BECKHOFF New Automation Technology

Handbuch | DE

TS6420

TwinCAT 2 | DataBase Server

Supplement | Communication

10.08.2023 | Version: 1.2

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 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 Produktbeschreibung

Der TwinCAT Database Server ermöglicht den Datenaustausch zwischen dem TwinCAT System und verschiedenen Datenbanksystemen. Für kleine Applikationen lässt er sich sogar ohne Eingriff in den vorhandenen Programmcode über einen Konfigurator nutzen. Für komplexe Aufgabenstellungen bietet der Database Server eine ausführliche Bibliothek von SPS-Funktionsbausteinen für ein Maximum an Flexibilität. So können direkt aus der SPS beispielsweise SQL-Befehle wie Insert oder Select verwendet werden. Um die SPS gegebenenfalls zu entlasten, besteht zusätzlich die Möglichkeit sogenannte Stored Procedures in den Datenbanken aufzurufen. Die vom entsprechenden SPS-Baustein übergebenen Parameter werden dann von der Datenbank im Zusammenhang mit der gespeicherten Prozedur verwendet und Ergebnisse werden, wenn gewünscht, an die Steuerung zurückgegeben.

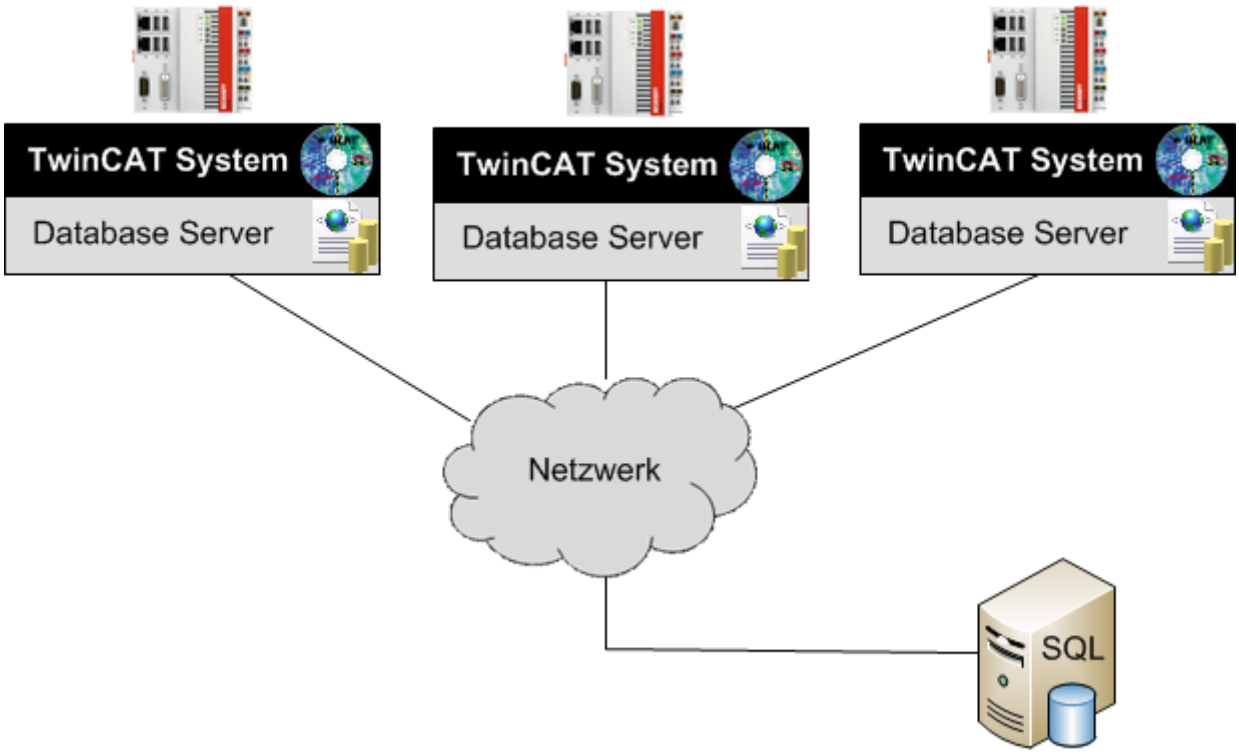
Der TwinCAT Database Server unterstützt derzeit elf verschiedene Datenbanksysteme [▶ 36]: MS SQL, MS SQL Compact, MS Access, MySQL, PostgreSQL, DB2, Oracle, Interbase, Firebird, ASCII (z.B. .txt oder .csv) sowie XML-Dateien.

Komponenten

- TwinCAT Database Server [▶ 13]: Ist ein Service der mit TwinCAT zusammen gestartet und gestoppt wird. Er bildet das Bindeglied zwischen TwinCAT System und Datenbank.
- Konfigurator [▶ 15]: Der TwinCAT Database Server Konfigurator ermöglicht die einfache visuelle Einstellung der Datenbankparameter, welche für die grundlegende Kommunikation mit der jeweiligen Datenbank benötigt werden.
- SPS-Bibliothek [▶ 60]: Die SPS-Bibliothek bietet die verschiedene Funktionsbausteine, um sich u.a. eine Datenbankverbindung oder eine neue Tabelle zu erstellen. Um Daten in völlig beliebige Tabellenstrukturen mit Insert-Befehlen zu schreiben oder sie per Select-Befehl zu lesen. Man kann Datenbankeinträge aktualisieren oder löschen. Gespeicherte Prozeduren können angestoßen werden.

Funktionsprinzip

Der Database Server kommuniziert innerhalb des TwinCAT Systems über ADS. Nach außen verbindet er sich mit der jeweils konfigurierten Datenbank. Mögliche Netzwerktopologien findet man hier [▶ 144].



3 Systemvoraussetzungen

Voraussetzungen

Technische Daten	TwinCAT Database Server
Zielsystem	Windows NT/2000/XP/Vista/7 PC (x86-kompatibel)
.NET Framework	.Net 2.0 oder höher
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 Database Server Supplements für Windows basierte Betriebssysteme. Es wird hierbei auf die folgenden Themen eingegangen:

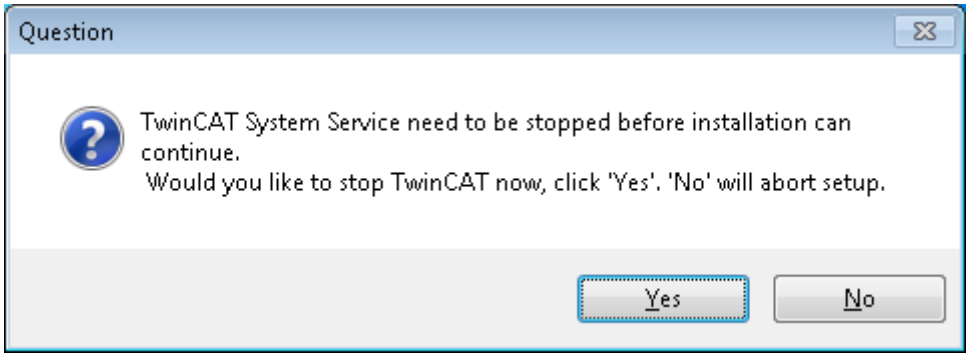
- Starten der Installation
- Nach der Installation

Starten der Installation

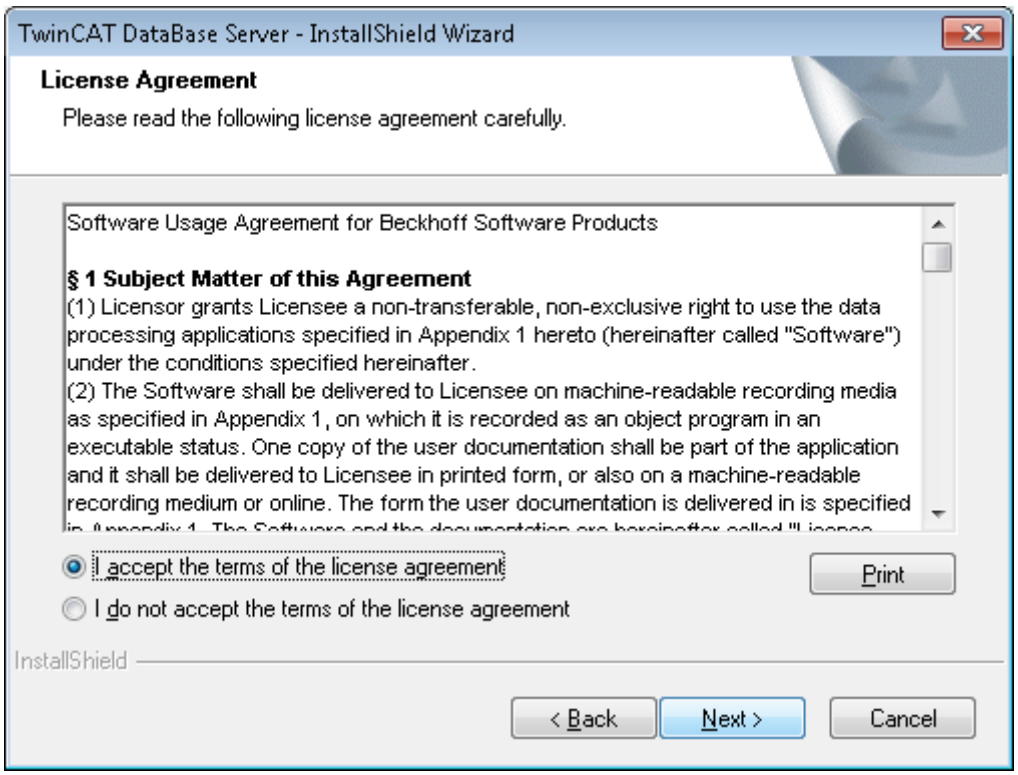
Um das Supplement zu installieren, führen Sie bitte die folgenden Schritte durch:

> Bitte starten Sie die Installation unter Windows "Als Administrator ausführen", indem Sie die Setup-Datei mit der rechten Maustaste anklicken und die entsprechende Option im Kontextmenü auswählen.

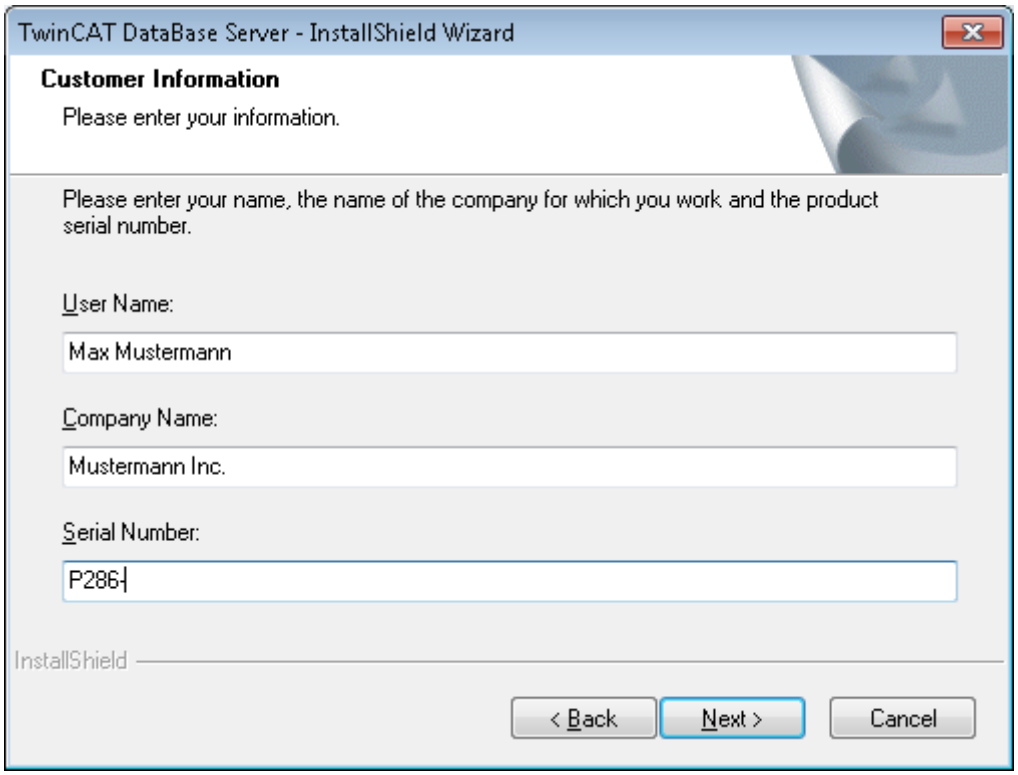
- Führen Sie einen Doppelklick auf die heruntergeladene Datei**"TcDatabaseSrv.exe"**aus.
- Wählen Sie eine **Sprache** aus, in der Sie die Software installieren möchten
- Das TwinCAT System muss gestoppt werden, bevor die Installation ausgeführt wird. Wählen Sie "Yes" um den TwinCAT System Service zu stoppen



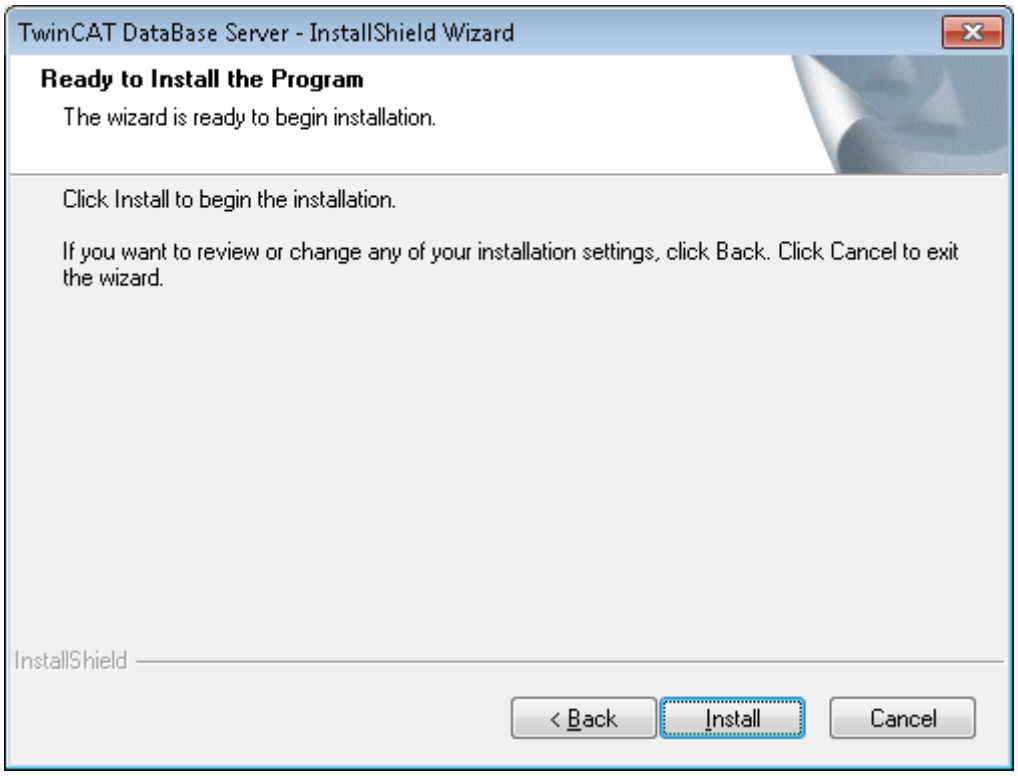
- Klicken Sie auf "Next" und akzeptieren Sie dann die **Endbenutzervereinbarung**



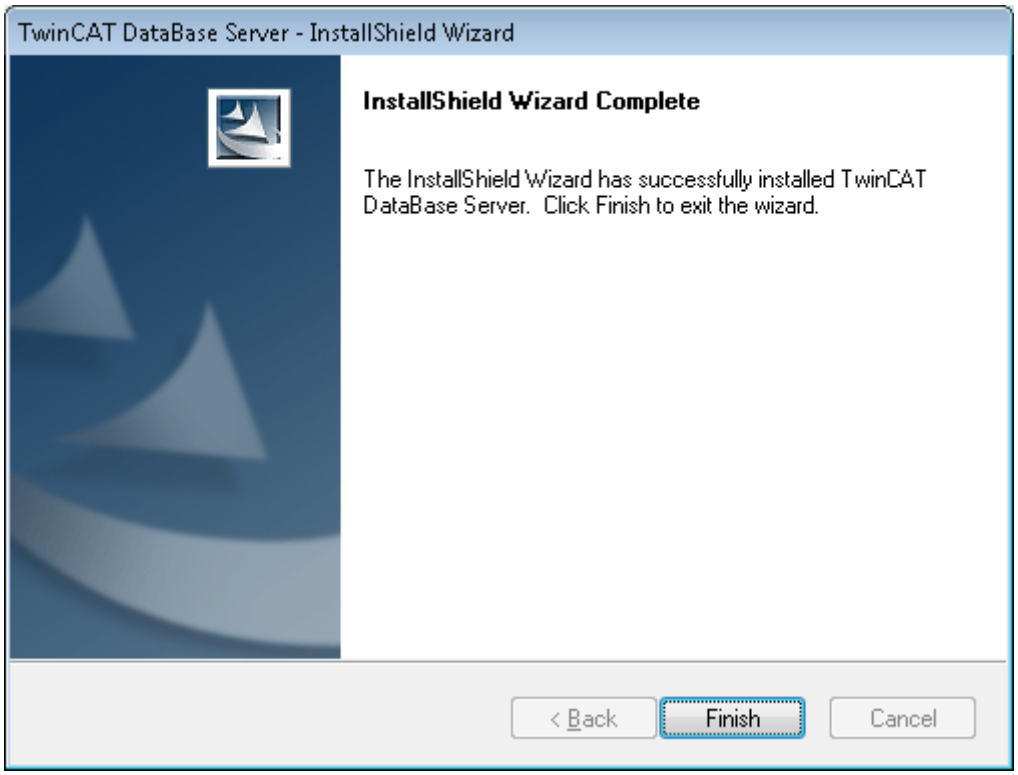
- Geben Sie Ihre **Benutzerdaten** ein



- Klicken Sie auf **"Install"** um die Installation zu starten



- Wählen Sie "**Finish**" um das Setup zu beenden



Nach der Installation

Nach der Installation kann das TwinCAT System wieder in den CONFIG Modus versetzt werden. Das Supplement-Produkt "TwinCAT Database Server" kann dann im vollen Umfang verwendet werden. Mit Hilfe des XML-Konfigurators kann der TwinCAT Database Server dann konfiguriert werden siehe hier [▶ 15].

5 Database Server Funktionsweise

Die Funktionsweise des TwinCAT Database Servers zeichnet sich durch verschiedene Kommunikationsrichtungen und der Konfiguration über eine XML-Datei aus.

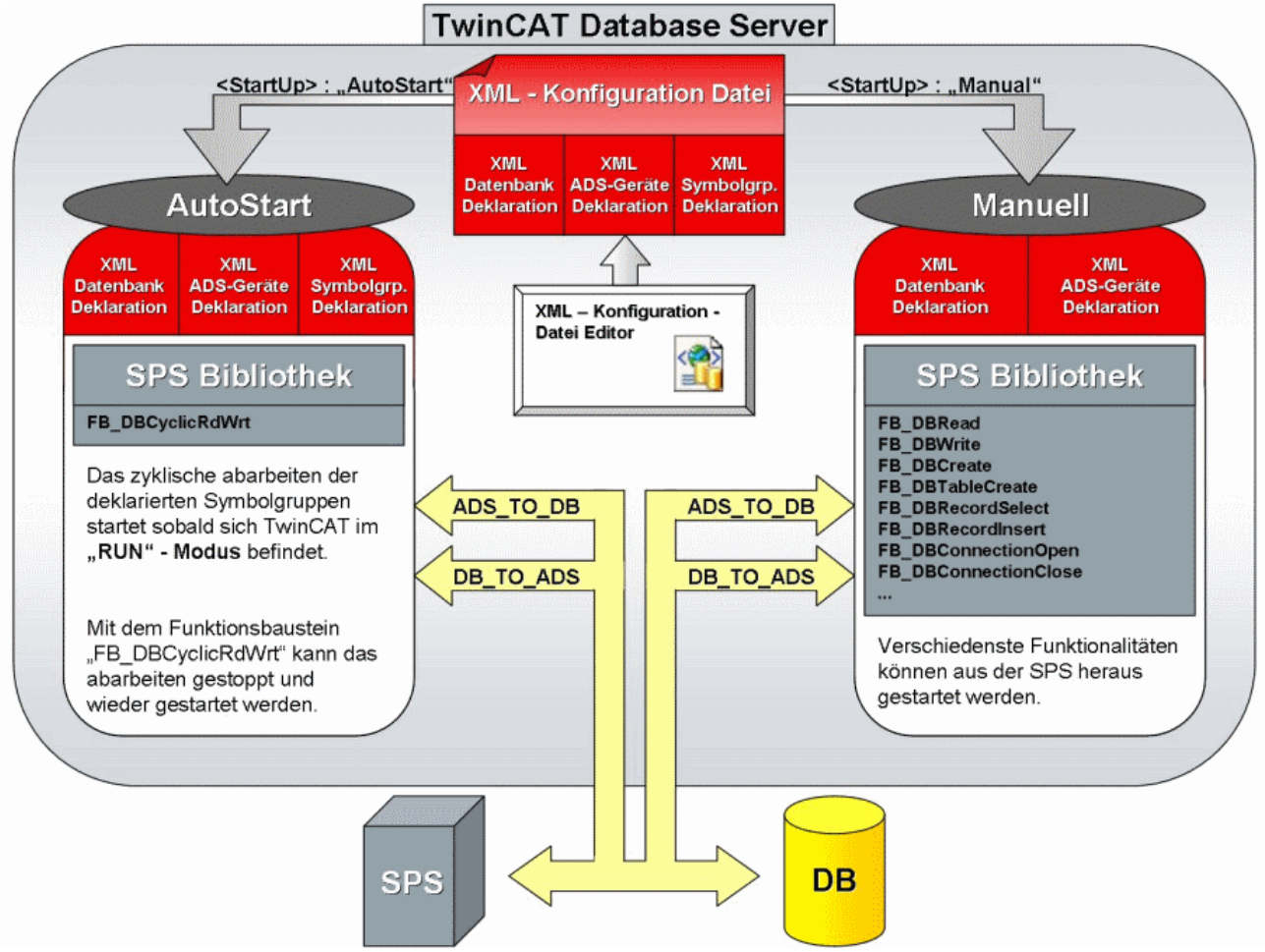
- Kommunikationsfluss in zwei Richtungen
  - **ADS_TO_DB**
    Prüfen eines TwinCAT-ADS-Gerätes (z.B. TwinCAT SPS) und Schreiben dieser Daten in eine "Datenbank"
  - **DB_TO_ADS**
    Prüfen einer "Datenbank" (z.B. SQL-Datenbank) und Schreiben dieser Daten per ADS in ein TwinCAT-ADS-Gerät (z.B. eine SPS)
- Die Konfiguration des "TwinCAT Database Server" wird mit Hilfe einer XML-Datei durchgeführt. In dieser Konfigurationsdatei werden die gewünschten "Datenbanken", ADS-Geräte (z.B. SPS-Laufzeitsysteme) und die Variablen beschrieben.
  Zwei Speicherarten:
  - **"Double"** Die kompatiblen Variablendatentypen lauten: BOOL, LREAL, REAL, INT, DINT, USINT, BYTE, UDINT, DWORD, UINT, WORD, SINT
  - **"Bytes"** Kompatibel für alle Variablendatentypen, insbesondere Strings und Datenstrukturen.

