenen Daten.
Done	BOOL	Schaltet auf TRUE, wenn der Funktionsbaustein erfolgreich ausgeführt wurde.
Busy	BOOL	TRUE, bis der Baustein einen Befehl ausführt hat, maximal für die Dauer des „Timeout“ am Eingang. Solange Busy = TRUE ist, akzeptieren die Eingänge keinen neuen Befehl. Es wird nicht die Zeit der Verbindung sondern die Empfangszeit überwacht.
Error	BOOL	Schaltet auf TRUE, wenn bei der Ausführung eines Befehls ein Fehler auftritt. Der befehlspezifische Fehlercode steht in nErrID.
ErrorID	DWORD	Enthält die Fehler-ID, wenn ein Fehler auftritt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

Sehen Sie dazu auch

 [UA_Connect](#) [► 235]

5.2.2.12 UA_NodeReleaseHandle



Dieser Funktionsbaustein gibt ein Knotenhandle frei.

Eingänge

```
VAR_INPUT
    Execute      : BOOL;
    ConnectionHdl : DWORD;
    NodeHdl      : DWORD;
    Timeout      : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Typ	Beschreibung
Execute	BOOL	Der Befehl wird durch eine steigende Flanke an diesem Eingang ausgelöst.
ConnectionHdl	DWORD	Verbindungshandle, das vorher vom Funktionsbaustein UA_Connect ausgegeben wurde.
NodeHdl	DWORD	Freizugebendes Knotenhandle.
Timeout	TIME	Zeit bis zum Abbruch der Funktion. DEFAULT_ADS_TIMEOUT ist eine globale Konstante, gesetzt auf 5 Sekunden.

Ausgänge

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

Name	Typ	Beschreibung
Done	BOOL	Schaltet auf TRUE, wenn der Funktionsbaustein erfolgreich ausgeführt wurde.
Busy	BOOL	TRUE, bis der Baustein einen Befehl ausführt hat, maximal für die Dauer des „Timeout“ am Eingang. Solange Busy = TRUE ist, akzeptieren die Eingänge keinen neuen Befehl. Es wird nicht die Zeit der Verbindung sondern die Empfangszeit überwacht.
Error	BOOL	Schaltet auf TRUE, wenn bei der Ausführung eines Befehls ein Fehler auftritt. Der befehlspezifische Fehlercode ist in ErrorID enthalten.
ErrorID	DWORD	Enthält den befehlspezifischen ADS Fehlercode des zuletzt ausgeführten Befehls.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

Sehen Sie dazu auch

UA_Connect [▶ 235]

5.2.2.13 UA_NodeReleaseHandleList



Dieser Funktionsbaustein gibt mehrere Knotenhandles frei.

Eingänge

```
VAR_INPUT
    Execute      : BOOL;
    ConnectionHdl : DWORD;
    NodeHdlCount : UINT;
END_VAR
```

```

NodeHdls      : ARRAY[1..nMaxNodeIDsInList] OF DWORD;
Timeout      : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR

```

Name	Typ	Beschreibung
Execute	BOOL	Der Befehl wird durch eine steigende Flanke an diesem Eingang ausgelöst.
ConnectionHdl	DWORD	Verbindungshandle, das vorher vom Funktionsbaustein UA_Connect ausgegeben wurde.
NodeHdlCount	UINT	Anzahl Knotenhandles.
NodeHdls	ARRAY	Array von Knotenhandles, die freizugeben sind.
Timeout	TIME	Zeit bis zum Abbruch der Funktion. DEFAULT_ADS_TIMEOUT ist eine globale Konstante, gesetzt auf 5 Sekunden.

Ausgänge

```

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

```

Name	Typ	Beschreibung
NodeErrorIDs	ARRAY	Array von Fehler-IDs, falls ein Knotenhandle nicht freigegeben werden konnte.
Done	BOOL	Schaltet auf TRUE, wenn der Funktionsbaustein erfolgreich ausgeführt wurde.
Busy	BOOL	TRUE, bis der Baustein einen Befehl ausführt hat, maximal für die Dauer des „Timeout“ am Eingang. Solange Busy = TRUE ist, akzeptieren die Eingänge keinen neuen Befehl. Es wird nicht die Zeit der Verbindung sondern die Empfangszeit überwacht.
Error	BOOL	Schaltet auf TRUE, wenn bei der Ausführung eines Befehls ein Fehler auftritt. Der befehlspezifische Fehlercode steht in nErrID.
ErrorID	DWORD	Enthält die Fehler-ID, wenn ein Fehler auftritt.

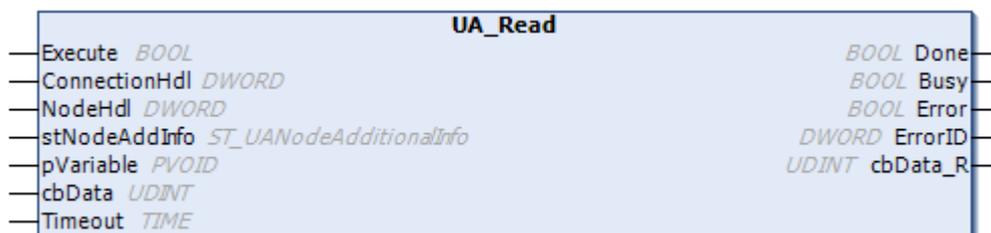
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

Sehen Sie dazu auch

 UA_Connect [▶ 235]

5.2.2.14 UA_Read



Dieser Funktionsbaustein liest Werte aus einem gegebenen Knoten- und Verbindungshandle.

 **Eingänge**

