BECKHOFF New Automation Technology

Handbuch | DE

TS6421

TwinCAT 2 | XML Data Server

Supplement | Communication

05.12.2022 | Version: 1.1

Inhaltsverzeichnis

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, für jede Installation und Inbetriebnahme die zu dem betreffenden Zeitpunkt veröffentliche 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 weiter entwickelt.
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 Sicherheitshinweise

Sicherheitsbestimmungen

Beachten Sie die folgenden Sicherheitshinweise und Erklärungen!
Produktspezifische Sicherheitshinweise finden Sie auf den folgenden Seiten oder in den Bereichen Montage, Verdrahtung, Inbetriebnahme usw.

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.

Erklärung der Symbole

In der vorliegenden Dokumentation werden die folgenden Symbole mit einem nebenstehenden Sicherheitshinweis oder Hinweistext verwendet. Die Sicherheitshinweise sind aufmerksam zu lesen und unbedingt zu befolgen!

⚠ GEFAHR

Akute Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht unmittelbare Gefahr für Leben und Gesundheit von Personen!

⚠ WARNUNG

Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht Gefahr für Leben und Gesundheit von Personen!

⚠ VORSICHT

Schädigung von Personen!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, können Personen geschädigt werden!

HINWEIS

Schädigung von Umwelt oder Geräten

Wenn der Hinweis neben diesem Symbol nicht beachtet wird, können Umwelt oder Geräte geschädigt werden.

Tipp oder Fingerzeig

Dieses Symbol kennzeichnet Informationen, die zum besseren Verständnis beitragen.

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.

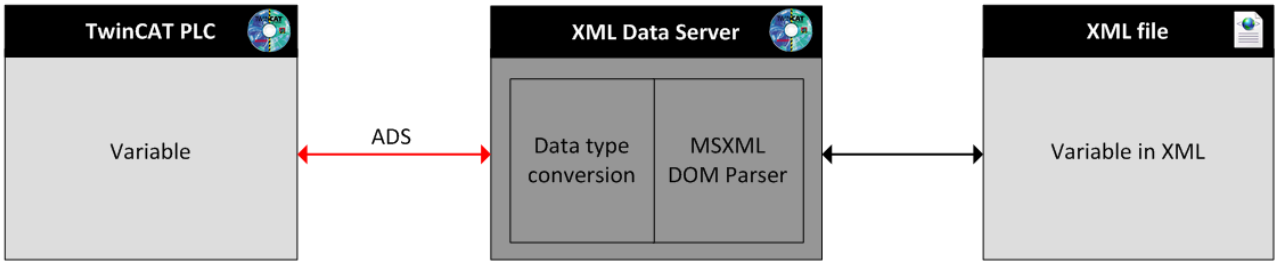
Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem https://www.beckhoff.de/secguide.

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

Der TwinCAT XML Data Server ermöglicht es, Variablen in der TwinCAT SPS mit Daten, die einer XML-Datei gespeichert sind, zu initialisieren. Außerdem ist es möglich, SPS Variablen formatiert in einer XML-Datei zu speichern. Die Struktur einer Variablen im XML-Dokument entspricht der Struktur der Variablen in der SPS. So ist es möglich, auch direkt auf einzelne Unterelemente einer Variablen zuzugreifen. Dabei werden nur die Unterelemente (Elemente einer Struktur oder Elemente eines Arrays) übertragen, die auch in der XML-Datei definiert sind. Beim Schreiben der SPS-Variablen gibt es jedoch auch die Möglichkeit, fehlende Elemente hinzuzufügen.



3 Systemvoraussetzungen

Technische Daten	TS6421 XML Data Server
Zielsystem	Windows NT/2000/XP/Vista/7 PC (x86-kompatibel)
Min. TwinCAT-Version	2.10.0
Min. TwinCAT-Level	TwinCAT PLC

4 Installation

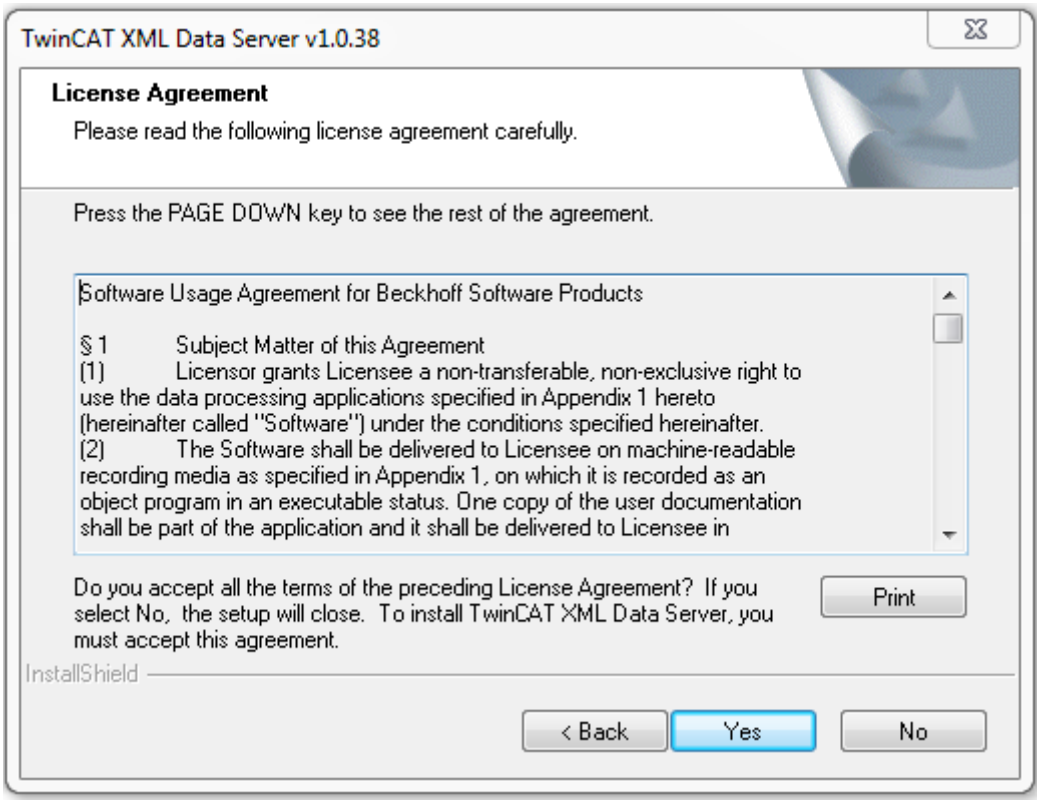
Dieser Teil der Dokumentation führt den Benutzer Schritt-für-Schritt durch den Installationsvorgang des TwinCAT XML Data Server Supplements für Windows-basierte Betriebssysteme. Es wird hierbei auf die folgenden Themen eingegangen:

- Starten der Installation
- Nach der Installation

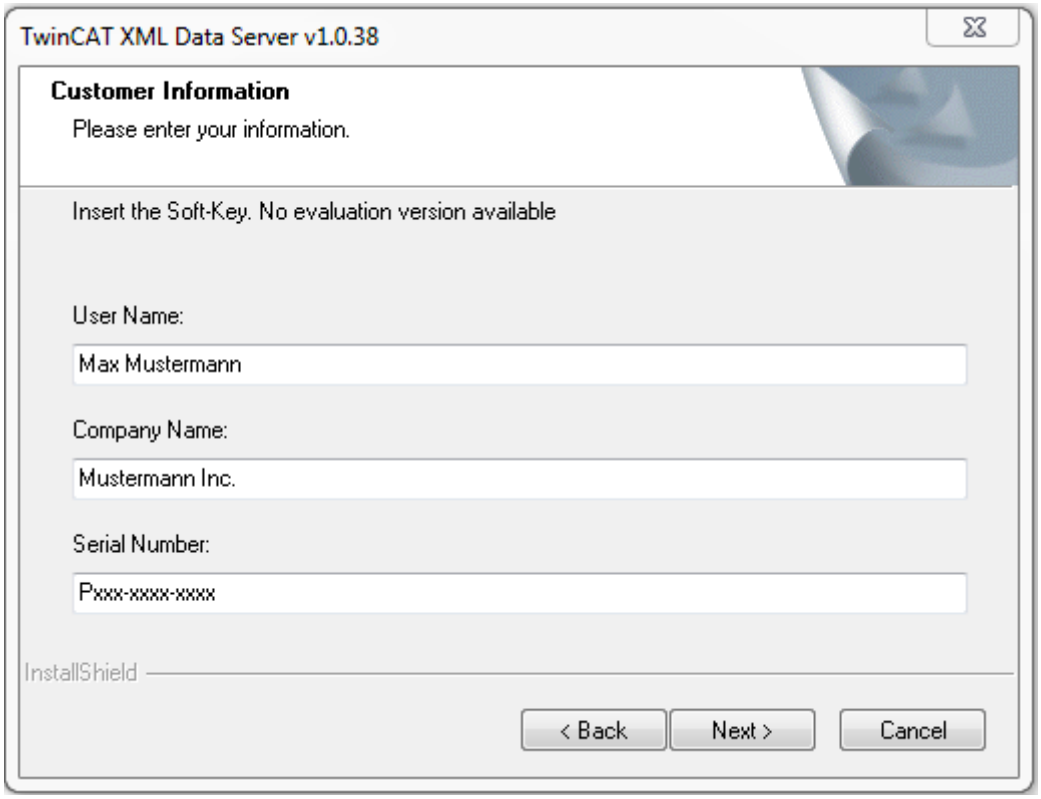
Starten der Installation

Um das Supplement zu installieren, führen Sie bitte die folgenden Schritte durch:

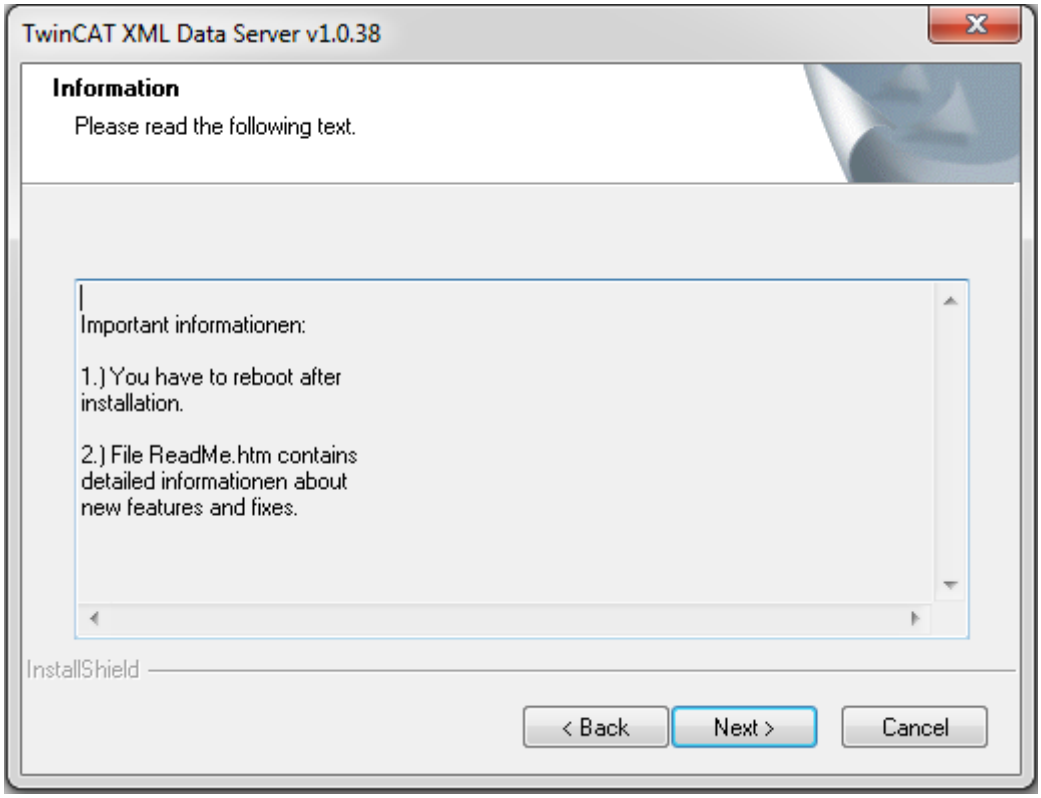
- Führen Sie einen Doppelklick auf die heruntergeladene Datei**"TcXmlDataSrv.exe"**aus.
  Starten Sie die Installation unter Windows mit der Option "Als Administrator ausführen", indem Sie die Setup-Datei mit der rechten Maustaste anklicken und die entsprechende Option im Kontextmenü auswählen.
- Wählen Sie eine **Sprache** aus, in der Sie die Software installieren möchten.
- Klicken Sie auf "Next" und akzeptieren Sie dann die **Endbenutzervereinbarung.**



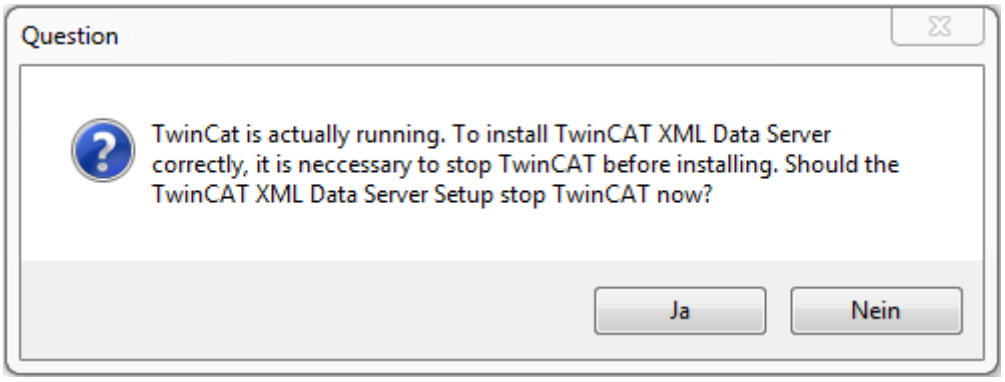
- Geben Sie Ihre **Benutzerdaten** ein:



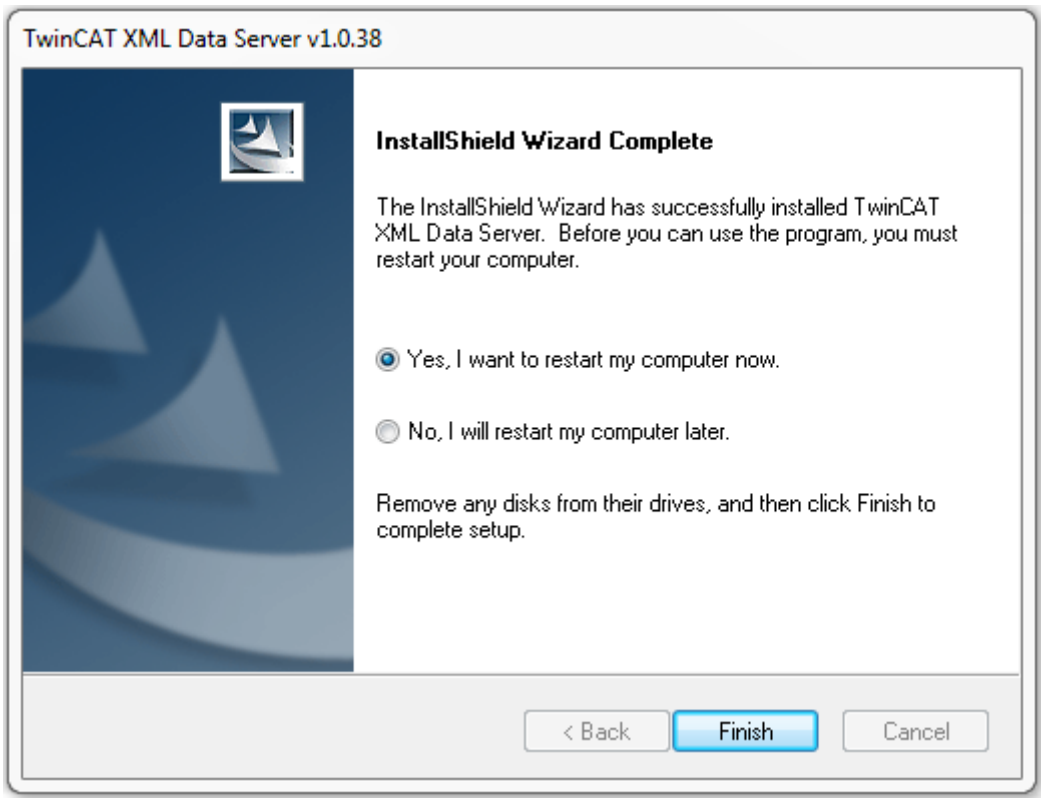
- Klicken Sie auf **"Next"** um die Installation zu starten.



- Das TwinCAT System muss gestoppt werden, bevor die Installation ausgeführt wird. Wählen Sie "Yes" um den TwinCAT System Service zu stoppen.



- Bestätigen Sie die folgende Meldung, sobald der TwinCAT Service gestoppt ist.
- Wählen Sie "**Finish**" um das Setup zu beenden.



Nach der Installation

Das Supplement-Produkt "TwinCAT XML Data Server" wird automatisch durch das Setup konfiguriert.

5 SPS-API

Die SPS-Bibliothek TcXmlDataSrv.Lib wird mit dem TwinCAT Xml Data Server mitgeliefert und während der Installation in den ...\TwinCAT\PLC\Lib-Ordner kopiert.

Es gibt jeweils zwei Funktionsbausteine zum Auslesen von Variablen aus der XML-Datei:

- FB_XmlSrvRead
- FB_XmlSrvReadByName

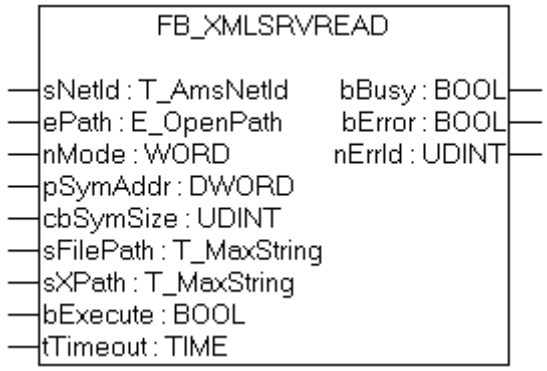
und zwei Funktionsbausteine zum Schreiben von SPS-Variablen in die XML-Datei:

- FB_XmlSrvWrite
- FB_XmlSrvWriteByName

Die erste Variante (FB_XMLSrvRead, FB_XMLSrvWrite) verwendet die Adresse und die Größe der Variable in der SPS, um die Variable zu spezifizieren. Die zweite Variante (FB_XMLSrvReadByName, FB_XMLSrvWriteByName) verwendet den Symbolnamen, um die Variable zu spezifizieren. Die erste Variante ist perfomanter, wird aber erst ab Build 1235 von TwinCAT Version 2.10 unterstützt. Für ältere Versionen von TwinCAT können nur die Funktionsbausteine FB_XMLSrvReadByName und FB_XMLSrvWriteByName verwendet werden. Zusätzlich muss der Pfad der XML-Datei und der Standort der Variablen im XML-Dokument im XPath-Format den Funktionsbausteinen als Eingangs-Parameter übergeben werden.

5.1 Funktionsbausteine

5.1.1 FB_XmlSrvRead



Mit dem Funktionsbaustein FB_XmlSrvRead kann eine SPS-Variable mit Daten aus einer XML-Datei initialisiert werden. Die Eingangsvariable sXPath muss dabei auf einen gültigen Knoten in der mit sFilePath angegebenen XML-Datei zeigen. Das Symbol, das initialisiert werden soll, wird anhand der übergebenen Symbol-Adresse und Symbol-Größe eindeutig identifiziert.

VAR_INPUT

```
VAR_INPUT
    sNetId      : T_AmsNetId;
    ePath       : E_OpenPath := PATH_GENERIC;
    nMode       : WORD := XMLSRV_SKIPMISSING;
    pSymAddr    : DWORD;
    cbSymSize   : UDINT;
    sFilePath   : T_MaxString;
    sXPath      : T_MaxString;
    bExecute    : BOOL;
    tTimeout    : TIME := T#60s;
END_VAR
```

sNetId: String mit der Netzwerkadresse des TwinCAT XML-Data Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

ePath: Über diesen Eingang kann ein TwinCAT - Systempfad auf dem Zielgerät zum Öffnen der Datei angewählt werden.

nMode: Über diesen Eingang ist das Verhalten beeinflussbar, wie die XML-Datei ausgewertet wird. Für den Befehl XmlSrvRead wird zurzeit nur der Modus XMLSRV_SKIPMISSING unterstützt.

pSymAddr: Adresse der SPS Variable, die mit den Daten aus der XML-Datei beschrieben werden soll.

cbSymSize: Größe der SPS Variable, die mit den Daten aus der XML-Datei beschrieben werden soll.

sFilePath: Enthält den Pfad- und Dateinamen der zu öffnenden Datei.
Der Pfad kann nur auf das lokale Filesystem des Rechners zeigen. Das bedeutet, Netzwerkpfade können hier nicht angegeben werden.

sXPath: Enthält die Adresse des Tags im XML-Dokument, aus der die Daten geschrieben werden soll. Die Adresse muss eine gültige XPath-Anweisung sein. Der Name des Tags muss dabei nicht dem Namen des Symbols entsprechen.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy   : BOOL;
    bError  : BOOL;
    nErrId  : UDINT;
END_VAR
```

bBusy: Dieser Ausgang wird bei der Aktivierung des Funktionsbausteins gesetzt und bleibt solange gesetzt, bis eine Rückmeldung erfolgt.

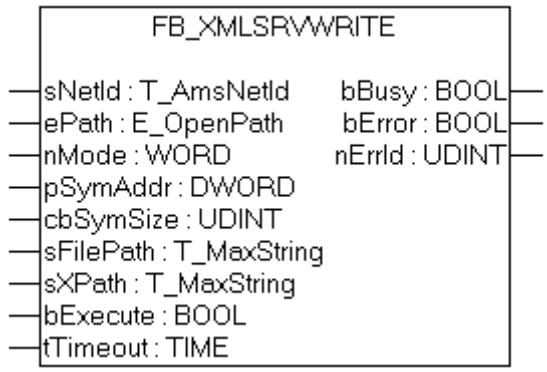
bError: Dieser Ausgang wird, nachdem der bBusy-Ausgang zurückgesetzt wurde, gesetzt, sollte ein Fehler bei der Übertragung des Kommandos erfolgen.

nErrId: Liefert bei einem gesetzten bError-Ausgang die TwinCAT Xml Data Server Fehlernummer [▶ 29].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 ab Build 1235	PC oder CX (x86)	TcXmlDataSrv.Lib

5.1.2 FB_XmlSrvWrite



Mit dem Funktionsbaustein FB_XmlSrvWrite kann der Wert einer SPS-Variablen in eine XML-Datei geschrieben werden. Die Eingangsvariable sXPath muss dabei auf einen gültigen Knoten in der mit sFilePath angegebenen XML-Datei zeigen. Das Symbol, das geschrieben werden soll, wird anhand der übergebenen Symbol-Adresse und Symbol-Größe eindeutig identifiziert.

VAR_INPUT