Zwei Funktionsmodi:

- **AutoStart:**
  Startet das zyklische Prüfen eines TwinCAT-ADS-Gerätes (ADS_TO_DB) oder einer Datenbank (DB_TO_ADS) automatisch, sobald sich das TwinCAT System im "RUN" - Modus befindet. Die zu überprüfenden SPS-Programme sollten als Bootprojekte laufen.
- **Manual:**
  Funktionen wie in eine Datenbank schreiben bzw. aus einer Datenbank lesen werden mit Funktionsbausteinen aus der SPS heraus gestartet.

In dem folgendem Schaubild ist die Funktionsweise des Servers dargestellt.

Zentrale Einheit des TwinCAT Database Server ist der XML-Konfigurationsdatei Editor mit dem alle notwendigen Einstellungen vorgenommen werden können. Die erzeugte Konfiguration kann dann in zwei verschiedenen Modi (AutoStart/Manuell) verwendet werden. Der TwinCAT Database Server bildet dann das Bindeglied zwischen SPS und Datenbank.



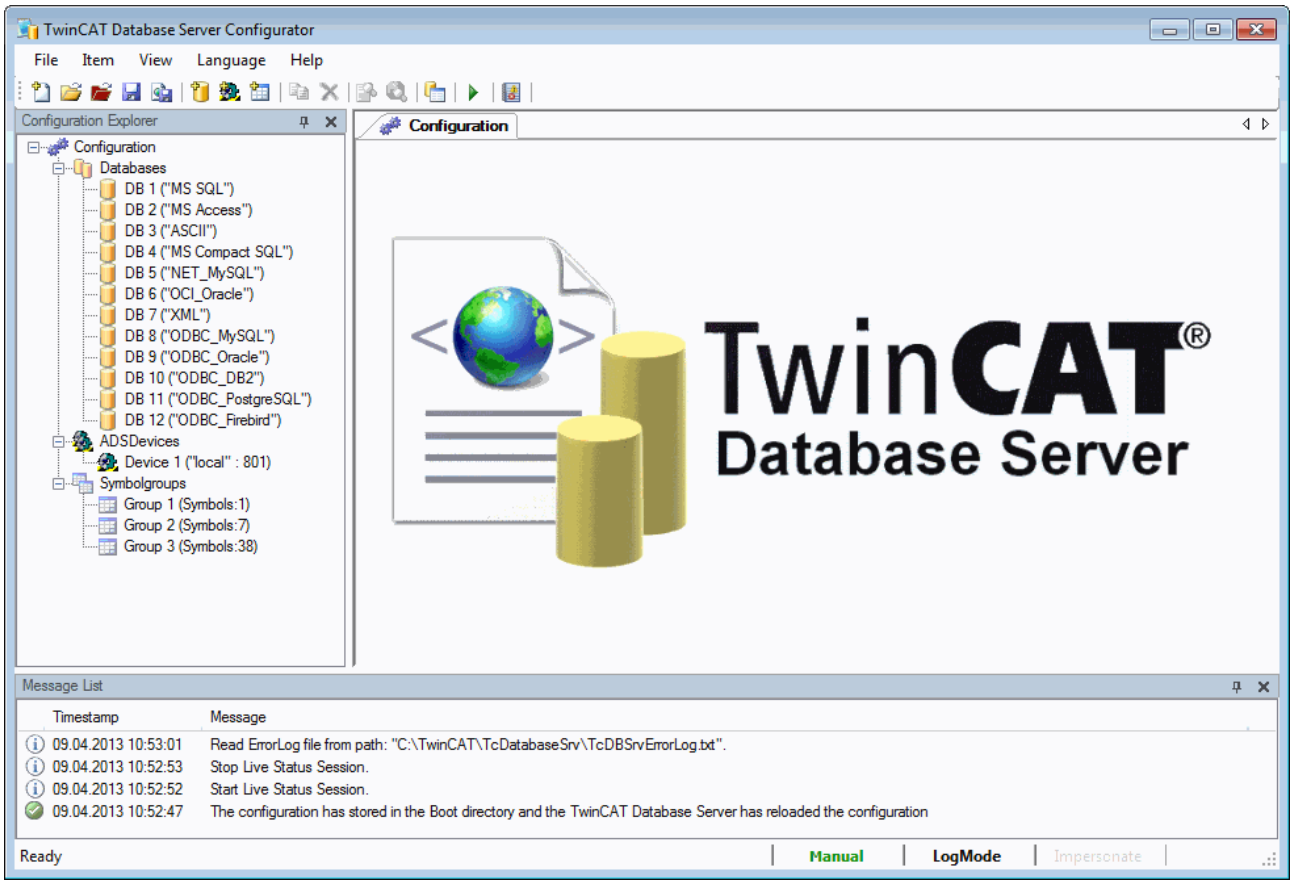
Wichtig!!

Beim Loggen von Strukturen von ADS-Geräten mit ARM-Prozessoren können Fehler aufgrund des Byte-Alignments auftreten.
Beim Loggen von Strukturen vom BC9000 mit Floating-Point Variablen können Fehler auftreten.

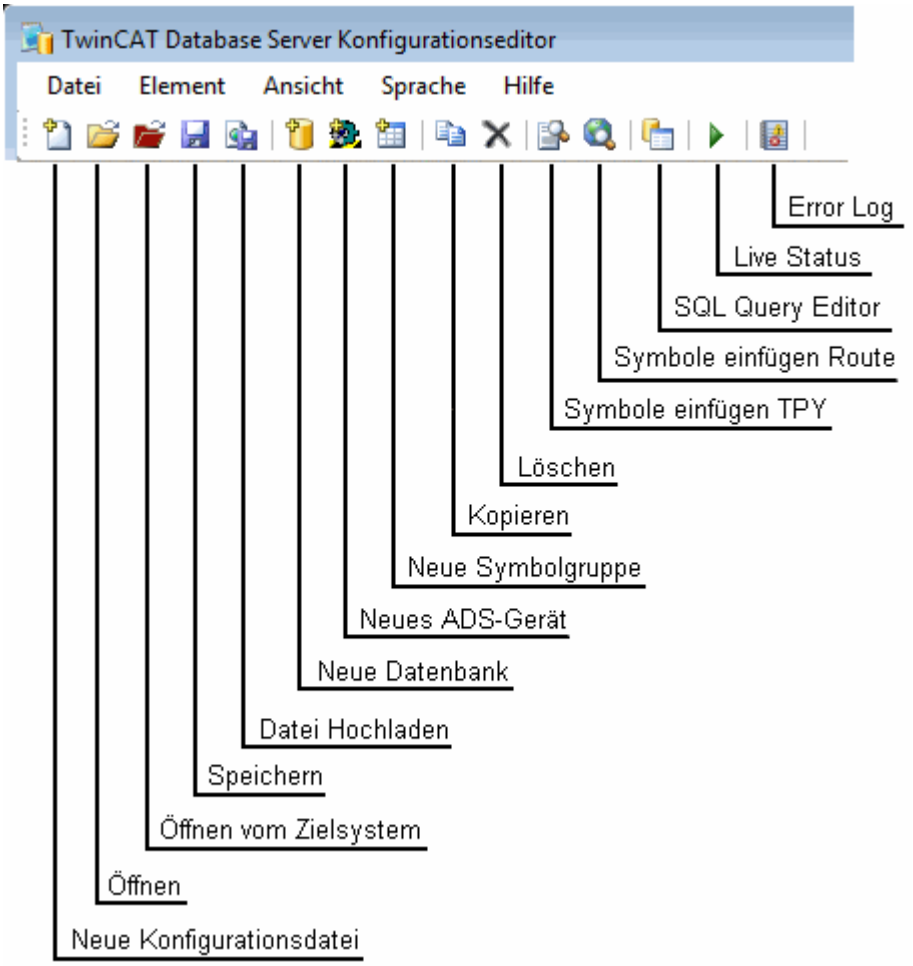
6 Konfiguration

6.1 XML - Konfigurationsdateieditor

Die Konfiguration des TwinCAT Database Servers erfolgt über eine XML - Konfigurationsdatei.
Die Einstellungen der Konfigurationsdatei können mit Hilfe des XML - Konfigurationsdateieditors auf schnelle und einfache Weise erstellt und überarbeitet werden. Neue Konfigurationsdateien können erstellt bzw. vorhandene Konfigurationsdateien können eingelesen und überarbeitet werden.



Die Menüleiste

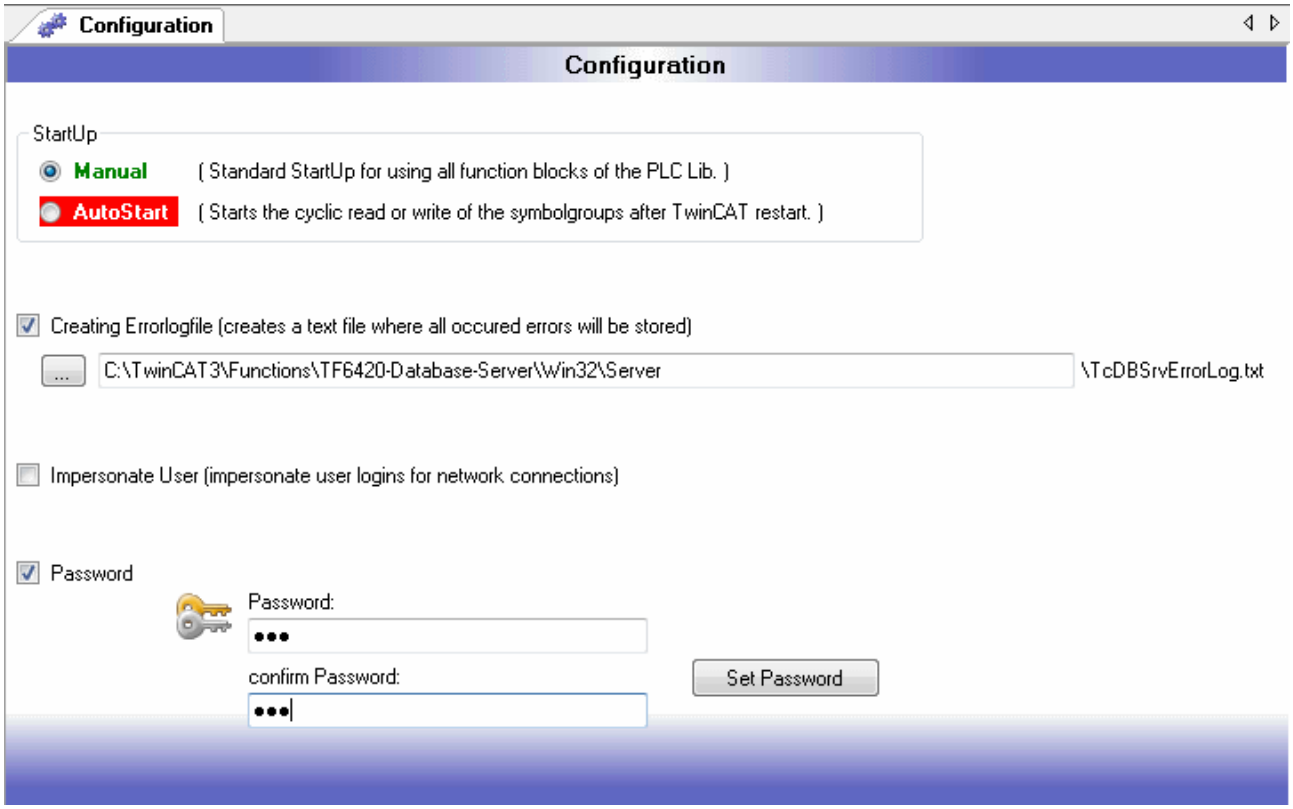


	Beschreibung
Neue Konfigurationsdatei	Erstellt eine leere Konfigurationsdatei mit allen Defaulteinstellungen.
Öffnen	Öffnet eine vorhandene XML-Konfigurationsdatei.
Speichern	Speichert alle vorgenommenen Änderungen und erstellt die Konfigurationsdatei mit dem Namen: "CurrentConfigDataBase.xml".
Datei Hochladen	Überträgt die erstellte Konfigurationsdatei zum Database Server, speichert sie im entsprechenden "Boot"-Verzeichnis und aktiviert die Konfiguration.
Neue Datenbank	Erstellt einen neuen Datenbank Konfigurationseintrag.
Neues ADS-Gerät	Erstellt einen neuen ADS-Gerät Konfigurationseintrag.
Neue Symbolgruppe	Erstellt einen neuen Symbolgruppen Konfigurationseintrag.
Löschen	Löscht den selektierten Konfigurationseintrag. Dies kann eine Datenbank, ein ADS-Gerät oder eine Symbolgruppe sein.
Symbole einfügen TPY	Importiert Symbole/Variablen aus einer TPY-Datei in die selektierte Symbolgruppe.
Symbole einfügen Route	Importiert Symbole/Variablen online von einem ADS Gerät (Route).
SQL Query Editor	Editor zum einfachen generieren von SQL Befehlen. Weitere Information hier [▶ 27]
Live Status	Zeigt den aktuellen Status des TF6420 Database Servers an und bietet die Möglichkeit, die zyklische Schreib-/Lesefunktion zu starten und/oder stoppen.
Error Log	Zeigt die geloggten Fehler aus der Fehlerlogdatei "TcDBSrvErrorLog.txt"

Optionsdialog

In diesem Dialog können benötigte Optionen für den TwinCAT Database Server eingestellt werden:

- **StartUp Option:**
  Zwei Varianten können eingestellt werden.
  - "Manual" => TwinCAT Database Server ist aktiv und wartet auf Funktionsaufrufe aus der PLC heraus mit Hilfe der Funktionsbausteine aus der TcDatabase.lib.
  - "AutoStart" => TwinCAT Database Server ist aktiv und startet den Logvorgang der eingestellten Symbolgruppen sobald das TwinCAT System sich im "RUN" Mode befindet.
    SPS-Programme aus denen Variablenwerte geloggt werden sollen, müssen als Bootprojekte laufen!
- **ErrorLog Option:**
  Aktivieren des Fehlerlogmechanismus. Auftretende Fehler werden in eine Textdatei geloggt. Die geloggten Fehler können dann zur Fehleranalyse benutzt werden.
  Zusätzlich kann der Pfad angegeben werden wo diese Textdatei gespeichert werden soll.
- **Impersonate Option:**
  Bei Netzwerkzugriff auf dateibasierenden Datenbanken wie Access-Datenbanken oder SQL Compact Datenbanken muss die Impersonate Option gesetzt werden, damit sich der TwinCAT Database Server auf dieses Netzwerklaufwerk verbinden kann. Weitere Informationen siehe Impersonate [▶ 54]. **Diese Funktion wird zurzeit nicht unter Windows CE unterstützt.**

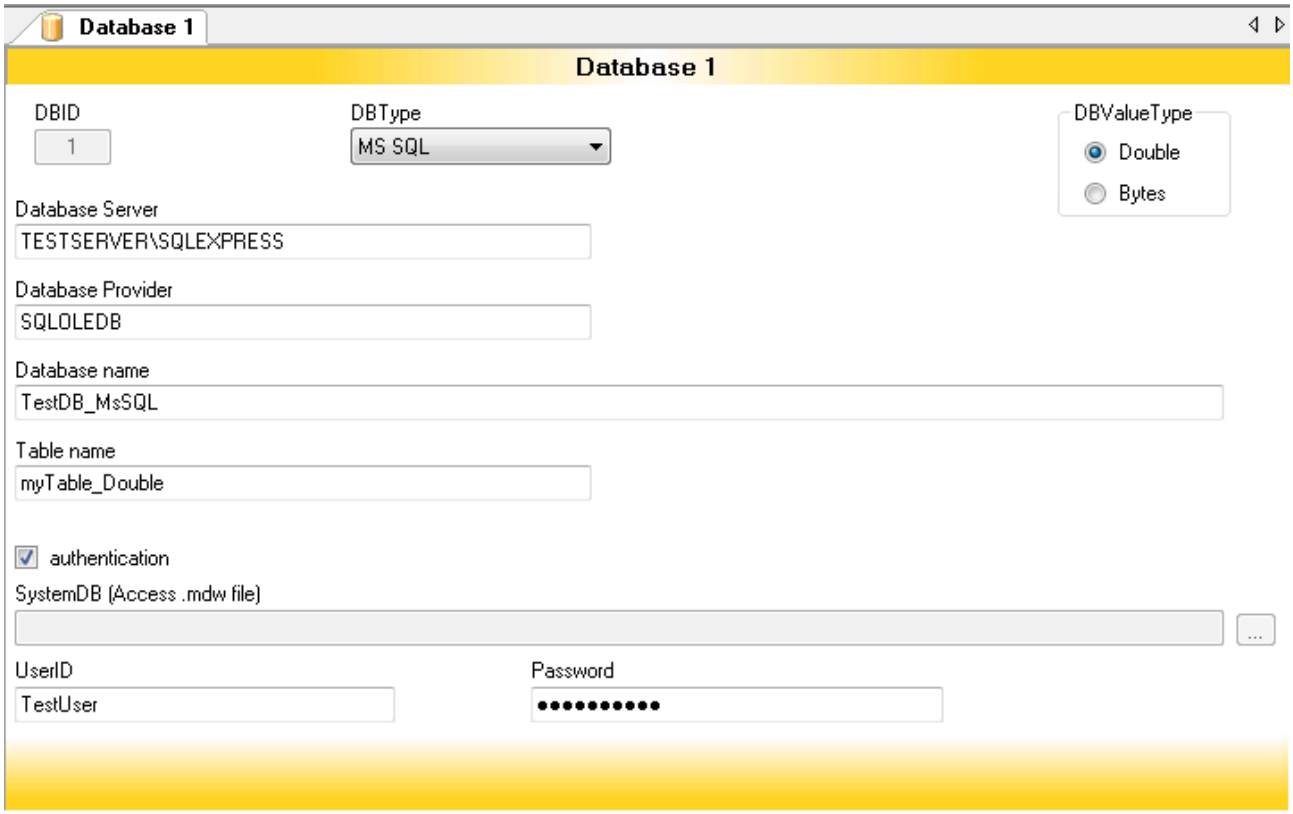


Datenbank Konfigurationsdialog

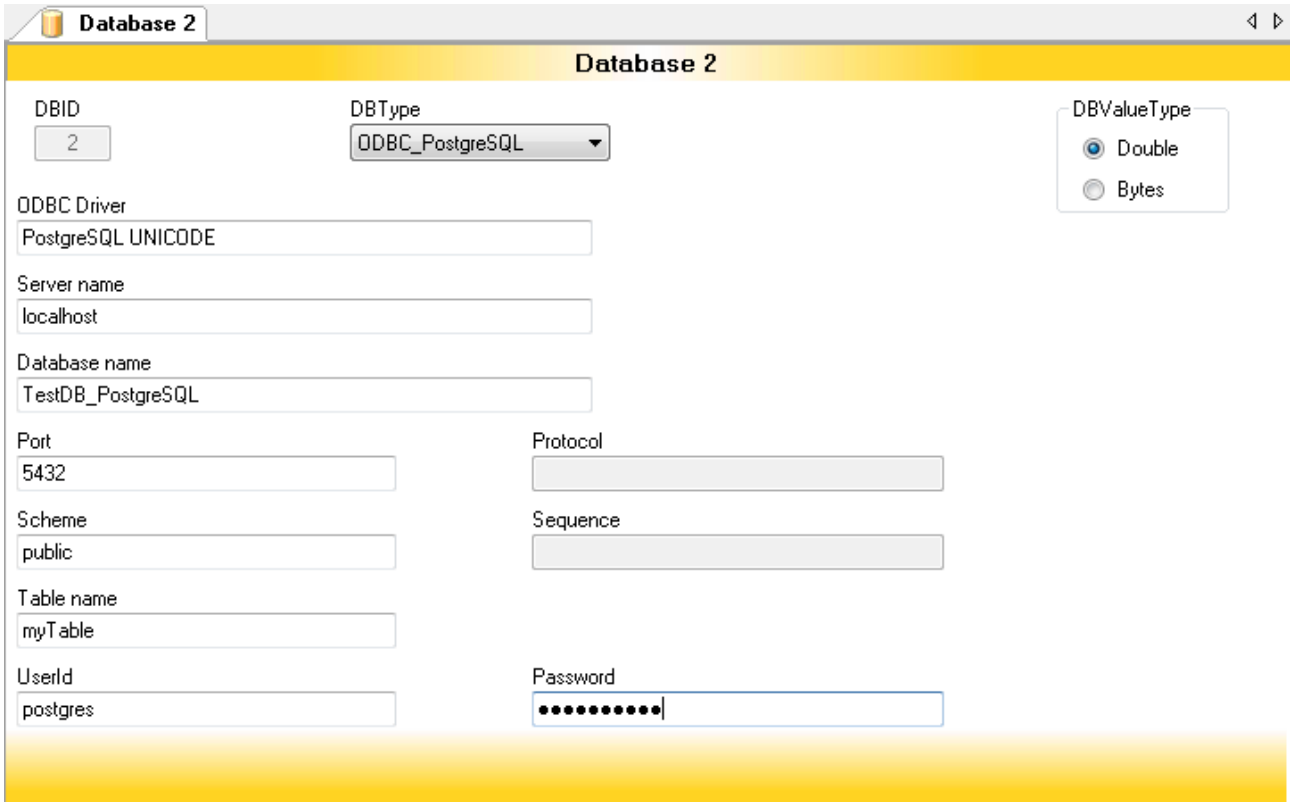
In diesem Dialog können alle benötigten Angaben für die Datenbankkommunikation eingetragen werden.