```
VAR_INPUT
  Execute           : BOOL;
  ConnectionHdl    : DWORD;
  NodeHdl          : DWORD;
  stNodeAddInfo    : ST_UANodeAdditionalInfo;
  pVariable        : PVOID;
  cbData           : UDINT;
  Timeout          : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Typ	Beschreibung
Execute	BOOL	Der Befehl wird durch eine steigende Flanke an diesem Eingang ausgelöst.
Connection Hdl	DWORD	Verbindungshandle, das vorher vom Funktionsbaustein UA_Connect ausgegeben wurde.
NodeHdl	DWORD	Knotenhandle, das zuvor vom Funktionsbaustein UA_NodeGetHandle ausgegeben wurde.
stNodeAddInfo	ST_UANodeAdditionalInfo	Definiert zusätzliche Informationen, z. B. welches Attribut aus dem UA-Namensraum gelesen (Standard: 'Value'-Attribut) oder welcher IndexRange verwendet werden soll. Wird durch STRUCT ST_UANodeAdditionalInfo [▶ 230] spezifiziert.
pVariable	PVOID	Zeiger auf Datenspeicher, wo die gelesenen Daten abgespeichert werden sollen.
cbData	UDINT	Bestimmt die Größe der zu lesenden Daten.
Timeout	TIME	Zeit bis zum Abbruch der Funktion. DEFAULT_ADS_TIMEOUT ist eine globale Konstante, gesetzt auf 5 Sekunden.

 **Ausgänge**

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

Name	Typ	Beschreibung
Done	BOOL	Schaltet auf TRUE, wenn der Funktionsbaustein erfolgreich ausgeführt wurde.
Busy	BOOL	TRUE, bis der Baustein einen Befehl ausführt hat, maximal für die Dauer des „Timeout“ am Eingang. Solange Busy = TRUE ist, akzeptieren die Eingänge keinen neuen Befehl. Es wird nicht die Zeit der Verbindung sondern die Empfangszeit überwacht.
Error	BOOL	Schaltet auf TRUE, wenn bei der Ausführung eines Befehls ein Fehler auftritt. Der befehlspezifische Fehlercode ist in nErrID enthalten.
ErrorID	UDINT	Enthält den befehlspezifischen ADS-Fehlercode des zuletzt ausgeführten Befehls.
cbData_R	UDINT	Anzahl zu lesender Bytes.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

Sehen Sie dazu auch

-  [UA_Connect](#) [▶ 235]
-  [UA_NodeGetHandle](#) [▶ 246]

5.2.2.15 UA_ReadList



Dieser Funktionsbaustein liest Werte aus mehreren gegebenen Knoten- und Verbindungshandles.

Eingänge

```

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	Typ	Beschreibung
Execute	BOOL	Der Befehl wird durch eine steigende Flanke an diesem Eingang ausgelöst.
ConnectionHdl	DWORD	Verbindungshandle, das vorher vom Funktionsbaustein UA_Connect [▶ 235] ausgegeben wurde.
NodeHdlCount	UINT	Anzahl Knotenhandles, die in der Eingangsvariablen NodeHdls gespeichert sind.
NodeHdls	ARRAY	Array von Knotenhandles, die vorher vom Funktionsbaustein UA_NodeGetHandle [▶ 246] oder UA_NodeGetHandleList [▶ 247] erhalten wurden.
stNodeAddInfo	ARRAY	Definiert zusätzliche Informationen, z. B. welches Attribut aus dem UA-Namensraum gelesen (Standard: 'Value'-Attribut) oder welcher IndexRange verwendet werden soll. Wird durch STRUCT ST_UANodeAdditionalInfo [▶ 230] spezifiziert.
pVariable	PVOID	Zeiger auf Datenspeicher, wo die gelesenen Daten abgespeichert werden sollen.
cbData	ARRAY	Bestimmt die Größe der zu lesenden Daten.
cbDataTotal	UDINT	Gesamtgröße der zu empfangenden Daten.
Timeout	TIME	Zeit bis zum Abbruch der Funktion. DEFAULT_ADS_TIMEOUT ist eine globale Konstante, gesetzt auf 5 Sekunden.

Ausgänge

```

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

```

Name	Typ	Beschreibung
Done	BOOL	Schaltet auf TRUE, wenn der Funktionsbaustein erfolgreich ausgeführt wurde.
Busy	BOOL	TRUE, bis der Baustein einen Befehl ausführt hat, maximal für die Dauer des „Timeout“ am Eingang. Solange Busy = TRUE ist, akzeptieren die Eingänge keinen neuen Befehl. Es wird nicht die Zeit der Verbindung sondern die Empfangszeit überwacht.
Error	BOOL	Schaltet auf TRUE, wenn bei der Ausführung eines Befehls ein Fehler auftritt. Der befehlspezifische Fehlercode steht in nErrID.
ErrorID	UDINT	Enthält den befehlspezifischen ADS-Fehlercode des zuletzt ausgeführten Befehls.
cbData_R	UDINT	Anzahl der gelesenen Bytes.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

5.2.2.16 UA_Write



Dieser Funktionsbaustein schreibt Werte in ein gegebenes Knoten- und Verbindungshandle.

Eingänge

```

VAR_INPUT
  Execute      : BOOL;
  ConnectionHdl : DWORD;
  NodeHdl     : DWORD;
  stNodeAddInfo : ST_UANodeAdditionalInfo;
  pVariable   : PVOID;
  cbData      : UDINT;
  Timeout     : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
    