```
VAR_INPUT
    sNetId      : T_AmsNetId;
    ePath       : E_OpenPath := PATH_GENERIC;
    nMode       : WORD := XMLSRV_SKIPMISSING;
    pSymAddr    : DWORD;
    cbSymSize   : UDINT;
    sFilePath   : T_MaxString;
    sXPath      : T_MaxString;
    bExecute    : BOOL;
    tTimeout    : TIME := T#60s;
END_VAR
```

sNetId: String mit der Netzwerkadresse des TwinCAT XML Data Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

ePath: Über diesen Eingang kann ein TwinCAT - Systempfad auf dem Zielgerät zum Öffnen der Datei angewählt werden.

nMode: Über diesen Eingang ist das Verhalten beeinflussbar, mit welchen Daten die XML-Datei beschrieben wird. Für den Befehl XmlSrvWrite gibt es den Modus XMLSRV_SKIPMISSING und XMLSRV_ADDMISSING. Im Modus XMLSRV_SKIPMISSING werden nur die Sub-Elemente eines SPS-Symbols in die XML-Datei geschrieben, die bereits vorher in der XML-Datei existierten. Im Modus XMLSRV_ADDMISSING werden in der XML-Datei fehlende Subelement der XML-Datei hinzugefügt.

pSymAddr: Adresse der SPS Variable, die in die XML-Datei geschrieben werden soll.

cbSymSize: Größe der SPS Variable, die in die XML-Datei geschrieben werden soll.

sFilePath: Enthält den Pfad- und Dateinamen der zu öffnenden Datei.
Der Pfad kann nur auf das lokale Filesystem des Rechners zeigen. Das bedeutet, Netzwerkpfade können hier nicht angegeben werden.

sXPath: Enthält die Adresse des Tags im XML-Dokument, aus der die Daten geschrieben werden soll. Die Adresse muss eine gültige XPath-Anweisung sein. Der Name des Tags muss dabei nicht dem Namen des Symbols entsprechen.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy   : BOOL;
    bError  : BOOL;
    nErrId  : UDINT;
END_VAR
```

bBusy: Dieser Ausgang wird bei der Aktivierung des Funktionsbausteins gesetzt und bleibt solange gesetzt, bis eine Rückmeldung erfolgt.

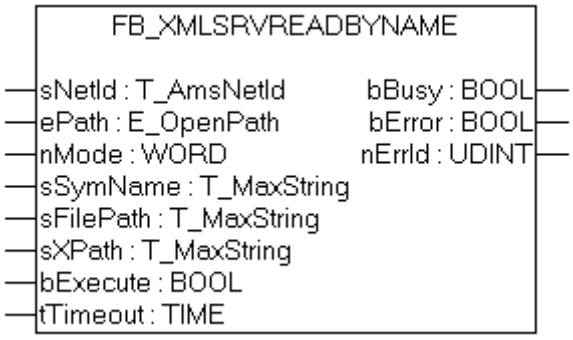
bError: Dieser Ausgang wird, nachdem der bBusy-Ausgang zurückgesetzt wurde, gesetzt, sollte ein Fehler bei der Übertragung des Kommandos erfolgen.

nErrId: Liefert bei einem gesetzten bError-Ausgang die TwinCAT Xml Data Server Fehlernummer [▶ 29].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 ab Build 1235	PC oder CX (x86)	TcXmlDataSrv.Lib

5.1.3 FB_XmlSrvReadByName



Mit dem Funktionsbaustein FB_XmlSrvReadByName kann eine Sps-Variable mit Daten aus einer XML-Datei initialisiert werden. Die Eingangsvariable sXPath muss dabei auf einen gültigen Knoten in der mit sFilePath angegebenen XML-Datei zeigen. Das Symbol, das initialisiert werden soll, wird anhand des Symbolnamens eindeutig identifiziert.

VAR_INPUT

```
VAR_INPUT
    sNetId      : T_AmsNetId;
    ePath       : E_OpenPath := PATH_GENERIC;
    nMode       : WORD := XMLSRV_SKIPMISSING;
    sSymName    : T_MaxString;
    sFilePath   : T_MaxString;
    sXPath      : T_MaxString;
    bExecute    : BOOL;
    tTimeout    : TIME := T#60s;
END_VAR
```

sNetId: String mit der Netzwerkadresse des TwinCAT XML Data Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

ePath: Über diesen Eingang kann ein TwinCAT - Systempfad auf dem Zielgerät zum Öffnen der Datei angewählt werden.

nMode: Über diesen Eingang ist das Verhalten beeinflussbar, wie die XML-Datei ausgewertet wird. Für den Befehl XmlSrvReadByName wird zurzeit nur der Modus XMLSRV_SKIPMISSING unterstützt.

sSymName: Name des SPS-Symbols, das mit den Daten aus der XML-Datei beschrieben werden soll.

sFilePath: Enthält den Pfad- und Dateinamen der zu öffnenden Datei.
Der Pfad kann nur auf das lokale Filesystem des Rechners zeigen. Das bedeutet, Netzwerkpfade können hier nicht angegeben werden.

sXPath: Enthält die Adresse des Tags im XML-Dokument, aus der die Daten geschrieben werden soll. Die Adresse muss eine gültige XPath-Anweisung sein. Der Name des Tags muss dabei nicht dem Namen des Symbols entsprechen.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy   : BOOL;
    bError  : BOOL;
    nErrId  : UDINT;
END_VAR
```

bBusy: Dieser Ausgang wird bei der Aktivierung des Funktionsbausteins gesetzt und bleibt solange gesetzt, bis eine Rückmeldung erfolgt.

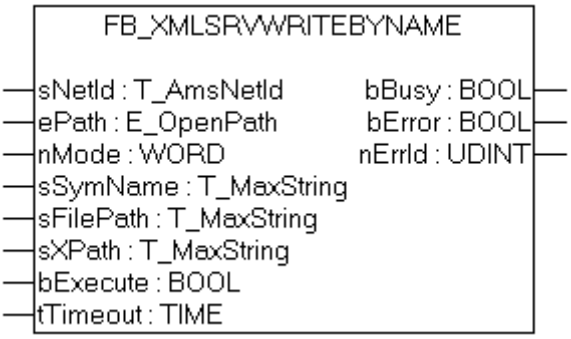
bError: Dieser Ausgang wird, nachdem der bBusy-Ausgang zurückgesetzt wurde, gesetzt, sollte ein Fehler bei der Übertragung des Kommandos erfolgen.

nErrId: Liefert bei einem gesetzten bError-Ausgang die TwinCAT Xml Data Server Fehlernummer [▶ 29].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0	PC oder CX (x86)	TcXmlDataSrv.Lib

5.1.4 FB_XmlSrvWriteByName



Mit dem Funktionsbaustein FB_XmlSrvWriteByName kann der Wert einer SPS-Variablen in eine XML-Datei geschrieben werden. Die Eingangsvariable sXPath muss dabei auf einen gültigen Knoten in der mit sFilePath angegebenen XML-Datei zeigen. Das Symbol, das geschrieben werden soll, wird anhand des Symbol-Namens eindeutig identifiziert.

VAR_INPUT