Der DBValueType gibt an von welchem Datentyp die "Value"-Spalte ist. Ist sie vom Datentyp "Double", so können SPS-Strukturen und Strings nicht geloggt werden. Ist sie vom Datentyp "Bytes", so können auch Strukturen und Strings geloggt werden.
Zusätzlich kann eingestellt werden, ob eine Authentifizierung bei der Datenbankkommunikation benötigt wird, oder nicht. Wenn ja werden die Felder **DBSystemDB**, **DBUserId** und **DBPassword** frei geschaltet. Das Feld **DBSystemDB** wird nur bei Access Datenbanken benötigt. Hier muss der Pfad von der MDW-Datei angegeben werden, in der alle Benutzernamen mit den dazugehörigen Passwörtern gespeichert sind. **DBUserId** ist der Benutzername und **DBPassword** das dazugehörige Passwort.

Konfigurationsbeschreibungen aller Datenbanktypen siehe: Datenbank-Konfigurationen [▶ 36]

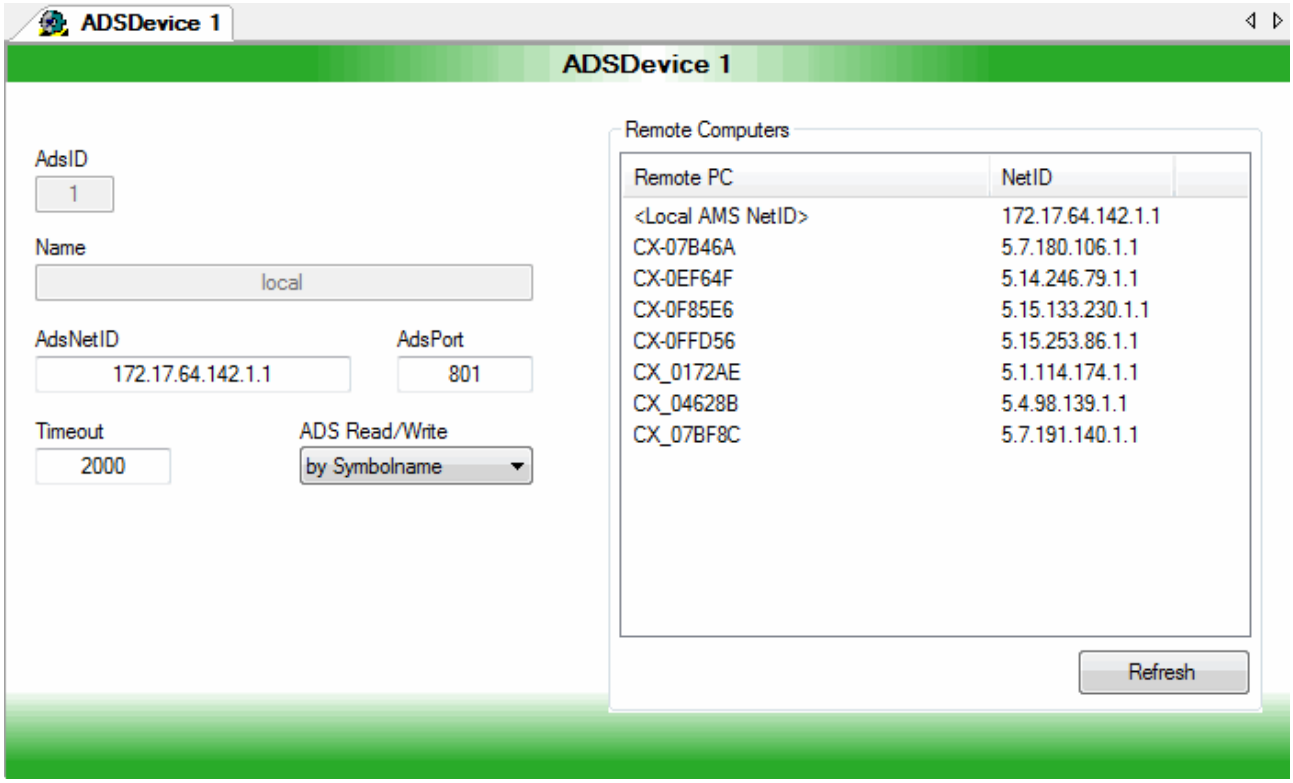


Werden ODBC-Datenbanktypen ausgewählt verändert sich die Einstellungsmaske. Weitere Informationen für die ODBC-Kommunikation wie Port, Protokoll, Driver, Schema und Sequenz müssen angegeben werden.



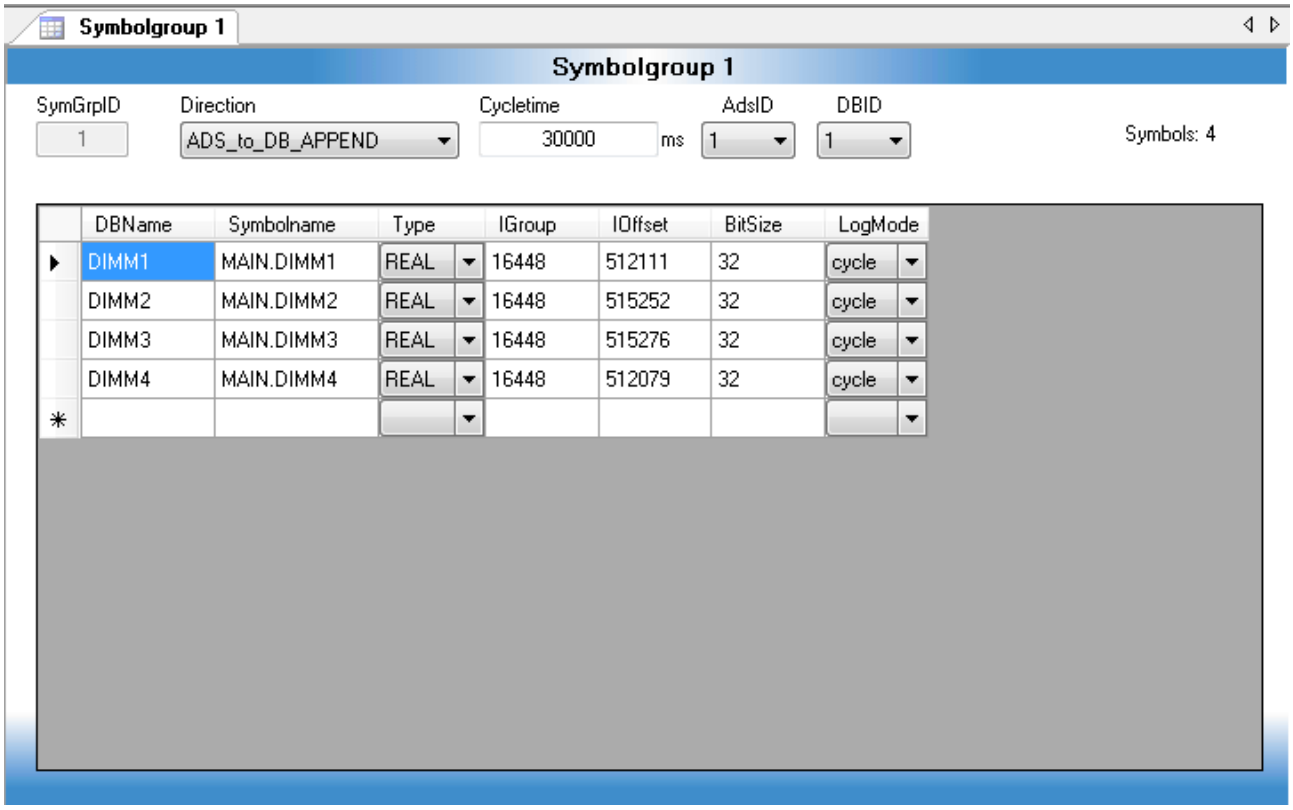
ADS-Geräte Konfigurationsdialog

In diesem Dialog können alle benötigten Angaben für die Kommunikation zu den ADS-Geräten eingetragen werden. Zur Vereinfachung der Eingabe werden alle eingestellten RemotePCs des TwinCAT Systems aufgelistet und können ausgewählt werden. Es können aber auch NetIDs eingetragen werden, die nicht zu einem RemotePc gehören. Des Weiteren kann angegeben werden, ob die Variablenwerte per Symbolnamen oder per IndexGroup / IndexOffset gelesen/beschrieben werden sollen.



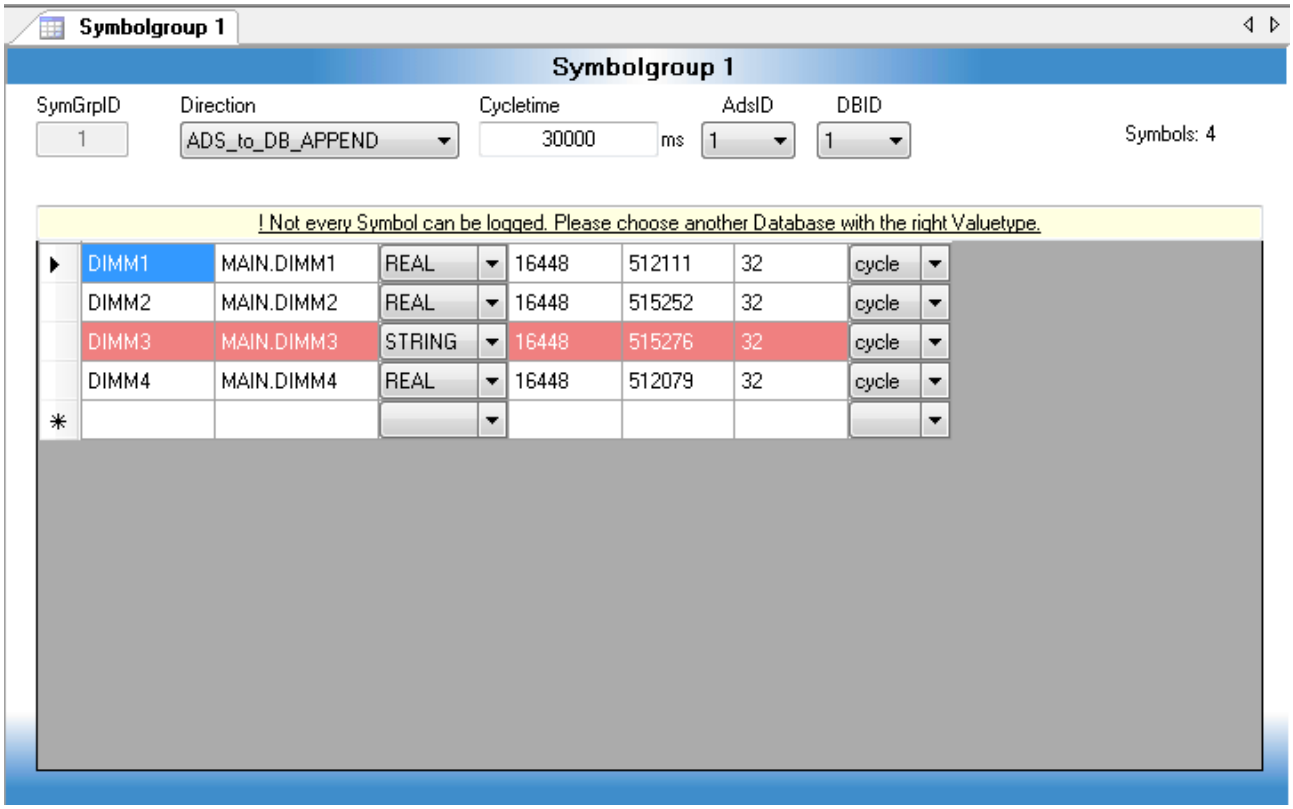
Symbolgruppen Konfigurationsdialog

In diesem Dialog können Symbolgruppen zusammengestellt und ADS-Geräten bzw. Datenbanken zugeordnet werden. Zusätzlich wird die Schreibrichtung der Symbolgruppe festgelegt. Es gibt verschiedene Einstellungsmöglichkeiten siehe Schreibrichtungsmodus [▶ 33].

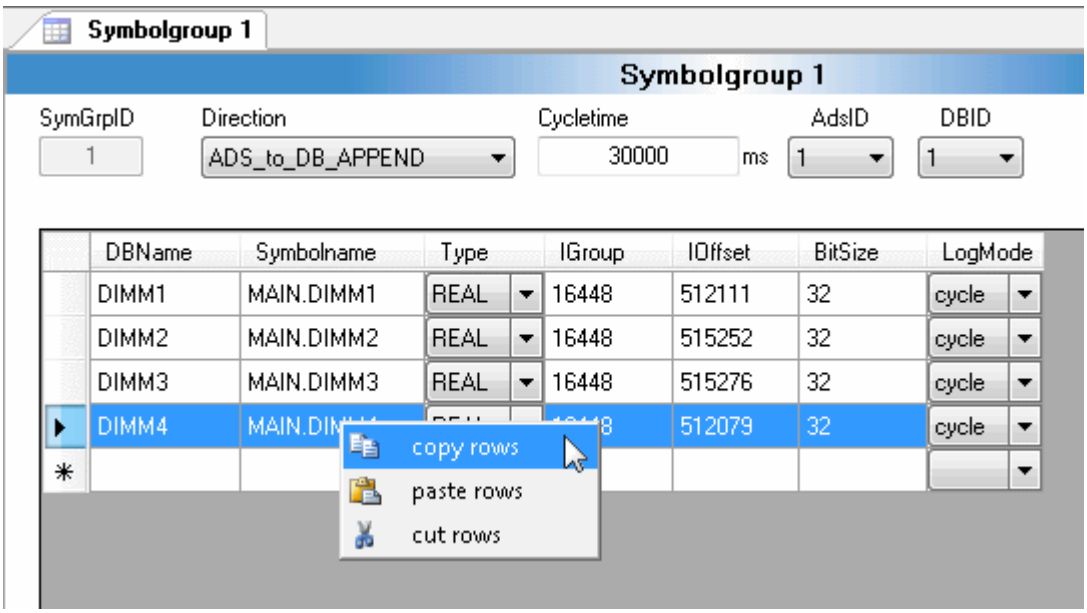


In diesem Dialog ist auch ein Symbolzähler enthalten. Er gibt die Anzahl der deklarierten Symbole der einzelnen Gruppen zurück. Sind mehr als 500 Symbole deklariert verfärbt sich diese Anzeige rot, da pro Symbolgruppe nicht mehr als 500 Symbole/Variablen deklariert werden dürfen.

Werden Datentypen auf Grund der Datenbankeinstellungen nicht unterstützt, werden diese Symbole rot unterlegt und eine Warnmeldung wird im oberen Bereich angezeigt.



Natürlich können Symbolreihen auch kopiert und eingefügt werden. Markieren Sie einfach eine Zeile oder einen Bereich einer Zeile und drücken Sie dann die rechte Maustaste. In dem sich öffnenden Kontextmenü wählen Sie die gewünschte Funktion aus.

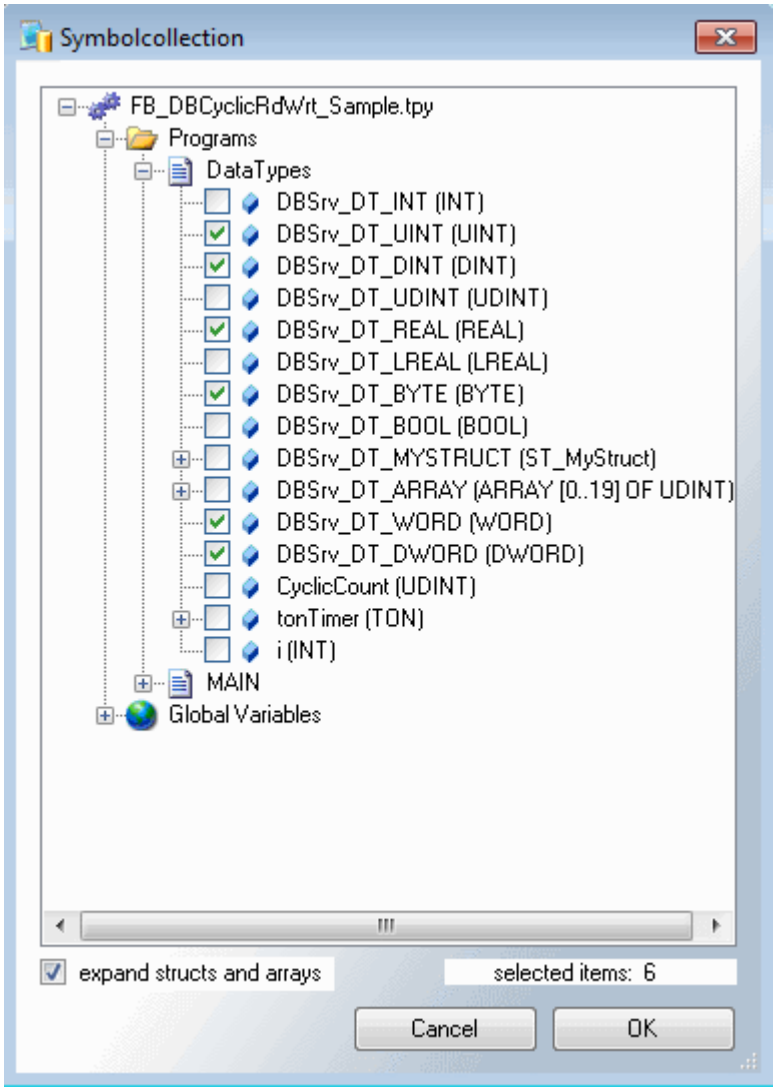


Importieren von Symbolen

Um auf einfache Art und Weise Symbole in die Symbolgruppe einzupflegen, stellt der TwinCAT Database XML Konfigurator zwei Möglichkeiten zur Verfügung. Zum einen können Variablen aus der, nach einem Kompile im PLC Control erzeugten, TPY-Datei eingelesen werden und zum anderen können sie unabhängig von der SPS bequem und direkt von der Steuerung durch den Target Browser ausgewählt werden.

Symbolliste aus TPY Datei

Alle Symbole werden im folgenden Dialog übersichtlich aufgelistet. Zusätzlich besteht die Möglichkeit jedes einzelne Element eines Arrays oder einer Struktur in die Symbolgruppe einzupflegen. So können Strukturen und Arrays die nur aus numerischen Datentypen bestehen auch beim DBValueTyp "Double" geloggt werden.

