BECKHOFF New Automation Technology

Handbuch | DE

TS6100

TwinCAT 2 | OPC UA Client

Supplement | Communication

03.04.2024 | Version: 1.0.0

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
3 Installation 10
3.1 Systemvoraussetzungen 10
3.2 Installation 10
3.3 Lizenzierung 12
4 Technische Einführung 15
4.1 Quick Start 15
4.2 Softwarearchitektur 18
4.3 Unterstützte Funktionen 19
4.4 Applikationsverzeichnisse 21
4.5 Lesen von Variablen 21
4.6 Schreiben von Variablen 23
4.7 Methodenaufrufe 25
4.8 Timestamp und StatusCode 28
4.9 Strukturen 28
4.10 Codegenerierung 31
4.11 PLCopen-Funktionsbausteine 35
5 SPS API 48
5.1 Tc2_OpcUa 48
5.1.1 Datentypen 48
5.1.2 Funktionsbausteine 49
5.2 Tc3_PLCopen_OpcUa 51
5.2.1 Datentypen 51
5.2.2 Funktionsbausteine 65
6 Beispiele 87
7 Anhang 88
7.1 Fehlerdiagnose 88
7.2 Statuscodes 88
7.2.1 ADS Return Codes 88
7.2.2 Client I/O 93
7.2.3 Client PLCopen 95
7.3 Support und Service 97

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

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

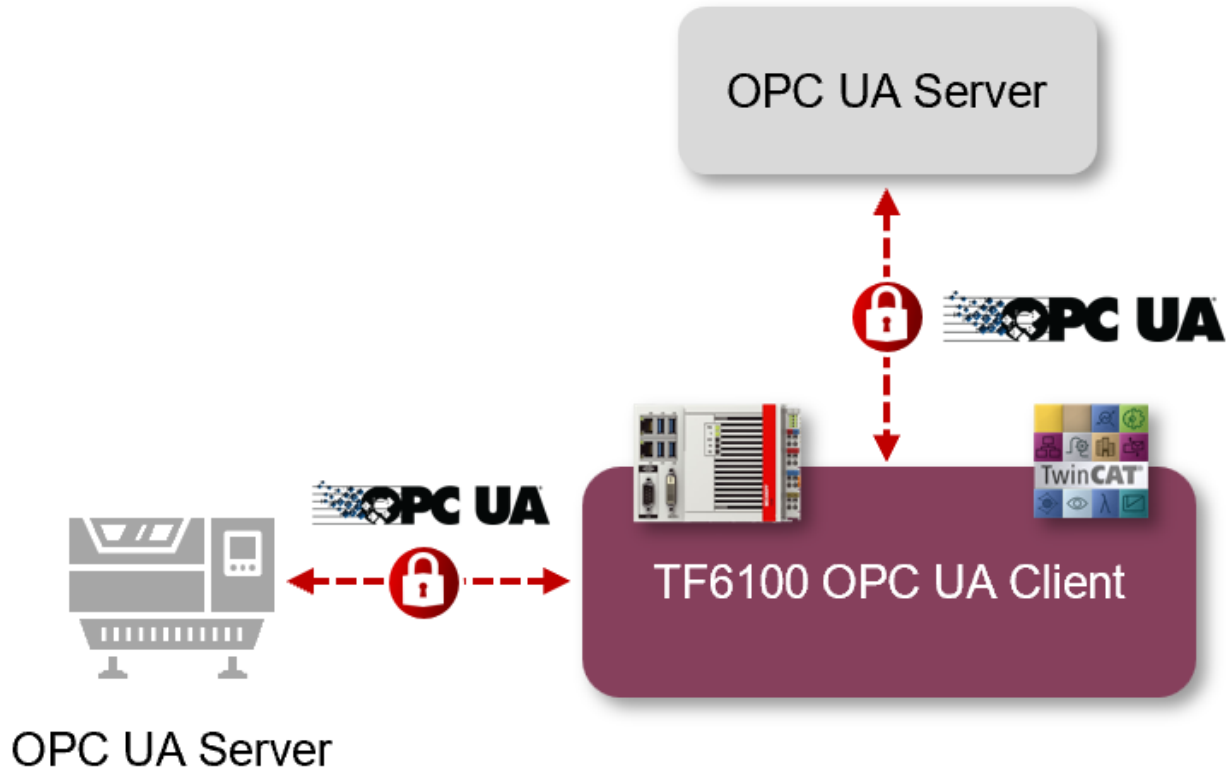
OPC Unified **A**rchitecture (OPC UA) ist die nächste Generation des klassischen OPC-Standards. Es handelt sich hierbei um ein weltweit standardisiertes Kommunikationsprotokoll, über das Maschinendaten hersteller- und plattformunabhängig ausgetauscht werden können. OPC UA integriert gängige Sicherheitsstandards 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 der Webseite der OPC Foundation.

Die TwinCAT 3 Function TF6100 OPC UA besteht aus mehreren Softwarekomponenten, welche einen Datenaustausch mit TwinCAT, basierend auf OPC UA, ermöglichen.
Die folgende Tabelle gibt einen Überblick über die einzelnen Produktbestandteile.

Software-Komponente	Beschreibung
TwinCAT OPC UA Server	Stellt eine OPC-UA-Server-Schnittstelle zur Verfügung, damit UA-Clients auf die TwinCAT-Laufzeit zugreifen können.
TwinCAT OPC UA Client	Stellt eine OPC-UA-Client-Funktionalität zur Verfügung, damit die Kommunikation mit anderen OPC UA Servern auf der Grundlage von PLCopen-genormten Funktionsbausteinen sowie einem einfach zu konfigurierenden I/O-Gerät möglich ist.
TwinCAT OPC UA Configurator	Grafische Benutzerschnittstelle für die Konfiguration des TwinCAT OPC UA Servers.
TwinCAT OPC UA Sample Client	Grafische Beispielimplementierung eines OPC UA Clients um einen ersten Verbindungstest mit dem TwinCAT OPC UA Server durchführen zu können.
TwinCAT OPC UA Gateway	Wrapper-Technologie, die sowohl eine OPC-COM-DA-Server-Schnittstelle als auch OPC-UA-Server-Aggregationsfähigkeiten zur Verfügung stellt.

Diese Dokumentation beschreibt den TwinCAT 3 OPC UA Client, bei welchem es sich um eine Softwarekomponente handelt, die eine OPC UA Client Schnittstelle für die TwinCAT-Runtime-Umgebung bereitstellt. Mit Hilfe des TwinCAT 3 OPC UA Clients können somit Verbindungen mit OPC UA Servern initiiert werden, um Daten mit diesen auszutauschen.



Die technischen Anwendungsfälle reichen hierbei von TCP-basierter (und demnach nicht echtzeitfähiger) Machine-to-Machine-Kommunikation bis hin zu Machine-to-Cloud-Kommunikation, wenn sich der zu verbindende OPC UA Server in der Cloud befinden sollte.

Der TwinCAT OPC UA Client steht technisch in zwei verschiedenen Varianten bereit:

1. Als ein TwinCAT-I/O-Gerät
2. Als PLC Funktionsbausteine

Weiterführende Informationen

- Für einen Überblick über eventuelle Funktionsunterschiede empfehlen wir unser Kapitel Unterstützte Funktionen [▶ 19].
- Bitte beachten Sie die Systemvoraussetzungen [▶ 10] zu diesem Produkt.
- Für einen schnellen Einstieg in das Produkt empfehlen wir unsere Kapitel Installation [▶ 10] und Quick Start [▶ 15].

3 Installation

3.1 Systemvoraussetzungen

Für die Installation und den Betrieb dieses Produkts gelten die folgenden Systemvoraussetzungen.

Client

Technische Daten	Beschreibung
Betriebssystem	Windows 10 Windows CE 6/7 Windows Server 2022
Zielplattformen	PC-Architektur (x86, x64, ARM)
.NET Framework	---
Minimale TwinCAT-Version	TwinCAT 3
Minimales TwinCAT-Installationslevel	TwinCAT 3 XAE, XAR
Benötigte TwinCAT-Lizenz	TF6100 TC3 OPC UA

Sample Server

Technische Daten	Beschreibung
Betriebssystem	Windows 10 (>= 21H2) Windows Server 2022
Zielplattformen	PC-Architektur (x86, x64)
.NET Framework	4.8.1
Minimale TwinCAT-Version	---
Minimales TwinCAT-Installationslevel	---
Benötigte TwinCAT-Lizenz	---

3.2 Installation

Die Installation dieser TwinCAT 3 Function kann, abhängig von der verwendeten TwinCAT-Version und dem Betriebssystem, auf unterschiedliche Arten erfolgen, welche im Folgenden näher beschrieben werden sollen.

HINWEIS

Updateinstallation

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

TwinCAT Package Manager

Wenn Sie TwinCAT 3.1 Build 4026 (und höher) auf dem Betriebssystem Microsoft Windows verwenden, können Sie diese Function über den TwinCAT Package Manager installieren, siehe Dokumentation zur Installation.

Normalerweise installieren Sie die Function über den entsprechenden Workload; dennoch können Sie die im Workload enthaltenen Pakete auch einzeln installieren. Diese Dokumentation beschreibt im Folgenden kurz den Installationsvorgang über den Workload.

Kommandozeilenprogramm TcPkg

Über das TcPkg **C**ommand **L**ine **I**nterface (CLI) können Sie sich die verfügbaren Workloads auf dem System anzeigen lassen:

```
tcpkg list -t workload
```

Über das folgende Kommando können Sie den Workload einer Function installieren.
Hier exemplarisch dargestellt am Beispiel des TF6100 TwinCAT OPC UA Client:

```
tcpkg install tf6100-opc-ua-client
```

TwinCAT Package Manager UI

Über das **U**ser **I**nterface (UI) können Sie sich alle verfügbaren Workloads anzeigen lassen und diese bei Bedarf installieren.
Folgen Sie hierzu den entsprechenden Anweisungen in der Oberfläche.

HINWEIS

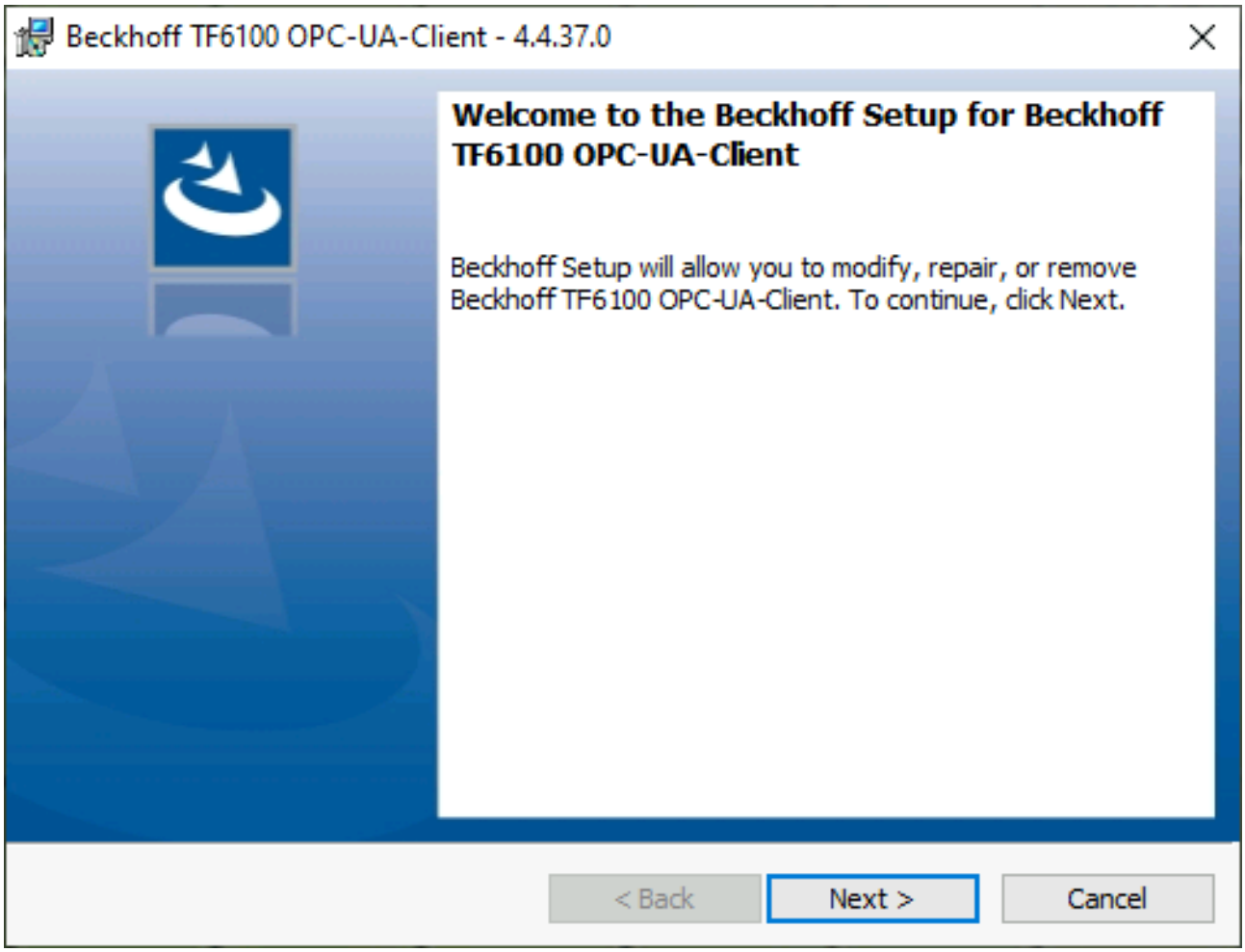
Unvorbereiteter TwinCAT-Neustart kann Datenverlust erzeugen

Die Installation dieser Function hat unter Umständen einen TwinCAT-Neustart zur Folge.
Stellen Sie sicher, dass keine kritischen TwinCAT-Applikationen auf dem System laufen oder fahren Sie diese zunächst geordnet herunter.

Setup

Wenn Sie TwinCAT 3.1 Build 4024 auf dem Betriebssystem Microsoft Windows verwenden, können Sie diese Function über ein Setup-Paket installieren, welches Sie auf der Beckhoff Webseite unter https://www.beckhoff.com/download herunterladen können.

Die Installation kann hierbei sowohl auf Engineering- als auch Runtime-Seite erfolgen, je nachdem, auf welchem System Sie die Function benötigen. Der folgende Screenshot zeigt exemplarisch die Setup-Oberfläche am Beispiel des TF6100 TwinCAT OPC UA Client-Setups.



Zur Durchführung des Installationsvorgangs, folgen Sie den entsprechenden Anweisungen im Setup-Dialog.

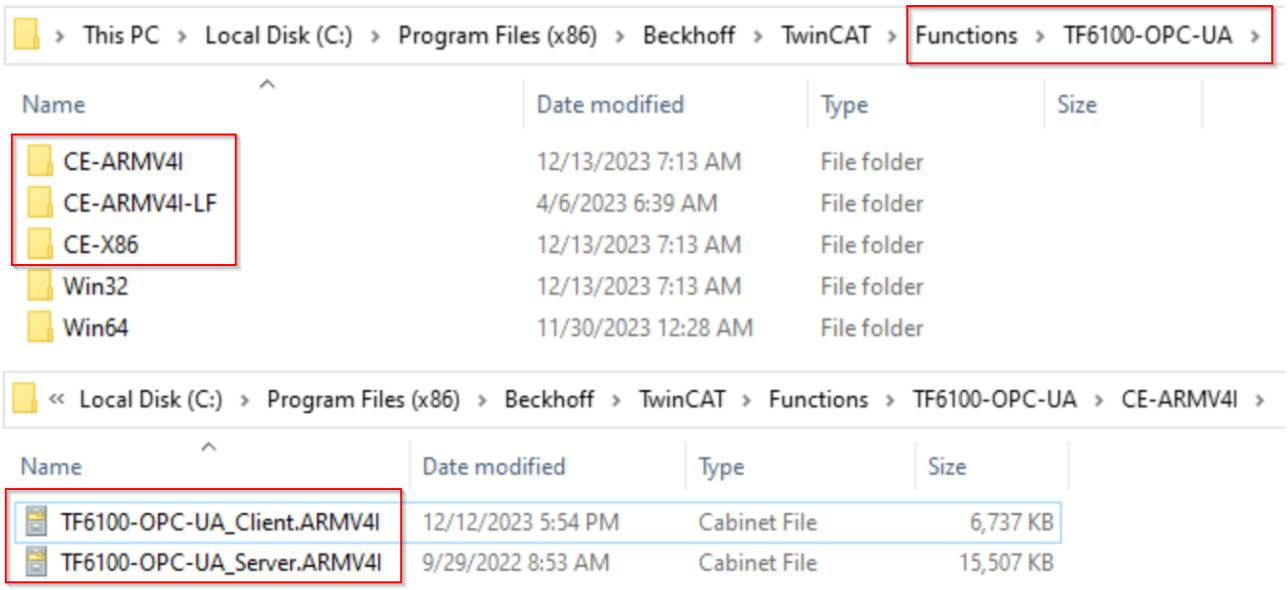
HINWEIS

Unvorbereiteter TwinCAT-Neustart kann Datenverlust erzeugen

Die Installation dieser Function hat unter Umständen einen TwinCAT-Neustart zur Folge.
Stellen Sie sicher, dass keine kritischen TwinCAT-Applikationen auf dem System laufen oder fahren Sie diese zunächst geordnet herunter.