```
VAR_INPUT
    sNetId      : T_AmsNetId;
    ePath       : E_OpenPath := PATH_GENERIC;
    nMode       : WORD := XMLSRV_SKIPMISSING;
    sSymName    : T_MaxString;
    sFilePath   : T_MaxString;
    sXPath      : T_MaxString;
    bExecute    : BOOL;
    tTimeout    : TIME := T#60s;
END_VAR
```

sNetId: String mit der Netzwerkadresse des TwinCAT XML Data Servers. Für den lokalen Rechner (default) kann auch ein Leerstring angegeben werden.

ePath: Über diesen Eingang kann ein TwinCAT - Systempfad auf dem Zielgerät zum Öffnen der Datei angewählt werden.

nMode: Über diesen Eingang ist das Verhalten beeinflussbar, mit welchen Daten die XML-Datei beschrieben wird. Für den Befehl XmlSrvWriteByte gibt es den Modus XMLSRV_SKIPMISSING und XMLSRV_ADDMISSING. Im Modus XMLSRV_SKIPMISSING werden nur die Sub-Elemente eines SPS-Symbols in die XML-Datei geschrieben, die bereits vorher in der XML-Datei existierten. Im Modus XMLSRV_ADDMISSING werden in der XML-Datei fehlende Subelement der XML-Datei hinzugefügt.

sSymName: Name des SPS-Symbols, das in die XML-Datei geschrieben werden soll.

sFilePath: Enthält den Pfad- und Dateinamen der zu öffnenden Datei.
Der Pfad kann nur auf das lokale Filesystem des Rechners zeigen. Das bedeutet, Netzwerkpfade können hier nicht angegeben werden.

sXPath: Enthält die Adresse des Tags im XML-Dokument, aus der die Daten geschrieben werden soll. Die Adresse muss eine gültige XPath-Anweisung sein. Der Name des Tags muss dabei nicht dem Namen des Symbols entsprechen.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy   : BOOL;
    bError  : BOOL;
    nErrId  : UDINT;
END_VAR
```

bBusy: Dieser Ausgang wird bei der Aktivierung des Funktionsbausteins gesetzt und bleibt solange gesetzt, bis eine Rückmeldung erfolgt.

bError: Dieser Ausgang wird, nachdem der bBusy-Ausgang zurückgesetzt wurde, gesetzt, sollte ein Fehler bei der Übertragung des Kommandos erfolgen.

nErrId: Liefert bei einem gesetzten bError-Ausgang die TwinCAT Xml Data Server Fehlernummer [▶ 29].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0	PC oder CX (x86)	TcXmlDataSrv.Lib

5.2 Funktionen

5.2.1 F_GetVersionTcXmlDataSrv

Mit dieser Funktion können Versionsinformationen der SPS-Bibliothek ausgelesen werden.

FUNCTION F_GetVersionTcXmlDataSrv : UINT

```
VAR_INPUT
    nVersionElement : INT;
END_VAR
```

nVersionElement : Versionselement, das gelesen werden soll. Mögliche Parameter:

- 1 : major number;
- 2 : minor number;
- 3 : revision number;

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0	PC oder CX (x86)	TcXmlDataSrv.Lib

5.3 Globale Konstanten

5.3.1 Globale Variablen

```
VAR_GLOBAL CONSTANT
    XMLSRV_AMSPORT :UINT :=10600;

    XMLSRV_IGR_CLOSE :UDINT := 121;
    XMLSRV_IGR_READ :UDINT := 122;
    XMLSRV_IGR_WRITE :UDINT := 123;
    XMLSRV_IGR_OPENREAD :UDINT := 124;
    XMLSRV_IGR_OPENWRITE :UDINT := 125;

    XMLSRV_SKIPMISSING :WORD := 0;
    XMLSRV_ADDMISSING :WORD := 1; (*for write commands*)

    XMLSRV_MAX_FRAGSIZE :UDINT := 16#40000;

    XMLSRVERROR_INTERNAL :UDINT:= 16#8000;
    XMLSRVERROR_NOTFOUND :UDINT:= 16#8001;
    XMLSRVERROR_PARSERERROR :UDINT:= 16#8002;
    XMLSRVERROR_INCOMPATIBLE :UDINT:= 16#8003;
    XMLSRVERROR_NOMEMORY :UDINT:= 16#8004;
    XMLSRVERROR_ADDNODE :UDINT:= 16#8005;
    XMLSRVERROR_INVALIDXPATH :UDINT:= 16#8006;
END_VAR
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0	PC oder CX (x86)	TcXmlDataSrv.Lib

6 Beispiele

Auf den folgenden Seiten finden Sie Beispiele (Samples), die den Umgang mit dem TwinCAT XML Data Server zeigen. Die Beispiele sind fortlaufend nummeriert und lassen sich als https://infosys.beckhoff.com/content/1031/tcxmldatasrv/Resources/11416819467.zip.

Getting Started [▶ 20]

Hier wird der generelle Umgang mit dem TwinCAT XML Data Server sowie das Arbeiten mit Strukturen und Arrays und der Zugriff auf einzelne Knotenpunkte behandelt.

Die Funktionsbausteine [▶ 22]

In vier Beispielen wird der Einsatz und die Konfiguration der Funktionsbausteine der SPS-Bibiothek *TcXmlDataSrv.Lib* vorgestellt. (Beispiel 1-4)

Weiterführende Beispiele [▶ 24]

Dieses Dokument enthält weiterführende Beispiele, die die einmalige Initialisierung bei SPS-Start oder das zyklische Schreiben darstellen. (Beispiel 5-6)

Produktionsbeispiel [▶ 26]

Das Produktionsbeispiel zeigt die Abarbeitung eines kleinen Produktionsauftrags. Es verwendet FB_XmlSrvReadByName und FB_XmlSrvWriteByName. (Beispiel 7)

6.1 Getting Started

Im Folgenden wird die grundlegende Arbeitsweise mit dem TwinCAT XML Data Server anhand kleiner Beispiele erklärt.

Primitive Datentypen

Folgende Primitive Datentypen werden unterstützt:

Datentyp	Beispiel in der SPS	Beispiel in XML

UDINT, DINT, UINT, INT, USINT, SINT, DWORD, WORD, BYTE	`value1 : DINT := -1;`
	`value2 : UDINT := 65535;`
		`<dataentry>`
		`<MAIN.value1>-1</MAIN.value1>`
		`<MAIN.value2>65535</MAIN.value2>`
		`</dataentry>`

LREAL, REAL	`value1 : LREAL = 1.2;`	`<dataentry>`
		`<MAIN.value1>1.2</MAIN.value1>`
		`</dataentry>`

STRING	`str1 : STRING = 'hallo';`	`<dataentry>`
		`<MAIN.str1>hallo</MAIN.str1>`
		`</dataentry>`

TIME, DATE, TOD,DT	`date1:DATE :=D#2005-05-04;`(* Time-Typen werden in der XML-Datei als DWORD gespeichert*)	`<dataentry>`
		`<MAIN.date1>1115164800</MAIN.date1>`
		`</dataentry>`

BOOL	`bool1:BOOL := TRUE;` `bool2:BOOL := FALSE;`	`<dataentry>` `<MAIN.bool1>true</MAIN.bool1>`

Datentyp	Beispiel in der SPS	Beispiel in XML
		<MAIN.bool2>false</MAIN.bool2></dataentry>

Bsp.: In der SPS ist die globale Variable `.Var1` vom Typen `DINT` definiert. Die XML-Datei ist dann wie folgt definiert:

```
<dataentry>
    <Var1>10</Var1>
</dataentry>
```

Die Inputvariablen des Funktionsbausteines FB_XmlSrvRead müssen dann wie folgt gesetzt werden:

```
fbRead.pSymAddr      := ADR(value1);

fbRead.cbSymSize     := SIZEOF(value1);

fbRead.sFilePath     := 'C:\Test.xml';

fbRead.sXPath        :='/dataentry/Var1';
```

Das Root-Element in der XML-Datei und der Name der Variablen sind in der XML-Datei frei wählbar. Es muss dann lediglich der neue Pfad in der Eingangsvariable sXPath geändert werden. In einer XML-Datei können auch mehrere Variablen definiert sein:

```
<dataentry>
    <Var1>10</Var1>
    <Var2>100</Var2>
    <Var3>
    <a>100</a>
    <b>10</b>
    </Var3>
</dataentry>
```

Um auf das Symbol `.Var3.a` zuzugreifen müsste `sXpath := '/dataentry/Var3.a'` gesetzt werden.

Strukturen

Strukturen in der XML-Datei haben den gleichen hierarchischen Aufbau wie in der SPS. Es gibt jedoch die Möglichkeit, einzelne Unterelemente in der XML-Datei zu überspringen. Die einzelnen Unterelmente der Struktur müssen die gleichen Namen besitzen wie in der SPS, ansonsten werden sie übersprungen. Wenn Unterelemente in der XML-Datei nicht in den korrekten Datentypen umgewandelt werden können, werden sie auch übersprungen.

Bsp: In der SPS ist die globale Variable `.Var2` vom Typen `ST_MYSTRUCT` definiert:

```
TYPE ST_MYSTRUCT:
STRUCT
    a:  UINT;
    b:  DINT;
    c:  LREAL;
    d:  STRING;
END_STRUCT
END_TYPE
```

Die XML-Datei könnte dann wie folgt aussehen:

```
<variables>
    <Var1>10</Var1>
    <Var2>              <!-- sXPath := '\variables\Var2' --!>
    <a>100</a>
    <b>-10</b>
    <c>1.2</c>
    <d>Hallo</d>
    </Var2>
</variables>
```

In diesem Fall sind alle Unterelemente vollständig und korrekt definiert, so dass die Variable vollständig initialisiert wird. Im folgenden Beispiel hingegen wird nur das Unterelement c serialisiert:

```
 <variables>
     <Var1>10</Var1>
     <Variable2>           <!-- sXPath := '\variables\Variable2' --!>
```

```
     <Info>dies ist ein Test</Info>
     <a>-100</a>
     <c>1.2</c>
    </Variable2>
</variables>
```

Unterelement a kann nicht konvertiert werden, weil es negativ ist und ein UINT gefordert wird. Das Unterelement b fehlt komplett. Das Tag <Info> wird übersprungen, da es in der SPS-Datei nicht definiert ist.

Arrays

Um den Index von Arrays anzugeben, muss bei den einzelnen Array-Elementen das Attribut "Index" verwendet werden. Es können auch einzelne Array-Elemente wegelassen werden. Diese werden dann einfach übersprungen.

Bsp.: In der SPS ist ein Variable `.array1` vom `Typen ARRAY[1..4] OF DINT` definiert. Die XML sieht dann wie folgt aus:

```
<dataentry>
    <array1 index="1">10</array1>
    <array1 index="2">10</array1>
    <array1 index="3">10</array1>
    <array1 index="4">10</array1>
</dataentry>
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0	PC oder CX (x86)	TcXmlDataSrv.Lib

6.2 Die Funktionsbausteine

Die folgenden Beispiele zeigen den Umgang mit den Funktionsbausteinen der tcXmlDataSrv-Bibliothek. Das **SPS-Projekt**, das die Beispiele enthält, können Sie https://infosys.beckhoff.com/content/1031/tcxmldatasrv/Resources/11416819467.zip

Alle Beispiele arbeiten mit der Struktur ST_MYSTRUCT, die wiederum die Struktur ST_INNTERSTRUCT enthält. Die beiden Strukturen sind im Folgenden dargestellt:

Die Strukturen

```
TYPE ST_MYSTRUCT:
STRUCT
   fReal      : REAL;
   bBool      : ARRAY [0..2] OF BOOL;
   stInner    : ST_INNTERSTRUCT;
END_STRUC
TEND_TYPE
```

```
TYPE ST_INNTERSTRUCT:
STRUCT
  nInteger    : INT;
  sString     : STRING;
END_STRUCT
END_TYPE
```

Sample 1: Schreibvorgang mit FB_XmlSrvWrite

Im ersten Schritt soll die Struktur ST_MyStruct in eine XML-Datei geschrieben werden. Der Modus wird auf XMLSRV_ADMISSING gesetzt, sodass die XML-Datei automatisch erstellt und die Struktur darin angelegt wird. Ordner werden nicht automatisch angelegt!Dieses Vorgehen empfiehlt bei größeren Strukturen auch dann, wenn die Datei später nur ausgelesen werden soll. So muss die XML-Datei nicht manuell angelegt werden, Fehler werden vermieden.

```
(* Sample1 creates an XML-file under the path C:\Test.xml and writes value1 to it.
   FUNCTIONBLOCK: FB_XmlSrvWrite *)
PROGRAM Sample1
VAR
   value1          : ST_MyStruct;
```

```
   fbXmlSrvWrite   : FB_XmlSrvWrite;
   bExecute        : BOOL;
   sFilePath       : T_MaxString := 'C:\Test.xml';  (* CE: '\Hard Disk\Test.xml' *)
   sXPath          : T_MaxString := '/dataentry/MAIN.value1';
END_VAR
```

```
fbXmlSrvWrite(
   nMode       := XMLSRV_ADDMISSING,
   pSymAddr    := ADR(value1),
   cbSymSize   := SIZEOF(value1),
   sFilePath   := sFilePath,
   sXPath      := sXPath,
   bExecute    := bExecute
);
bExecute:= TRUE;
```

Sample 2: Schreibvorgang mit FB_XmlSrvWriteByName

Sample 2 führt zum gleichen Ergebnis wie Sample 1, verwendet aber den FB_XmlSrvWriteByName-Baustein. Sample 1 ist allerdings performanter.

```
(* Sample2 creates an XML-file under the path C:\Test.xml and writes value1 to it.
   FUNCTIONBLOCK: FB_XmlSrvWriteByName *)
PROGRAM Sample2
VAR
   value1          : ST_MyStruct;
   fbXmlSrvWrite   : FB_XmlSrvWriteByName;
   bExecute        : BOOL;
   sSymName        : T_MaxString := 'Sample2.value1';
   sFilePath       : T_MaxString := 'C:\Test.xml';  (* CE: '\Hard Disk\Test.xml' *)
   sXPath          : T_MaxString := '/dataentry/MAIN.value1';
END_VAR
```

```
fbXmlSrvWrite(
   nMode       := XMLSRV_ADDMISSING,
   sSymName    := sSymName,
   sFilePath   := sFilePath,
   sXPath      := sXPath,
   bExecute    := bExecute
);
bExecute:= TRUE;
```

XML-Datei

Bei beiden Beispielen wird die folgende XML-Datei *'Test.xml'* unter *C:* erstellt. Damit die XML-Datei bei beiden Varianten den gleichen Inhalt enthält, wird als sXPath der gleiche Pfad *'/dataentry/MAIN.value1'* gewählt, obwohl value1 nicht in der Main, sondern direkt in den entsprechenden Programmen liegt. sSymName (Sample 2) gibt hingegen den Ort der Variablen in TwinCAT an: *'Sample2.value1'!*

```
<dataentry>
    <MAIN.value1>
        <fReal>0</fReal>
        <bBool index="0">false</bBool>
        <bBool index="1">false</bBool>
        <bBool index="2">false</bBool>
        <stInner>
            <nInteger>0</nInteger>
            <sString></sString>
        </stInner>
    </MAIN.value1>
</dataentry>
```

Sample 3: Lesevorgang mit FB_XmlSrvRead

Im Folgenden soll die Struktur aus der in Sample 1 bzw. Sample 2 erstellten XML-Datei wieder eingelesen werden. Sample 3 nutzt hierzu den FB_XmlSrvRead-Baustein.

```
(* Sample3 reads an XML-file (C:\Test.xml) FUNCTIONBLOCK: FB_XmlSrvRead *)
PROGRAM Sample3
VAR
   value1          : ST_MyStruct;
   fbXmlSrvRead    : FB_XmlSrvRead;
   bExecute        : BOOL;
   sFilePath       : T_MaxString := 'C:\Test.xml';  (* CE: '\Hard Disk\Test.xml' *)
   sXPath          : T_MaxString := '/dataentry/MAIN.value1';
END_VAR
```

```
fbXmlSrvRead(
   pSymAddr       := ADR(value1),
   cbSymSize      := SIZEOF(value1),
   sFilePath      := sFilePath,
   sXPath         := sXPath,
   bExecute       := bExecute
);
bExecute:= TRUE;
```