```

Name	Typ	Beschreibung
Execute	BOOL	Der Befehl wird durch eine steigende Flanke an diesem Eingang ausgelöst.
ConnectionHdl	DWORD	Verbindungshandle, das vorher vom Funktionsbaustein UA_Connect [► 235] ausgegeben wurde.
NodeHdl	DWORD	Knotenhandle, der zuvor vom Funktionsbaustein UA_NodeGetHandle [► 246] ausgegeben wurde.
stNodeAddInfo	ST_UANodeAdditionalInfo	Definiert zusätzliche Informationen, z. B. auf welchen IndexRange oder welches Attribut geschrieben werden soll (standardmäßig wird das 'Value'-Attribut verwendet). Wird durch STRUCT ST_UANodeAdditionalInfo [► 230] spezifiziert.
pVariable	PVOID	Zeiger auf zu schreibende Daten.
cbData	UDINT	Legt die Größe der zu schreibenden Werte fest.
Timeout	TIME	Zeit bis zum Abbruch der Funktion. <code>DEFAULT_ADS_TIMEOUT</code> ist eine globale Konstante, gesetzt auf 5 Sekunden.

Ausgänge

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

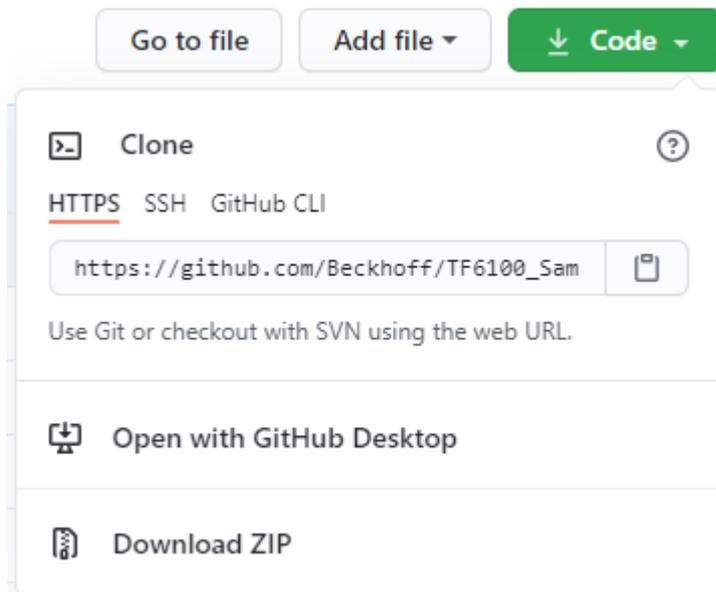
Name	Typ	Beschreibung
Done	BOOL	Schaltet auf TRUE, wenn der Funktionsbaustein erfolgreich ausgeführt wurde.
Busy	BOOL	TRUE, bis der Baustein einen Befehl ausführt hat, maximal für die Dauer des „Timeout“ am Eingang. Solange Busy = TRUE ist, akzeptieren die Eingänge keinen neuen Befehl. Es wird nicht die Zeit der Verbindung sondern die Empfangszeit überwacht.
Error	BOOL	Schaltet auf TRUE, wenn bei der Ausführung eines Befehls ein Fehler auftritt. Der befehlspezifische Fehlercode ist in ErrorID enthalten.
ErrorID	DWORD	Enthält den befehlspezifischen ADS-Fehlercode des zuletzt ausgeführten Befehls.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCOpen_OpcUa

6 Beispiele

Beispielcode und -konfigurationen für dieses Produkt können über das entsprechende Repository auf GitHub bezogen werden: https://github.com/Beckhoff/TF6100_Samples . Sie haben dort die Möglichkeit das Repository zu klonen oder ein ZIP File mit dem Sample herunterzuladen.



Es existieren folgende Samples:

Name	TwinCAT-Versi- on	Beschreibung
TF6100_OpcUa_Client_Sample	TwinCAT 3	Dieses Sample beinhaltet Beispielcode für verschiedene Funktionen des TwinCAT OPC UA Clients (PLCOpen-Funktionsbausteine). Dazu gehören Browse, Connect, HistoryUpdate, MethodCall, Read und Write. Es ist zusätzlich das Server-Sample für den Zugriff enthalten.
TF6100_OpcUa_Server_Sample	TwinCAT 3	Dieses Sample beinhaltet eine SPS mit umfangreicher Bereitstellung von SPS-Daten für den TwinCAT OPC UA Server (OPC UA Data Access).
TS6100_OpcUa_Client_Sample	TwinCAT 2	Dieses Sample beinhaltet Beispielcode für verschiedene Funktionen des TwinCAT OPC UA Clients (PLCOpen-Funktionsbausteine). Dazu gehören MethodCall, Read und Write.

7 Anhang

7.1 Fehlerdiagnose

Im Folgenden werden für alle Komponenten des OPC UA-Setups mögliche Fehler in Form einer Tabelle dargestellt. Zusätzlich werden zu den jeweiligen Fehlern hilfreiche Hinweise gegeben, aus welchen Gründen diese Fehler auftreten und wie sie behoben werden können.

7.1.1 Server

Verhalten	Hinweise
Ein OPC UA Client sieht den PLC Namespace nicht	TF6100 Setup Version 3.x und kleiner: Dieser Status ist ein Hinweis auf eine fehlende Lizenz. Überprüfen Sie, ob Sie eine gültige TF6100 Lizenz aktiviert haben.
Ein OPC UA Client bekommt bei Auslesen von Nodes den StatusCode 0x810e0000	TF6100 Setup Version 4.x: Dieser Status ist ein Hinweis auf eine fehlende Lizenz. Überprüfen Sie, ob Sie eine gültige TF6100 Lizenz aktiviert haben.
Die über Kommentare/Attribute freigegebenen Variablen werden nicht im OPC UA Server angezeigt	Überprüfen Sie, ob die Symboldatei ordnungsgemäß auf die Steuerung übertragen wurde (z. B. Auswahlkästchen im SPS-Projekt), im Bootverzeichnis vorhanden ist und der Pfad zur Symboldatei in der Konfigurationsdatei des Servers auf die richtige Symboldatei verweist. Sie können auch über die DeviceState Node im jeweiligen Namespace eventuell aufgetretene ErrorMessageS überprüfen. Hier wird z. B. auch eingetragen, wenn die Symboldatei nicht gefunden wurde. Überprüfen Sie auch die ordnungsgemäße Schreibweise der Kommentare/Attribute.
Die vom OPC UA Client geforderte Samplingrate/PublishingInterval werden vom Server nicht eingehalten	OPC UA Client/Server ist kein Echtzeitprotokoll, d. h. dass es keine Garantie gibt, dass der Server eine vom Client geforderte Samplingrate oder PublishingInterval auch immer zu 100% einhält. Die zur Verfügung stehenden Samplingraten und PublishingIntervalle können in der Konfigurationsdatei des Servers eingesehen und bei Bedarf modifiziert werden (<AvailableSamplingRates> und <MinPublishingInterval>).
Ein OPC UA Client kann sich nicht mit dem Server verbinden, obwohl der Server im Windows Task Manager angezeigt wird. Es erscheint die Fehlermeldung „Host unreachable“ (o.ä.)	Überprüfen Sie, ob gegebenenfalls Firewall-Einstellungen eine Kommunikation mit dem Server verhindern. Der Serverport muss für eine eingehende TCP-Kommunikation geöffnet sein, damit sich ein Client verbinden kann.