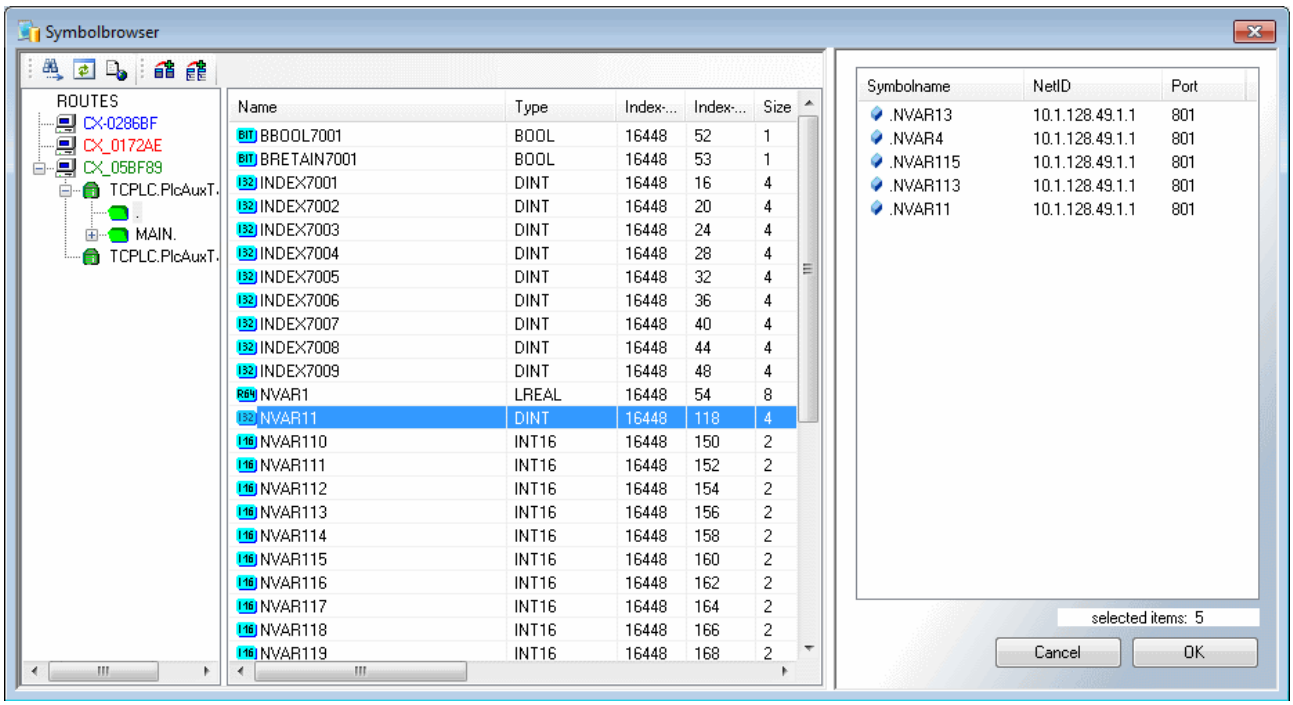


Direkt aus der Steuerung mit dem Target Browser

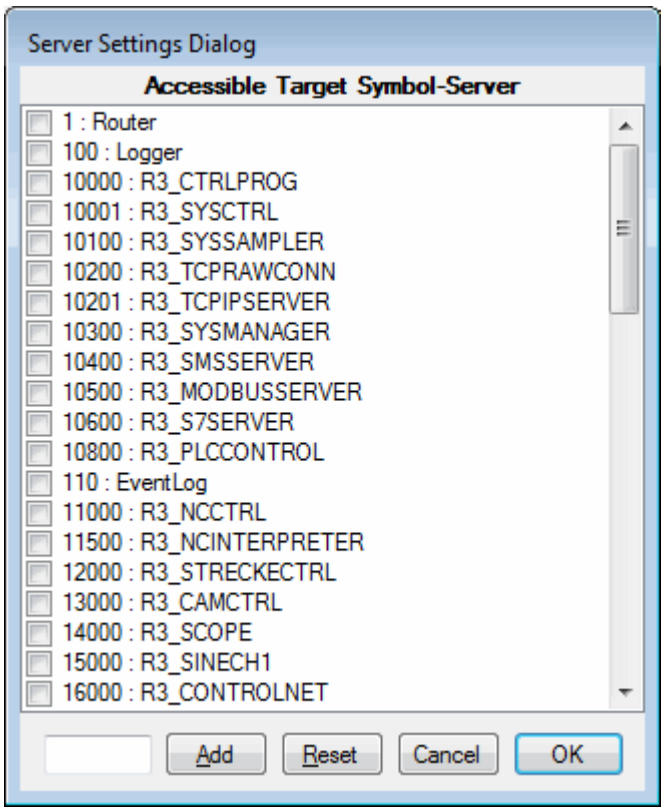
Mit dem Target Browser können Symbole hinzugefügt werden. Der Target Browser ist in drei Teile unterteilt. Der erste Teil zeigt einen Baum mit dem ersten Eintrag: ROUTES. Darunter werden alle im TwinCAT System Manager angemeldeten Geräte angezeigt. Die Farbe des Targets gibt Aufschluss über den Systemzustand: Rot = nicht erreichbar (Stop-Modus), Blau = Konfig-Modus, Grün = System in Run-Modus.

Der zweite Teil zeigt eine detaillierte Liste der Unterpunkte des gewählten Elementes der Baum-Struktur.

Im dritten Teil werden alle ausgewählten Symbole aufgelistet.



Im Target Browser können durch das Symbol mit dem Fernglas in der Toolbar auch neue ADS Ports hinzugefügt werden. Der folgende Dialog zeigt die Eingabemöglichkeit der Port-Nummer. So kann beispielsweise im System Manager im EtherCAT Device Image ein ADS Port aktiviert und hier im Konfigurator eingetragen werden. So können Werte direkt von den EtherCAT Klemmen in eine Datenbank geschrieben werden, ohne dafür die SPS zu verwenden.

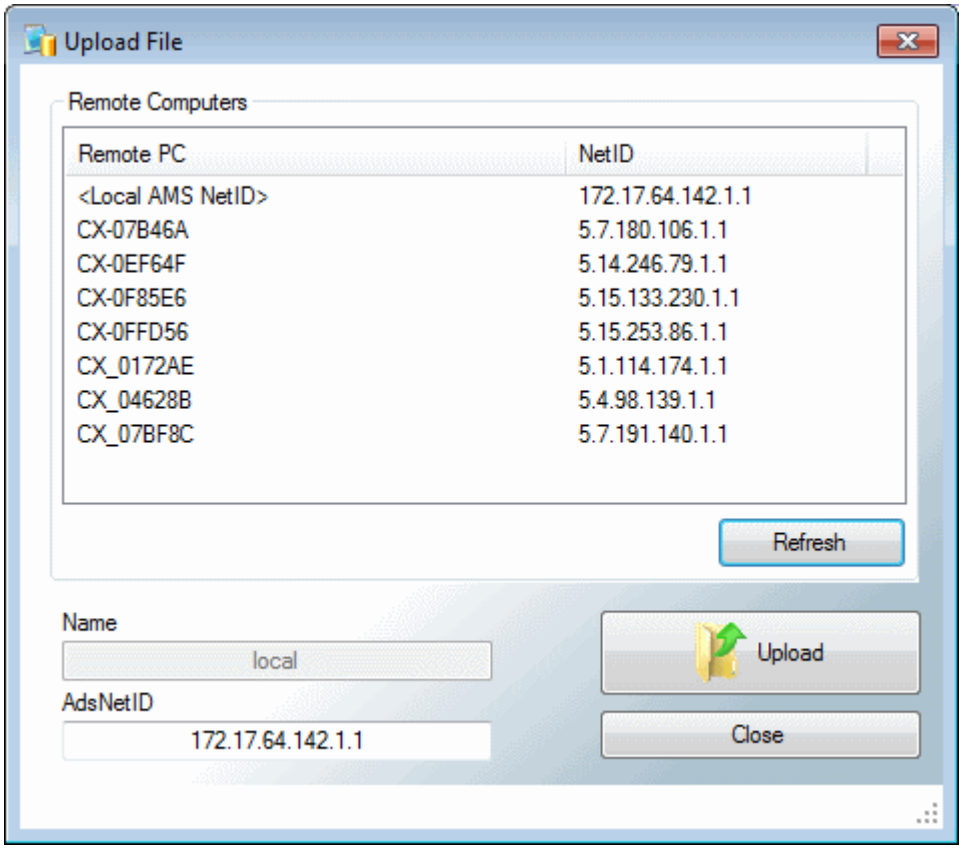


Hochladen der XML-Konfigurationsdatei zum TwinCAT Database Server

Die erstellte XML-Konfigurationsdatei muss zum Aktivieren in das entsprechende "TwinCAT\Boot" Verzeichnis kopiert werden und durch einen Restart des TwinCAT Systems, oder durch den Funktionsbaustein FB_DBReloadConfig zum TwinCAT Database Server hochgeladen werden.

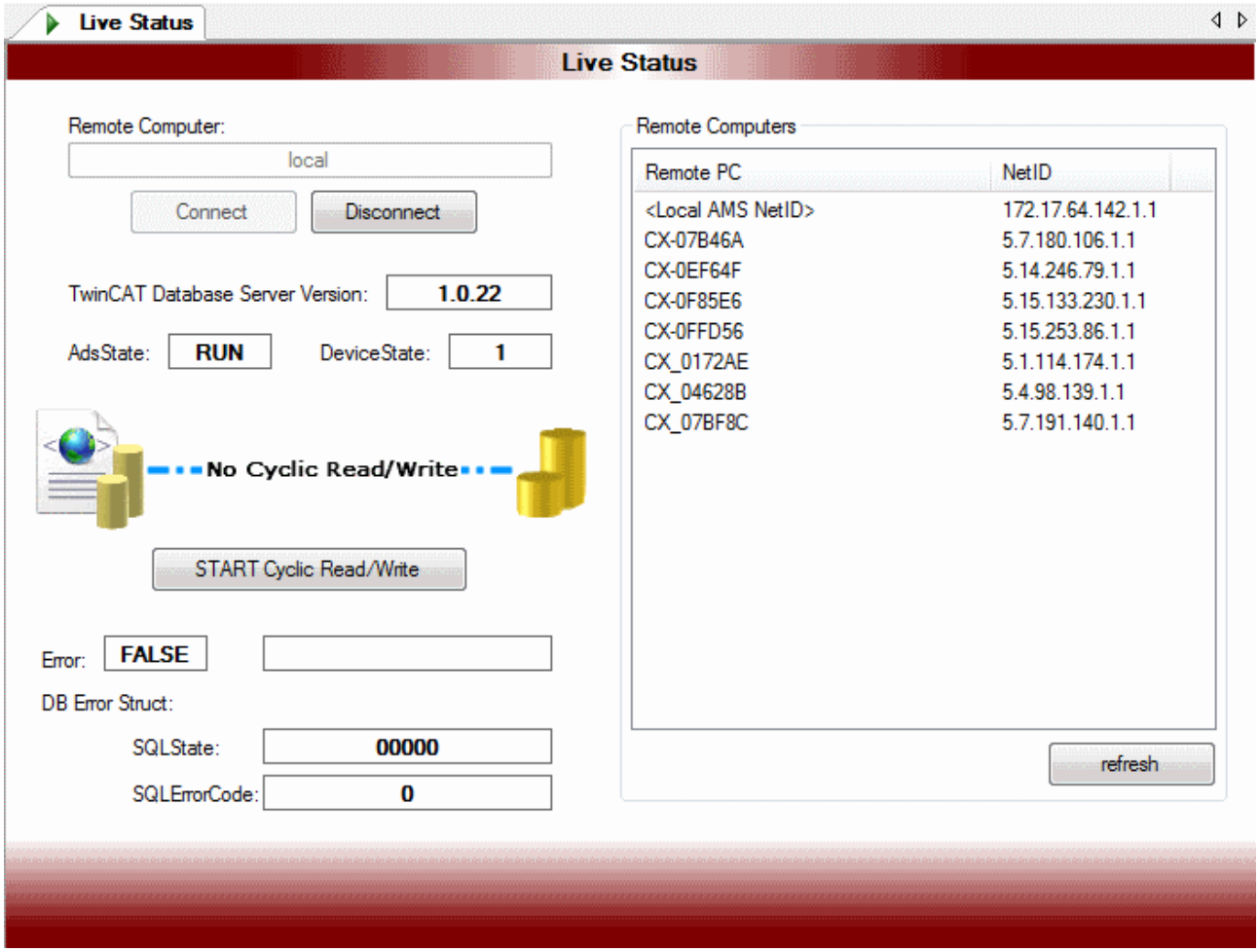
Eine weitere Möglichkeit besteht darin, die erzeugte Datei mit Hilfe des "Upload"-Dialogs zu aktivieren. Es muss nur das gewünschte ADS-Gerät ausgewählt und der Vorgang mit dem Upload Button gestartet werden. Die XML-Konfigurationsdatei wird dann dem TwinCAT Database Server übermittelt, ins Boot-Verzeichnis gespeichert und eingelesen.

Um den Service nutzen zu können muss auf dem Host PC, auf dem der XML-Konfigurationsdatei Editor läuft, TwinCAT installiert und im RUN-Modus sein. Des Weiteren muss der TwinCAT Database Server auf dem entsprechenden ADS-Gerät gestartet sein.



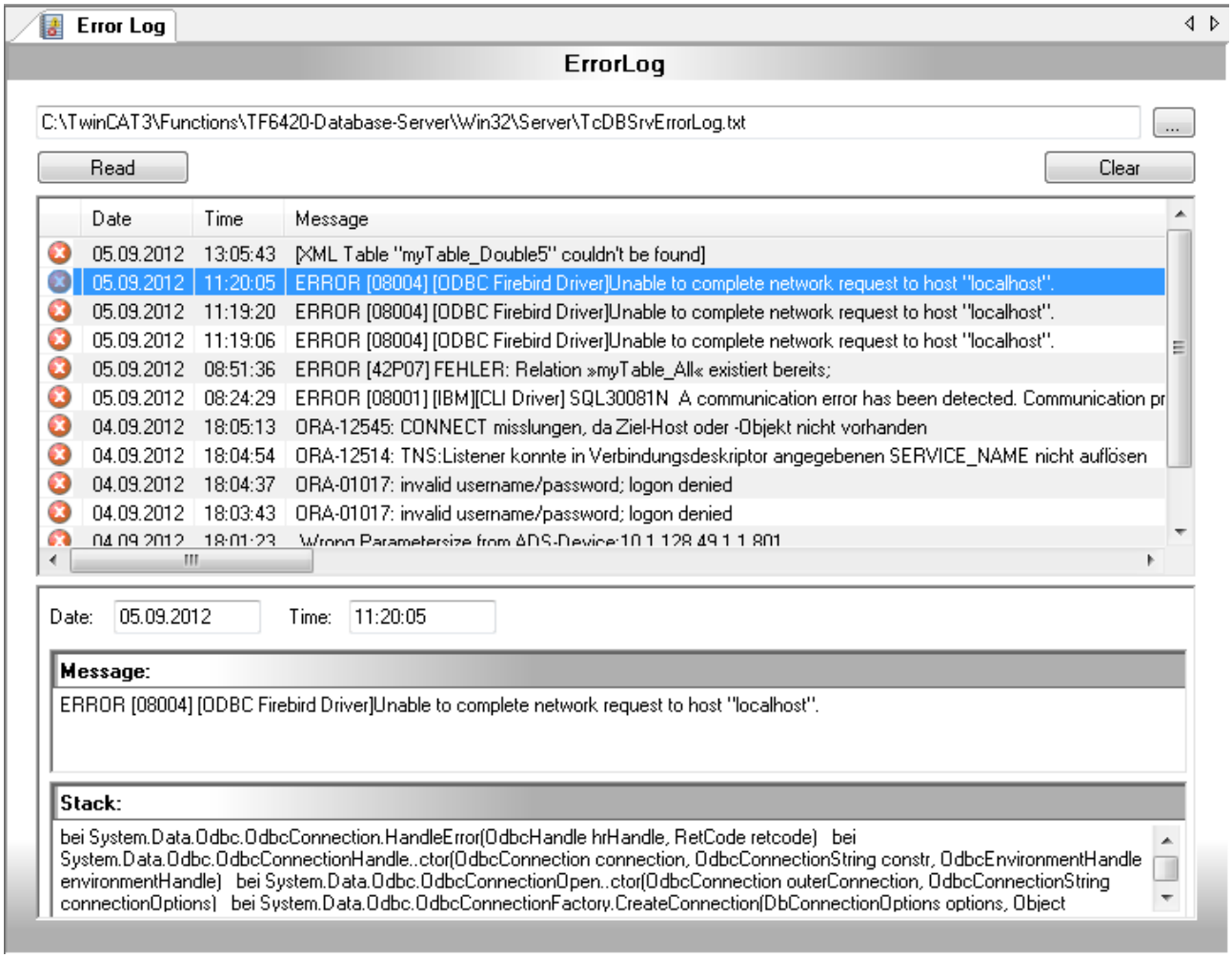
Live Status Dialog des TwinCAT Database Server

In diesem Dialog wird der aktuelle Status des Database Servers angezeigt. Es ist auch möglich den Status des Database Servers eines Remote Computers zu überwachen. Zusätzlich besteht die Möglichkeit die zyklische Log Funktion zu starten oder stoppen. Wenn ein Fehler während des zyklischen Log Vorgangs auftreten sollte, würde, der sqlstate und der Fehlercode vom aufgetretenen Fehler angezeigt.



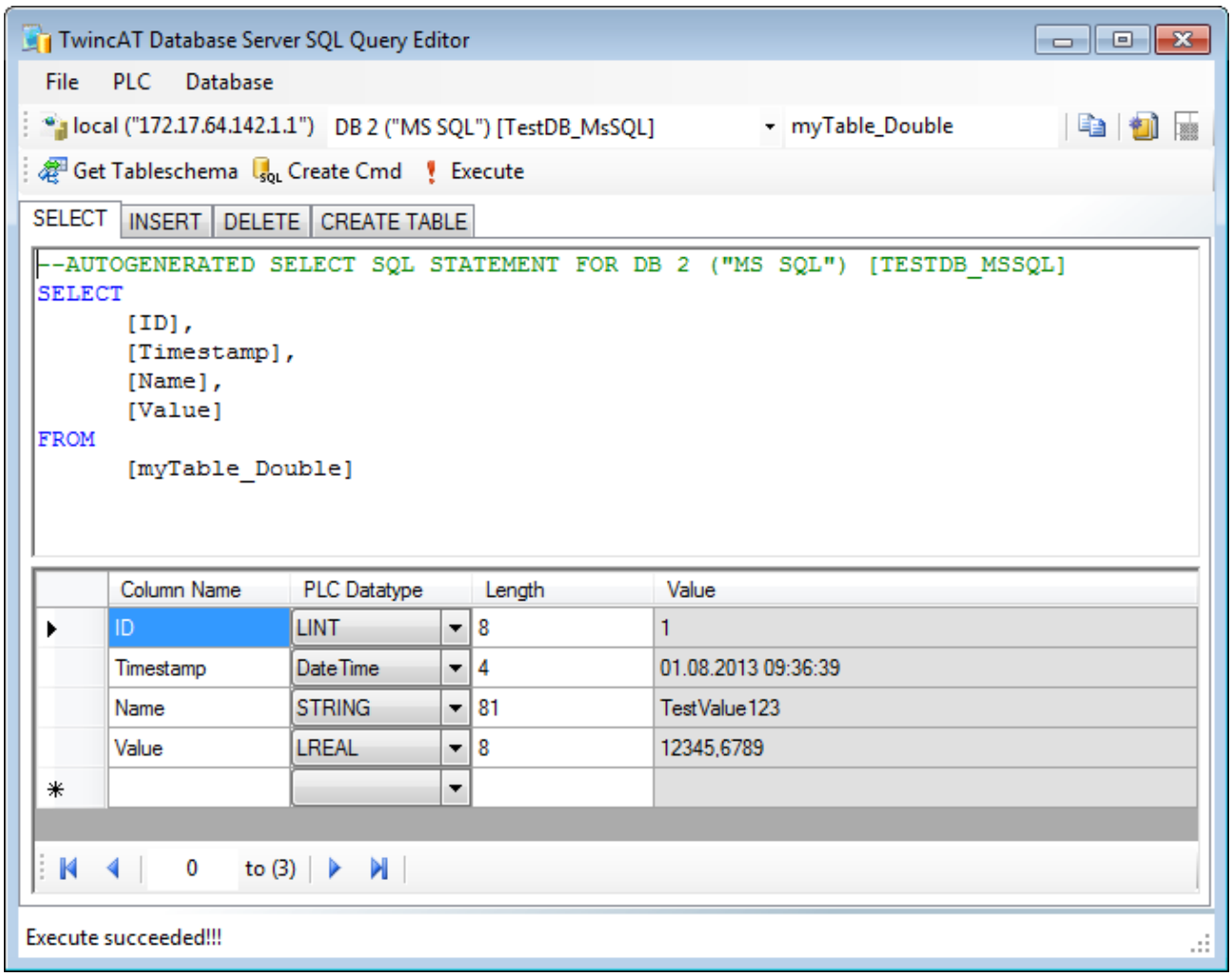
Error Log (TcDBSrvErrorLog.txt)

In diesem Dialog werden alle Einträge der Errorlogdatei übersichtlich aufgelistet. Zusätzlich besteht die Möglichkeit die Errorlogdatei zu leeren.