Sample 4: Lesevorgang mit FB_XmlSrvReadByName

Sample 4 zeigt den Lesevorgang unter Verwendung des FB_XmlSrvReadByName-Bausteins.

```
(* Sample4 reads an XML-file (C:\Test.xml) FUNCTIONBLOCK: FB_XmlSrvReadByName *)
PROGRAM Sample4
VAR
   value1         : ST_MyStruct;
   fbXmlSrvRead   : FB_XmlSrvReadByName;
   bExecute       : BOOL;
   sSymName       : T_MaxString := 'Sample4.value1';
   sFilePath      : T_MaxString := 'C:\Test.xml';  (* CE: '\Hard Disk\Test.xml' *)
   sXPath         : T_MaxString := '/dataentry/MAIN.value1';
END_VAR
```

```
fbXmlSrvRead(
   sSymName       := sSymName,
   sFilePath      := sFilePath,
   sXPath         := sXPath,
   bExecute       := bExecute
);
bExecute:= TRUE;
```

Bei Sample 4 ist erneut (wie bei Sample 2) zu beachten, dass sich sSymName und sXPath unterscheiden: sXPath gibt den Pfad innerhalb der XML-Datei an. Dieser wurde in Sample 1 bzw. Sample 2 festgelegt. sSymName hingegen gibt den Symbolnamen der TwinCAT-Variablen an und lautet deshalb *'Sample4.value1'*.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0	PC oder CX (x86)	TcXmlDataSrv.Lib

6.3 Weiterführende Beispiele

Die folgenden Beispiele zeigen verschiedene Anwendungsarten des TwinCAT XML Data Servers. Das **SPS-Projekt**, das die Beispiele enthält, können Sie https://infosys.beckhoff.com/content/1031/tcxmldatasrv/Resources/11416819467.zip

Beispiel 5 zeigt eine einmalige Initialisierung bei Programmstart, Beispiel 6 zyklische und ereignisgesteuerte Schreibvorgänge. Beide Beispiele arbeiten mit der schon aus den Beispielen 1-4 bekannten Struktur ST_MYSTRUCT, die wiederum die Struktur ST_INNTERSTRUCT enthält. Die beiden Strukturen sind im Folgenden dargestellt:

Die Strukturen

```
TYPE ST_MYSTRUCT:
STRUCT
   fReal       : REAL;
   bBool       : ARRAY [0..2] OF BOOL;
   stInner     : ST_INNTERSTRUCT;
END_STRUCT
END_TYPE
```

```
TYPE ST_INNTERSTRUCT:
STRUCT
  nInteger     : INT;
  sString      : STRING;
END_STRUCT
END_TYPE
```

Sample 5: Einmalige Initalisierung bei Programmstart

Sample 5 zeigt die einmalige Initialisierung von value1 bei Programmstart. Zum Einsatz kommt der Funktionsbaustein FB_XmlSrvRead.

```
(* Sample5 reads and initializes value1 when the PLC is started FUNCTIONBLOCK: FB_XmlSrvRead *)
PROGRAM Sample5
VAR
   value1        : ST_MyStruct;
   fbXmlSrvRead  : FB_XmlSrvRead;
   bExecute      : BOOL;
   sFilePath     : T_MaxString := 'C:\Test.xml'; (* CE: '\Hard Disk\Test.xml' *)
   sXPath        : T_MaxString := '/dataentry/MAIN.value1';
   nState        : INT := 0;
END_VAR
```

```
CASE nState OF
0: (* initialize *)
   fbXmlSrvRead(
   pSymAddr  := ADR(value1),
   cbSymSize := SIZEOF(value1),
   sFilePath := sFilePath,
   sXPath    := sXPath,
   bExecute  := bExecute
   );
   fbXmlSrvRead(bExecute:= TRUE);
   nState:= 1;

1: (* wait for read operation *)
   fbXmlSrvRead(bExecute:= FALSE);
   IF NOT fbXmlSrvRead.bBusy AND NOT fbXmlSrvRead.bError THEN
     nState:= 2;
   ELSIF fbXmlSrvRead.bError THEN
     nState:= 100;
   END_IF

2: (* operations *)
   ;

100:(* errorState *)
   ;

END_CASE
```

Sample 6: Zyklisches und ereignisgesteuertes Schreiben

Das folgende Beispiel erzeugt alle 20 Sekunden eine neue XML-Datei und beschreibt diese mit der bekannten Struktur. Der Dateiname wird immer aus dem aktuellen Windows-Datum, der -Uhrzeit und einem String generiert. Außerdem kann der Schreibprozess durch Drücken eines Schalters (bzw. Setzen der Schaltervariable bButton) gestartet werden. Sollte der Schreibprozess in einer Sekunde mehrfach ausgelöst werden, wird die aktuelle Datei überschrieben.

```
(* Sample6: Every 20s value1 will be written into a new XML-File named after the current date and
   time. Furthermore you can activate the printing procedure by pressing a button (or setting the
   corresponding variable *)
PROGRAM Sample6
VAR
   value1            : ST_MyStruct;
   fbXmlSrvWrite     : FB_XmlSrvWrite;

   sFileFolder       : T_MaxString :='C:\'; (* CE: '\Hard Disk\' *)
   sFileName         : T_MaxString:= '_test.xml';
   sFilePathWrite    : T_MaxString
   (*sFilePathWrite = sFileFolder + time + sFileName*)

   sXPathWrite       : T_MaxString :='/dataentry/MAIN.value1';
   ntGetTime         : NT_GetTime;
   stMyTimestruct    : TIMESTRUCT;
   iState            : INT := 1;
   bTwentySec        : BOOL:= FALSE;
   bButton           : BOOL:= FALSE;
   bTwentySecOver    : BOOL;
   triggerWrite      : R_TRIG;
   triggerButton     : R_TRIG;
END_VAR
```

```
triggerButton(CLK:= bButton);

CASE iState OF
0: (* idle state *)
   ;

1: (* initialize *)
   fbXmlSrvWrite(nMode:=XMLSRV_ADDMISSING, pSymAddr:= ADR(value1),
     cbSymSize:= SIZEOF(value1));
   ntGetTime(START:= TRUE, TIMESTR=>stMyTimestruct);  (* get Windows time *)IF NOT ntGetTime.BUSY AN
D NOT ntGetTime.ERR THEN
     iState:= 2;
   ELSIF ntGetTime.ERR THEN
     iState:= 100;
   END_IF

2: (* working state *)
   (* change some values - replace with production-process *)
   value1.stInner.nInteger:= value1.stInner.nInteger + 1;
   IF value1.stInner.nInteger = 32767 THEN
     value1.stInner.nInteger:= 0;
   END_IF(* get Windows time *)
   ntGetTime(START:= FALSE);
   IF NOT ntGetTime.BUSY AND NOT ntGetTime.ERR THEN
     ntGetTime(START:= TRUE, TIMESTR=>stMyTimestruct);
   ELSIF ntGetTime.ERR THEN
     iState:= 100;
   END_IF(* check if 20s have passed*)IF stMyTimestruct.wSecond = 0  OR stMyTimestruct.wSecond = 20
 OR stMyTimeStruct.wSecond = 40  THEN
     bTwentySecOver:= TRUE;
   ELSE
     bTwentySecOver:= FALSE;
   END_IF(* if 20s have passed => trigger writing-process *)
   triggerWrite(CLK:=bTwentySecOver);
   IF (triggerWrite.Q OR triggerButton.Q) AND NOT fbXmlSrvWrite.bBusy
                      AND NOT fbXmlSrvWrite.bError THEN (* create filename *)
     sFilePathWrite:= CONCAT(sFileFolder, SYSTEMTIME_TO_STRING(stMyTimestruct)); (* set folder + tim
e *)
     sFilePathWrite:= DELETE(STR:= sFilePathWrite, LEN:= 4 ,
                  POS:= LEN(STR:=sFilePathWrite)-3); (* delete milliseconds *)
     sFilePathWrite:= REPLACE(STR1:= sFilePathWrite , STR2:= '.' , L:= 1,
                   P:= LEN(STR:=sFilePathWrite)-2); (* replace colon with point *)
     sFilePathWrite:= REPLACE(STR1:= sFilePathWrite , STR2:= '.' , L:= 1,
                   P:= LEN(STR:=sFilePathWrite)-5); (* replace colon with point *)
     sFilePathWrite:= CONCAT(sFilePathWrite, sFileName);  (* add filename (default: test) *)
(* write *)
     fbXmlSrvWrite(sFilePath:=sFilePathWrite, sXPath:=sXPathWrite, bExecute:= TRUE);
   ELSIF fbXmlSrvWrite.bError THEN
     iState:= 100;
   END_IF(* reset fbXmlSrvWrite *)IF fbXmlSrvWrite.bBusy AND NOT ntGetTime.ERR THEN
     fbXmlSrvWrite(bExecute:= FALSE);
   ELSIF ntGetTime.ERR THEN
     iState:= 100;
   END_IF

100: (* error state*)
   ;

END_CASE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0	PC oder CX (x86)	TcXmlDataSrv.Lib

6.4 Sample 7 (Produktionsbeispiel)

Hier finden Sie ein Beispiel für einen Produktionsauftrag. Dabei werden Produktionsdaten wie Bauteillänge, Bauteilbreite etc. in einer Struktur zur Initialisierung ausgelesen (XML Read), die Produktion mit einer entsprechenden Stückzahl durchgeführt und der Produktionsauftrag mit einem Eintrag in der XML Datei (Write) beendet. Um das Programm zu starten, muss zuvor das XML File an die dem Datei-Pfad entsprechende Stelle gespeichert werden und im SPS-Programm die Variable bStart auf TRUE gesetzt werden.

Das **SPS-Projekt**, das die Beispiele enthält, können Sie https://infosys.beckhoff.com/content/1031/tcxmldatasrv/Resources/11416819467.zip

Variablendeklaration