Ein OPC UA Client sieht zwar die Endpunkte des Servers, eine Verbindung mit diesen schlägt jedoch mit der Fehlermeldung „Host unreachable“ fehl	Überprüfen Sie, ob die Namensauflösung in Ihrem Netzwerk ordnungsgemäß funktioniert und der Server unter seinem Hostnamen erreichbar ist. Auch wenn sich der OPC UA Client augenscheinlich mit der IP-Adresse des Servers verbindet (z. B. opc.tcp://192.168.0.1:4840), um auf die Endpunkte des Servers zuzugreifen, so returniert der Server in seinen Endpunkten dennoch immer den eigenen Hostnamen. Wenn sich nun der Client direkt mit einem der Endpunkte verbindet, so verwendet er wieder den Hostnamen des Servers. Im Falle einer nicht funktionierenden Namensauflösung schlägt dann die Verbindung fehl.
Ein OPC UA Client sieht zwar die Endpunkte des Servers, eine Verbindung mit einem sicheren Endpunkt schlägt jedoch fehl. Es erscheint die Fehlermeldung „BadSecurityChecksFailed“	Überprüfen Sie, ob der Server dem Client-Zertifikat vertraut. Die notwendigen Konfigurationsschritte können Sie im Abschnitt Zertifikatsaustausch [► 108] nachlesen. Hier ist zusätzlich darauf zu achten, dass ein Signaturhashalgorithmus der SHA 2-Gruppe (SHA256, SHA384, SHA512) zur Signierung des Client-Zertifikats verwendet wird. Werden veraltete Algorithmen wie SHA1 verwendet, lässt der TwinCAT OPC UA-Server keine Verbindung zu.
Bei Verwendung eines SQL Servers zur Speicherung von Historical Access Informationen, werden die Werte nicht zur SQL-Datenbank hinzugefügt	Überprüfen Sie die Zugangsdaten zum SQL Server und dass der SQL Server auch im Netzwerk erreichbar ist. Stellen Sie auch sicher, dass Sie ein „Big Windows“-Betriebssystem auf dem TwinCAT OPC UA Server verwenden, da SQL Server nicht unter Windows CE für Historical Access verwendet werden können (SQL Compact hingegen schon).

Verhalten	Hinweise
Beim Auslesen von Variablen erhält ein OPC UA Client die Fehlermeldung „BadDeviceFailure“	Dies ist ein Hinweis darauf, dass das zugehörige ADS-Gerät nicht erreichbar ist, z. B. wenn kein SPS-Programm gestartet ist. Überprüfen Sie die Konnektivität mit dem ADS-Gerät und stellen Sie sicher, dass die entsprechende Laufzeit aktiv ist.
Arrays werden nicht voll aufgelöst im Namespace dargestellt	Standardmäßig werden Arrays von einfachen Datentypen nicht „aufgeklappt“ im Namensraum dargestellt. Einzelne Array-Indizes sind dennoch über die sogenannte IndexRange Funktion von OPC UA adressierbar. Ein OPC UA Client sollte diese Funktion entsprechend unterstützen. Ist dies nicht der Fall, so kann über einen Optionsschalter in der Konfigurationsdatei des Servers bewirkt werden, dass ein Array „aufgeklappt“ dargestellt und somit jedes einzelne Arrayelement als separate Node adressierbar ist. Dieser Optionsschalter ist im Abschnitt Arrays [► 55] beschrieben.

7.1.2 Client I/O

Verhalten	Hinweise
Beim Erzeugen eines neuen I/O Clients durch Angabe der Server URL mit dem Hostnamen des Servers kann anschliessend keine Verbindung über den AddNodes Dialog aufgebaut werden.	Bitte überprüfen Sie, ob die Namensauflösung in Ihrem Netzwerk funktioniert und versuchen Sie es alternativ noch einmal über die IP-Adresse des Servers.
Einige Konfigurationselemente aus dem I/O Client sind nicht vorhanden, obwohl sie laut Dokumentation da sein sollten.	In diesem Fall wurde vermutlich nach einem TF6100 Update die System Manager Beschreibungsdatei (TMC) nicht aktualisiert. Bitte führen Sie den Befehl "Reload TMC" aus dem Kontextmenü des I/O Clients aus um die Beschreibungsdatei neu zu laden.
Schreibbefehle auf eine Variable werden nicht ausgeführt bzw. kommen nicht am Server an.	Bitte überprüfen Sie, ob der Ausgang "Write Enable" am I/O Client aktiviert wurde.

7.1.3 Client PLCopen

Verhalten	Hinweise
Der Versuch einen Structured Data Type von einem Server auszulesen oder zu schreiben schlägt fehl.	Structured Data Types werden vom PLCopen-basierten Client nicht unterstützt. Bitte verwenden Sie hierfür den I/O Client.

7.1.4 Gateway

Verhalten	Hinweise
Das Gateway kann sich nicht mit dem Server verbinden.	Eine der möglichen Ursachen ist, dass eine alte Konfiguration benutzt wird. Wenn es beispielsweise ein neues Server-Zertifikat gibt, merkt das Gateway dies erst, wenn der konfigurierte Endpoint gelöscht und unter anderem Namen wieder eingefügt wird. Bei gleichem Endpoint oder einem neuen Endpoint mit gleichem Namen würde das Gateway die Verbindungsinformationen aus einem Cache verwenden und infolgedessen keine Verbindung mehr zum Server herstellen können.

7.2 Statuscodes

7.2.1 ADS Return Codes

Gruppierung der Fehlercodes:

Globale Fehlercodes: 0x0000 [▶ 260]... (0x9811_0000 ...)

Router Fehlercodes: 0x0500 [▶ 260]... (0x9811_0500 ...)

Allgemeine ADS Fehler: 0x0700 [▶ 261]... (0x9811_0700 ...)

RTime Fehlercodes: 0x1000 [▶ 263]... (0x9811_1000 ...)

Globale Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x0	0	0x98110000	ERR_NOERROR	Kein Fehler.
0x1	1	0x98110001	ERR_INTERNAL	Interner Fehler.
0x2	2	0x98110002	ERR_NORTIME	Keine Echtzeit.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Zuweisung gesperrt - Speicherfehler.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Postfach voll – Es konnte die ADS Nachricht nicht versendet werden. Reduzieren der Anzahl der ADS Nachrichten pro Zyklus bringt Abhilfe.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Falsches HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Ziel-Port nicht gefunden – ADS Server ist nicht gestartet oder erreichbar.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Zielrechner nicht gefunden – AMS Route wurde nicht gefunden.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unbekannte Befehl-ID.