Windows CE

Wenn Sie als Betriebssystem Microsoft Windows CE verwenden, können Sie diese Function über die jeweiligen CAB-Dateien installieren, welche mit dem Setup bzw. TcPkg Workload ausgeliefert werden. Die CAB-Dateien werden üblicherweise in dem Unterverzeichnis CE-ARMC4I und CE-X86 relativ vom Installationsverzeichnis der Function abgelegt.



Von dort können Sie per Dateitransfer auf das Windows CE-Gerät übertragen und dort ausgeführt werden. Die CAB-Dateien installieren und registrieren dann die Function auf dem jeweiligen System.

Verwenden Sie jeweils die CAB-Datei passend zu Ihrem System. Konkret bedeutet dies:

- CE-ARMV4I: ARM-basierte Geräte, z. B. CX8190, CX9020
- CE-X86: x86-basierte Geräte, z. B. CX51xx, CX52xx, CX20xx

Der Dateitransfer der CAB-Datei auf das Gerät kann entweder über die CF/SD-Karte oder den im Windows CE integrierten FTP-Server erfolgen.

Geräteneustart

Nach der Installation dieser Function ist ein Geräteneustart erforderlich, damit die Function verwendet werden kann.

3.3 Lizenzierung

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:

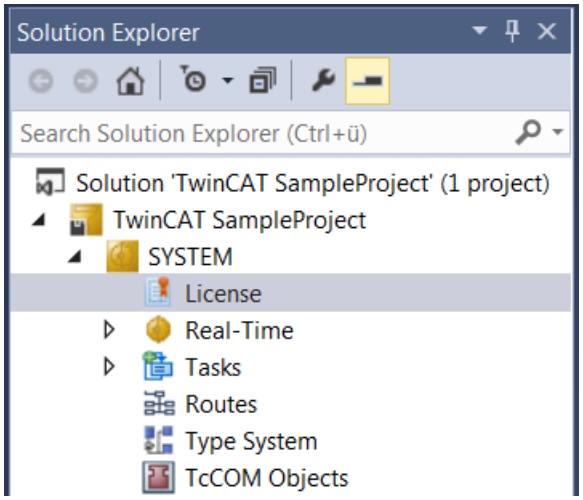
- Lizenzierung einer 7-Tage Testversion [▶ 13]
- Lizenzierung einer Vollversion [▶ 14]

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

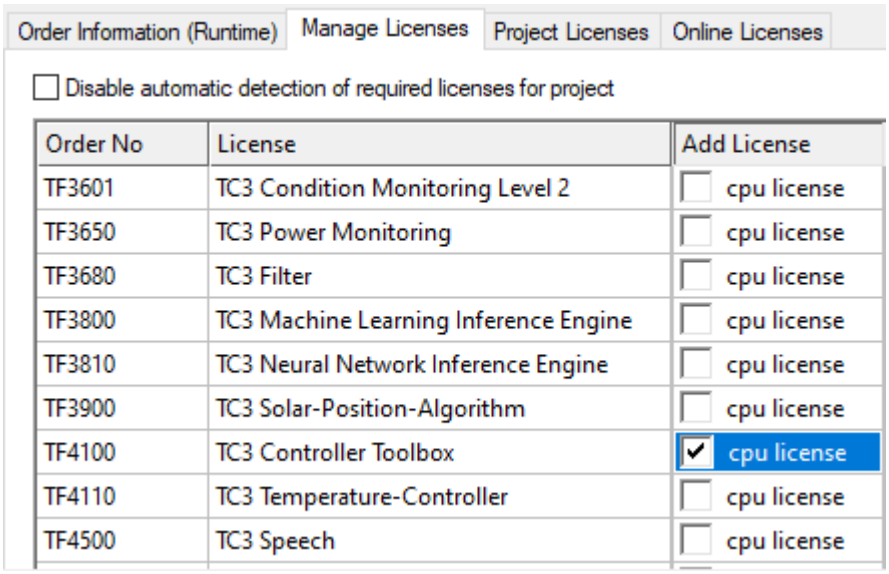
Lizenzierung der 7-Tage-Testversion einer TwinCAT 3 Function

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.**



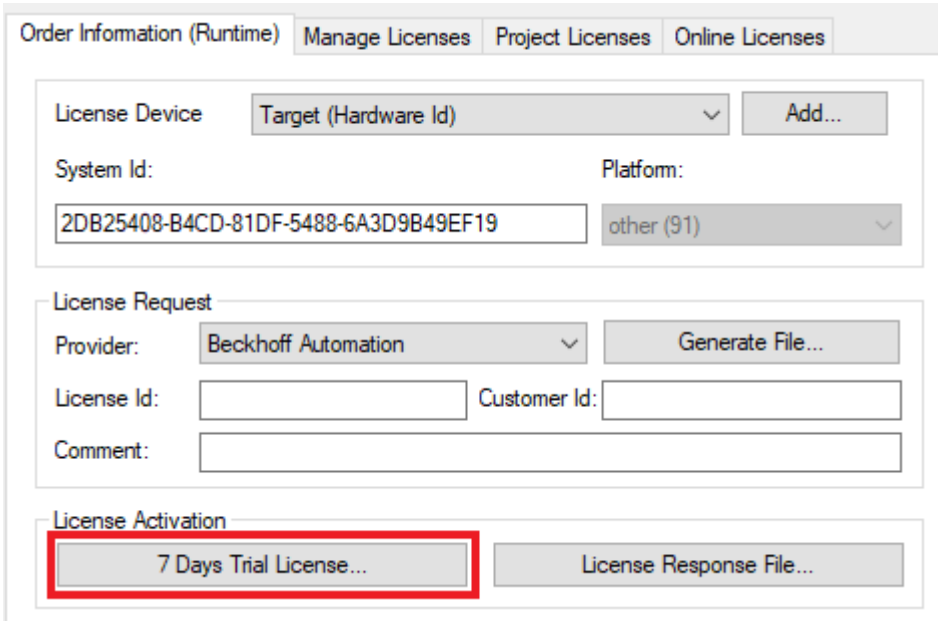
   ⇨ Der TwinCAT-3-Lizenzmanager öffnet sich.
5. Öffnen Sie die Registerkarte **Manage Licenses.** Aktivieren Sie in der Spalte **Add License** das Auswahlkästchen für die Lizenz, die Sie Ihrem Projekt hinzufügen möchten (z. B. „TF4100 TC3 Controller Toolbox“).



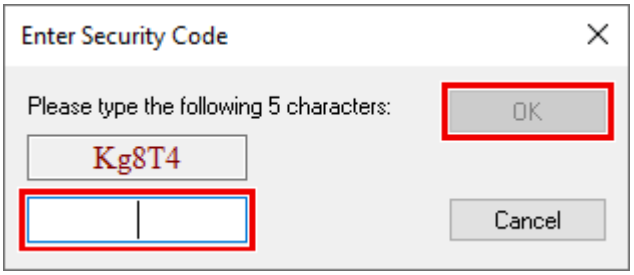
Order No	License	Add License
TF3601	TC3 Condition Monitoring Level 2	cpu license
TF3650	TC3 Power Monitoring	cpu license
TF3680	TC3 Filter	cpu license
TF3800	TC3 Machine Learning Inference Engine	cpu license
TF3810	TC3 Neural Network Inference Engine	cpu license
TF3900	TC3 Solar-Position-Algorithm	cpu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

6. Öffnen Sie die Registerkarte **Order Information (Runtime).**
   ⇨ In der tabellarischen Übersicht der Lizenzen wird die zuvor ausgewählte Lizenz mit dem Status „missing“ angezeigt.

7. 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.



8. Geben Sie den Code genauso ein, wie er angezeigt wird, und bestätigen Sie ihn.
9. 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“.

4 Technische Einführung

4.1 Quick Start

Das folgende Kapitel ermöglicht einen Schnelleinstieg in den TwinCAT OPC UA I/O Client. In dieser Anleitung wird die Verbindung zu einem Sample OPC UA Server eingerichtet, welcher einige Variablen in seinem Namensraum anbietet. Diese Variablen werden zum Prozessabbild des TwinCAT OPC UA I/O Clients hinzugefügt und anschließend mit SPS-Variablen verknüpft.

Sample OPC UA Server

Der Sample OPC UA Server wird zusammen mit dem TF6100 OPC UA Client Setup ausgeliefert und befindet sich in dem Installationsverzeichnis [▶ 21] des Clients. Alternativ können Sie anstelle des Sample OPC UA Servers auch den TwinCAT OPC UA Server verwenden, um ein paar Variablen über OPC UA bereitzustellen.

Im Folgenden werden die Handlungsschritte ihrer Reihenfolge nach genauer beschrieben:

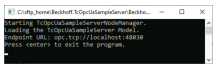
- Starten des Sample OPC UA Servers
- Erstellen eines TwinCAT-Projekts
- Auslesen der OPC UA Variablen
- Starten der Codegenerierung

Starten des Sample OPC UA Servers

1. Navigieren Sie im Windows Explorer zum Installationsverzeichnis der TF6100 FunctionSample und anschließend in das Unterverzeichnis „SampleServer“.
2. Starten Sie dort die Datei *TcOpcUaSampleServer.exe* als Administrator.

⇨ Der Server wird in einem Konsolenfenster gestartet und ist anschließend unter der folgenden OPC UA URL erreichbar:

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



Die Meldung bzgl. der eingeschränkten Laufzeit können Sie bestätigen.

Erstellen eines TwinCAT-Projekts

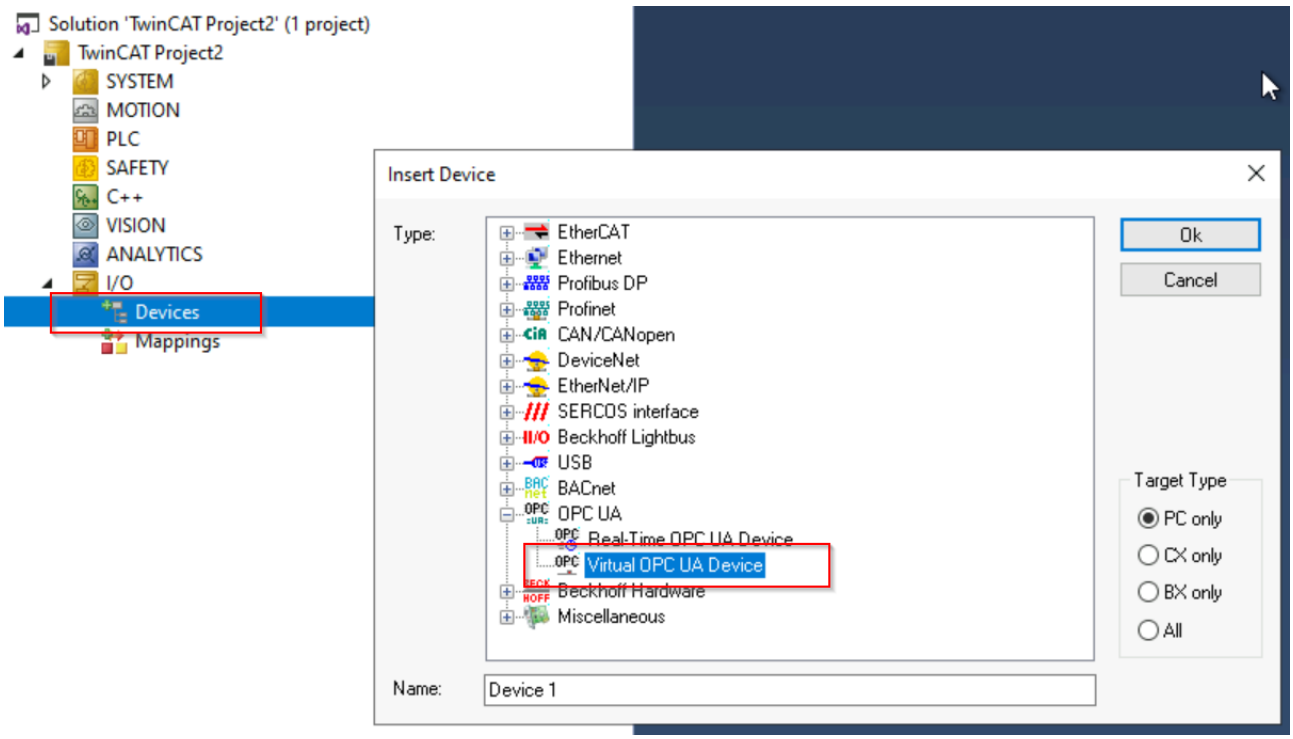
1. Öffnen Sie die TwinCAT XAE Shell.
2. Wählen Sie im Menü **Datei** den Befehl **Neu > Projekt**.
3. Fügen Sie dem Projekt ein SPS-Projekt hinzu.

⇨ Ein neues TwinCAT-Projekt inklusive SPS-Projekt wurde erstellt.

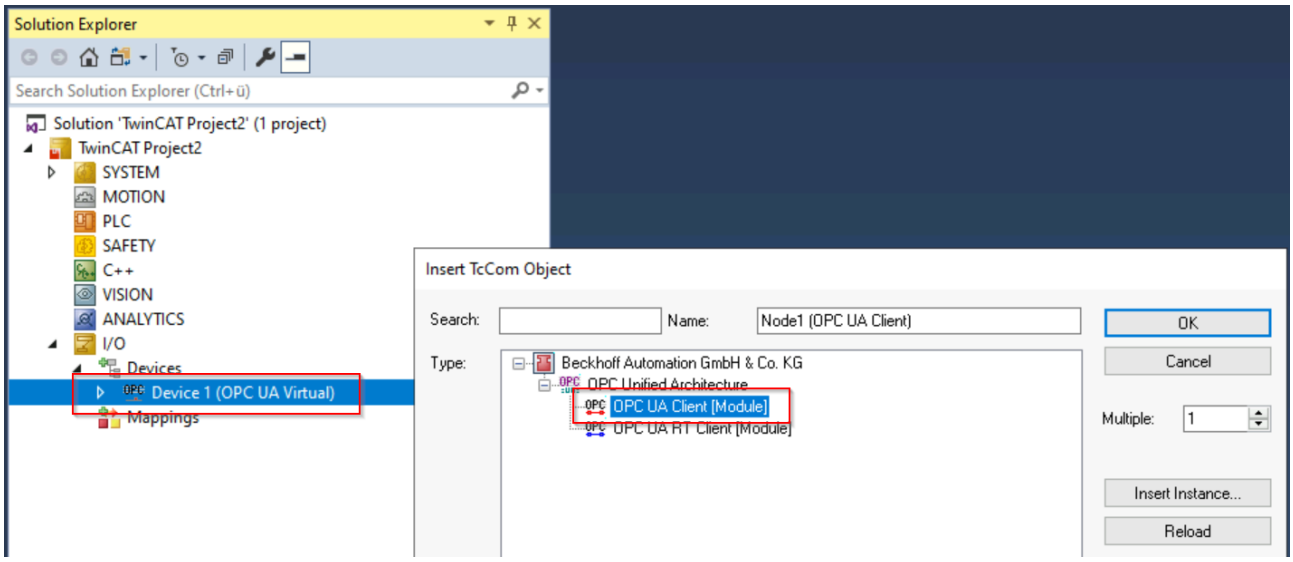
Auslesen der OPC UA Variablen

✓ In diesem Schritt wird der TwinCAT OPC UA Client verwendet, um eine Verbindung zum Server herzustellen und die dort vorhandenen Variablen einzulesen.