```
PROGRAM Sample7
VAR
   fbXmlSrvReadByName      : FB_XmlSrvReadByName;
   fbXmlSrvWriteByName     : FB_XmlSrvWriteByName;

   value           : ST_MyProductionStruct;

   state           : INT := 0;
   R_Edge          : R_TRIG;
   bStart          : BOOL;
   bError          : BOOL;
   nErrId          : UDINT;
END_VAR
```

Struktur ST_MyProductionStruct

```
TYPE ST_MyProductionStruct :
STRUCT
   rLength     : REAL;
   rWidth      : REAL;
   rHeight     : REAL;
   iQuantity   : INT;
   iCounter    : INT;
   bReady      : BOOL;
   stInfo      : STRING;
END_STRUCT
END_TYPE
```

SPS Programm

```
(* The production data are read, the production is carried out and, according to the
quantity and the production order, completed with an entry in the XML file.
To start the program the XML file needs to be stored at the corresponding place
of the data path and the variable bStart needs to be set TRUE in the PLC program. *)
R_Edge (CLK := bStart);
IF R_Edge.Q THEN
    state := 1;
END_IFCASE state OF
0: (* idle state *)
       ;

1: (* init state *)
    fbXmlSrvReadByName( sNetId     := '',
            sSymName   := 'Sample7.value',
            sFilePath  := 'C:\Production1.xml',
            sXPath     := '/dataentry/MAIN.value',
            bExecute   := TRUE,
            tTimeout   := t#10s,
            bError     => bError,
            nErrId     => nErrId);
    state := 2;

2:
    fbXmlSrvReadByName(bExecute := FALSE);
    IF NOT fbXmlSrvReadByName.bBusy AND NOT fbXmlSrvReadByName.bError THEN
    state := 3;
    ELSIF fbXmlSrvReadByName.bError THEN
    state := 100;
    END_IF

3:  (* working state *)
    IF value.bReady = TRUE THEN
    value.stInfo := 'The order was already processed!';
    (* replace your production XML file! *)
    state := 4;
    RETURN;
    END_IF

  (* Call production program with
    new length, width and height here *)

    value.iCounter := value.iCounter + 1;
    IF value.iCounter = value.iQuantity THEN
```

```
    value.bReady := TRUE;
    state := 4;
    END_IF

4:  (* documentation state *)
    fbXmlSrvWriteByName( sNetId     := '',
             nMode      := XMLSRV_SKIPMISSING,
             sSymName   := 'Sample7.value',
             sFilePath  := 'C:\Production1.xml',
             sXPath     := '/dataentry/MAIN.value',
             bExecute   := TRUE,
             tTimeout   := t#10s,
             bError     => bError,
             nErrId     => nErrId);
    state := 5;

5:
    fbXmlSrvWriteByName(bExecute := FALSE);
    IF NOT fbXmlSrvWriteByName.bBusy AND NOT fbXmlSrvWriteByName.bError THEN
    state := 0;
    ELSIF fbXmlSrvWriteByName.bError THEN
    state := 100;
    END_IF

100:(* error state *)
    ;
END_CASE
```

XML Datei

```
<dataentry>
    <MAIN.value>
        <rLength>65.85</rLength>
        <rWidth>30</rWidth>
        <rHeight>2.5</rHeight>
        <iQuantity>500</iQuantity>
        <iCounter>0</iCounter>
        <bReady>false</bReady>
        <stInfo></stInfo>
    </MAIN.value>
</dataentry>
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0	PC oder CX (x86)	TcXmlDataSrv.Lib

7 Anhang

7.1 Fehlercodes des TwinCAT Xml Data Servers

Voraussetzungen

Offset + Fehlercode	Bereich	Beschreibung
0x00000000 + TwinCAT System Fehler	0x00000000-0x00007800	TwinCAT System Fehler (ADS-Fehlercodes inklusive)
0x00008000 + Interner TwinCAT Xml Data Server Fehler [▶ 29]	0x00008000-0x0080FF	Interne Fehler des TwinCAT Xml Data Servers

7.2 Interne Fehlercodes des TwinCAT XML Data Servers

Voraussetzungen

Code	Beschreibung	Symbolischer Name
0x00008000	Interner Fehler	XMLSRVERROR_INTERNAL
0x00008001	Handle nicht gefunden.	XMLSRVERROR_NOTFOUND
0x00008002	Fehler beim Parsen der Xml-Datei	XMLSRVERROR_PARSERERROR
0x00008003	Inkompatibler Datentyp	XMLSRVERROR_INCOMPATIBLE
0x00008004	Fehler beim Allozieren von Speicher	XMLSRVERROR_NOMEMORY
0x00008005	Fehler beim Hinzufügen eines XML-Knotens	XMLSRVERROR_ADDNODE
0x00008006	Ungültiger sXPath	XMLSRVERROR_INVALIDXPATH
0x00008007	Ungültiger String in TC	XMLSRVERROR_INVALIDSTRING
0x00008800	Ungültiges Handle des Clients	XMLSRVERROR_INVALIDCLIENTHANDLE
0x00008900	Verwendung der falschen TcXmlDataSrv.exe (für TC2 statt TC3)	XMLSRVERROR_INVALIDTWINCATVERSION

Mehr Informationen:
www.beckhoff.de/ts6421

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