0x9	9	0x98110009	ERR_BADTASKID	Ungültige Task-ID.
0xA	10	0x9811000A	ERR_NOIO	Kein IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unbekannter AMS-Befehl.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 Fehler.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port nicht verbunden.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Ungültige AMS-Länge.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Ungültige AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installations-Level ist zu niedrig –TwinCAT 2 Lizenzfehler.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	Kein Debugging verfügbar.
0x12	18	0x98110012	ERR_PORTDISABLED	Port deaktiviert – TwinCAT System Service nicht gestartet.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port bereits verbunden.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 Fehler.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync Fehler.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	Keine Index-Map für AMS Sync vorhanden.
0x18	24	0x98110018	ERR_INVALIDAMSSPORT	Ungültiger AMS-Port.
0x19	25	0x98110019	ERR_NOMEMORY	Kein Speicher.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP Sendefehler.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host nicht erreichbar.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Ungültiges AMS Fragment.
0x1D	29	0x9811001D	ERR_TLSSSEND	TLS Sendefehler – Secure ADS Verbindung fehlgeschlagen.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Zugriff Verweigert – Secure ADS Zugriff verweigert.

Router Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Lockierter Speicher kann nicht zugewiesen werden.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	Die Größe des Routerspeichers konnte nicht geändert werden.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	Das Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	Das Debug Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	Der Porttyp ist unbekannt.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	Router ist nicht initialisiert.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	Die Portnummer ist bereits vergeben.
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	Der Port ist nicht registriert.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	Die maximale Portanzahl ist erreicht.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	Der Port ist ungültig.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	Der Router ist nicht aktiv.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	Das Postfach hat die maximale Anzahl für fragmentierte Nachrichten erreicht.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	Fragment Timeout aufgetreten.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	Port wird entfernt.

Allgemeine ADS Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	Allgemeiner Gerätefehler.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service wird vom Server nicht unterstützt.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Ungültige Index-Gruppe.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Ungültiger Index-Offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Lesen oder Schreiben nicht gestattet.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parametergröße nicht korrekt.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Ungültige Daten-Werte.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Gerät nicht betriebsbereit.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Gerät beschäftigt.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Ungültiger Kontext vom Betriebssystem - Kann durch Verwendung von ADS Bausteinen in unterschiedlichen Tasks auftreten. Abhilfe kann die Multitasking-Synchronisation in der SPS geben.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Nicht genügend Speicher.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARG	Ungültige Parameter-Werte.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Nicht gefunden (Dateien,...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax-Fehler in Datei oder Befehl.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objekte stimmen nicht überein.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Objekt ist bereits vorhanden.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol nicht gefunden.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Symbol-Version ungültig – Kann durch einen Online-Change auftreten. Erzeuge einen neuen Handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Gerät (Server) ist im ungültigen Zustand.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode nicht unterstützt.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHANDINVALID	Notification Handle ist ungültig.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification-Client nicht registriert.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDL	Keine weiteren Handles verfügbar.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Größe der Notification zu groß.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Gerät nicht initialisiert.