1. Fügen Sie dem TwinCAT-Projekt ein neues I/O-Gerät hinzu.
   Als Gerätetyp verwenden Sie „OPC UA Virtual Device“

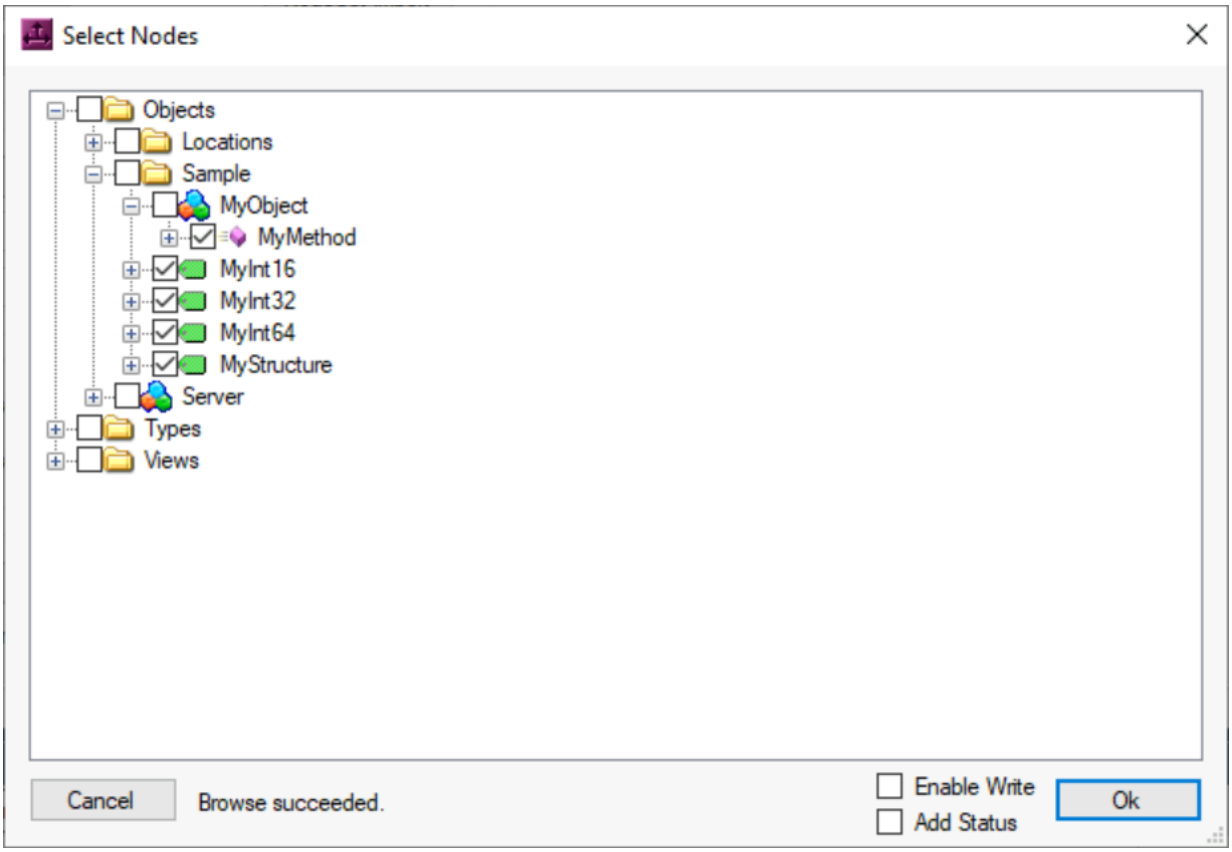


2. Fügen Sie dem Gerät einen OPC UA Client hinzu



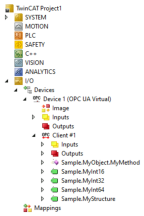
3. Öffnen Sie die Einstellungen des OPC UA Clients, indem Sie einen Doppelklick auf den Client ausführen.
4. Navigieren Sie zur Registerkarte **Settings**. Tragen Sie dort die Server-URL des OPC UA Servers ein. In diesem Beispiel lautet diese „opc.tcp://localhost:48030“.

5. Klicken Sie auf **Add Nodes**. Es wird eine Verbindung zum Server aufgebaut und der Adressraum des Servers wird in einem separaten Dialog angezeigt.



6. Selektieren Sie die obengezeigten Nodes und klicken Sie den Button **Ok**.

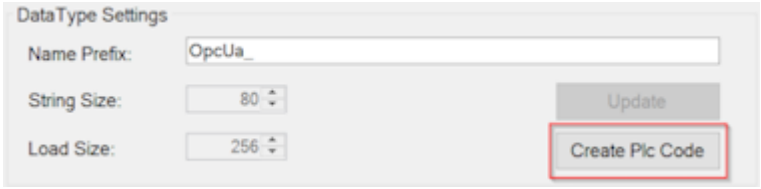
⇨ Die Variablen und die Methode zum Prozessabbild des Clients wurden hinzugefügt.



Starten der Codegenerierung

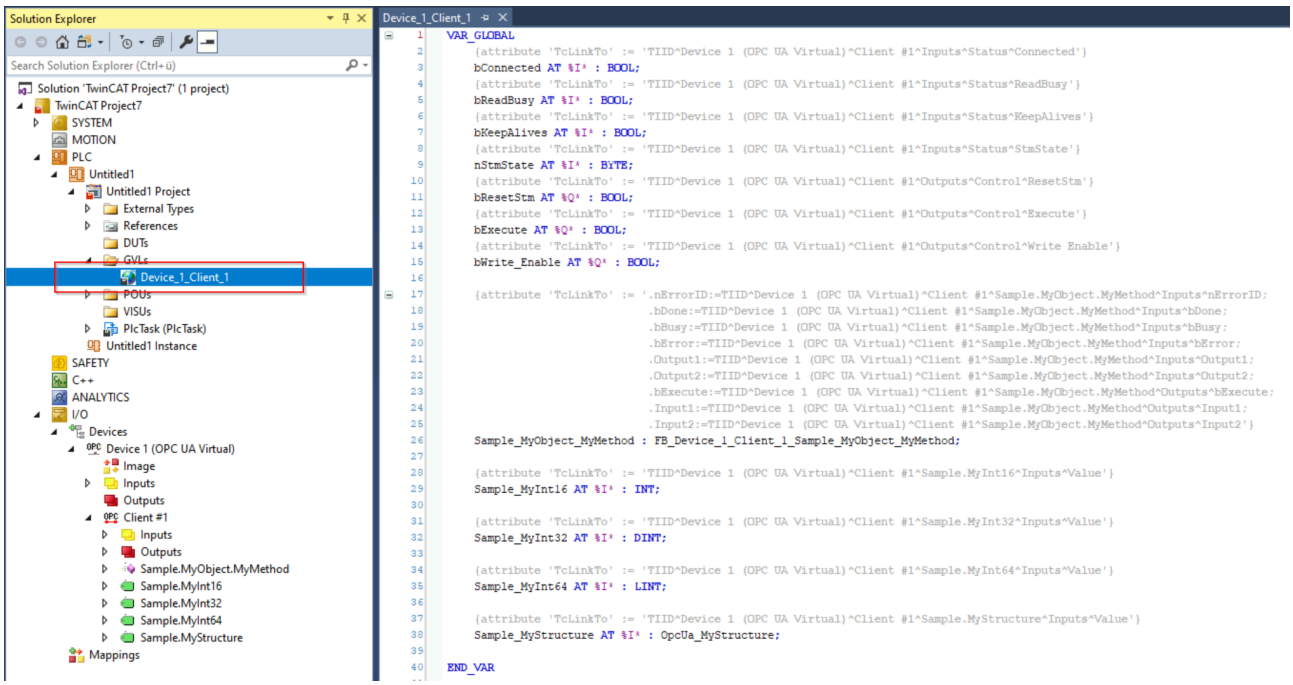
✓ Die automatische Codegenerierung soll zur Erzeugung von SPS-Variablen passend zu den hinzugefügten OPC UA Nodes verwendet werden. Die generierten SPS-Variablen werden automatisch mit den Nodes verlinkt. Alternativ können Sie das Mapping auch manuell durchführen.

1. Doppelklicken Sie auf den OPC UA Client.
2. Wählen Sie in der Registerkarte **Settings** die Sektion **DataType Settings.**



3. Klicken Sie den Button **Create Plc Code**, welcher die Codegenerierung startet.

⇨ Der Codegenerator erstellt in dem vorhandenen SPS-Projekt eine GVL deren Namen sich aus dem OPC UA Client Gerätenamen ableitet. Innerhalb der GVL wurden nun automatisch SPS-Variablen angelegt und über das Pragma „TcLinkTo“ mit den entsprechenden Nodes im Prozessabbild des I/O-Geräts verlinkt.



4. Aktivieren Sie die Konfiguration.

⇨ Die Werte der OPC UA Nodes werden aus dem Server ausgelesen und über das Mapping in die SPS-Variablen geschrieben.

Weitere Informationen zum Aufruf der Methode

Für das Aufrufen der Methode ist weitere SPS-Logik notwendig. Hierfür wurde bereits ein passender Funktionsbaustein durch die Codegenerierung erzeugt und mit den entsprechenden Ein-/Ausgabeparametern versehen, siehe Kapitel Methodenaufrufe [▶ 25].

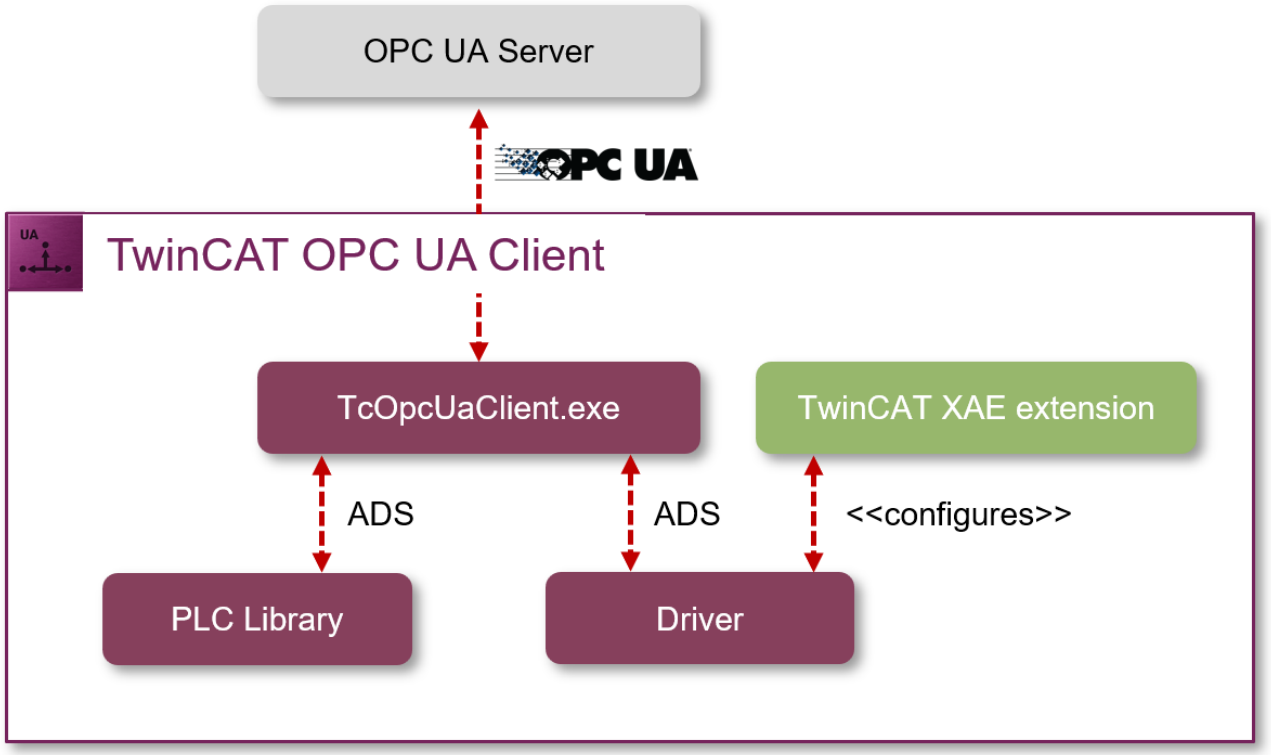
4.2 Softwarearchitektur

Die interne Softwarearchitektur dieses Produkts müssen Sie für den Betrieb der Software nicht kennen – sie kann jedoch im Einzelfall von Interesse sein. Deshalb stellen wir Sie Ihnen im Folgenden kurz vor.

Der TwinCAT OPC UA Client besteht im Wesentlichen aus folgenden Komponenten:

- Dem Prozess im Betriebssystem
- Dem Kommunikationstreiber in der Echtzeit
- Der TwinCAT-XAE-Extension
- Der SPS-Bibliothek

Das Zusammenspiel der einzelnen Komponenten wird durch das folgende Schaubild näher beschrieben:



Prozess im Betriebssystem

Der Prozess im Betriebssystem (TcOpcUaClient.exe) kümmert sich um die OPC UA Protokollfunktionen und stellt diese über einen ADS-Server zur Verfügung, sodass die TwinCAT-Echtzeitkomponenten (z. B. der Treiber oder die SPS-Bibliothek) darauf zugreifen können.

Kommunikationstreiber in der Echtzeit

Der Treiber in der Echtzeit ist zuständig für die Kommunikation des I/O-Geräts mit dem Prozess im Betriebssystem. Er wandelt die konfigurierten Prozessdaten in ADS-Telegramme um, welche mit dem Prozess ausgetauscht werden, um dort in OPC UA Befehle umgewandelt zu werden. Für die Konfiguration der Kommunikationsverbindung sowie Auswahl der Datenpunkte, gibt es eine Engineering-Komponente im TwinCAT XAE.

TwinCAT XAE Extension

Die Engineering-Komponente im TwinCAT XAE stellt eine grafische Konfigurationsschnittstelle für das I/O-Gerät bereit. Das heißt, dort können Variablen von OPC UA Servern ausgelesen- und zum Prozessabbild hinzugefügt werden, damit sie später zur Laufzeit verarbeitet werden können.

SPS-Bibliothek

Die SPS-Bibliothek stellt die OPC UA Funktionen des Prozesses im Betriebssystem für die SPS-Logik zur Verfügung. Ähnlich wie der Treiber, kommuniziert die SPS-Bibliothek somit per ADS mit dem Prozess, um auf die OPC UA Funktionen zugreifen zu können.

4.3 Unterstützte Funktionen

Der TwinCAT OPC UA Client ermöglicht den Zugriff auf einen OPC UA Server direkt aus einer Echtzeitlogik heraus.

Allgemeiner Funktionsumfang

OPC UA definiert einen großen Funktionsumfang, welcher sich unter Umständen nicht 1:1 auf eine SPS-Echtzeitumgebung abbilden lässt. Die folgende Tabelle gibt einen Überblick über den aktuellen Funktionsumfang des TwinCAT OPC UA Client. Fehlende Features werden zukünftig -wie gewohnt- in Form von Updates ausgeliefert.

Feature	PLCopen Funktionsbausteine	I/O Client
Polling	x	x
Subscriptions	-	x
Methodenaufrufe	x	x
Basisdatentypen nach IEC61131	x	x
Strukturen	-	x
Arrays von Basisdatentypen nach IEC61131	x	x
Arrays von Strukturen	-	x
Arrays mit fester Länge	x	x
Arrays mit dynamischer Länge	-	-
Security auf Transportebene mit X.509 Zertifikaten (self-signed + CA)	x	x
Security auf Applikationsebene mit Benutzername/Password	x	x
Security auf Applikationsebene mit X.509 Zertifikaten	x	x
Kommunikation mit None/None Endpunkt	x	x
Kommunikation mit Basic128 Endpunkt	x	x
Kommunikation mit Basic128Rsa15 Endpunkt	x	x
Kommunikation mit Basic256 Endpunkt	x	x
Kommunikation mit Basic256Sha256 Endpunkt	x	x

Basisdatentypen nach IEC61131

Zum Lesen und Schreiben von Daten muss der Datentyp der OPC UA Node dem TwinCAT-Datentyp zugeordnet werden („Mapping“). Die Zuordnung von Basisdatentypen wird in dem standardisierten Informationsmodell „PLCopen OPC UA Information Model for IEC 61131-3“ beschrieben und ist im Folgenden aufgelistet. Dieses Mapping können Sie sowohl auf die PLCopen-Funktionsbausteine als auch auf den TwinCAT OPC UA I/O Client anwenden.

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

4.4 Applikationsverzeichnisse

Diese Applikation verwendet verschiedene Verzeichnisse, um relevante Informationen abzuspeichern, wie z. B. Konfigurations- oder Zertifikatsdateien.

Installationsverzeichnis

Das Basis-Installationsverzeichnis der Applikation ist relativ zum TwinCAT-Installationsverzeichnis.

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

Unterhalb dieses Verzeichnisses wird die Applikation dann in folgendes Verzeichnis installiert:

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

Zertifikatsverzeichnis

Zertifikatsdateien, welche zum Aufbau einer gesicherten Kommunikationsverbindung verwendet werden, werden in folgendem Verzeichnis abgelegt:

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

Logdateien

Logdateien werden in folgendem Verzeichnis abgelegt:

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

4.5 Lesen von Variablen

Über den TwinCAT OPC UA I/O Client können Variablenwerte aus einem OPC UA Server ausgelesen werden. Das Sampling der Werte kann hierbei über verschiedene Mechanismen erfolgen, welche im Folgenden näher beschrieben werden.