SQL Query Editor

Der SQL Query Editor hilf beim Erzeugen von SQL Befehlen und testen der konfigurierten Datenbankverbindungen. Weiter Informationen hier [▶ 27]



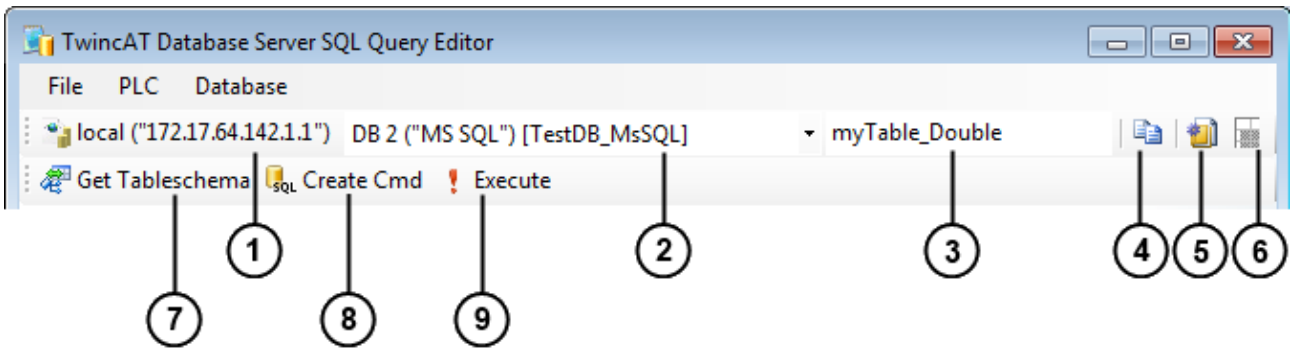
6.2 SQL Query Editor

Der TwinCAT Database Server verbindet zwei Welten miteinander. Zum einen das Know How von Automations-Ingenieuren und das der IT Datenbank Administratoren. Um beide Welten besser miteinander zu vereinen, wurde der SQL Query Editor entwickelt. In diesem können SQL-Befehle generiert und mit den Standard SPS Bausteinen des TwinCAT Database Servers getestet werden. Zusätzlich bietet er die Möglichkeit die erzeugten Befehle und Strukturen nach TwinCAT zu kopieren und exportieren.

Es Besteht die Möglichkeit Datensätze zu selektieren, integrieren, löschen, und sogar ganze Tabellen zu erzeugen. Beim Generieren von SQL-Befehlen wird auch die Datenbank spezifische Syntax berücksichtigt.

Die Unterschiedlichen Funktionen werden mit Hilfe der Standard ADS Funktionsbausteine, der TwinCAT Database Server SPS Bibliothek, und den bereits konfigurierten Datenbankverbindungen ausgeführt.

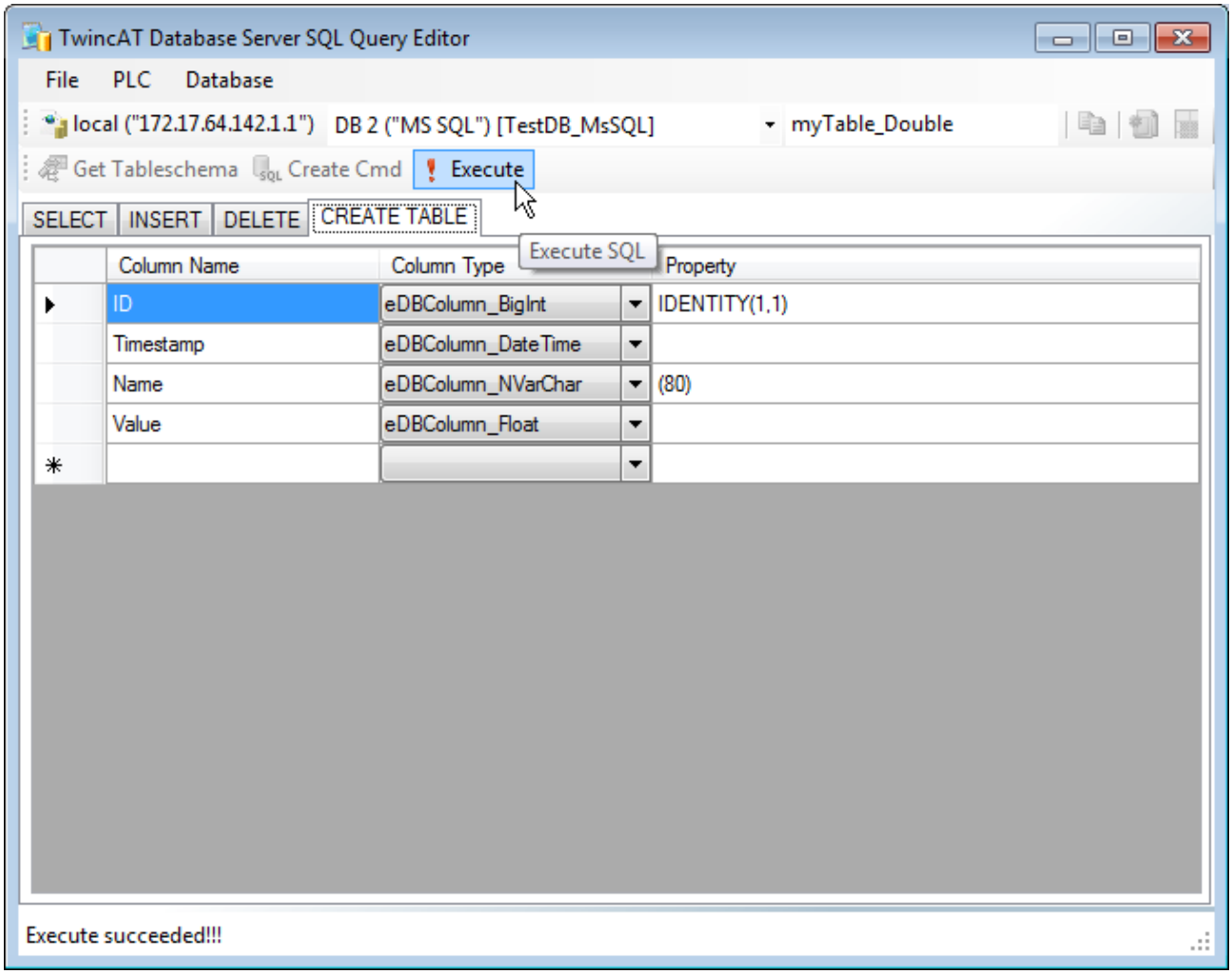
Die Menüleiste



		Beschreibung
1	**Ziel TwinCAT Database Server**	Auswahl des Ziel TwinCAT Database Server mit dem kommuniziert werden soll.
2	**Datenbank**	Auswahl der zu verwendende Datenbankkonfiguration
3	**Tabelle**	Tabellenname für die SQL-Kommandoerzeugung
4	**Kopieren für SPS**	Kopiert den erzeugten SQL-Befehl in korrekter SPS Syntax in die Zwischenablage
5	**Export TC2**	Erzeugt eine Exportdatei der SELECT SPS Struktur für TwinCAT 2
6	**Export TC3**	Erzeugt eine Exportdatei der SELECT SPS Struktur für TwinCAT 3
7	**Lese Tabellenschema**	Liest die Tabellenstruktur der angegebenen Tabelle aus
8	**Erzeuge SQL**	Erzeugt SQL-Befehle unter Berücksichtigung der unterschiedlichen Datenbanksyntax
9	**Ausführen**	Führt den erzeugten SQL-Befehl aus

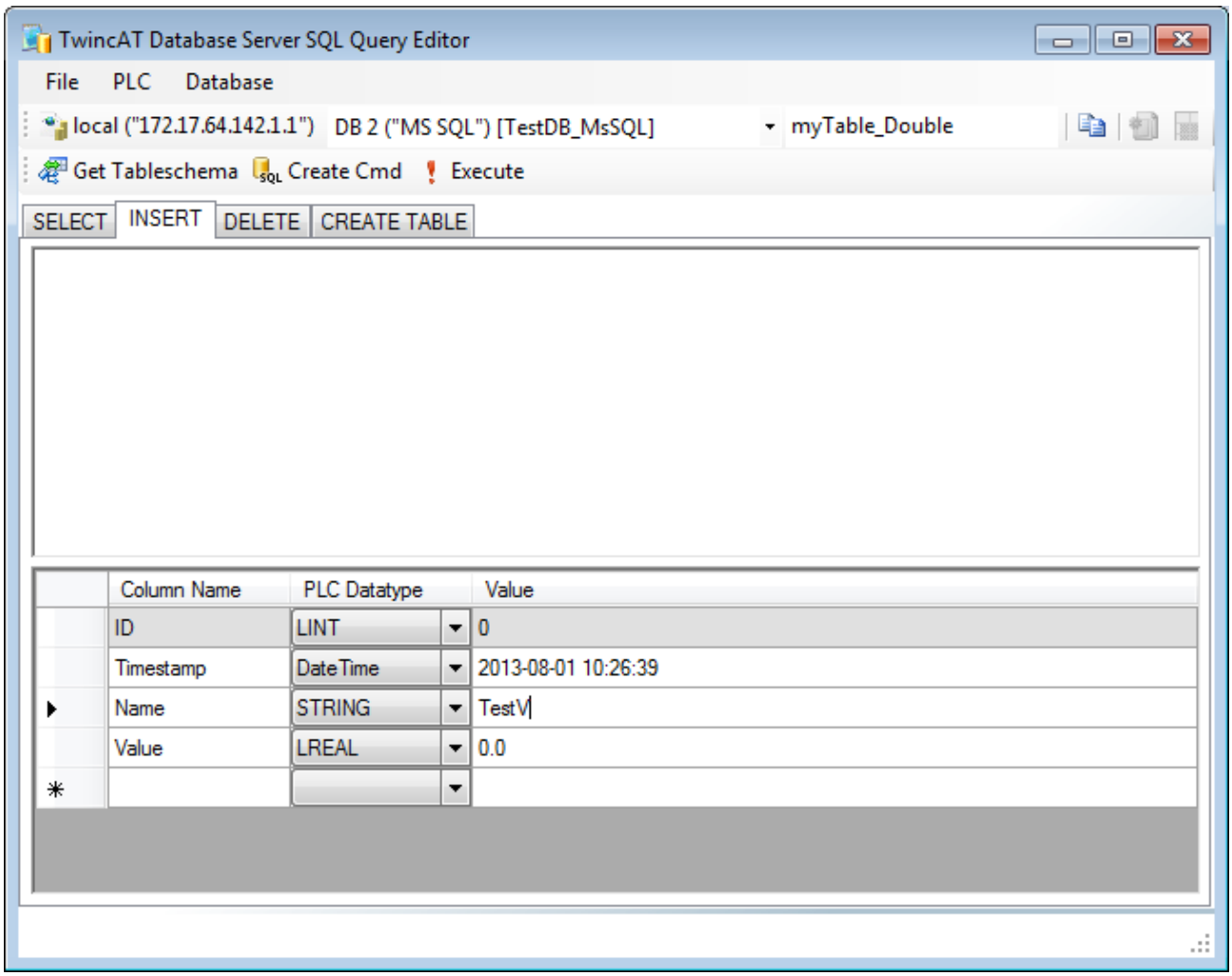
SQL "CREATE TABLE"

Mit dem Karteireiter "CREATE TABLE" können beliebige Tabellen an der ausgewählten Datenbank erzeugt werden. Intern wird dies über den Baustein FB_DBTableCreate durchgeführt. Im Tabellentextfeld (3) kann der Name der neu zu erzeugender Tabelle angegeben werden.

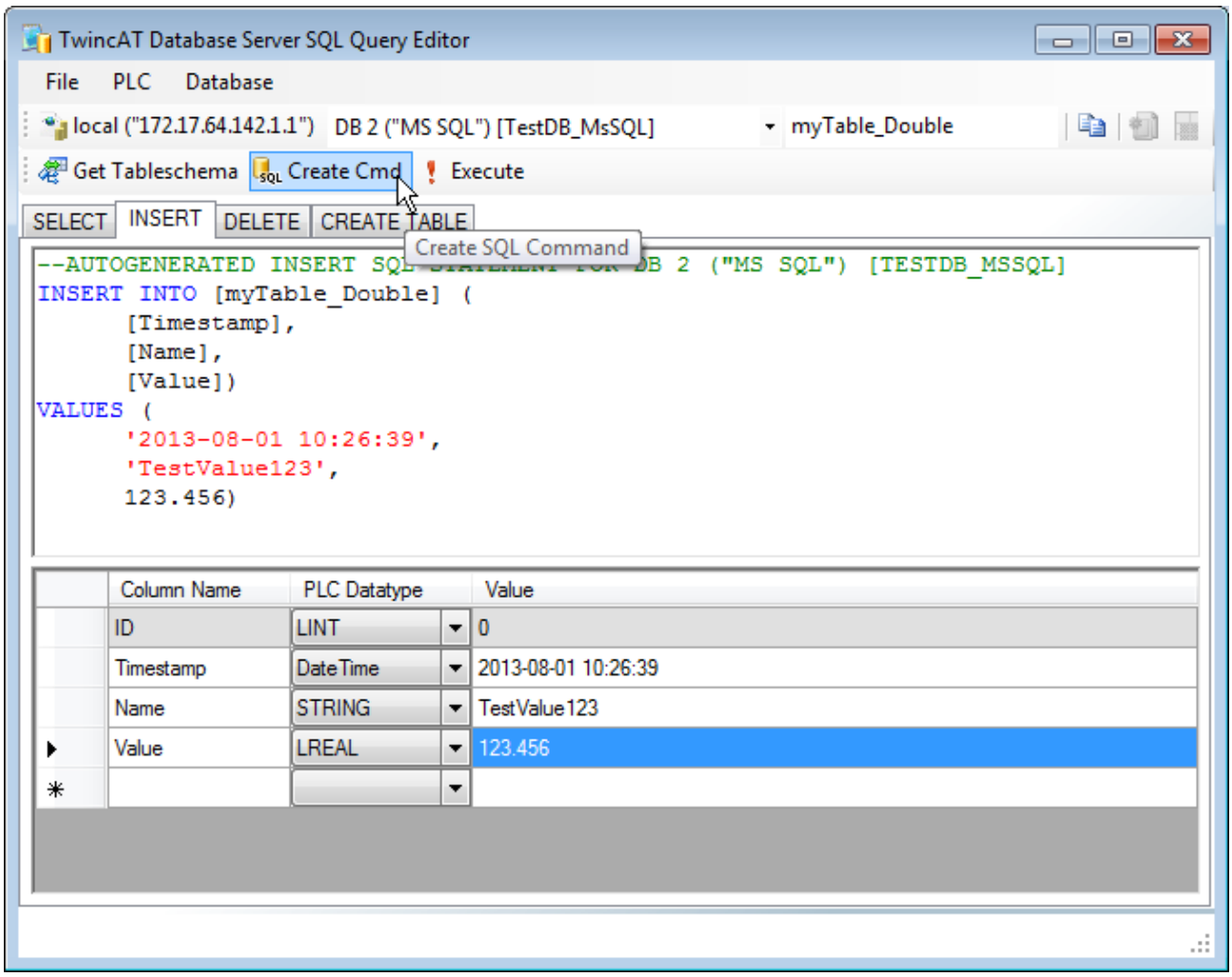


SQL "INSERT" Befehle

Mit dem Karteireiter "INSERT" können INSERT-SQL-Befehle auf einfache und schnelle Art und Weise generiert werden. Es kann entweder die einzelnen Spalten der Tabelle im unteren Bereich selbst erzeugt werden, oder mit Hilfe des "Get Tableschema" Buttons die komplette Tabellenstruktur ausgelesen werden.

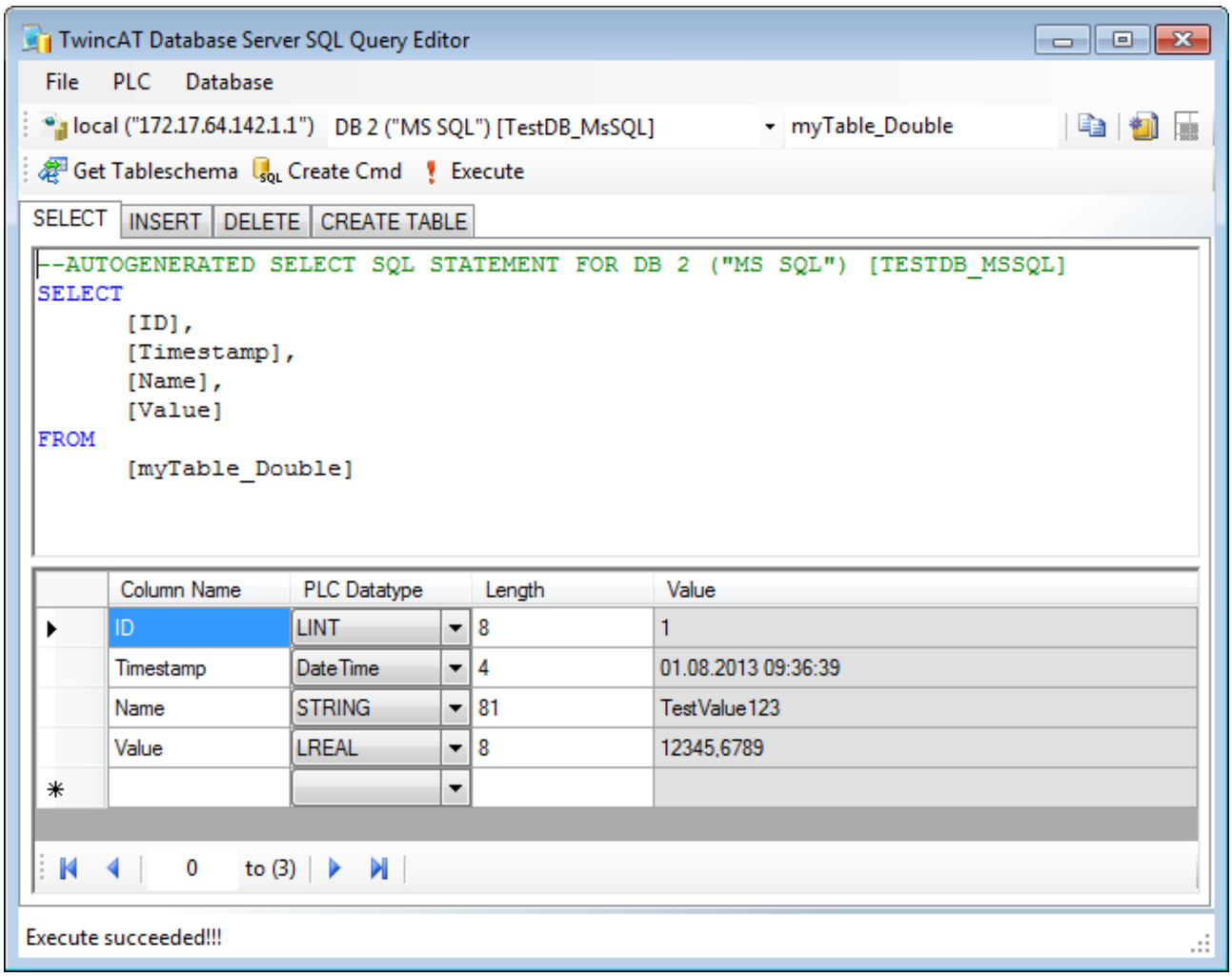


Wenn alle Spaltenwerte eingegeben wurden, und der Button "Create Cmd" gedrückt wird, wird der INSERT-SQL-Befehl in der richtigen Datenbanksyntax erzeugt. Danach muss nur noch der Befehl mit dem "Execute" Button abgesetzt werden. Intern wird für diese Funktion der SPS Baustein FB_DBRecordINSERT_EX verwendet.



SQL "SELECT" Befehle

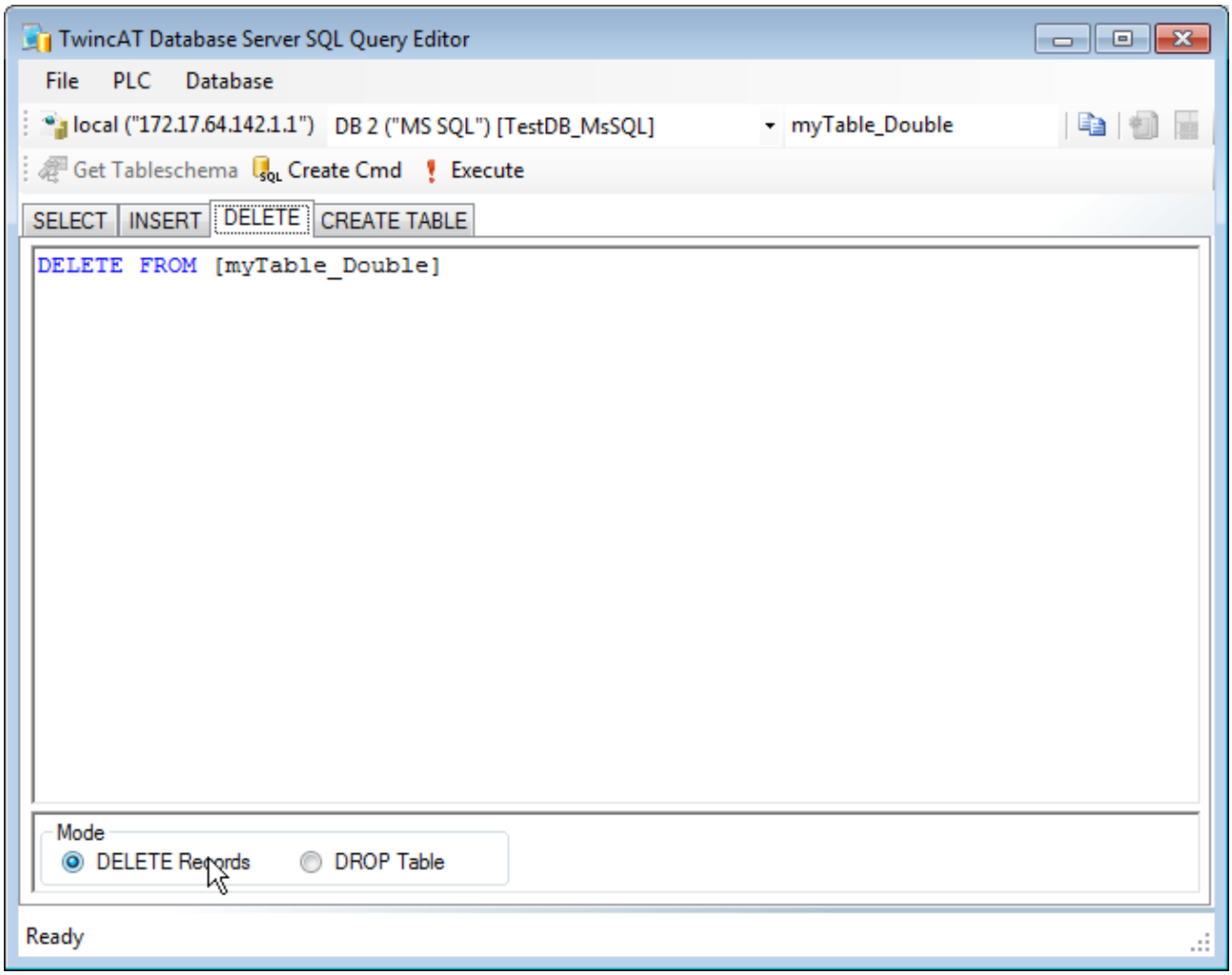
Mit dem Karteireiter "SELECT" können Datensätze einer Tabelle ausgelesen werden. Hierfür muss im unteren Bereich eine Struktur mit Standard SPS Datentypen erzeugt werden, welche die ausgelesenen Daten aufnimmt. Dies kann von Hand, oder mit Hilfe des "Get Tableschema" Buttons erfolgen. Die erzeugte Struktur kann dann mit Hilfe der Export Buttons in das jeweilige TwinCAT SPS Projekt importiert werden. Intern wird zum absetzen des erzeugten SELECT-SQL-Befehlt der SPS Baustein FB_DBRecordArraySelect verwendet.