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Gerät hat einen Timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface Abfrage fehlgeschlagen.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Falsches Interface angefordert.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class-ID ist ungültig.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object-ID ist ungültig.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Anforderung steht aus.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Anforderung wird abgebrochen.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal-Warnung.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Ungültiger Array-Index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol nicht aktiv.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Zugriff verweigert.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Fehlende Lizenz.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	Lizenz abgelaufen.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	Lizenz überschritten.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Lizenz ungültig.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	Lizenzproblem: System-ID ist ungültig.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	Lizenz nicht zeitlich begrenzt.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Lizenzproblem: Zeitpunkt in der Zukunft.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	Lizenz-Zeitraum zu lang.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception beim Systemstart.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	Lizenz-Datei zweimal gelesen.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Ungültige Signatur.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Zertifikat ungültig.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public Key vom OEM nicht bekannt.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	Lizenz nicht gültig für diese System.ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo-Lizenz untersagt.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Funktions-ID ungültig.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Außerhalb des gültigen Bereiches.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Ungültiges Alignment.

Hex	Dec	HRESULT	Name	Beschreibung
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Ungültiger Plattform Level.
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Kontext – Weiterleitung zum Passiv-Level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Kontext – Weiterleitung zum Dispatch-Level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Kontext – Weiterleitung zur Echtzeit.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Clientfehler.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Dienst enthält einen ungültigen Parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling-Liste ist leer.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var-Verbindung bereits im Einsatz.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	Die aufgerufene ID ist bereits in Benutzung.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNC TIMEOUT	Timeout ist aufgetreten – Die Gegenstelle antwortet nicht im vorgegebenen ADS Timeout. Die Routeneinstellung der Gegenstelle kann falsch konfiguriert sein.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Fehler im Win32 Subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Ungültiger Client Timeout-Wert.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port nicht geöffnet.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	Keine AMS Adresse.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Interner Fehler in Ads-Sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Überlauf der Hash-Tabelle.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Schlüssel in der Tabelle nicht gefunden.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESVM	Keine Symbole im Cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Ungültige Antwort erhalten.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port ist verriegelt.
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	Die Anfrage wurde abgebrochen.

RTime Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x1000	4096	0x98111000	RTERR_INTERNAL	Interner Fehler im Echtzeit-System.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer-Wert nicht gültig.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task-Pointer hat den ungültigen Wert 0 (null).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack-Pointer hat den ungültigen Wert 0 (null).
0x1004	4100	0x98111004	RTERR_PrioEXISTS	Die Request Task Priority ist bereits vergeben.
0x1005	4101	0x98111005	RTERR_NOMORETCB	Kein freier TCB (Task Control Block) verfügbar. Maximale Anzahl von TCBs beträgt 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	Keine freien Semaphoren zur Verfügung. Maximale Anzahl der Semaphoren beträgt 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	Kein freier Platz in der Warteschlange zur Verfügung. Maximale Anzahl der Plätze in der Warteschlange beträgt 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	Ein externer Synchronisations-Interrupt wird bereits angewandt.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	Kein externer Sync-Interrupt angewandt.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Anwendung des externen Synchronisierungs-Interrupts ist fehlgeschlagen.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Aufruf einer Service-Funktion im falschen Kontext