Die verschiedenen Einstellungen in diesem Zusammenhang werden in den Konfigurationsseiten des I/O-Geräts durchgeführt. Im Bereich **Process Data Configuration** werden verschiedene Parameter aufgelistet, um die verschiedenen Modi der Datenerfassung von einem Server zu beeinflussen.

Der TwinCAT OPC UA I/O Client bietet drei verschiedene Modi zur Datenaufnahme an:

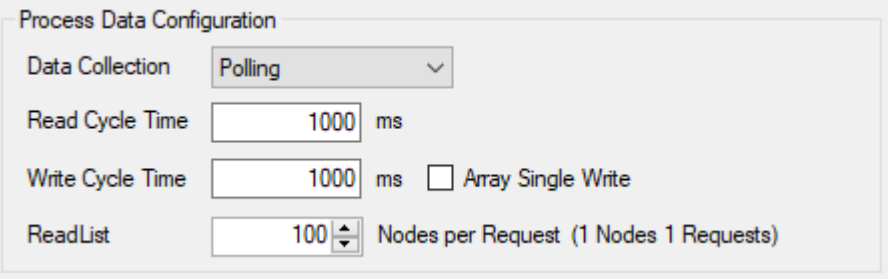
- Polling (Zyklisches Lesen/Schreiben)
- Subscriptions

- OnTrigger

Polling (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**
>
> 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.

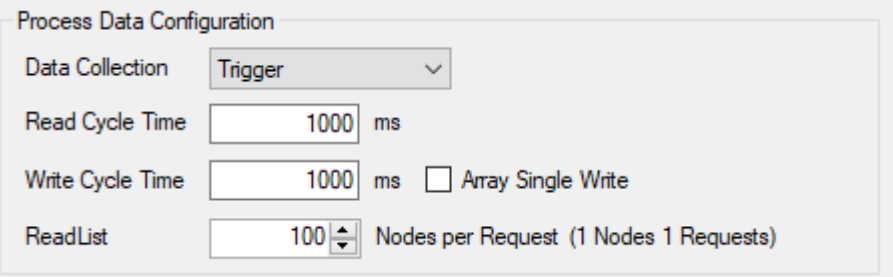


Parameter	Beschreibung
Read Cycle Time	Spezifiziert, wie schnell Variablen zyklisch gelesen werden.
Write Cycle Time	Legt fest, wie häufig ein Schreibbefehl auf dem OPC UA-Kanal ausgelöst wird. Wenn sich ein Variablenwert innerhalb einer spezifizierten Zykluszeit mehrfach ändert, wird nur der letzte 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.

OnTrigger

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.

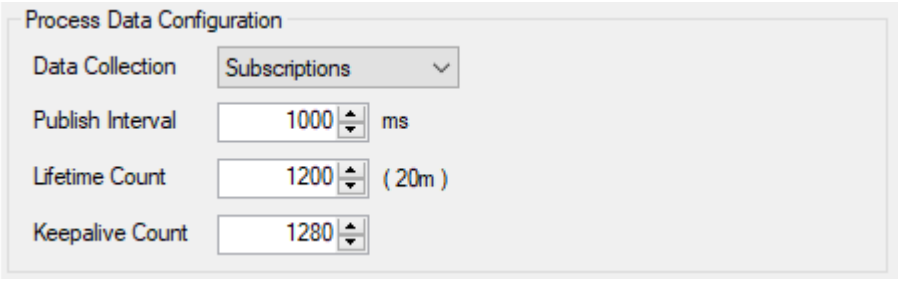
Beim Schreiben hingegen wird bei gesetzter Trigger-Variable in jedem Zyklus ein Wert geschrieben. Hierbei wird keine Werteänderung betrachtet.



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).



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.

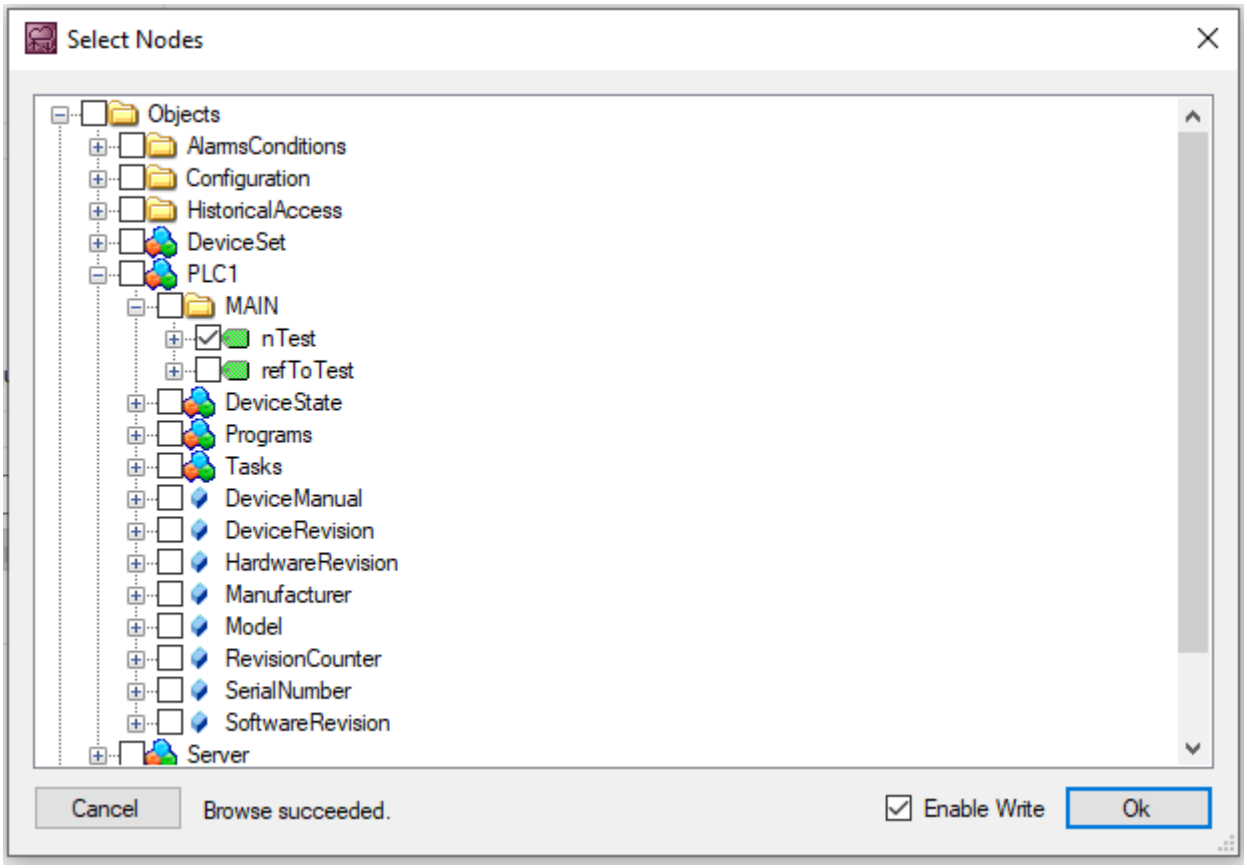
4.6 Schreiben von Variablen

Um das Schreiben von Variablen zu aktivieren, müssen mehrere Voraussetzungen erfüllt sein:

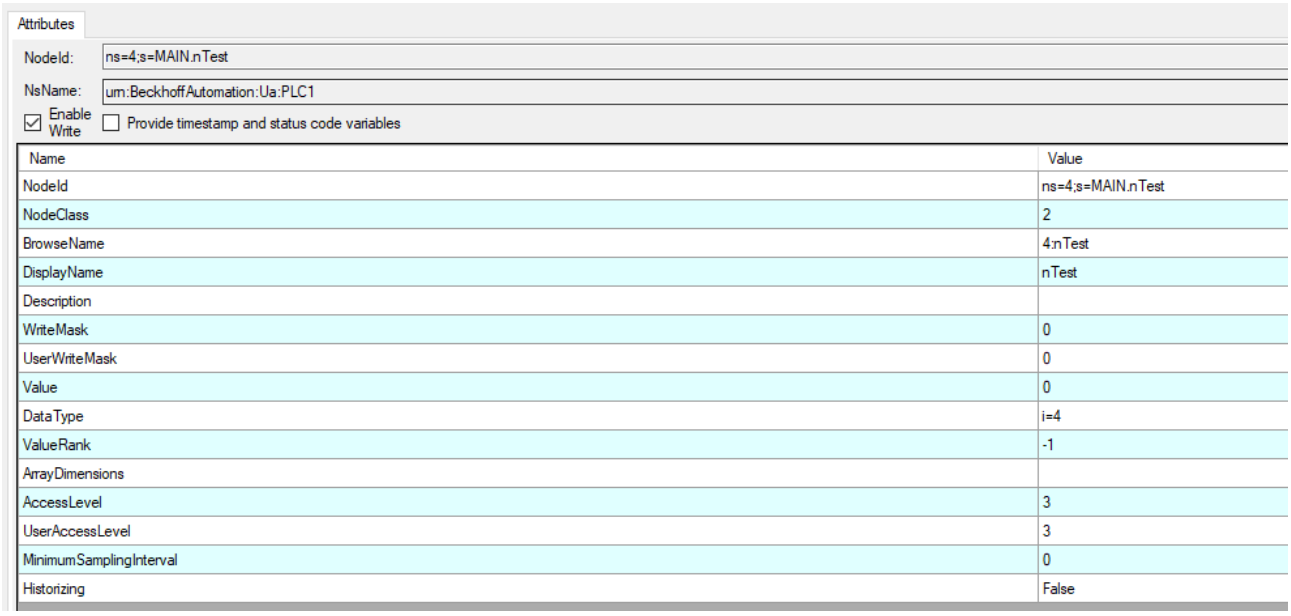
1. An der Variablen muss das Flag "Enable Write" gesetzt sein. Dies kann entweder während des Hinzufügens über den Button **Add Nodes** erfolgen oder nachträglich in den Parametereinstellungen der Variablen.
2. Vor einem Schreibkommando muss der Ausgang "Write Enable" am I/O Client global aktiviert werden. 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 würde.

Setzen von Enable Write an einer Variablen

Damit für eine Variable nicht nur ein Input (Read)-Element, 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:



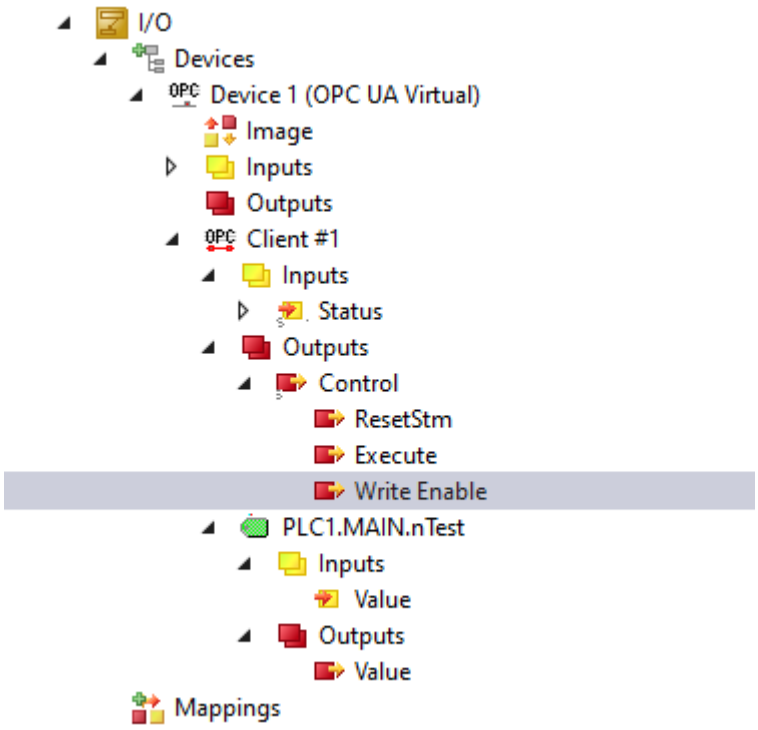
Alternativ kann diese Einstellung auch nachträglich noch über die Konfigurationsparameter der Variablen im Prozessabbild aktiviert/deaktiviert werden.



Name	Value
NodeId	ns=4;s=MAIN.nTest
NodeClass	2
BrowseName	4:nTest
DisplayName	nTest
Description	
WriteMask	0
UserWriteMask	0
Value	0
DataType	i=4
ValueRank	-1
ArrayDimensions	
AccessLevel	3
UserAccessLevel	3
MinimumSamplingInterval	0
Historizing	False

Globales Aktivieren des Schreibzugriffs

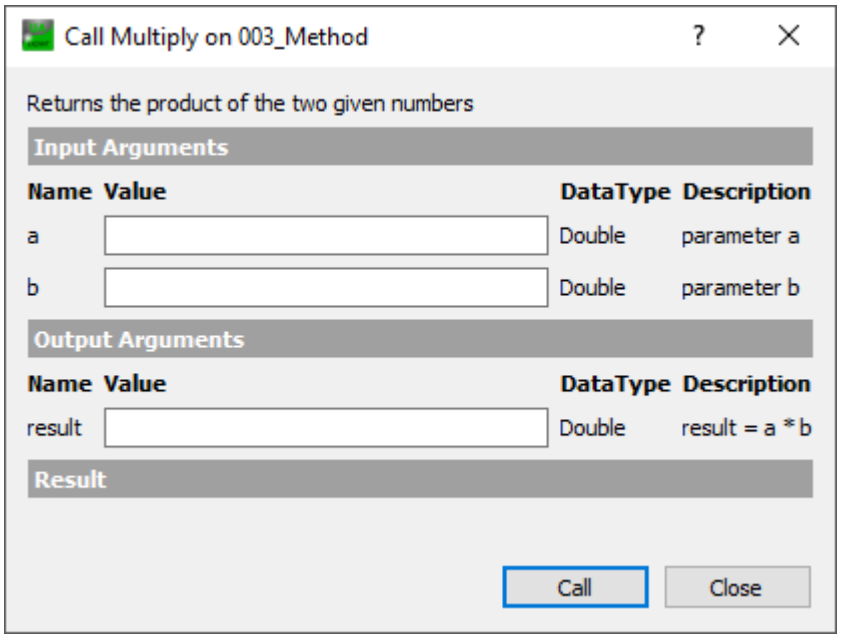
Vor dem Absenden von Schreibkommandos müssen diese global freigeschaltet werden. Dies erfolgt durch das Setzen der Ausgangsvariablen "Write Enable" am I/O Client:



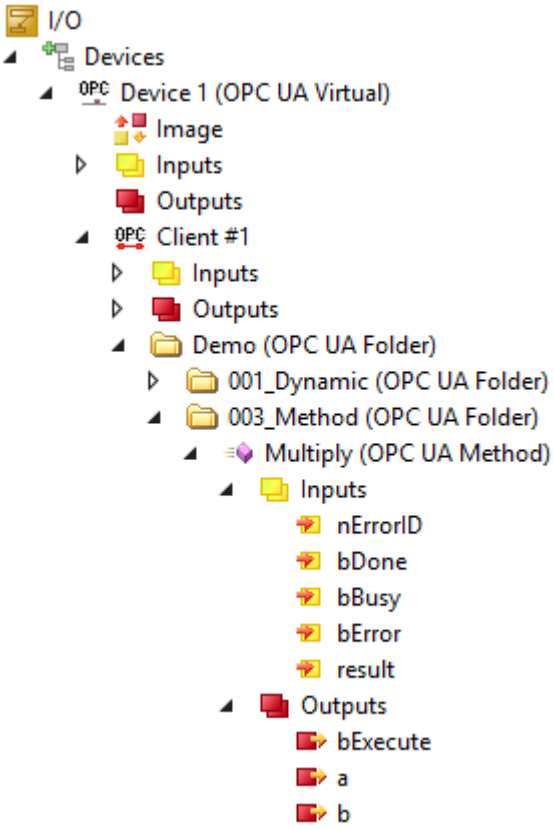
4.7 Methodenaufrufe

Der TwinCAT OPC UA I/O Client unterstützt den Aufruf von Servermethoden. Sie können eine Methode zum Prozessabbild hinzufügen, wie jede andere Variable. Die „Eingangsargumente“ der Methode sind dann als Ausgangsvariablen im Prozessabbild verfügbar, wohingegen die „Ausgangsargumente“ als Eingangsvariablen hinzugefügt werden. Zusätzliche Eingangs- und Ausgangsvariablen, z. B. bExecute, bBusy, bError, werden dem Prozessabbild hinzugefügt, sodass die Methode aufgerufen werden kann.