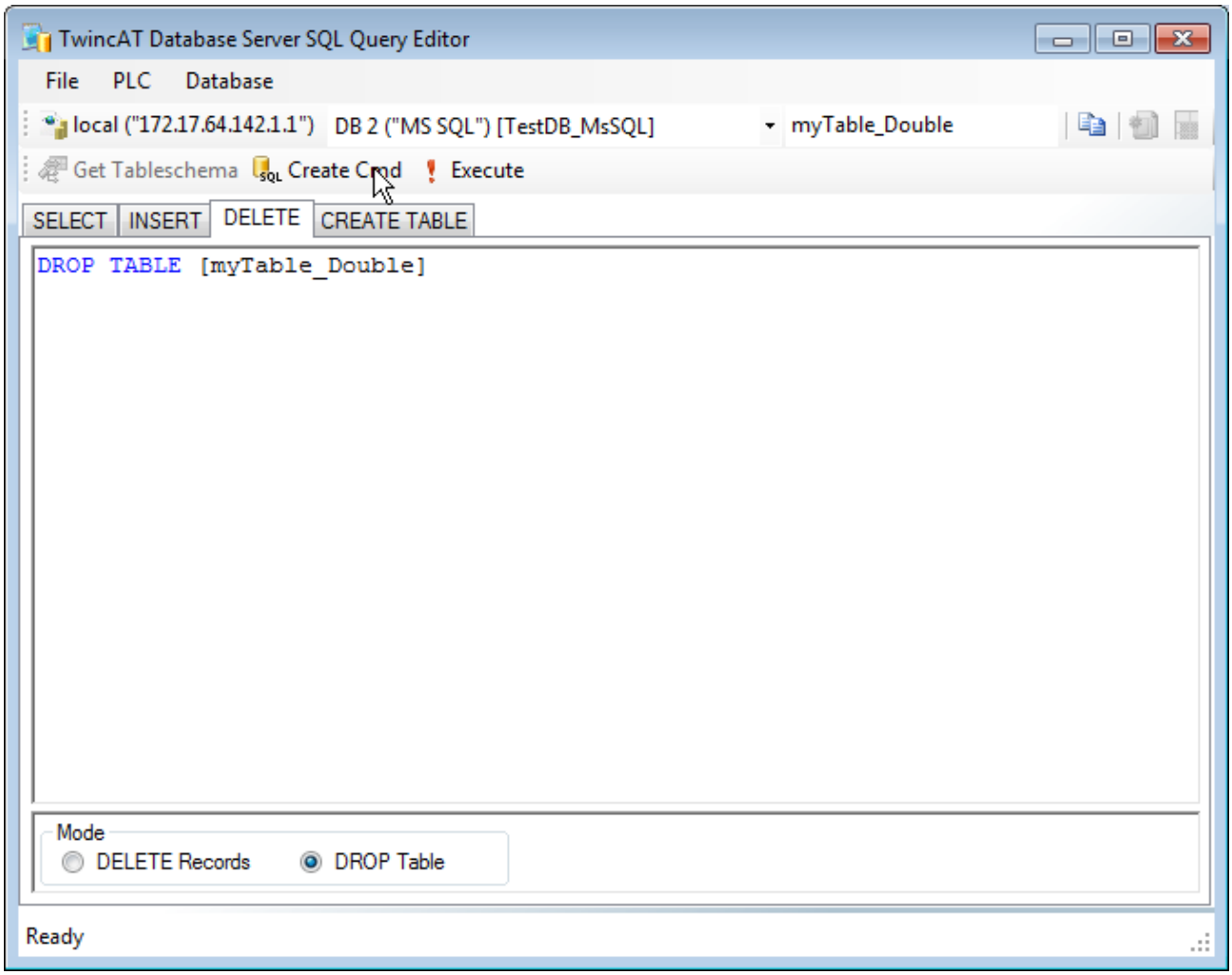
SQL "DELETE" Befehle

Der SQL Query Editor besitz auch die Möglichkeit Datensätze und sogar ganze Tabelle zu löschen. Zu diesem Zweck gibt es den Karteireiter "DELETE". Hier können DELETE-SQL-Befehlt generiert und abgesetzt werden. Intern wird dies über den Baustein FB_DBRecordDelete umgesetzt.

DELETE Datensätze



DELETE Tabellen



6.3 Schreibrichtungsmodus

Der TwinCAT Database Server verfügt über vier verschiedene Schreibrichtungsmodi, welche im Folgenden erläutert werden.

DB_TO_ADS

Mit diesem Schreibmodus werden zyklisch Variablenwerte aus einer Datenbank ausgelesen und die gelesen Werte in Variablen der SPS geschrieben.

ADS_TO_DB_APPEND

Mit diesem Schreibmodus werden Variablenwerte zyklisch aus der SPS in eine Datenbank geschrieben. Es wird jedes Mal ein neuer Datensatz erzeugt und am Ende der Tabelle / Datei angehängt.

ADS_TO_DB_UPDATE

Mit diesem Schreibmodus werden Variablenwerte zyklisch aus der SPS ausgelesen und die ausgelesenen Werte mit den Datensätzen der Datenbank verglichen. Bei Wertunterschieden wird dann der entsprechende Datensatz mit dem neuen Wert geändert.

ADS_TO_DB_RINGBUFFER

Mit diesem Schreibmodus kann die Anzahl an Datensätzen bzw. das Alter von Datensätzen festgelegt werden.
Dieser Schreibmodus ist verfügbar beim zyklischen Loggen mit Hilfe der Symbolgruppen und beim Loggen mit dem Funktionsbaustein FB_DBWrite.

Der "RingBuffer" Modus ist für alle Datenbanktypen verfügbar. Auch das Loggen in ASCII-Dateien kann mit diesem Modus beeinflusst werden.
"RingBuffer"-Arten:
Der RingBuffer kann auf zwei verschiedene Arten benutzt werden:

- "RingBuffer_Time"
- "RingBuffer_Count"

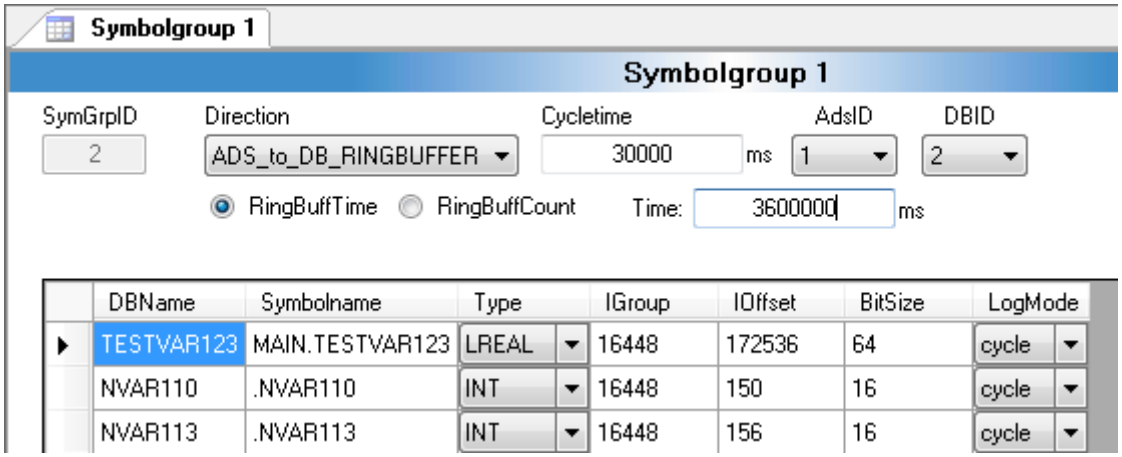
RingBuffer "Time":
Bei diesem Modus kann eine Zeit angegeben werden, die das Alter eines Datensatzes festlegt. Wird dieses Alter überschritten, wird der betroffene Datensatz gelöscht.

RingBuffer "Count":
Bei diesem Modus kann eine Maximalanzahl an Datensätzen festgelegt werden. Ist die Maximalanzahl erreicht, werden die ältesten Datensätze gelöscht, um Platz für die neuen zu schaffen.

Deklarieren des RingBuffer Modus:

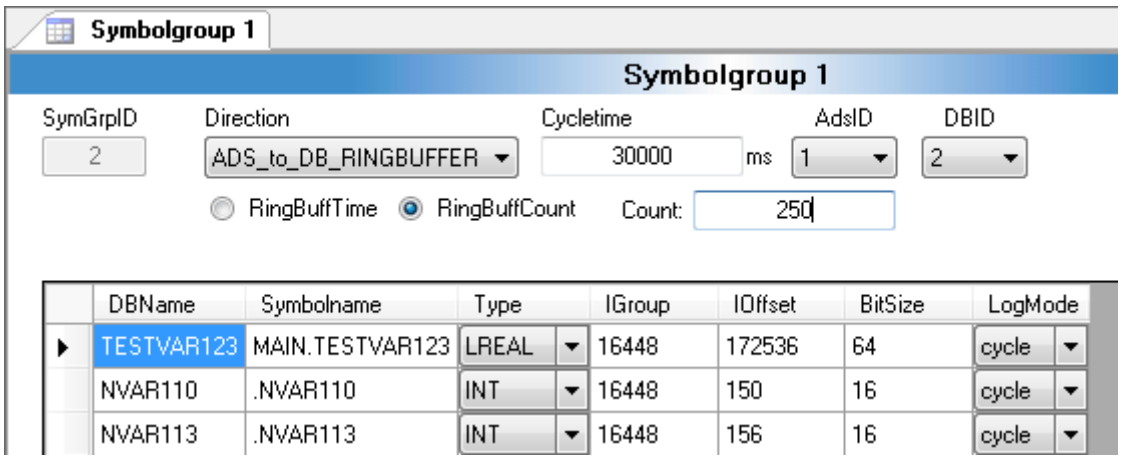
Tab. 1: XML-Konfiguration file editor:

RingBuffer_Time



Die Zeit wird in Millisekunden angegeben.

RingBuffer_Count



Tab. 2: FB_DBWrite:

RingBuffer_Time

```
FB_DBWrite1(
    sNetID:= ,
    hDBID:= 1,
    hAdsID:= 1,
    sVarName:= 'MAIN.DIMM1',
    sDBVarName:= 'DIMM1',
    eDBWriteMode:= eDBWriteMode_RingBuffer_Time,
    tRingBufferTime:= T#1h,
    bExecute:= TRUE,
    tTimeout:= T#15s,
    bBusy=> busy,
    bError=> err,
    nErrID=> errid);
```

RingBuffer_Count

```
FB_DBWrite1(
    sNetID:= ,
    hDBID:= 1,
    hAdsID:= 1,
    sVarName:= 'MAIN.DIMM1',
    sDBVarName:= 'DIMM1',
    eDBWriteMode:= eDBWriteMode_RingBuffer_Count,
    nRingBufferCount:= 250,
    bExecute:= TRUE,
    tTimeout:= T#15s,
    bBusy=> busy,
    bError=> err,
    nErrID=> errid);
```

6.4 Eigenschaften und Verwendung der Konfigurationsdatei

Speicherort der XML-Konfigurationsdatei:

Der Speicherort der Konfigurationsdatei ist festgelegt.

- Unter CE liegt die Konfigurationsdatei im Ordner "\Hard Disk\TwinCAT\Boot" (Werden Änderungen mit dem XML-Konfigurationsdateieditor vorgenommen, muss die Konfigurationsdatei immer wieder neu in diesen Ordner kopiert werden.)
- Bei PCs liegt die Konfigurationsdatei im Ordner "C:\TwinCAT\Boot"

Laden der XML-Konfigurationsdatei:

Die Konfigurationsdatei wird beim Starten der TcDatabaseSrv.exe einmalig eingelesen.
Durch den Funktionsbaustein **"FB_DBReloadConfig"** kann die Konfigurationsdatei nachträglich neu eingelesen werden. (Dies ist nur möglich, wenn das zyklische Lesen/Schreiben nicht gestartet ist)
Nach jedem Neustart des TwinCAT Systems wird die XML-Konfigurationsdatei neu geladen.

Modus 1 (StartUp = "AutoStart")

Beim Starten der TcDatabaseSrv.exe wird automatisch die XML-Konfigurationsdatei eingelesen. Da der Tag StartUp in der Konfigurationsdatei mit dem Wert "AutoStart" belegt ist, beginnt der Database Server sofort, die Verbindung zu allen deklarierten Datenbanken und ADS-Geräten herzustellen. Außerdem werden alle Variablen, die in den Symbolgruppen beschrieben sind, in die entsprechende Datenbank geloggt bzw. mit den Werten aus der entsprechenden Datenbank beschrieben. Dieser Vorgang wird mit dem in der Konfigurationsdatei angegebenen Zykluszeit durchgeführt. Der Vorgang wird so lange fortgesetzt, bis der Database Server aus der SPS heraus mit Hilfe des Funktionsblocks **"FB_DBCyclicRdWrt"** gestoppt wird.

Modus 2 (StartUp = "Manual")

Beim Starten der TcDatabaseSrv.exe wird automatisch die XML-Konfigurationsdatei eingelesen. Da der Tag StartUp in der Konfigurationsdatei mit dem Wert "Manual" belegt ist, wird nichts weiter ausgeführt. Danach wartet der Database Server auf Anweisungen aus der SPS heraus.

"FB_DBCyclicRdWrt"

Alle Sektoren der Konfigurationsdatei werden für diesen Funktionsbaustein verwendet.
Verbindung zu den in der XML-Konfigurationsdatei deklarierten Datenbanken und ADS-Geräten wird hergestellt und das zyklische Loggen mit allen Symbolgruppen wird gestartet.

Für alle weiteren Funktionsblöcke werden nur die deklarierten Datenbanken und ADS-Geräte aus der Konfigurationsdatei verwendet. Die deklarierten Symbolgruppen werden nicht beachtet. Wie z.B. beim folgenden Funktionsblock

"FB_DBWrite"

Es wird eine Verbindung zu der ausgewählten Datenbank (hDBID) und dem ausgewählten ADS-Gerät (hAdsID) hergestellt. Danach wird die im Funktionsblock angegebene Variable vom ADS-Gerät gelesen und in die Datenbank geloggt.

6.5 Datenbanken

6.5.1 Deklarieren der verschiedenen Datenbanktypen

Mircosoft SQL Datenbank [▶ 40]	
-DBValueType:	Wollen Sie nur Alphanummerische Datentypen loggen und Boolean wählen Sie **"Double"** aus. Möchten Sie auch Strukturen und Strings mitloggen wählen Sie **"Bytes"** aus
-DBType:	Wählen Sie **"MS SQL"** aus. →PLC: **eDBType_Sequal_Server**.
-DBServer:	Geben Sie hier den Namen ihres SQL-Servers an. Z.B. ("TESTSERVER\SQLEXPRESS")
-DBProvider:	**"SQLOLEDB"** oder der Provider des SQL Native Clients z.B. **"SQLNCLI10"**
-DBName	DBName enthält den Namen der Datenbank
-DBTable:	DBTable enthält den Namen der Tabelle.

Mircosoft Compact SQL Datenbank [▶ 41]	
-DBValueType:	Wollen Sie nur Alphanummerische Datentypen loggen und Boolean wählen Sie **"Double"** aus. Möchten Sie auch Strukturen und Strings mitloggen wählen Sie **"Bytes"** aus
-DBType:	Wählen Sie **"MS Compact SQL"** aus. →PLC: **eDBType_Mobile_Server**.
-DBServer:	Wird nicht benötigt.
-DBProvider:	Wird nicht benötigt.
-DBUrl	DBUrl enthält den Pfad zu der SDF-Datei. Z.B. ("C:\TwinCAT\TcDatabaseSrv\Samples\TestDB.sdf")
-DBTable:	DBTable enthält den Namen der Tabelle.

Microsoft Access Datenbank [▶ 41]	
-DBValueType:	Wollen Sie nur Alphanummerische Datentypen loggen und Boolean wählen Sie **"Double"** aus. Möchten Sie auch Strukturen und Strings mitloggen wählen Sie **"Bytes"** aus
-DBType:	Wählen Sie **"MS Access"** aus. →PLC: **eDBType_Access**.
-DBServer:	Wird nicht benötigt.
-DBProvider:	Access 2000 - Access 2003: Der Provider lautet **"Microsoft.Jet.OLEDB.4.0"**. Access 2007: Der Provider lautet **"Microsoft.ACE.OLEDB.12.0"**.

Microsoft Access Datenbank [▶ 41]	
-DBUrl	DBUrl enthält den Pfad zu der MDB-Datei. Z.B. ("C:\TwinCAT\TcDatabaseSrv\Samples\TestDB.mdb")
-DBTable:	DBTable enthält den Namen der Tabelle.

ASCII - Datei [▶ 44]	
-DBValueType:	Wollen Sie nur Alphanummerische Datentypen loggen und Boolean wählen Sie **"Double"** aus. Möchten Sie auch Strukturen und Strings mitloggen wählen Sie **"Bytes"** aus
-DBType:	Wählen Sie **"ASCII"** aus. →PLC: **eDBType_ASCII**.
-DBServer:	Wird nicht benötigt.
-DBProvider:	Wird nicht benötigt.
-DBUrl	DBUrl enthält den Pfad zu der ASC-Datei oder TXT-Datei oder CSV-Datei. Z.B. ("C:\TwinCAT\TcDatabaseSrv\Samples\TestDB.asc")
-DBTable:	Wird nicht benötigt. Es können keine Tabellen in ASCII-Dateien angelegt werden.

NET-MySQL Datenbank [▶ 43]	
-DBValueType:	Wollen Sie nur Alphanummerische Datentypen loggen und Boolean wählen Sie **"Double"** aus. Möchten Sie auch Strukturen und Strings mitloggen wählen Sie **"Bytes"** aus
-DBType:	Wählen Sie **"NET_MySQL"** aus. →PLC: **eDBType_NET_MySQL**.
-ODBC Driver:	Wird nicht benötigt.
-Server name:	Enthält den Namen des Servers bzw. die IP-Adresse/Namen des Hosts.
-Database name:	Enthält den Namen der Datenbank.
-Port:	Enthält den Port über den sich verbinden soll. (Standard 3306)
-Protocol:	Wird nicht benötigt.
-Scheme:	Wird nicht benötigt.
-Sequence:	Wird nicht benötigt.
-Table name:	Enthält den Namen der Tabelle in die geschrieben bzw. gelesen werden soll.
-UserId:	Enthält den Namen des Benutzers, der sich am Server anmelden soll.
-Password:	Enthält das Passwort, welches zur Authentifizierung verwendet werden soll.

ODBC-MySQL Datenbank [▶ 43]	
-DBValueType:	Wollen Sie nur Alphanummerische Datentypen loggen und Boolean wählen Sie **"Double"** aus. Möchten Sie auch Strukturen und Strings mitloggen wählen Sie **"Bytes"** aus
-DBType:	Wählen Sie **"ODBC_MySQL"** aus. →PLC: **eDBType_ODBC_MySQL**.
-ODBC Driver:	Fügen Sie den Namen des ODBC-Drivers ein. ("MySQL ODBC 3.51 Driver")
-Server name:	Enthält den Namen des Servers bzw. die IP-Adresse/Namen des Hosts.
-Database name:	Enthält den Namen der Datenbank.
-Port:	Enthält den Port, über den sich der ODBC-Treiber verbinden soll.
-Protocol:	Wird nicht benötigt.
-Scheme:	Wird nicht benötigt.
-Sequence:	Wird nicht benötigt.
-Table name:	Enthält den Namen der Tabelle in die geschrieben bzw. gelesen werden soll.
-UserId:	Enthält den Namen des Benutzers, der sich am Server anmelden soll.
-Password:	Enthält das Passwort, welches zur Authentifizierung verwendet werden soll.