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel VT-x Erweiterung wird nicht unterstützt.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x Erweiterung ist nicht aktiviert im BIOS.
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Fehlende Funktion in Intel VT-x Erweiterung.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Aktivieren von Intel VT-x schlägt fehl.

Spezifische positive HRESULT Return Codes:

HRESULT	Name	Beschreibung
0x0000_0000	S_OK	Kein Fehler.
0x0000_0001	S_FALSE	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch ein negatives oder unvollständiges Ergebnis erzielt wurde.
0x0000_0203	S_PENDING	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch noch kein Ergebnis vorliegt.
0x0000_0256	S_WATCHDOG_TIMEOUT	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch eine Zeitüberschreitung eintrat.

TCP Winsock-Fehlercodes

Hex	Dec	Name	Beschreibung
0x274C	10060	WSAETIMEDOUT	Verbindungs Timeout aufgetreten - Fehler beim Herstellen der Verbindung, da die Gegenstelle nach einer bestimmten Zeitspanne nicht ordnungsgemäß reagiert hat, oder die hergestellte Verbindung konnte nicht aufrecht erhalten werden, da der verbundene Host nicht reagiert hat.
0x274D	10061	WSAECONNREFUSED	Verbindung abgelehnt - Es konnte keine Verbindung hergestellt werden, da der Zielcomputer dies explizit abgelehnt hat. Dieser Fehler resultiert normalerweise aus dem Versuch, eine Verbindung mit einem Dienst herzustellen, der auf dem fremden Host inaktiv ist—das heißt, einem Dienst, für den keine Serveranwendung ausgeführt wird.
0x2751	10065	WSAEHOSTUNREACH	Keine Route zum Host - Ein Socketvorgang bezog sich auf einen nicht verfügbaren Host.

Weitere Winsock-Fehlercodes: Win32-Fehlercodes

7.2.2 Client I/O

Die zu einem virtuellen OPC UA-Gerät gehörenden OPC UA Client-Module bieten verschiedene Statusvariablen sowie Kontrollvariablen an. Nachfolgend findet sich die Erläuterung dieser Variablen.

Lesen der Statuscodes

i Bitte beachten Sie, dass der Statuscode der State Machine hier in hexadezimaler Schreibweise aufgeführt ist. Sollte der Code in TwinCAT als Dezimalzahl angezeigt werden, muss er für die Interpretation konvertiert werden.

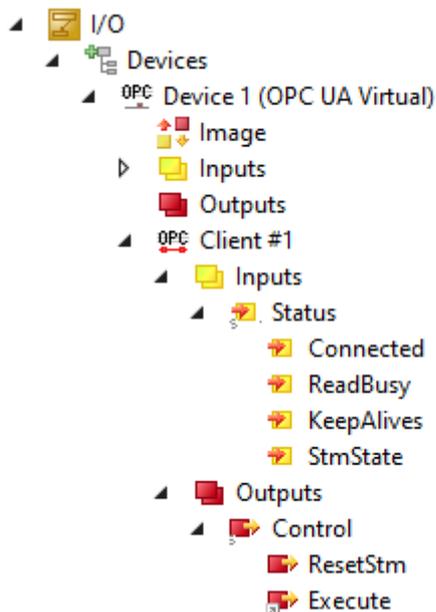


Abb. 1: OPCUAClientModulesStatusCodes

Variable	Schema	0	1- State Machine-Status (state machinestate)	2- Keep alive-Zähler beim Verwenden von Subscriptions (keep alive count if using subscriptions)	3- Verbindungsstatus (&Read Busy) (connection state(&read busy))
➡ Status	0x0123	-	0 = Initialisieren (init)		0 = false(&off)
			1 = Verbinden (connect)		1 = true(&off)
			2 = Namespaces auflösen (resolve namespace)		2 = false(&on)
			3 = Node Handles abfragen (get node handles)		3 = true(&on)
			4 = Zyklisches Lesen/Schreiben (continuous read/write)		
			5 = Lesen/Schreiben über Trigger-Variablen (triggered read/write)		
			6 = Warten auf Benachrichtigungen bezgl. Datenänderungen (awaiting data change notifications (subscriptions))		
			7 = Verbindung trennen (disconnect)		
			8 = Zurücksetzen (reset)		
➡ Control	0x0123	-	-	-	0 = Standard (default) 1 = State Machine zurücksetzen (reset state machine) 2 = Ausführen (beim Lesen/Schreiben mit Trigger-Variablen) (execute (in triggered read mode))
Variable	Datentyp		Beschreibung		
➡ Connected	BIT		1 TRUE 0= FALSE.		
➡ ReadBusy	BIT		1 TRUE 0= FALSE. Diese Funktion ist nur beim Lesen und Schreiben über Trigger-Variablen aktiv.		
➡ KeepAlives	BIT4		Zeigt die Anzahl an gezählten KeepAlive-Nachrichten. Ist nur beim Lesen und Schreiben mithilfe von Subscriptions aktiv.		
➡ StmState	BYTE		Abzulesen in der obigen Tabelle.		
➡ ResetStm	BIT		Der Client wird zurückgesetzt, wenn dieses Bit auf 1 gesetzt wird.		

Variable	Schema	0	1- State Machine-Status (state machinestate)	2- Keep alive- Zähler beim Ver- wenden von Sub- scriptions (keep alive count if using subs- criptions)	3- Verbindungssta- tus (&Read Busy) (connection state(&read busy))
 Execute		BIT		1 TRUE 0= FALSE. Wenn dieses Bit beim Lesen und Schreiben über Trigger-Variablen auf 1 gesetzt wird, wird gelesen/geschrieben. Wenn dieses Bit gesetzt bleibt, besteht kein Unterschied zum zyklischen Lesen/Schreiben.	

7.2.3 Client PLCopen

Die Funktionsbausteine des TwinCAT OPC UA Client besitzen eigene Fehlercodes, die das Auftreten eines Fehlers signalisieren und mit einer ErrorID weitere Informationen zum aufgetretenen Problem anzeigen. Es können sowohl TwinCAT-ADS-Fehlermeldungen ([ADS Return Codes \[▶ 260\]](#)) mit dem Highword 0x0000 als auch eigene Fehlermeldungen aus dem Client oder der SPS-Bibliothek mit dem Highword 0xE4DD auftreten.

Verwendete TwinCAT-ADS-Fehler sind beispielsweise folgende:

Hex	Name	Beschreibung
0x 0000 0705	DEVICE_INVALIDSIZE	Parametergröße nicht korrekt
0x 0000 0706	DEVICE_INVALIDDATA	Ungültige Parameterwerte
0x 0000 070A	DEVICE_NOMEMORY	Nicht genügend Speicher

Diese Fehlercodeliste führt die möglichen eigenen Fehlerwerte auf:

Hex	Name	Beschreibung
0x E4DD 0001	UAC_E_FAIL	Aufruf von UA Service fehlgeschlagen