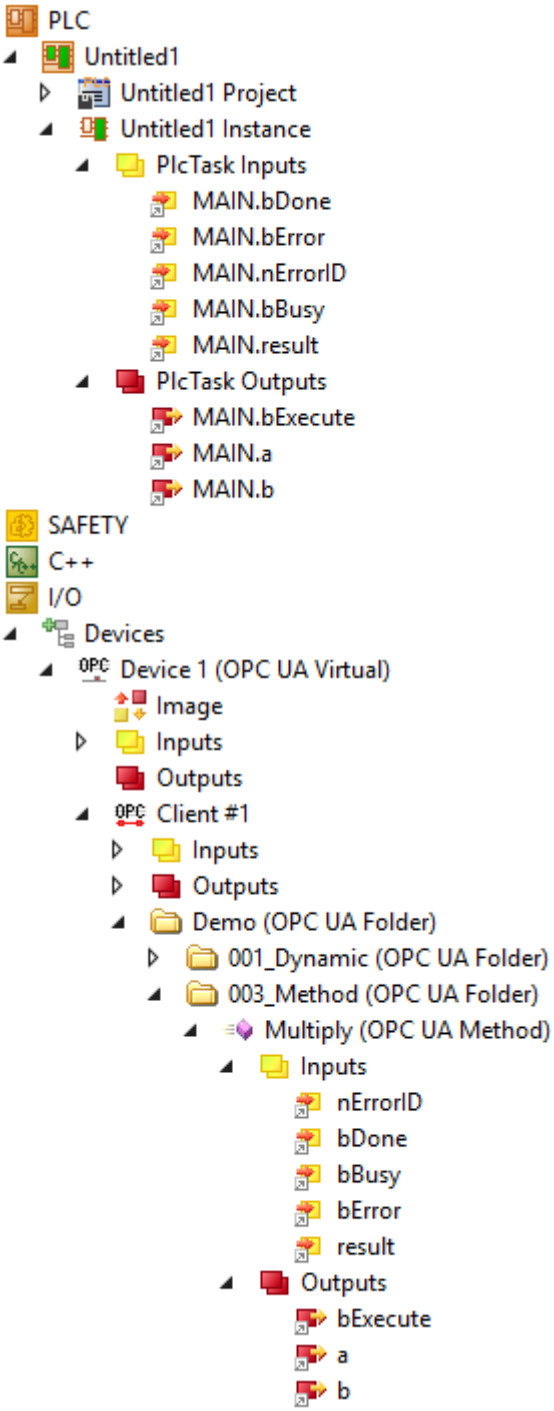
Beispiel: Methode auf Server



Beispiel: Methode nach dem Hinzufügen zum Prozessabbild



Sie können dann ein Mapping zwischen den Eingangs-/Ausgangsvariablen und den SPS-Variablen erstellen.



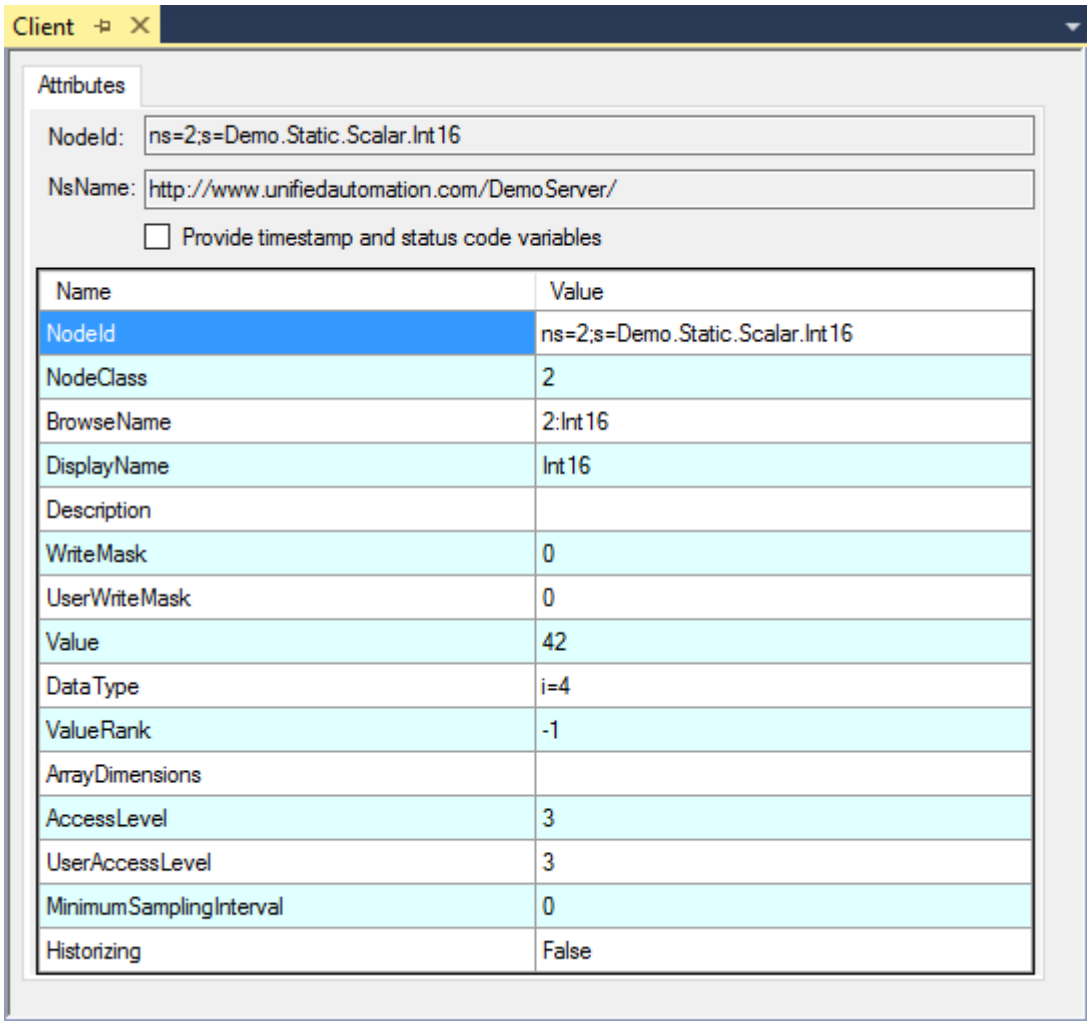
Aufruf einer Methode

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 ob er erfolgreich war.

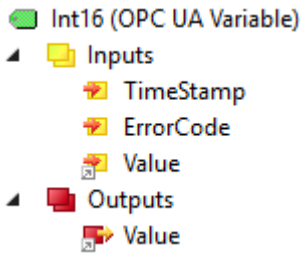
a	LREAL	3
b	LREAL	42
result	LREAL	126
bExecute	BOOL	TRUE
nErrorID	DINT	0
bDone	BOOL	TRUE
bError	BOOL	FALSE
bBusy	BOOL	FALSE

4.8 Timestamp und StatusCode

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.



4.9 Strukturen

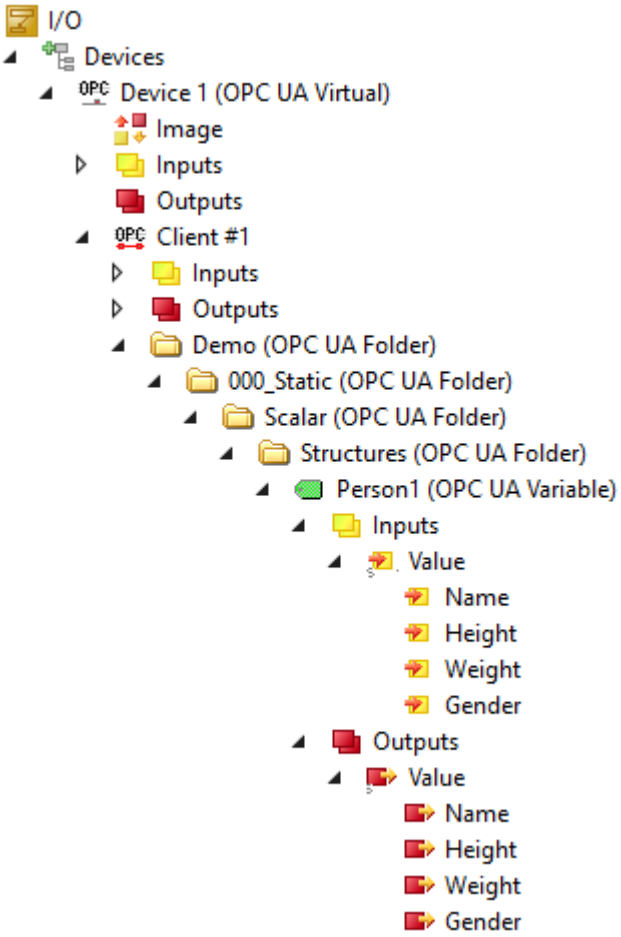
Der OPC UA I/O Client unterstützt Lese-/Schreibvorgänge bei strukturierten Datentypen (StructuredTypes). StructuredTypes können Sie dem Prozessabbild genau so, wie jede andere Variable, hinzufügen. Beim Hinzufügen eines StructuredTypes zum Prozessabbild wird der zu parsende Typ dem Typsystem von TwinCAT 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).

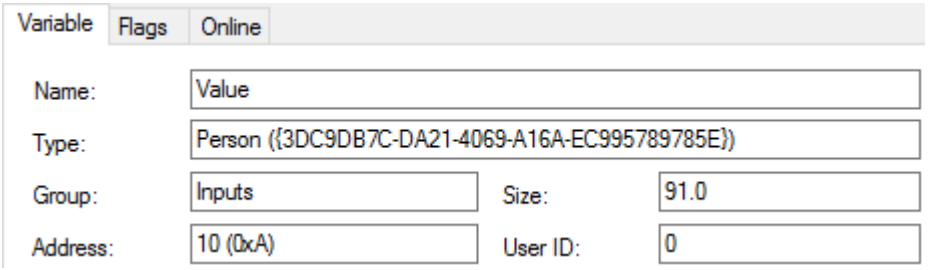
Beispiel: StructuredTypes im Prozessabbild



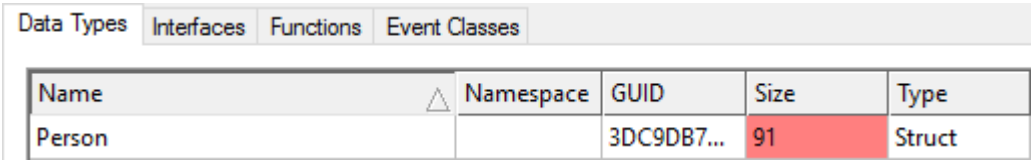
Nachdem Sie einen Knoten dem Prozessabbild hinzugefügt haben, enthält das Prozessabbild den Knoten und die strukturellen Informationen des Typs, z. B., ob eine einzige Membervariable des Knotens gelesen oder geschrieben werden soll.

StructuredTypes im Typsystem von TwinCAT

Der Datentyp wird dem Typsystem von TwinCAT hinzugefügt. Die „Value“ Tree Items haben dann diesen Datentyp.



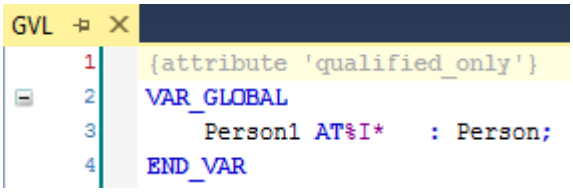
Sie können den Datentyp auch im Typsystem von TwinCAT unter **SYSTEM > Type System** einsehen.

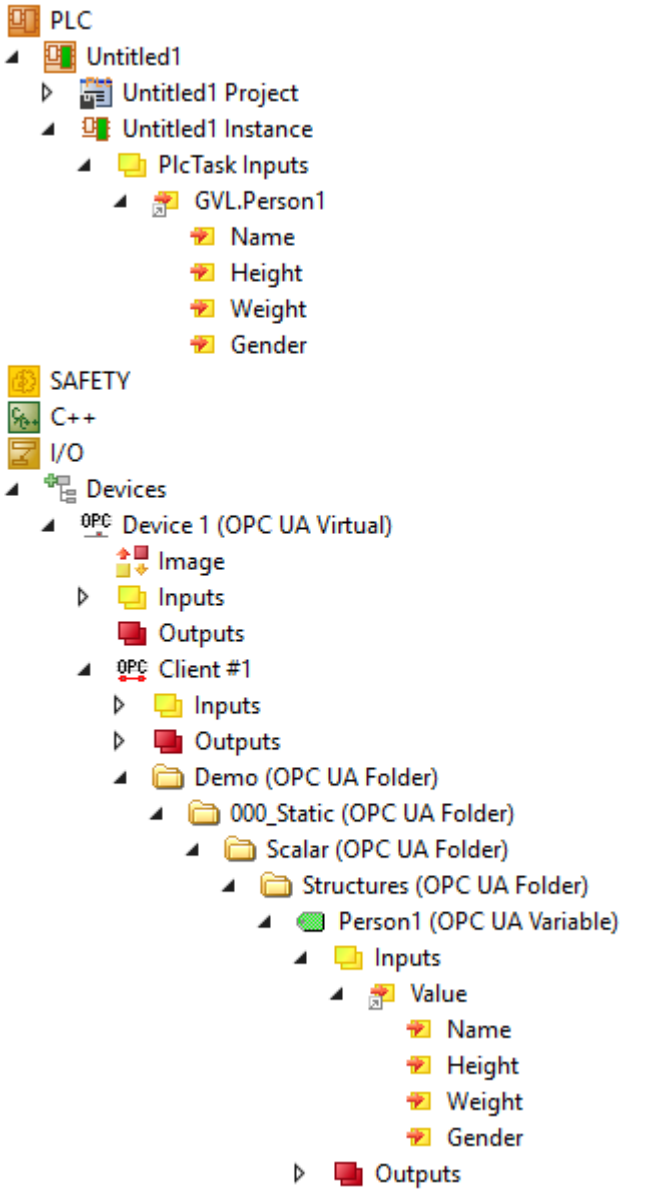


Zur Unterscheidung des Datentyps von anderen Datentypen können Sie in den Einstellungen des OPC UA Clients ein Präfix hinzufügen.

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.





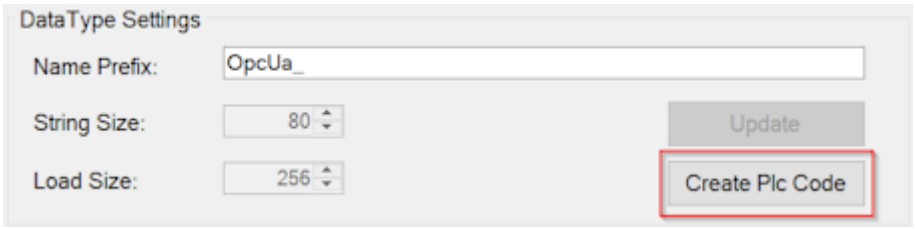
MAIN [Online]

TwinCAT_Project5.Untitled1.MAIN

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

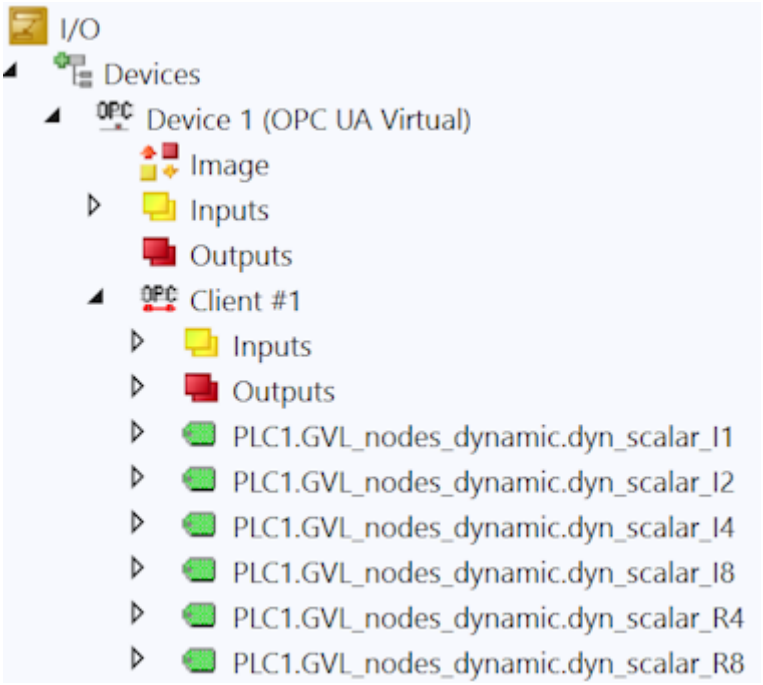
4.10 Codegenerierung

Mit Hilfe der automatischen Codegenerierung lassen sich schnell und einfach SPS-Variablen für die im I/O-Prozessabbild konfigurierten OPC UA Nodes erzeugen und automatisch mit diesen verlinken. Diese Funktion steht im Konfigurationsdialog des I/O Clients über den Button **Create Plc Code** zur Verfügung. Diese Funktion benötigt als Voraussetzung ein existierendes SPS-Projekt in der aktuellen Solution.