ODBC-PostgreSQL Datenbank [▶ 46]	
-DBValueType:	Wollen Sie nur Alphanummerische Datentypen loggen und Boolean wählen Sie **"Double"** aus. Möchten Sie auch Strukturen und Strings mitloggen wählen Sie **"Bytes"** aus
-DBType:	Wählen Sie **"ODBC_PostgreSQL"** aus. →PLC: **eDBType_ODBC_PostgreSQL**.

ODBC-PostgreSQL Datenbank [▶ 46]	
-ODBC Driver:	Fügen Sie den Namen des ODBC-Drivers ein. ("PostgreSQL UNICODE")
-Server name:	Enthält den Namen des Servers bzw. die IP-Adresse/Namen des Hosts .
-Database name:	Enthält den Namen der Datenbank.
-Port:	Enthält den Port, über den sich der ODBC-Treiber verbinden soll.
-Protocol:	Wird nicht benötigt.
-Scheme:	Enthält den Namen des zu verwendenden Schemas.
-Sequence:	Wird nicht benötigt.
-Table name:	Enthält den Namen der Tabelle in die geschrieben bzw. gelesen werden soll.
-UserId:	Enthält den Namen des Benutzers, der sich am Server anmelden soll.
-Password:	Enthält das Passwort, welches zur Authentifizierung verwendet werden soll.

ODBC-DB2 Datenbank [▶ 47]	
-DBValueType:	Wollen Sie nur Alphanummerische Datentypen loggen und Boolean wählen Sie **"Double"** aus. Möchten Sie auch Strukturen und Strings mitloggen wählen Sie **"Bytes"** aus
-DBType:	Wählen Sie **"ODBC_DB2"** aus. →PLC: **eDBType_ODBC_DB2**.
-ODBC Driver:	Fügen Sie den Namen des ODBC-Drivers ein. ("IBM DB2 ODBC DRIVER")
-Server name:	Enthält den Namen des Servers bzw. die IP-Adresse/Namen des Hosts.
-Database name:	Enthält den Namen der Datenbank.
-Port:	Enthält den Port, über den sich der ODBC-Treiber verbinden soll.
-Protocol:	Enthält den Namen des Protokolls, mit dem die Verbindung hergestellt werden soll. ("TCPIP")
-Scheme:	Enthält den Namen des zu verwendenden Schemas.
-Sequence:	Wird nicht benötigt.
-Table name:	Enthält den Namen der Tabelle in die geschrieben bzw. gelesen werden soll.
-UserId:	Enthält den Namen des Benutzers, der sich am Server anmelden soll.
-Password:	Enthält das Passwort, welches zur Authentifizierung verwendet werden soll.

OCI-Oracle Datenbank (Oracle Calling Interface) [▶ 43]	
-DBValueType:	Wollen Sie nur Alphanummerische Datentypen loggen und Boolean wählen Sie **"Double"** aus. Möchten Sie auch Strukturen und Strings mitloggen wählen Sie **"Bytes"** aus
-DBType:	Wählen Sie **"OCI_Oracle"** aus. →PLC: **eDBType_OCI_Oracle**.
-ODBC Driver:	Wird nicht benötigt
-Server name:	Enthält den Hostnamen / IP-Adresse des Servers.
-Database name:	Enthält den Namen der Datenbank bzw. ServiceName.
-Port:	Port über den sich verbunden werden soll. (Standard: 1521)
-Protocol:	Protokol mit dem sich verbunden werden soll. (Standard: TCP)
-Scheme:	Enthält den Namen des zu verwendenden Schemas.
-Sequence:	Enthält den Namen der zu verwendenden Sequenz.
-Table name:	Enthält den Namen der Tabelle in die geschrieben bzw. gelesen werden soll.
-UserId:	Enthält den Namen des Benutzers, der sich am Server anmelden soll.
-Password:	Enthält das Passwort, welches zur Authentifizierung verwendet werden soll.

ODBC-Oracle Datenbank [▶ 43]	
-DBValueType:	Wollen Sie nur Alphanummerische Datentypen loggen und Boolean wählen Sie **"Double"** aus. Möchten Sie auch Strukturen und Strings mitloggen wählen Sie **"Bytes"** aus
-DBType:	Wählen Sie **"ODBC_Oracle"** aus. →PLC: **eDBType_ODBC_Oracle**.
-ODBC Driver:	Fügen Sie den Namen des ODBC-Drivers ein. ("Microsoft ODBC for Oracle")
-Server name:	Wird nicht benötigt.

ODBC-Oracle Datenbank [▶ 43]	
-Database name:	Enthält den Namen der Datenbank.
-Port:	Wird nicht benötigt.
-Protocol:	Wird nicht benötigt.
-Scheme:	Enthält den Namen des zu verwendenden Schemas.
-Sequence:	Enthält den Namen der zu verwendenden Sequenz.
-Table name:	Enthält den Namen der Tabelle in die geschrieben bzw. gelesen werden soll.
-UserId:	Enthält den Namen des Benutzers, der sich am Server anmelden soll.
-Password:	Enthält das Passwort, welches zur Authentifizierung verwendet werden soll.

ODBC-InterBase Datenbank [▶ 47]	
-DBValueType:	Wollen Sie nur Alphanummerische Datentypen loggen und Boolean wählen Sie **"Double"** aus. Möchten Sie auch Strukturen und Strings mitloggen wählen Sie **"Bytes"** aus
-DBType:	Wählen Sie **"ODBC_InterBase"** aus. →PLC: **eDBType_ODBC_InterBase**.
-ODBC Driver:	Fügen Sie den Namen des ODBC-Drivers ein. ("Firebird/InterBase(r) driver")
-Server name:	Enthält den Namen des Servers bzw. die IP-Adresse/Namen des Hosts.
-Database name:	Enthält den Namen der Datenbank.
-ClientDll:	Enthält den Pfad zur fbclient.dll.
-Table name:	Enthält den Namen der Tabelle in die geschrieben bzw. gelesen werden soll.
-UserId:	Enthält den Namen des Benutzers, der sich am Server anmelden soll.
-Password:	Enthält das Passwort, welches zur Authentifizierung verwendet werden soll.

ODBC-Firebird Datenbank [▶ 48]	
-DBValueType:	Wollen Sie nur Alphanummerische Datentypen loggen und Boolean wählen Sie **"Double"** aus. Möchten Sie auch Strukturen und Strings mitloggen wählen Sie **"Bytes"** aus
-DBType:	Wählen Sie **"ODBC_Firebird"** aus. →PLC: **eDBType_ODBC_Firebird**.
-ODBC Driver:	Fügen Sie den Namen des ODBC-Drivers ein. ("Firebird/InterBase(r) driver")
-Server name:	Enthält den Namen des Servers bzw. die IP-Adresse/Namen des Hosts.
-Database name:	Enthält den Namen der Datenbank.
-ClientDll:	Enthält den Pfad zur fbclient.dll.
-Table name:	Enthält den Namen der Tabelle in die geschrieben bzw. gelesen werden soll.
-UserId:	Enthält den Namen des Benutzers, der sich am Server anmelden soll.
-Password:	Enthält das Passwort, welches zur Authentifizierung verwendet werden soll.

XML Datenbank [▶ 45]	
-DBValueType:	Wollen Sie nur Alphanummerische Datentypen loggen und Boolean wählen Sie **"Double"** aus. Möchten Sie auch Strukturen und Strings mitloggen wählen Sie **"Bytes"** aus
-DBType:	Wählen Sie **"XML"** aus. →PLC: **eDBType_XML**.
-DBServer:	Enthält den Namen der Datenbank.
-DBProvider:	Wird nicht benötigt.
-DBUrl	DBUrl enthält den Pfad zu der XML-Datei. z.B. ("C:\TwinCAT\TcDatabaseSrv\Samples\TestDB.xml") Die XSD-Datei muss im gleichen Verzeichnis liegen und denselben Dateinamen tragen. z.B. ("C:\TwinCAT\TcDatabaseSrv\Samples\TestDB.xsd")
-DBTable:	DBTable enthält den Namen der Tabelle.

6.5.2 Microsoft SQL Datenbank

Die Variablenwerte werden in einer Microsoft SQL Datenbank gespeichert.

Kompatible Versionen: Microsoft SQL Datenbank 2000/2005/2008. Deklarationen siehe "Deklaration verschiedener Datenbanken" [▶ 36]

In folgender Tabellenstruktur werden die Variablenwerte gespeichert.

Spaltenname	Datentyp	Null zulässig	Eigenschaft
ID	bigint	nein	IDENTITY(1,1)
Timestamp	datetime	nein	
Name	ntext	nein	
ValueType="Double"			
Value	float	nein	
ValueType="Bytes"			
Value	varbinary	nein	

In der Spalte **„ID“** wird eine AutoID erzeugt. Das heißt der Wert in dieser Spalte wird immer um 1 erhöht. Diese Funktionalität ermöglicht die Eigenschaft IDENTITY.
In die Spalte **„Timestamp“** wird der Speicherzeitpunkt des Datensatzes gespeichert.
In der Spalte **„Name“** steht der Name der Variable.
In der Spalte **„Value“** wird der Wert der Variable gespeichert.

Mit folgendem SQL-Kommando wird die Tabelle erzeugt

```
/*ValueType="Double"*/
CREATE TABLE myTable(
    ID          bigint IDENTITY(1,1)   NOT NULL,
    Timestamp   datetime               NOT NULL,
    Name        ntext                  NOT NULL,
    Value       float                  NOT NULL
)

/*ValueType="Bytes"*/
CREATE TABLE myTable(
    ID          bigint IDENTITY(1,1)   NOT NULL,
    Timestamp   datetime               NOT NULL,
    Name        ntext                  NOT NULL,
    Value       varbinary              NOT NULL
)
```

E_DBColumnTypes	MS SQL	PLC Control
eDBColumn_BigInt	bigint	T_ULARGE_INTEGER (TcUtilities.lib)
eDBColumn_Integer	integer	DINT
eDBColumn_SmallInt	smallint	INT
eDBColumn_TinyInt	tinyint	SINT
eDBColumn_Bit	bit	BYTE
eDBColumn_Money	money	LREAL
eDBColumn_Float	float	LREAL
eDBColumn_Real	real	REAL
eDBColumn_DateTime	datetime	DT
eDBColumn_NText	ntext	STRING
eDBColumn_NChar	nchar	STRING
eDBColumn_Image	image	ARRAY OF BYTE
eDBColumn_NVarChar	nvarchar	STRING
eDBColumn_Binary	binary	ARRAY OF BYTE
eDBColumn_VarBinary	varbinary	ARRAY OF BYTE

6.5.3 Microsoft Access Datenbank

Die Variablenwerte werden in einer Microsoft Access Datenbank gespeichert.

Datenbankdateien von Access 2000 und Access 2003 (*.mdb) sind ebenso kompatibel wie die Datenbankdateien von Access 2007 (*.accdb). Es müssen nur unterschiedliche Provider bei der Deklaration in der XML - Konfigurationsdatei angegeben werden. siehe "Deklaration verschiedener Datenbanken" [▶ 36]

In folgender Tabellenstruktur werden die Variablenwerte gespeichert.

Spaltenname	Datentyp	Eigenschaft
ID	AutoNumber	Field Size:= "Long Integer"; New Values:= "Increment"
Timestamp	Date/Time	Format:= "General Date"
Name	Text	
ValueType="Double"		
Value	Number	Field Size:= "Double"
ValueType="Bytes"		
Value	OLE Object	

In der Spalte **„ID“** wird eine AutoID erzeugt. Das heißt der Wert in dieser Spalte wird immer um 1 erhöht.
In die Spalte **„Timestamp“** wird der Speicherzeitpunkt des Datensatzes gespeichert.
In der Spalte **„Name“** steht der Name der Variable.
In der Spalte **„Value“** wird der Wert der Variable gespeichert.

E_DBColumnTypes	MS Access	PLC Control
eDBColumn_BigInt	Integer4	DINT
eDBColumn_Integer	Integer2	INT
eDBColumn_SmallInt	Integer2	SINT
eDBColumn_TinyInt	Integer1	SINT
eDBColumn_Bit	YESNO	BYTE
eDBColumn_Money	Currency	LREAL
eDBColumn_Float	Double	LREAL
eDBColumn_Real	Single	REAL
eDBColumn_DateTime	DATETIME	DT
eDBColumn_NText	Text	STRING
eDBColumn_NChar	VarChar	STRING
eDBColumn_Image	OLEOBJECT	ARRAY OF BYTE
eDBColumn_NVarChar	VarChar	STRING
eDBColumn_Binary	OLEOBJECT	ARRAY OF BYTE
eDBColumn_VarBinary	OLEOBJECT	ARRAY OF BYTE

Wichtig !
Bei Embedded Systemen die Datenbank nicht auf der Compact Flash Karte speichern.
Entweder die Datenbank im RAM benutzen, also nicht im Ordner "Hard Disk" speichern, oder in einen Netzwerkordner ablegen. Zu viele Schreibzyklen auf der Compact Flash Karte können die Lebensdauer der Compact Flash Karte minimieren.

6.5.4 SQL Compact Datenbank

Die Variablenwerte werden in einer Microsoft SQL-Compact Datenbank gespeichert.
Microsoft SQL Server 2005 Compact Edition stellt eine kompakte Datenbank dar, die ideal zur Einbettung in mobile und Desktopanwendungen geeignet ist. Diese bietet Entwicklern ein mit anderen Editionen vom SQL Server gemeinsames Programmiermodell zum Entwicklung systemeigener und verwalteter Anwendungen.

Dieser SQL Server hat einen geringen Ressourcenbedarf und stellt dennoch die notwendige Funktionalität für relationale Datenbanken bereit, wie etwa einen robusten Datenspeicher, einen optimierenden Abfrageprozessor, und zuverlässige, skalierbare Verbindungsfunktionen.

Kompatible Version: Microsoft Compact SQL Datenbank 3.5

In folgender Tabellenstruktur werden die Variablenwerte gespeichert.

Spaltenname	Datentyp	Null zulässig	Eigenschaft
ID	bigint	nein	IDENTITY(1,1)
Timestamp	datetime	nein	
Name	ntext	nein	
ValueType = "Double"			
Value	float	nein	
ValueType = "Bytes"			
Value	image	nein	

In der Spalte **„ID“** wird eine AutoID erzeugt. Das heißt der Wert in dieser Spalte wird immer um 1 erhöht. Diese Funktionalität ermöglicht die Eigenschaft IDENTITY.
In die Spalte **„Timestamp“** wird der Speicherzeitpunkt des Datensatzes gespeichert.
In der Spalte **„Name“** "steht der Name der Variable.
In der Spalte **„Value“** wird der Wert der Variable gespeichert.

Mit folgendem SQL-Kommando wird die Tabelle erzeugt

```
/* ValueType = "Double"*/
CREATE TABLE myTable(
    ID          bigint IDENTITY(1,1)   NOT NULL,
    Timestamp   datetime               NOT NULL,
    Name        ntext                  NOT NULL,
    Value       float                  NOT NULL
)

/*ValueType = "Bytes"*/
CREATE TABLE myTable(
    ID          bigint IDENTITY(1,1)   NOT NULL,
    Timestamp   datetime               NOT NULL,
    Name        ntext                  NOT NULL,
    Value       image                  NOT NULL
)
```

E_DBColumnTypes	MS Compact SQL	PLC Control
eDBColumn_BigInt	bigint	T_ULARGE_INTEGER (TcUtilities.lib)
eDBColumn_Integer	integer	DINT
eDBColumn_SmallInt	smallint	INT
eDBColumn_TinyInt	tinyint	SINT
eDBColumn_Bit	bit	BYTE
eDBColumn_Money	money	LREAL
eDBColumn_Float	float	LREAL
eDBColumn_Real	real	REAL
eDBColumn_DateTime	datetime	DT
eDBColumn_NText	ntext	STRING
eDBColumn_NChar	nchar	STRING
eDBColumn_Image	image	ARRAY OF BYTE
eDBColumn_NVarChar	nvarchar	STRING
eDBColumn_Binary	binary	ARRAY OF BYTE
eDBColumn_VarBinary	varbinary	ARRAY OF BYTE

Wichtig !
Bei Embedded Systemen die Datenbank nicht auf der Compact Flash Karte speichern.
Entweder die Datenbank im RAM benutzen, also nicht im Ordner "Hard Disk" speichern, oder in einen Netzwerkordner ablegen. Zu viele Schreibzyklen auf der Compact Flash Karte können die Lebensdauer der Compact Flash Karte minimieren.

6.5.5 ODBC - MySQL Datenbank

Die Variablenwerte werden in einer MySQL Datenbank gespeichert.

In folgender Tabellenstruktur werden die Variablenwerte gespeichert.

Spaltenname	Datentyp	Null zulässig	Eigenschaft
ID	INTEGER	nein	IDENTITY(1,1)
Timestamp	DATETIME	nein	
Name	VARCHAR(50)	nein	
ValueType="Double"			
Value	DOUBLE	nein	
ValueType="Bytes"			
Value	BLOB	nein	

In der Spalte **„ID“** wird eine AutoID erzeugt. Das heißt der Wert in dieser Spalte wird immer um 1 erhöht. Diese Funktionalität ermöglicht die Eigenschaft IDENTITY.
In die Spalte **„Timestamp“** wird der Speicherzeitpunkt des Datensatzes gespeichert.
In der Spalte **„Name“** steht der Name der Variable.
In der Spalte **„Value“** wird der Wert der Variable gespeichert.

E_DBColumnTypes	MySQL	PLC Control
eDBColumn_BigInt	BIGINT	T_ULARGE_INTEGER (TcUtilities.lib)
eDBColumn_Integer	INT	DINT
eDBColumn_SmallInt	SMALLINT	INT
eDBColumn_TinyInt	TINYINT	SINT / INT
eDBColumn_Bit	CHAR(1)	STRING
eDBColumn_Money	DEZIMAL(18,4)	LREAL
eDBColumn_Float	DOUBLE	LREAL
eDBColumn_Real	FLOAT	REAL
eDBColumn_DateTime	DATETIME	DT
eDBColumn_NText	TEXT	STRING
eDBColumn_NChar	CHAR	STRING
eDBColumn_Image	BLOB	ARRAY OF BYTE
eDBColumn_NVarChar	VARCHAR(254)	STRING
eDBColumn_Binary	BLOB	ARRAY OF BYTE
eDBColumn_VarBinary	BLOB	ARRAY OF BYTE

6.5.6 OCI / ODBC - Oracle Datenbank

Die Variablenwerte werden in einer Oracle Datenbank gespeichert.

In folgender Tabellenstruktur werden die Variablenwerte gespeichert.

Spaltenname	Datentyp	Null zulässig	Eigenschaft
ID	NUMBER	nein	IDENTITY(1,1)

Spaltenname	Datentyp	Null zulässig	Eigenschaft
Timestamp	DATE	nein	
Name	VARCHAR2	nein	
ValueType="Double"			
Value	FLOAT	nein	
ValueType="Bytes"			
Value	BLOB	nein	

Um die Funktionalität der AutoId zu bekommen, wird eine so genannte Sequenz in dem von Ihnen verwendeten Schema, mit folgenden Eigenschaften, erstellt:

Name: z.B. "AUTO_INCREMENT_Tabelle1$"

Typ: "Aufsteigend"

Minimum: "1"

Maximum: "1.0E27"

Intervall: "1"

Cache: "no Cache"

In der Spalte **„ID“** wird eine AutoID erzeugt. Das heißt der Wert in dieser Spalte wird immer um 1 erhöht. Diese Funktionalität ermöglicht die erzeugte Sequenz.
In die Spalte **„Timestamp“** wird der Speicherzeitpunkt des Datensatzes gespeichert.
In der Spalte **„Name“** steht der Name der Variable.
In der Spalte **„Value“** wird der Wert der Variable gespeichert.

E_DBColumnTypes	Oracle	PLC Control
eDBColumn_BigInt	DECIMAL(15,0)	T_LARGE_INTEGER (TcUtilities.lib)
eDBColumn_Integer	INTEGER	T_LARGE_INTEGER
eDBColumn_SmallInt	SMALLINT	T_LARGE_INTEGER
eDBColumn_TinyInt	SMALLINT	T_LARGE_INTEGER
eDBColumn_Bit	CHAR(1)	STRING
eDBColumn_Money	DECIMAL(18,4)	LREAL
eDBColumn_Float	DOUBLE PRECISION	LREAL
eDBColumn_Real	FLOAT	LREAL
eDBColumn_DateTime	DATE	DT
eDBColumn_NText	VARCHAR(254)	STRING
eDBColumn_NChar	CHAR(254)	STRING
eDBColumn_Image	BLOB	ARRAY OF BYTE
eDBColumn_NVarChar	NVARCHAR(254)	STRING
eDBColumn_Binary	BLOB	ARRAY OF BYTE
eDBColumn_VarBinary	BLOB	ARRAY OF BYTE

6.5.7 ASCII - File

Die Variablenwerte werden in einem ASCII-File gespeichert.
Die Werte werden getrennt durch ein Semikolon in das ASCII-File geschrieben.
Die erzeugte Datei kann dann in andere Tabellenkalkulationsprogramme, wie z.B. "Microsoft Excel" importiert und weiterverarbeitet werden.