0x E4DD 0100	UAC_E_CONNECTED	Server bereits verbunden
0x E4DD 0101	UAC_E_CONNECT	Allgemeiner Fehler beim Aufbau einer Verbindung
0x E4DD 0102	UAC_E_UASECURITY	UA Security konnte nicht eingerichtet werden
0x E4DD 0103	UAC_E_ITEMEXISTS	Element ID bereits vorhanden
0x E4DD 0104	UAC_E_ITEMNOTFOUND	Element existiert nicht
0x E4DD 0105	UAC_E_ITEMTYPE	Ungültiger oder nicht unterstützter Elementtyp
0x E4DD 0106	UAC_E_CONVERSION	Variablentypen können nicht konvertiert werden
0x E4DD 0107	UAC_E_SUSPENDED	Gerät hängt. Bitte später erneut versuchen...
0x E4DD 0108	UAC_E_TYPE_NOT_SUPPORTED	Konvertierung Variablentyp wird nicht unterstützt.
0x E4DD 0109	UAC_E_NSNAME_NOTFOUND	Kein Namensraum mit dem angegebenen Namen gefunden.
0x E4DD 0110	UAC_E_CONNECT_NOTFOUND	Verbindung fehlgeschlagen: Ziel-Host konnte nicht gefunden werden.
0x E4DD 0111	UAC_E_TIMEOUT	Timeout: d. h. Ziel-Host antwortet nicht
0x E4DD 0112	UAC_E_INVALIDHDL	Sitzungshandle ungültig
0x E4DD 0113	UAC_E_INVALIDNODEID	UA-Knoten-ID unbekannt
0x E4DD 0114	UAC_E_INVAL_IDENTIFIER_TYPE	Bezeichnertyp der UaNodeld ungültig
0x E4DD 0115	UAC_E_IDENTIFIER_NOTSUPP	Bezeichnertyp UaNodeld wird nicht unterstützt
0x E4DD 0116	UAC_E_INVAL_NODE_HDL	Ungültiges Knotenhandle
0x E4DD 0117	UAC_E_UAREADFAILED	UA Read aus unbekannter Ursache fehlgeschlagen
0x E4DD 0118	UAC_E_UAWRITEFAILED	UA Write aus unbekannter Ursache fehlgeschlagen
0x E4DD 0119	UAC_E_INVAL_NODEMETHOD_HDL	Ungültiges Methodenhandle
0x E4DD 011A	UAC_E_CALL_FAILED	Aufruf fehlgeschlagen, Ursache unbekannt
0x E4DD 011B	UAC_E_CALLDECODE_FAILED	Aufruf erfolgreich, Decodierung Rückgabewert fehlgeschlagen
0x E4DD 011C	UAC_E_NOTMAPPEDTYPE	Nicht zugeordneter Datentyp in Rückgabewert
0x E4DD 011D	UAC_E_CALL_FAILED_BADINTERNAL	Aufruf fehlgeschlagen mit UA_BadInternal
0x E4DD 011E	UAC_E_METHODIDINVALID	Unbekannte MethodenID (bei Aufruf zurückgegeben, auch wenn von GetMethodHdl bereitgestellt)
0x E4DD 011F	UAC_E_TOOMUCHDIM	Methodenaufruf hat Parameter mit mehr als 3 Dimensionen zurückgegeben - wird nicht unterstützt.
0x E4DD 0120	UAC_E_CALL_FAILED_INVALIDARG	Aufruf fehlgeschlagen mit OpcUa_BadInvalidArgument
0x E4DD 0121	UAC_E_CALL_FAILED_TYPEMISMATCH	Aufruf fehlgeschlagen mit UAC_E_CALL_FAILED_TYPEMISMATCH
0x E4DD 0122	UAC_E_CALL_FAILED_OUTOFRANGE	Aufruf fehlgeschlagen mit UAC_E_CALL_FAILED_OUTOFRANGE
0x E4DD 0123	UAC_E_CALL_FAILED_BADSTRUCTURE	Aufruf fehlgeschlagen mit OpcUa_BadStructureMissing
0x E4DD 0124	UAC_E_CALL_TYPEMISMATCH_OUTPARAM	Aufruf erfolgreich, aber keine Typenübereinstimmung der bereitgestellten Ausgabeinformation
0x E4DD 0125	UAC_E_NONVALIDTYPEINFO	Knoten hat unzureichende Typinformationen
0x E4DD 0126	UAC_E_INVALIDATTRIBID	Zugriff auf ungültiges Attribut von Knoten

Hex	Name	Beschreibung
0x E4DD 0128	UAC_E_NOTSUPPORTED	Das Kommando wird vom verbundenen UaServer nicht unterstützt, z. B. beim Aufruf von UA_HistoryUpdate.
0x E4DE 0100	UAC_E_INVALID_ARRAY_LENGTH	Es wurde eine ungültige, nicht zu DataValueCount passende, Array-Länge am UA_HistoryUpdate zugewiesen.
0x E4DE 0101	UAC_E_INVALID_DATASIZE	Es wurde ein Datenwert mit ungültiger Datentypgröße am UA_HistoryUpdate zugewiesen. Alle zugewiesenen DataValues müssen vom gleichen Datentyp sein.
0x E4DE 0102	UAC_E_SUBERROR	Ein unterlagerter Fehler an mindestens einem der übertragenen Datenwerte wurde ausgegeben. Siehe ValueErrorIDs am UA_HistoryUpdate.

7.3 Support und Service

Beckhoff und seine weltweiten Partnerfirmen bieten einen umfassenden Support und Service, der eine schnelle und kompetente Unterstützung bei allen Fragen zu Beckhoff Produkten und Systemlösungen zur Verfügung stellt.

Downloadfinder

Unser [Downloadfinder](#) beinhaltet alle Dateien, die wir Ihnen zum Herunterladen anbieten. Sie finden dort Applikationsberichte, technische Dokumentationen, technische Zeichnungen, Konfigurationsdateien und vieles mehr.

Die Downloads sind in verschiedenen Formaten erhältlich.

Beckhoff Niederlassungen und Vertretungen

Wenden Sie sich bitte an Ihre Beckhoff Niederlassung oder Ihre Vertretung für den [lokalen Support und Service](#) zu Beckhoff Produkten!

Die Adressen der weltweiten Beckhoff Niederlassungen und Vertretungen entnehmen Sie bitte unserer Internetseite: www.beckhoff.com

Dort finden Sie auch weitere Dokumentationen zu Beckhoff Komponenten.

Beckhoff Support

Der Support bietet Ihnen einen umfangreichen technischen Support, der Sie nicht nur bei dem Einsatz einzelner Beckhoff Produkte, sondern auch bei weiteren umfassenden Dienstleistungen unterstützt:

- Support
- Planung, Programmierung und Inbetriebnahme komplexer Automatisierungssysteme
- umfangreiches Schulungsprogramm für Beckhoff Systemkomponenten

Hotline: +49 5246 963-157

E-Mail: support@beckhoff.com

Beckhoff Service

Das Beckhoff Service-Center unterstützt Sie rund um den After-Sales-Service:

- Vor-Ort-Service
- Reparaturservice
- Ersatzteilservice
- Hotline-Service

Hotline: +49 5246 963-460
E-Mail: service@beckhoff.com

Beckhoff Unternehmenszentrale

Beckhoff Automation GmbH & Co. KG

Hülshorstweg 20
33415 Verl
Deutschland

Telefon: +49 5246 963-0
E-Mail: info@beckhoff.com
Internet: www.beckhoff.com

Mehr Informationen:
www.beckhoff.de/ts6100

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