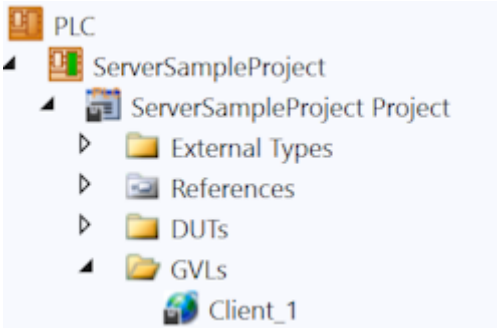


Nach dem Aufruf dieser Funktion wird eine neue **G**lobale **V**ariablen**l**iste (GVL) mit dem Namen des I/O Clients im SPS-Projekt angelegt. Anschließend werden alle OPC UA Nodes eingelesen und entsprechende Variablen in der GVL angelegt. Jede Variable erhält hierbei das TcLinkTo-Attribut für eine automatische Verlinkung mit der entsprechenden Variablen aus dem I/O-Prozessabbild.

Beispiel: Im I/O-Teil von TwinCAT XAE wurde ein TwinCAT OPC UA I/O Client mit dem Namen „Client 1“ angelegt, zu welchem diverse OPC UA Nodes von einem Server hinzugefügt wurden.



Nach dem Aufruf der Codegenerierung wurde nun in dem (bereits vorhandenen) SPS-Projekt eine neue GVL mit dem Namen „Client_1“ angelegt. Diese enthält dann entsprechende SPS-Variablen für die einzelnen Nodes, welche dann über das TcLinkTo-Attribut automatisch verlinkt wurden.



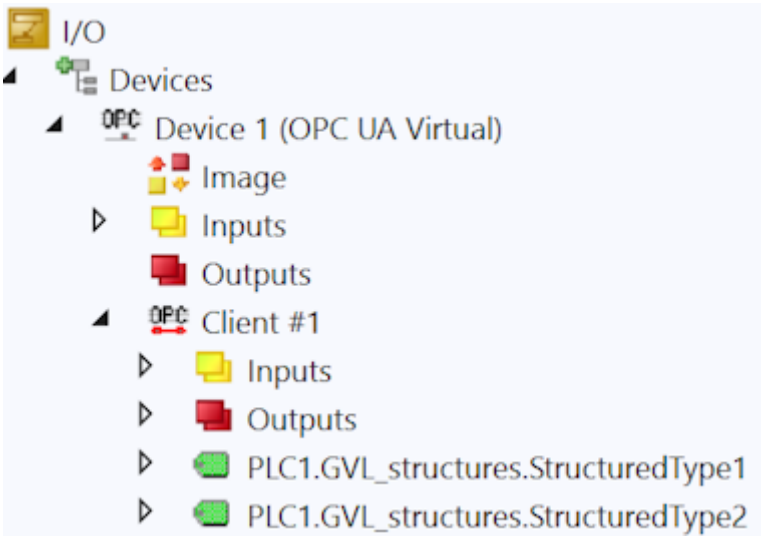
```
VAR_GLOBAL

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

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

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

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

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

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

END_VAR
```

Zusätzlich werden auch die Steuervariablen von dem jeweiligen I/O Client als Variablen in der GVL angelegt und verlinkt (im Screenshot oben nicht dargestellt).

Codegenerierung für Strukturen

Eine OPC UA Node, welche einen sogenannten StructuredDataType repräsentiert, wird von der Codegenerierung ebenfalls berücksichtigt und eine entsprechende Variable in der GVL angelegt. Da der StructuredDataType als nativer Datentyp im TwinCAT-Typsystem angelegt wurde, kann mit ihm wie mit einer normalen Struktur umgegangen werden.

Beispiel: Im Prozessabbild des I/O-Clients wurden zwei StructuredDataTypes von einem Server hinzugefügt. Die Datentypen der StructuredDataTypes auf dem Server lauten ST_Complex1 und ST_Complex2 (im Screenshot unten nicht sichtbar).



Durch die Codegenerierung wurde nun eine entsprechende GVL mit zwei Variablen aus dem jeweils automatisch erzeugten TwinCAT-Datentyp angelegt, welcher dem jeweiligen StructuredDataType entspricht.

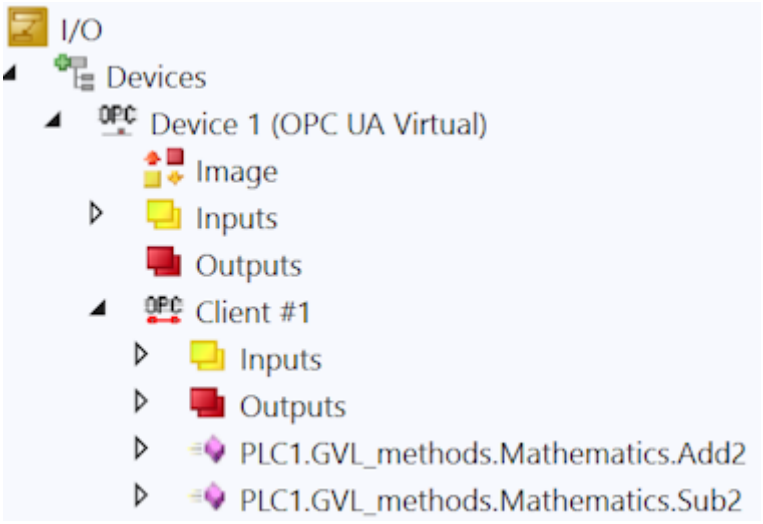
```
VAR_GLOBAL

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

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

END_VAR
```

Codegenerierung für Methoden

Eine OPC UA Methode hat Ein- und Ausgabeparameter, welche entsprechend an die Methode übergeben bzw. von ihr zurückgegeben werden. Des Weiteren ist eine Methode ein in sich geschlossener Aufruf; sie muss explizit vom Client gestartet werden. Diese Methodik ist im Prozessabbild des I/O-Clients an der

Methode entsprechend abgebildet und wird bei der Codegenerierung auch berücksichtigt. Für eine Methode wird, anders als bei normalen Variablen oder Strukturen, ein eigener Funktionsbaustein angelegt, welcher dann in der GVL referenziert wird.

Beispiel: Im Prozessabbild des I/O-Clients wurden zwei Methoden von einem Server hinzugefügt.



Durch die Codegenerierung wurden nun eine entsprechende GVL sowie zwei Funktionsbausteine angelegt, welche die Ein-/Ausgangs- und Steuervariablen der jeweiligen Methode repräsentieren.

```
VAR_GLOBAL

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

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

END_VAR
```

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

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

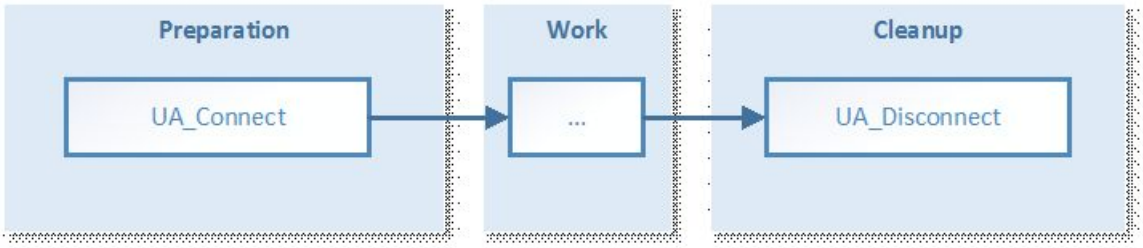
4.11 PLCopen-Funktionsbausteine

Der TwinCAT OPC UA Client bietet mehrere Möglichkeiten, direkt aus der Steuerungslogik heraus mit einem oder mehreren OPC UA Servern zu kommunizieren. Zum einen gibt es ein TwinCAT-I/O-Gerät, welches eine eine einfache, Mapping-basierte Schnittstelle bietet. Zum anderen stehen durch die PLCopen genormte Funktionsbausteine zur Verfügung, über die eine Verbindung mit einem OPC UA Server direkt aus der SPS-Logik heraus initiiert werden kann. Die Handhabung dieser Bausteine sollen im Folgenden näher beschrieben werden. Dieser Artikel besteht aus den folgenden Sektionen:

- Workflow
- Bestimmung der Kommunikatonsparameter
- Herstellen einer Verbindung
- Auslesen von Variablen
- Schreiben von Variablen
- Aufruf von Methoden

Workflow

Der allgemeine Workflow bei der Verwendung der PLCopen-Funktionsbausteine lässt sich wie folgt schematisch darstellen:



In der Vorbereitungsphase werden die Kommunikationsparameter eingerichtet und eine Verbindung zum Server aufgebaut. Anschliessend erfolgt die Ausführung der gewünschten Funktion (Lesen, Schreiben, Methodenaufrufe), gefolgt vom Trennen der Kommunikationsverbindung.

Bestimmung der Kommunikationsparameter

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.:

- NodeID
- NamespaceIndex und entsprechender NamespaceURI
- DataType
- MethodNodeID und ObjectNodeID

Die folgende Dokumentation verwendet den generischen OPC UA Client UA Expert als Beispiel. Dieser Client kann über die Webseite von Unified Automation erworben werden: www.unified-automation.com.

Knoten sind durch die folgenden drei Attribute gekennzeichnet, welche die sogenannte NodeID bilden:

- 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

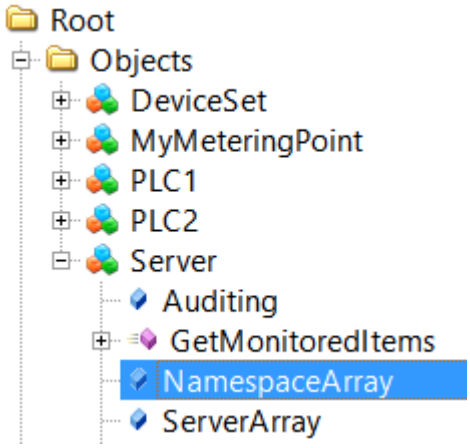
Diese Attribute stellen die sogenannte *NodeID* dar – die Darstellung eines Knotens auf einem OPC UA Server – und werden von vielen nachfolgenden Funktionsbausteinen benötigt.

Mithilfe der Software 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, zum Beispiel:

NodeId	NodeId
NamespaceIndex	5
IdentifierType	String
Identifier	MAIN.nCounter

Nach der OPC-UA-Spezifikation kann der NamespaceIndex 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 [▶ 70]. 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, zum 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

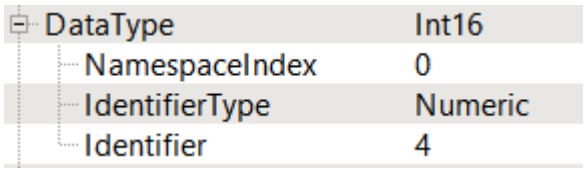
Im obigen Abschnitt wird beispielhaft eine NodeID gezeigt, in der der NamespaceIndex 5 ist. Nach dem in der Abbildung gezeigten NamespaceArray ist der entsprechende NamespaceURI urn*://SVENG-NB04/BeckhoffAutomation/Ua/PLC1*. Dieser URI kann nun für den Funktionsbaustein UA_GetNamespaceIndex verwendet werden. Der OPC UA Server stellt sicher, dass der URI immer derselbe bleibt, auch nach einem Neustart.

Korrekten NamespaceIndex beachten

Da sich der gezeigte NamespaceIndex verändern kann, sollten für die spätere Nutzung mit anderen Funktionsbausteinen, z. B. UA_Read [▶ 82], UA_Write [▶ 85], zur Auflösung des korrekten NamespaceIndex immer der NamespaceURI in Kombination mit dem Funktionsbaustein UA_GetNamespaceIndex verwendet werden.

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 eine 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, zum Beispiel:



In diesem Fall ist der Datentyp (DataType) „Int16“. Dieser muss einem äquivalenten Datentyp in der SPS zugeordnet werden, z. B. „INT“.

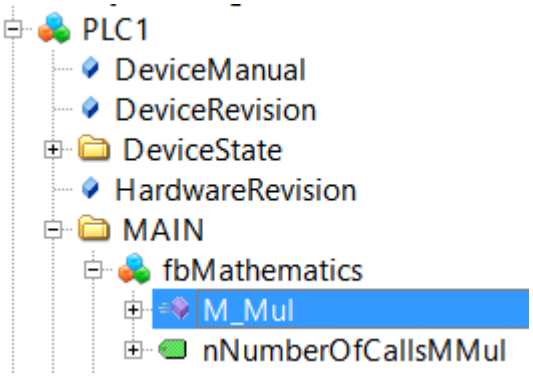
MethodNodeID und ObjectNodeID

Beim Aufruf von Methoden aus dem OPC-UA-Namensraum sind zwei Identifier erforderlich, wenn der Methodenhandle unter Verwendung des Funktionsbausteins UA_MethodGetHandle [▶ 76] erfasst wird:

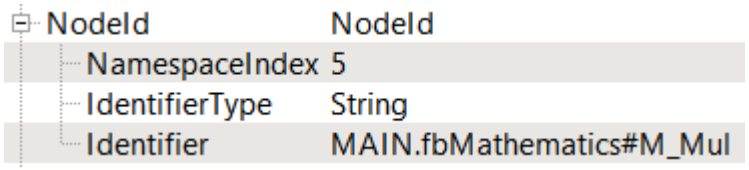
- ObjectNodeID: Identifiziert das UA-Objekt, das die Methode enthält.
- MethodNodeID: Identifiziert die Methode selbst.

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.

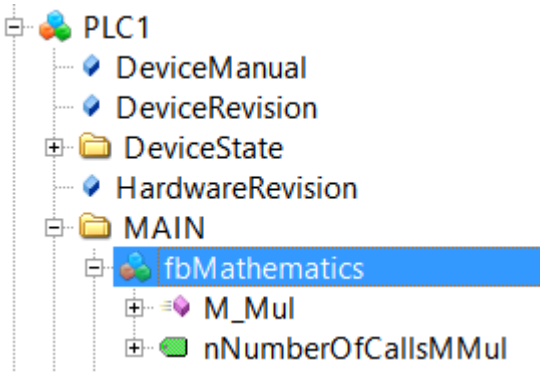
Beispiel Methode M_Mul:



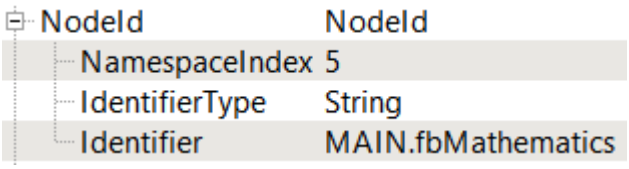
Der Method Identifier ist dann im Attributes-Panel sichtbar.



Beispiel Objekt fbMathematics:



Der Object Identifier ist dann im Attributes-Panel sichtbar.



Herstellen einer Verbindung

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.

Die folgenden Funktionsbausteine sind erforderlich, um eine Verbindung zu einem OPC UA Server herzustellen und später die Sitzung zu unterbrechen: UA_Connect [▶ 67], UA_Disconnect [▶ 69].

Lesen Sie zunächst den Abschnitt Wie Kommunikationsparameter zu bestimmen sind, um bestimmte UA-Funktionalitäten besser verstehen zu können (z. B., wie NodeIdentifier bestimmt werden können).

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.

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:

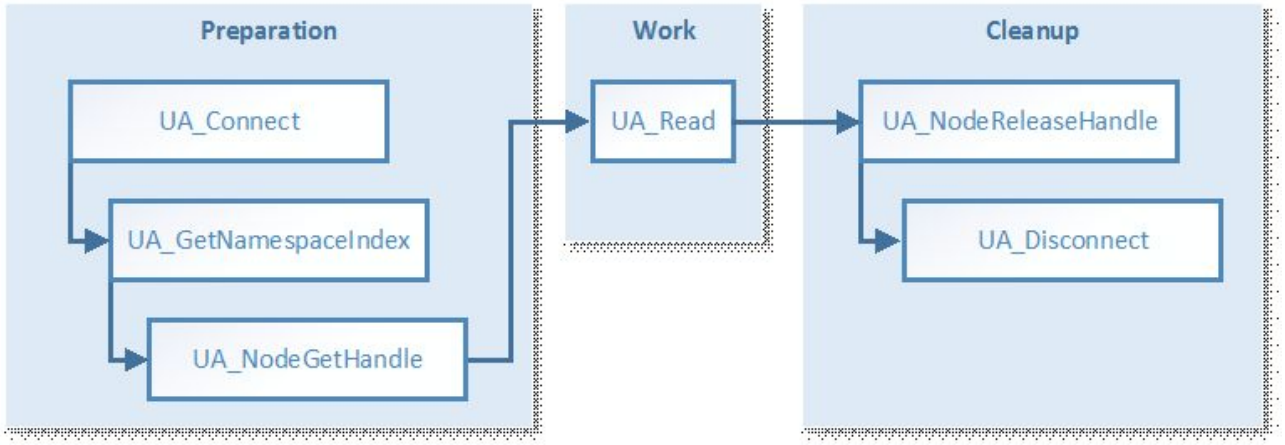
```
CASE iState OF

  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
```

Lesen von Variablen

Im nachfolgenden Abschnitt wird beschrieben, wie Sie die Funktionsbausteine TcX_PLCopen_OpcUa verwenden, um einen OPC-UA-Knoten von einem lokalen oder remote OPC UA Server auszulesen. 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 [▶ 67], UA_GetNamespaceIndex [▶ 70], UA_NodeGetHandle [▶ 78], UA_Read [▶ 82], UA_NodeReleaseHandle [▶ 80], UA_Disconnect [▶ 69].

Der schematische Arbeitsablauf jedes TwinCAT OPC UA Clients kann in drei verschiedene Phasen kategorisiert werden: Preparation, Work und Cleanup.

Der in diesem Abschnitt beschriebene Verwendungsfall kann wie folgt visualisiert werden:



- 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):
  - 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).
- 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).
- 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).
- Der Funktionsbaustein UA_NodeReleaseHandle erfordert einen Connection Handle (von UA_Connect) und einen Knotenhandle (von UA_NodeGetHandle).

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
```

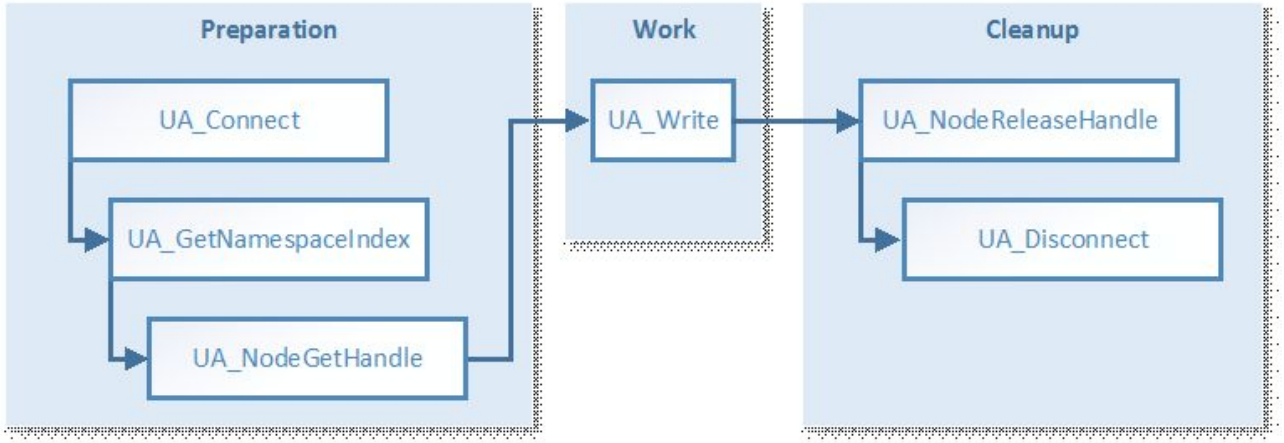
Schreiben von Variablen

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. 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 [▶ 67], UA_GetNamespaceIndex [▶ 70], UA_NodeGetHandle [▶ 78], UA_Write [▶ 85], UA_NodeReleaseHandle [▶ 80], UA_Disconnect [▶ 69].

Der schematische Arbeitsablauf jedes TwinCAT OPC UA Clients kann in drei verschiedene Phasen kategorisiert werden: Preparation, Work und Cleanup.

Der in diesem Abschnitt beschriebene Verwendungsfall kann wie folgt visualisiert werden:



- 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):
  - 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).
- 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).
- 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).
- Der Funktionsbaustein UA_NodeReleaseHandle erfordert einen Connection Handle (von UA_Connect) und einen Knotenhandle (von UA_NodeGetHandle).

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:
```

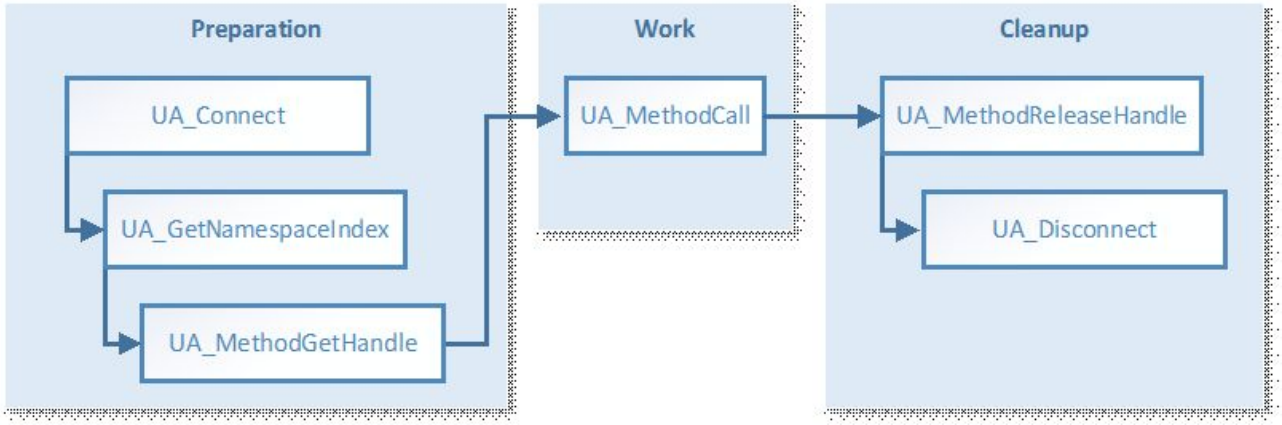
```
    [...]

END_CASE
```

Aufrufen von Methoden

Im nachfolgenden Abschnitt wird beschrieben, wie Sie die Funktionsbausteine TcX_PLCopen_OpcUa verwenden, um Methoden auf einem lokalen oder remote OPC UA Server aufzurufen. 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 [▶ 67], UA_GetNamespaceIndex [▶ 70], UA_MethodGetHandle [▶ 76], UA_MethodCall [▶ 74], UA_MethodReleaseHandle [▶ 77], UA_Disconnect [▶ 69].

Der schematische Arbeitsablauf jedes TwinCAT OPC UA Clients kann in drei verschiedene Phasen kategorisiert werden: Preparation, Work und Cleanup.

Der in diesem Abschnitt beschriebene Verwendungsfall kann wie folgt visualisiert werden:



- 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):
  - 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).
- 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).
- 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. Informationen 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).

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

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
```

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;
    nCummulatedSessionCount       : 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.
nCummulatedSessionCount	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 ST_OpcUAServerInfo [▶ 48] enthaltenen Server-Informationen ab.

5.1.1.3 E_OpcUAServerStatus

E_OpcUAServerStatus repräsentiert den Laufzeitstatus eines TwinCAT OPC UA Servers.

Syntax

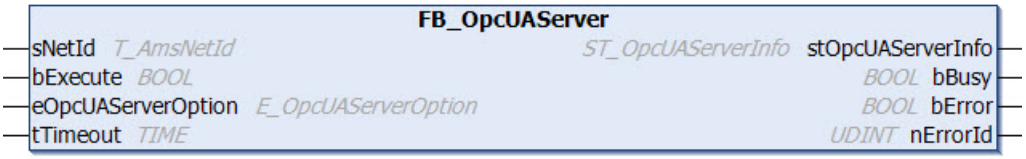
```
TYPE E_OpcUAServerStatus
(
    eOPCUAServerStatus_None,
    eOPCUAServerStatus_Alive,
    eOPCUAServerStatus_NotResponding
);
END_TYPE
```

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	E_OpcUAServerOption [▶ 48]	Gibt die auszuführende Operation an.
tTimeout	TIME	ADS Timeout

Ausgänge

Name	Typ	Beschreibung
stOpcUAServerInfo	ST_OpcUAServerInfo [▶ 48]	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 [▶ 49]	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
DataType	OPC UA DataType
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_ReferenceTypeId	ReferenceTypeId
eUABRM_IsForward	IsForward
eUABRM_ReferenceTypeInfo	ReferenceTypeInfo
eUABRM_NodeClass	NodeClass
eUABRM_BrowseName	BrowseName
eUABRM_DisplayName	DisplayName
eUABRM_TypeDefinition	TypeDefinition
eUABRM_TargetInfo	TargetInfo
eUABRM_All	All

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_Undefinied      := -1,
```

```
    eUAType_Null            := 0,
    eUAType_Boolean         := 1,
    eUAType_SByte           := 2,
    eUAType_Byte            := 3,
    eUAType_Int16           := 4,
    eUAType_UInt16          := 5,
    eUAType_Int32           := 6,
    eUAType_UInt32          := 7,
    eUAType_Int64           := 8,
    eUAType_UInt64          := 9,
    eUAType_Float           := 10,
    eUAType_Double          := 11,
    eUAType_String          := 12,
    eUAType_DateTime        := 13,
    eUAType_Guid            := 14,
    eUAType_ByteString      := 15,
    eUAType_XmlElement      := 16,
    eUAType_NodeId          := 17,
    eUAType_ExpandedNodeId  := 18,
    eUAType_StatusCode      := 19,
    eUAType_QualifiedName   := 20,
    eUAType_LocalizedText   := 21,
    eUAType_ExtensionObject := 22,
    eUAType_DataValue       := 23,
    eUAType_Variant         := 24,
    eUAType_DiagnosticInfo  := 25
)DINT;
END_TYPE
```

Werte

Name	Beschreibung
eUAType_Undefinied	Undefinied
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
bIncludeSubtypes	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 [▶ 61])

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 [▶ 56].
eSecurityPolicyUri	Sicherheitsrichtlinien-Uri. Verfügbare Sicherheitsrichtlinien-Uri siehe E_UASecurityPolicy [▶ 56].
eTransportProfileUri	Transportprofil-Uri. Verfügbare Transportprofil-Uri siehe E_UATransportProfile [▶ 57];
stUserIdentTokenType	Struktur mit Authentifizierungsdaten für die Anmeldung am OPC UA-Server. Vollständige Beschreibung unter ST_UAUserIdentityTokenType [▶ 63].
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:

- StartinIndex 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 [▶ 53])
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 [▶ 70] bestimmt werden.
nReserved	Platzhalter
sIdentifier	Bezeichner wie im UA Namensraum gezeigt (Attribut 'Identifier').
eIdentifierType	Typ der Variablen, beschrieben mittels E_UAIdentifierType [▶ 55].

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 [▶ 51]).
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 [▶ 60]).

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
stReferenceTypeId	NodeId des Referenztyps (z.B. Organizes, HasChild, HasTypeDefinition, …) als Datentyp ST_UANodeId [▶ 61].
bIsForward	Gibt an ob es sich bei der Referenz um eine Forward oder Backward Referenz handelt.
stNodeId	NodeId als Datentyp ST_UAExpandedNodeId [▶ 59].
stBrowseName	BrowseName der Referenz.
stDisplayName	DisplayName der Referenz (ST_UALocalizedText [▶ 61]).
eNodeClass	NodeClass der Referenz (E_UANodeClassMask [▶ 55]).
stTypeDefinition	Typdefinition (HasTypeDefinition) (ST_UAExpandedNodeId [▶ 59]).

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 [▶ 58]..
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 [▶ 71] 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 [▶ 64]	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

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 [▶ 63].

Syntax

```
{attribute 'qualified_only'}
TYPE UAHAUpdateStatusCode :
(
    HistorianRaw          := 0,          // A raw data value.
    HistorianCalculated   := 1,   // A data value which was calculated.
    HistorianInterpolated := 2, // A data value which was interpolated.
    Reserved              := 3,              // Undefined.
    HistorianPartial      := 4,      // A data value which was calculated with an incomplete interva
l.
    HistorianExtraData    := 8,    // A raw data value that hides other data at the same timestamp.
    HistorianMultiValue   := 16   // Multiple values match the Aggregate criteria (i.e. multiple min
imum values at different timestamps within the same interval).
) UDINT;
END_TYPE
```

Werte

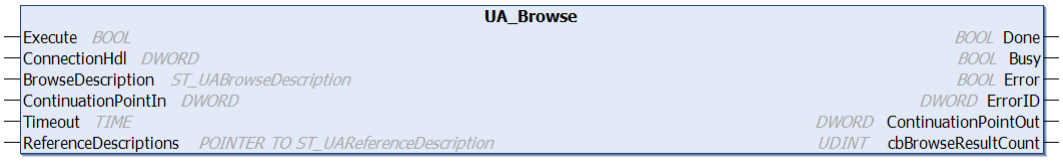
Name	Beschreibung
HistorianRaw	HistorianRaw
HistorianCalculated	HistorianCalculated
HistorianInterpolated	HistorianInterpolated
Reserved	Reserved
HistorianPartial	HistorianPartial
HistorianExtraData	HistorianExtraData
HistorianMultiValue	HistorianMultiValue

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 [▶ 58]	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

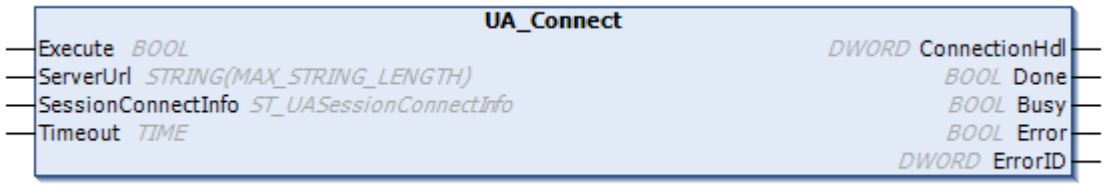
```
VAR_OUTPUT
    Done                : BOOL;
    Busy                : BOOL;
    Error               : BOOL;
    ErrorID             : DWORD;
    ContinuationPointOut : DWORD;
    cbBrowseResultCnt   : UDINT;
END_VAR
```

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 befehlsspezifische Fehlercode ist in ErrorID enthalten.
ErrorID	DWORD	Enthält den befehlsspezifischen 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 [▶ 59])
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

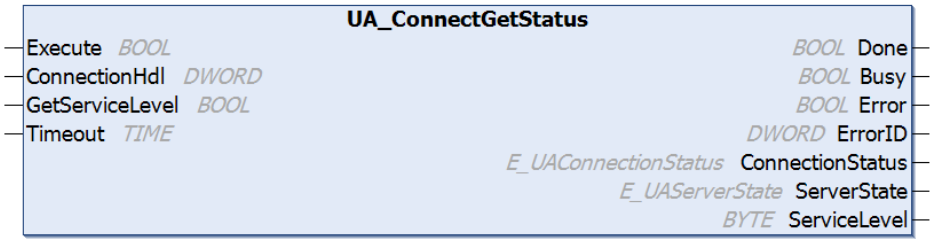
```
VAR_OUTPUT
    ConnectionHdl : DWORD;
    Done          : BOOL;
    Busy          : BOOL;
    Error         : BOOL;
    ErrorID       : DWORD;
END_VAR
```

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 befehlsspezifische Fehlercode ist in ErrorID enthalten.
ErrorID	DWORD	Enthält den befehlsspezifischen 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



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 [▶ 53] 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 ServiceLevel des OPC UA Servers aus.
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
    Done             : BOOL;
    Busy             : BOOL;
    Error            : BOOL;
    ErrorID          : DWORD;
    ConnectionStatus : E_UAConnectionStatus;
    ServerState      : E_UAServerState;
    ServiceLevel     : BYTE;
END_VAR
```

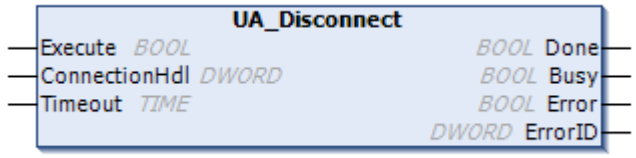
Name	Typ	Beschreibung
Done	BOOL	Schaltet auf TRUE, wenn der Funktionsbaustein erfolgreich ausgeführt wurde.
Busy	BOOL	TRUE, bis der Baustein einen Befehl ausführt hat, maximal für die Dauer des „Timeout“ am Eingang. Solange Busy = TRUE ist, akzeptieren die Eingänge keinen neuen Befehl. Es wird nicht die Zeit der Verbindung sondern die Empfangszeit überwacht.
Error	BOOL	Schaltet auf TRUE, wenn bei der Ausführung eines Befehls ein Fehler auftritt. Der befehlsspezifische Fehlercode ist in ErrorID enthalten.
ErrorID	DWORD	Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls.
ConnectionStatus	E_UAConnectionStatus	Verbindungsstatus (siehe E_UAConnectionStatus [▶ 53]).
ServerState	E_UAServerState	Serverstatus (siehe E_UAServerState [▶ 57]).
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

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 befehlsspezifische Fehlercode ist in nErrID enthalten.
ErrorID	DWORD	Enthält den befehlsspezifischen 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 [▶ 82]oder UA_Write [▶ 85]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;
```

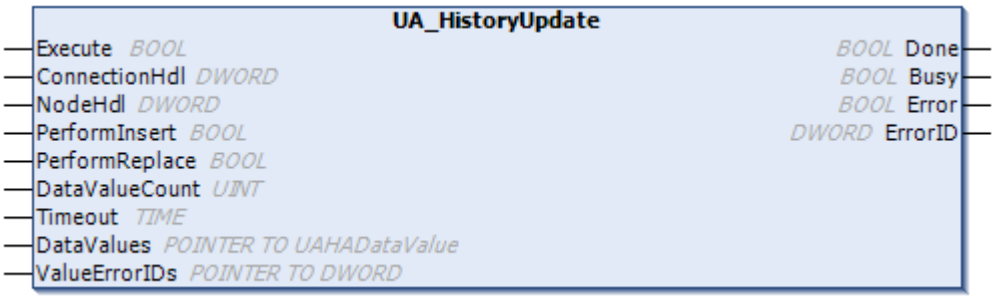
```
    Busy            : BOOL;
    Error           : BOOL;
    ErrorID         : DWORD;
END_VAR
```

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 befehlsspezifische Fehlercode ist in ErrorID enthalten.
ErrorID	DWORD	Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls.

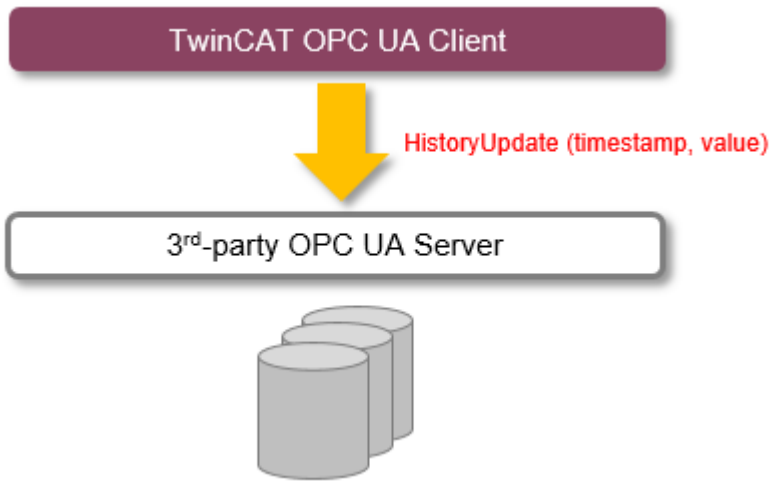
Voraussetzungen

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

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 befehlsspezifische Fehlercode ist in ErrorID enthalten.
ErrorID	DWORD	Enthält den befehlsspezifischen ADS Fehlercode des zuletzt ausgeführten Befehls.

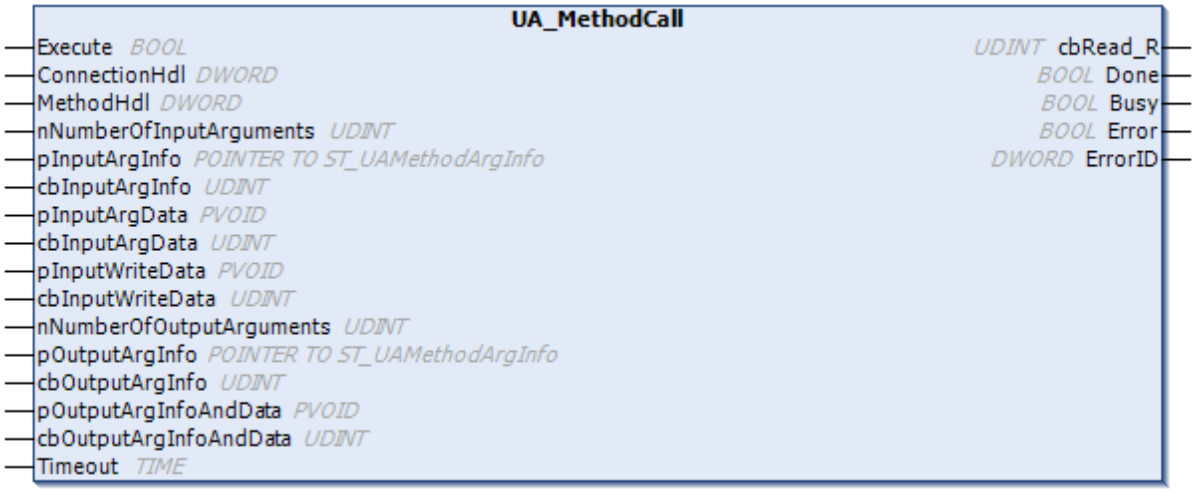
Anzahl der übergebenen Werte

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

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 [▶ 67] und letzterer durch UA_MethodGetHandle [▶ 76] 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 [▶ 61] (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 befehlsspezifische Fehlercode ist in nErrID enthalten.
ErrorID	UDINT	Enthält den befehlsspezifischen 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 [▶ 74] 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 [▶ 61]).
MethodNodeID	ST_UANodeID	Methoden-Knoten-ID der aufzurufenden Methode. Entspricht dem ID-Atttribut im UA-Namensraum. (Typ: UA_Connect [▶ 67]).
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 [▶ 74] 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 befehlsspezifische Fehlercode ist in nErrID enthalten.
ErrorID	UDINT	Enthält den befehlsspezifischen 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

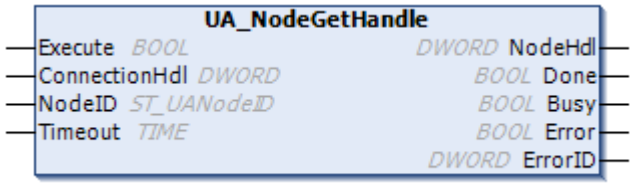
```
VAR_OUTPUT
    Done     : BOOL;
    Busy     : BOOL;
    Error    : BOOL;
    ErrorID  : UDINT;
END_VAR
```

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 befehlsspezifische Fehlercode ist in nErrID enthalten.
ErrorID	UDINT	Enthält den befehlsspezifischen ADS Fehlercode des zuletzt ausgeführten Befehls.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.2.10 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 [▶ 70].
Timeout	TIME	Zeit bis zum Abbruch der Funktion. DEFAULT_ADS_TIMEOUT ist eine globale Konstante, gesetzt auf 5 Sekunden.

Ausgänge

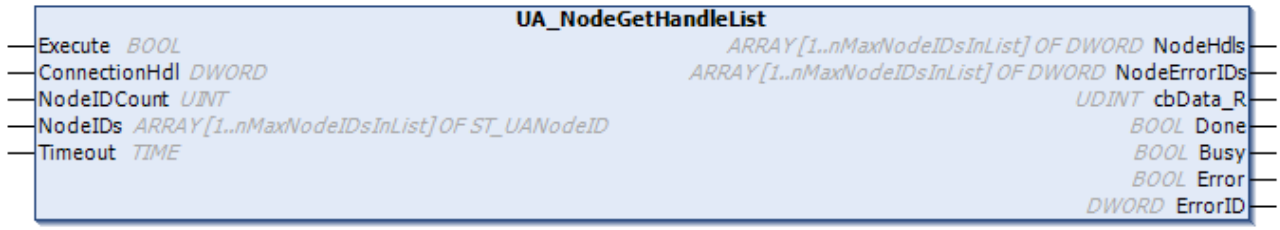
```
VAR_OUTPUT
    NodeHdl     : DWORD;
    Done        : BOOL;
    Busy        : BOOL;
    Error       : BOOL;
    ErrorID     : DWORD;
END_VAR
```

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 befehlsspezifische Fehlercode ist in ErrorID enthalten.
ErrorID	DWORD	Enthält den befehlsspezifischen ADS Fehlercode des zuletzt ausgeführten Befehls.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.2.11 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 [▶ 61] erstellt wurden.
Timeout	TIME	Zeit bis zum Abbruch der Funktion. DEFAULT_ADS_TIMEOUT ist eine globale Konstante, gesetzt auf 5 Sekunden.

Ausgänge

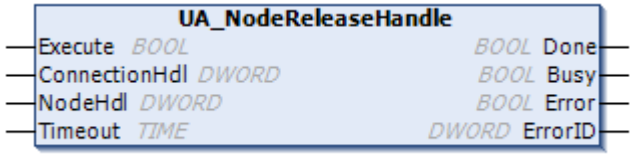
```
VAR_OUTPUT
    NodeHdls     : ARRAY[1..nMaxNodeIDsInList] OF DWORD;
    NodeErrorIDs : ARRAY[1..nMaxNodeIDsInList] OF DWORD;
    cbData_R     : UDINT;
    Done         : BOOL;
    Busy         : BOOL;
    Error        : BOOL;
    ErrorID      : DWORD;
END_VAR
```

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 befehlsspezifische 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

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 befehlsspezifische Fehlercode ist in ErrorID enthalten.
ErrorID	DWORD	Enthält den befehlsspezifischen ADS Fehlercode des zuletzt ausgeführten Befehls.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT 3.1	Win32, Win64, CE-X86, CE-ARM	Tc3_PLCopen_OpcUa

5.2.2.13 UA_NodeReleaseHandleList

Dieser Funktionsbaustein gibt mehrere Knotenhandles frei.

Eingänge

```
VAR_INPUT
    Execute          : BOOL;
    ConnectionHdl    : DWORD;
    NodeHdlCount     : UINT;
    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 befehlsspezifische 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

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.
ConnectionHdl	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 [▶ 62] 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 befehlsspezifische Fehlercode ist in nErrID enthalten.
ErrorID	UDINT	Enthält den befehlsspezifischen 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

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 [▶ 67] ausgegeben wurde.
NodeHdlCount	UINT	Anzahl Knotenhandles, die in der Eingangsvariablen NodeHdls gespeichert sind.
NodeHdls	ARRAY	Array von Knotenhandles, die vorher vom Funktionsbaustein UA_NodeGetHandle [▶ 78] oder UA_NodeGetHandleList [▶ 79] 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 [▶ 62] 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

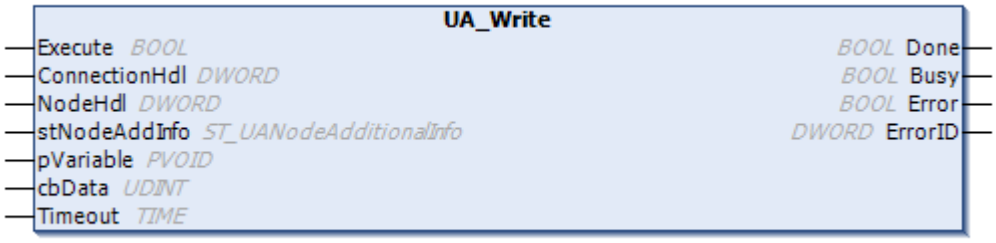
```
VAR_OUTPUT
    Done       : BOOL;
    Busy       : BOOL;
    Error      : BOOL;
    ErrorID    : UDINT;
    cbData_R   : UDINT;
END_VAR
```

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 befehlsspezifische Fehlercode steht in nErrID.
ErrorID	UDINT	Enthält den befehlsspezifischen 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 [▶ 67] ausgegeben wurde.
NodeHdl	DWORD	Knotenhandle, der zuvor vom Funktionsbaustein UA_NodeGetHandle [▶ 78] 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 [▶ 62] 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. DEFAULT_ADS_TIMEOUT ist eine globale Konstante, gesetzt auf 5 Sekunden.

Ausgänge

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

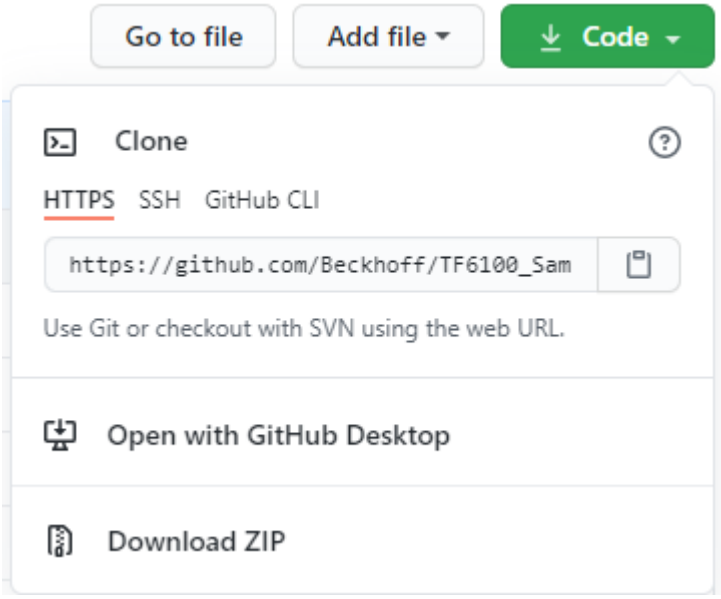
Name	Typ	Beschreibung
Done	BOOL	Schaltet auf TRUE, wenn der Funktionsbaustein erfolgreich ausgeführt wurde.
Busy	BOOL	TRUE, bis der Baustein einen Befehl ausführt hat, maximal für die Dauer des „Timeout“ am Eingang. Solange Busy = TRUE ist, akzeptieren die Eingänge keinen neuen Befehl. Es wird nicht die Zeit der Verbindung sondern die Empfangszeit überwacht.
Error	BOOL	Schaltet auf TRUE, wenn bei der Ausführung eines Befehls ein Fehler auftritt. Der befehlsspezifische Fehlercode ist in ErrorID enthalten.
ErrorID	DWORD	Enthält den befehlsspezifischen 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 clonen oder ein ZIP-File mit dem Sample herunterzuladen.



Es existieren folgende Samples:

Name	TwinCAT-Version	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 Tipps gegeben, aus welchen Gründen diese Fehler auftreten und wie sie behoben werden können.

Verhalten	Abhilfe
Der Versuch, mit Hilfe der PLCopen-Funktionsbausteine, einen StructuredDataType 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.
Bei Ausführung der PLCopen-Funktionsbausteine bleibt der Busy Ausgang auf TRUE, es wird jedoch kein Error ausgegeben und auch der Timeout wird nicht getriggert.	Dies ist ein Indiz für zu wenig ADS-Routerspeicher. Bitte erhöhen Sie Ihren Routerspeicher in den entsprechenden Einstellungen vom TwinCAT-ADS-Router.
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.
Schreibbfehle auf eine Variable werden vom I/O-Client nicht ausgeführt bzw. kommen nicht am Server an.	Bitte überprüfen Sie, ob der Ausgang "Write Enable" am I/O-Client aktiviert wurde.

7.2 Statuscodes

7.2.1 ADS Return Codes

Gruppierung der Fehlercodes:
Globale Fehlercodes: 0x0000 [▶ 88]... (0x9811_0000 ...)
Router Fehlercodes: 0x0500 [▶ 89]... (0x9811_0500 ...)
Allgemeine ADS Fehler: 0x0700 [▶ 90]... (0x9811_0700 ...)
RTime Fehlercodes: 0x1000 [▶ 92]... (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_INVALIDAMSPORT	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_TLSSEND	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-Syncronisation in der SPS geben.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Nicht genügend Speicher.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	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_NOTIFYHNDINVALID	Notification Handle ist ungültig.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification-Client nicht registriert.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	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_SYNCTIMEOUT	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_NOMORESYM	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

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.

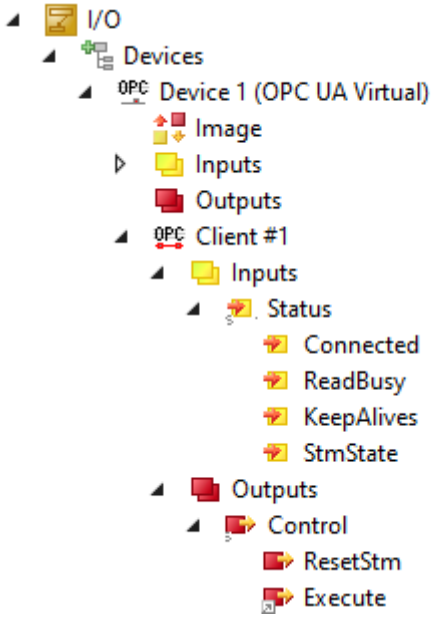


Abb. 1: OPCUAClientModulesStatusCodes

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 Verwenden von Subscriptions (keep alive count if using subscriptions)	3- Verbindungsstatus (&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 [▶ 88]) 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 UaNodeId ungültig
0x E4DD 0115	UAC_E_IDENTIFIER_NOTSUPP	Bezeichnertyp UaNodeId 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.com/ts6100

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