Das ASCII-File wird in folgender Struktur erstellt:

```
[Timestamp];[NAME];[VALUE]
[JJJJ-MM-DD hh:mm:ss];[Variablenname];[Variablenwert]
[JJJJ-MM-DD hh:mm:ss];[Variablenname];[Variablenwert]
[JJJJ-MM-DD hh:mm:ss];[Variablenname];[Variablenwert]
[JJJJ-MM-DD hh:mm:ss];[Variablenname];[Variablenwert]
```

In die Spalte **„Timestamp“** wird der Speicherzeitpunkt des Datensatzes gespeichert.
In der Spalte **„Name“** steht der Name der Variable.
In der Spalte **„Value“** wird der Wert der Variable gespeichert.

Wichtig !
Bei Embedded Systemen die Datenbank nicht auf der Compact Flash Karte speichern.
Entweder die Datenbank im RAM benutzen, also nicht im Ordner "Hard Disk" speichern, oder in einem Netzwerkordner ablegen. Zu viele Schreibzyklen auf der Compact Flash Karte können die Lebensdauer der Compact Flash Karte minimieren.

6.5.8 XML - Database

Die Variablenwerte werden in einer XML-Datei gespeichert. Aufbau der Datenbank, Tabellen und Spalten werden in einer XSD-Datei definiert. Mit den Bausteinen FB_DBCreate [▶ 71] und FB_DBTableCreate [▶ 72] können die XML-Datei und die XSD-Datei erzeugt werden. Weiter Informationen zum Arbeiten mit XML-Dateien mit dem TwinCAT Database Server siehe hier [▶ 49].

```
<?xml version="1.0" encoding="UTF-8" ?>
<TestDB_XML xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
xs:noNamespaceSchemaLocation="TestDB_XML.xsd">
  <myTable_Double>
    <row ID="1" Timestamp="2012-03-08T12:45:08" Name="TestValue1" Value="222.222" />
    <row ID="2" Timestamp="2012-03-08T12:45:14" Name="TestValue1" Value="222.222" />
    <row ID="3" Timestamp="2012-03-08T12:45:18" Name="TestValue1" Value="222.222" />
    <row ID="4" Timestamp="2012-03-08T12:45:22" Name="TestValue1" Value="222.222" />
    <row ID="5" Timestamp="2012-03-08T12:45:23" Name="TestValue1" Value="222.222" />
 </myTable_Double>
</TestDB_XML>
```

In folgender Tabellenstruktur werden die Variablenwerte gespeichert

Spaltenname	Datentyp	Null zulässig	Eigenschaft
ID	xsd:long	nein	
Timestamp	xsd:dateTime	nein	
Name	xsd:string	nein	80
ValueType="Double"			
Value	xsd:double	nein	
ValueType="Bytes"			
Value	xsd:hexBinary	nein	längenwert

In der Spalte **„ID“** wird eine AutoID erzeugt. Das heißt der Wert in dieser Spalte wird immer um 1 erhöht.
In die Spalte **„Timestamp“** wird der Speicherzeitpunkt des Datensatzes gespeichert.
In der Spalte **„Name“** steht der Name der Variable.
In der Spalte **„Value“** wird der Wert der Variable gespeichert.

E_DBColumnTypes	XML	PLC Control
eDBColumn_BigInt	xsd:long	T_LARGE_INTEGER (TcUtilities.lib)
eDBColumn_Integer	xsd:int	DINT
eDBColumn_SmallInt	xsd:short	INT
eDBColumn_TinyInt	xsd:byte	BYTE
eDBColumn_Bit	xsd:boolean	BOOL
eDBColumn_Money	xsd:double	LREAL
eDBColumn_Float	xsd:double	LREAL
eDBColumn_Real	xsd:double	LREAL
eDBColumn_DateTime	xsd:dateTime	DT

E_DBColumnTypes	XML	PLC Control
eDBColumn_NText	xsd:string	STRING
eDBColumn_NChar	xsd:string	STRING
eDBColumn_Image	xsd:hexBinary	ARRAY OF BYTE
eDBColumn_NVarChar	xsd:string	STRING
eDBColumn_Binary	xsd:hexBinary	ARRAY OF BYTE
eDBColumn_VarBinary	xsd:hexBinary	ARRAY OF BYTE

Wichtig !

Bei Embedded Systemen die Datenbank nicht auf der Compact Flash Karte speichern.
Entweder die Datenbank im RAM benutzen, also nicht im Ordner "Hard Disk" speichern, oder in einem Netzwerkordner ablegen. Zu viele Schreibzyklen auf der Compact Flash Karte können die Lebensdauer der Compact Flash Karte minimieren.

6.5.9 ODBC - PostgreSQL Datenbank

Die Variablenwerte werden in einer PostgreSQL Datenbank gespeichert.

In folgender Tabellenstruktur werden die Variablenwerte gespeichert.

Spaltenname	Datentyp	Null zulässig	Eigenschaft
id	serial	nein	IDENTITY(1,1)
timestamp	timestamp	nein	
name	text	nein	
ValueType="Double"			
value	double precision	nein	
ValueType="Bytes"			
value	bytea	nein	

Um die Funktionalität der AutoId zu bekommen, wird eine so genannte Sequenz erstellt mit folgenden Eigenschaften:

Name: "mytable_ID_seq"

Erhöhungsschritt: "1"

Minimum: "1"

In der Spalte **„ID“** wird eine AutoID erzeugt. Das heißt der Wert in dieser Spalte wird immer um 1 erhöht. Diese Funktionalität ermöglicht die erstellte Sequenz "mytable_ID_seq".
In die Spalte **„Timestamp“** wird der Speicherzeitpunkt des Datensatzes gespeichert.
In der Spalte **„Name“** steht der Name der Variable.
In der Spalte **„Value“** wird der Wert der Variable gespeichert.

E_DBColumnTypes	PostgreSQL	PLC Control
eDBColumn_BigInt	bigint	T_ULARGE_INTEGER (TcUtilities.lib)
eDBColumn_Integer	integer	DINT
eDBColumn_SmallInt	smallint	INT
eDBColumn_TinyInt	smallint	INT
eDBColumn_Bit	bit	STRING
eDBColumn_Money	money	LREAL
eDBColumn_Float	double precision	LREAL
eDBColumn_Real	real	REAL
eDBColumn_DateTime	timestamp	DT

E_DBColumnTypes	PostgreSQL	PLC Control
eDBColumn_NText	text	STRING
eDBColumn_NChar	character	STRING
eDBColumn_Image	bytea	ARRAY OF BYTE
eDBColumn_NVarChar	character varying	STRING
eDBColumn_Binary	bytea	ARRAY OF BYTE
eDBColumn_VarBinary	bytea	ARRAY OF BYTE

6.5.10 ODBC - DB2 Datenbank

Die Variablenwerte werden in einer DB2 Datenbank gespeichert.

In folgender Tabellenstruktur werden die Variablenwerte gespeichert.

Spaltenname	Datentyp	Null zulässig	Eigenschaft
ID	BIGINT	nein	IDENTITY (1,1)
Timestamp	TIMESTAMP	nein	
Name	VARCHAR	nein	
ValueType="Double"			
Value	DOUBLE	nein	
ValueType="Bytes"			
Value	BLOB	nein	

In der Spalte **„ID“** wird eine AutoID erzeugt. Das heißt der Wert in dieser Spalte wird immer um 1 erhöht. Diese Funktionalität ermöglicht die Eigenschaft IDENTITY.
In die Spalte **„Timestamp“** wird der Speicherzeitpunkt des Datensatzes gespeichert.
In der Spalte **„Name“** steht der Name der Variable.
In der Spalte **„Value“** wird der Wert der Variable gespeichert.

E_DBColumnTypes	DB2	PLC Control
eDBColumn_BigInt	BIGINT	T_ULARGE_INTEGER (TcUtilities.lib)
eDBColumn_Integer	INT	DINT
eDBColumn_SmallInt	SMALLINT	INT
eDBColumn_TinyInt	SMALLINT	INT
eDBColumn_Bit	VARCHAR(1)	STRING(1)
eDBColumn_Money	DECIMAL(18,4)	LREAL
eDBColumn_Float	DOUBLE PRECISION	LREAL
eDBColumn_Real	FLOAT	LREAL
eDBColumn_DateTime	TIMESTAMP	DT
eDBColumn_NText	LONG VARCHAR	STRING
eDBColumn_NChar	CHAR(254)	STRING
eDBColumn_Image	BLOB	ARRAY OF BYTE
eDBColumn_NVarChar	NVARCHAR(254)	STRING
eDBColumn_Binary	BLOB	ARRAY OF BYTE
eDBColumn_VarBinary	BLOB	ARRAY OF BYTE

6.5.11 ODBC - InterBase Datenbank

Die Variablenwerte werden in einer InterBase Datenbank gespeichert.

In folgender Tabellenstruktur werden die Variablenwerte gespeichert.

Spaltenname	Datentyp	Null zulässig	Eigenschaft
ID	BIGINT	nein	IDENTITY(1,1)
Timestamp	TIMESTAMP	nein	
Name	TEXT	nein	
ValueType="Double"			
Value	FLOAT	nein	
ValueType="Bytes"			
Value	BLOB	nein	

In der Spalte **„ID“** wird eine AutoID erzeugt. Das heißt der Wert in dieser Spalte wird immer um 1 erhöht.
In die Spalte **„Timestamp“** wird der Speicherzeitpunkt des Datensatzes gespeichert.
In der Spalte **„Name“** steht der Name der Variable.
In der Spalte **„Value“** wird der Wert der Variable gespeichert.

E_DBColumnTypes	InterBase	PLC Control
eDBColumn_BigInt	BIGINT	T_ULARGE_INTEGER (TcUtilities.lib)
eDBColumn_Integer	INTEGER	DINT
eDBColumn_SmallInt	SMALLINT	INT
eDBColumn_TinyInt	SMALLINT	INT
eDBColumn_Bit	CHAR(1)	STRING
eDBColumn_Money	DEZIMAL(18,4)	LREAL
eDBColumn_Float	FLOAT	REAL
eDBColumn_Real	DOUBLE PRECISION	LREAL
eDBColumn_DateTime	Timestamp	DT
eDBColumn_NText	VARCHAR(254)	STRING
eDBColumn_NChar	CHAR(254)	STRING
eDBColumn_Image	BLOB	ARRAY OF BYTE
eDBColumn_NVarChar	VARCHAR(254)	STRING
eDBColumn_Binary	BLOB	ARRAY OF BYTE
eDBColumn_VarBinary	BLOB	ARRAY OF BYTE

6.5.12 ODBC - Firebird Datenbank

Die Variablenwerte werden in einer Firebird Datenbank gespeichert.

In folgender Tabellenstruktur werden die Variablenwerte gespeichert.

Spaltenname	Datentyp	Null zulässig	Eigenschaft
ID	BIGINT	nein	IDENTITY(1,1)
Timestamp	TIMESTAMP	nein	
Name	TEXT	nein	
ValueType="Double"			
Value	FLOAT	nein	
ValueType="Bytes"			
Value	BLOB	nein	

In der Spalte **„ID“** wird eine AutoID erzeugt. Das heißt der Wert in dieser Spalte wird immer um 1 erhöht.

In die Spalte **„Timestamp“** wird der Speicherzeitpunkt des Datensatzes gespeichert.
In der Spalte **„Name“** steht der Name der Variable.
In der Spalte **„Value“** wird der Wert der Variable gespeichert.

E_DBColumnTypes	FireBird	PLC Control
eDBColumn_BigInt	BIGINT	T_ULARGE_INTEGER (TcUtilities.lib)
eDBColumn_Integer	INTEGER	DINT
eDBColumn_SmallInt	SMALLINT	INT
eDBColumn_TinyInt	SMALLINT	INT
eDBColumn_Bit	CHAR(1)	STRING
eDBColumn_Money	DEZIMAL(18,4)	LREAL
eDBColumn_Float	FLOAT	REAL
eDBColumn_Real	DOUBLE PRECISION	LREAL
eDBColumn_DateTime	Timestamp	DT
eDBColumn_NText	VARCHAR(254)	STRING
eDBColumn_NChar	CHAR(254)	STRING
eDBColumn_Image	BLOB	ARRAY OF BYTE
eDBColumn_NVarChar	VARCHAR(254)	STRING
eDBColumn_Binary	BLOB	ARRAY OF BYTE
eDBColumn_VarBinary	BLOB	ARRAY OF BYTE

6.5.13 Zusatzinformationen

6.5.13.1 Microsoft SQL Server Hinweise

Logs im Windows Eventlog:

- Error Event „Report Server Windows Service (SQLEXPRESS) kann nicht mit der Berichtsserver-Datenbank verbunden werden.“
  - Im SQL Configuration Manager unter SQL Server 2005 Services den SQL Server Reporting Services (SQLEXPRESS) stoppen und den Start Mode auf Manual setzen. Der Reporting Service wird nicht vom Database Server benötigt
- Information Event „'TcDataLogger'-Datenbank wird gestartet“
  - Im SQL Server Management Studio Express unter Datenbanken/TcDataLogger mit einem rechten Mausklick auf Eigenschaften gehen und dann unter Optionen die Option „Automatisch schließen“ auf „False“ setzen. Diese Option wird nicht benötigt, der Database Server öffnet und schließt die Datenbank automatisch.

Es gibt eine Möglichkeit das Loggen in den Windows Eventlog zu unterdrücken. Es werden dann überhaupt keine Events mehr geloggt. Es kann nicht zwischen den unterschiedlichen Eventtypen unterschieden werden.

- Im SQL Configuration Manager unter SQL Server 2005 Services den SQL Server (SQLEXPRESS) auswählen mit einem Reckklick der Maus auf Properties. Unter dem Advanced Tab gibt es den Unterpunkt „Startup Parameters“. Die einzelnen Parameter sind mit Semikolon getrennt. Den Parameter –n hinzufügen. Danach den Service neu starten.

Ab diesen Zeitpunkt werden keine Events mehr vom SQL Server geloggt.

6.5.13.2 XML - Informationen

XML Datei verwenden als Datenbank mit dem TwinCAT Database Server [▶ 49]

XPath Querries auf eine XML Datei anwenden mit dem TwinCAT Database Server [▶ 52]

Nähere Informationen zu XML-Schemas finden Sie hier: http://www.edition-w3.de/TR/2001/REC-xmlschema-0-20010502/

XML als Datenbank

XSD-Schema für Standard-Tabellenstruktur:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:simpleType name="bigint">
    <xsd:restriction base="xsd:long" />
  </xsd:simpleType>
  <xsd:simpleType name="datetime">
    <xsd:restriction base="xsd:dateTime" />
  </xsd:simpleType>
  <xsd:simpleType name="ntext_80">
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="80" />
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="float">
    <xsd:restriction base="xsd:double" />
  </xsd:simpleType>
  <xsd:complexType name="myTable_Double_Type">
    <xsd:sequence>
      <xsd:element minOccurs="0" maxOccurs="unbounded" name="row">
        <xsd:complexType>
          <xsd:attribute name="ID" type="bigint" />
          <xsd:attribute name="Timestamp" type="datetime" />
          <xsd:attribute name="Name" type=" ntext_80" />
          <xsd:attribute name="Value" type="float" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="TestDB_XML">
    <xsd:complexType>
      <xsd:sequence minOccurs="1" maxOccurs="1">
        <xsd:element name="myTable_Double" type="myTable_Double_Type" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XML-Datei für Standard-Tabellenstruktur (Beispiel)

```
<?xml version="1.0" encoding="UTF-8"?>
<TestDB_XML xmlns:xs="http://www.w3.org/2001/XMLSchema-
instance" xs:noNamespaceSchemaLocation="TestDB_XML.xsd">
  <myTable_Double>
    <row ID="1" Timestamp="2012-03-08T12:45:08" Name="TestValue1" Value="222.222" />
    <row ID="2" Timestamp="2012-03-08T12:45:14" Name="TestValue1" Value="222.222" />
    <row ID="3" Timestamp="2012-03-08T12:45:18" Name="TestValue1" Value="222.222" />
    <row ID="4" Timestamp="2012-03-08T12:45:22" Name="TestValue1" Value="222.222" />
    <row ID="5" Timestamp="2012-03-08T12:45:23" Name="TestValue1" Value="222.222" />
 </myTable_Double>
</TestDB_XML>
```

Datatypes für XML-Tabellen

```
  <xsd:simpleType name="bigint">
    <xsd:restriction base="xsd:long" />
  </xsd:simpleType>

  <xsd:simpleType name="datetime">
    <xsd:restriction base="xsd:dateTime" />
  </xsd:simpleType>

 <xsd:simpleType name="ntext_80"> <!-- Länge kann individuell angegeben werden -->
   <xsd:restriction base="xsd:string">
     <xsd:maxLength value="80" />
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="float">
    <xsd:restriction base="xsd:double" />
  </xsd:simpleType>

  <xsd:simpleType name="binary_1"> <!-- Länge kann individuell angegeben werden -->
   <xsd:restriction base="xsd:hexBinary">
     <xsd:maxLength value="1" />
    </xsd:restriction>
```

```
 </xsd:simpleType>

 <xsd:simpleType name="bit">
   <xsd:restriction base="xsd:boolean" />
 </xsd:simpleType>

<xsd:simpleType name="image_1"> <!-- Länge kann individuell angegeben werden -->
  <xsd:restriction base="xsd:hexBinary">
     <xsd:maxLength value="1" />
   </xsd:restriction>
 </xsd:simpleType>

 <xsd:simpleType name="integer">
   <xsd:restriction base="xsd:int" />
 </xsd:simpleType>

 <xsd:simpleType name="money">
   <xsd:restriction base="xsd:double" />
 </xsd:simpleType>

 <xsd:simpleType name="nchar_50"> <!-- Länge kann individuell angegeben werden -->
  <xsd:restriction base="xsd:string">
     <xsd:maxLength value="50" />
   </xsd:restriction>
 </xsd:simpleType>

 <xsd:simpleType name="nvarchar_50"> <!-- Länge kann individuell angegeben werden -->
  <xsd:restriction base="xsd:string">
     <xsd:maxLength value="50" />
   </xsd:restriction>
 </xsd:simpleType>

 <xsd:simpleType name="real">
   <xsd:restriction base="xsd:double" />
 </xsd:simpleType>

 <xsd:simpleType name="smallint">
   <xsd:restriction base="xsd:short" />
 </xsd:simpleType>

 <xsd:simpleType name="tinyint">
   <xsd:restriction base="xsd:byte" />
 </xsd:simpleType>

 <xsd:simpleType name="varbinary_1"> <!-- Länge kann individuell angegeben werden -->
  <xsd:restriction base="xsd:hexBinary">
     <xsd:maxLength value="1" />
   </xsd:restriction>
</xsd:simpleType>
```

Datentyp Zuordnung XML => SPS

E_DBColumnTypes	XML	PLC Control
eDBColumn_BigInt	xsd:long	T_ULARGE_INTEGER
eDBColumn_Integer	xsd:int	DINT
eDBColumn_SmallInt	xsd:short	INT
eDBColumn_TinyInt	xsd:byte	BYTE
eDBColumn_Bit	xsd:boolean	BOOL
eDBColumn_Money	xsd:double	LREAL
eDBColumn_Float	xsd:double	LREAL
eDBColumn_Real	xsd:double	LREAL
eDBColumn_DateTime	xsd:dateTime	DT
eDBColumn_NText	xsd:string	STRING
eDBColumn_NChar	xsd:string	STRING
eDBColumn_Image	xsd:hexBinary	ARRAY OF BYTE
eDBColumn_NVarChar	xsd:string	STRING
eDBColumn_Binary	xsd:hexBinary	ARRAY OF BYTE
eDBColumn_VarBinary	xsd:hexBinary	ARRAY OF BYTE

Erzeugen/Auslesen von Datensätzen in/aus der XML-Datei

Zum Erzeugen von Datensätzen können Standard SQL-Befehle verwendet werden. Die SQL-INSERT Befehle werden vom TwinCAT Database Server interpretiert und auf die XML-Datei in Form von XML-Nodes umgesetzt. Die SQL SELECT Befehle werden vom TwinCAT Database Server in Form von XPath Queries auf die XML-Datei umgesetzt.

Beispiele für unterstützte INSERT Befehle:

```
INSERT INTOmyTable_Double (ID, Timestamp, Name, Value) VALUES(1, CURRENT_TIMESTAMP, 'TestValue1' , 1234.5678)
INSERT INTOmyTable_Double (Timestamp, Name) VALUES(CURRENT_TIMESTAMP, 'TestValue1');
INSERT INTOmyTable_Double VALUES(1, CURRENT_TIMESTAMP, 'TestValue1', 1234.5678);
INSERT INTOmyTable_Double VALUES(1, '2010-01-06 12:13:14', 'TestValue1', 1234.5678);
```

Beispiele für unterstützte SELECT Befehle:

```
SELECT ID, Timestamp, Name, Value FROM myTable_Double;
SELECT* FROM myTable_Double;
SELECT Timestamp, Name FROM myTable_Double
SELECT* FROM myTable_Double WHERE Name = 'TestValue1';
SELECT* FROM myTable_Double WHERE ID > 1;
```

Unterstützte Funktionsbausteine

FB_DBCreate

FB_DBCyclicRdWrt

FB_DBRead

FB_DBRecordArraySelect

FB_DBRecordDelete

FB_DBRecordInsert

FB_DBRecordInsert_EX

FB_DBRecordSelect

FB_DBRecordSelect_EX

FB_DBTableCreate

FB_DBWrite

XML Standard XPath Funktion

XPath Typen

Es gibt 3 verschiedene Arten per XPath Werte aus einer XML-Datei zu lesen.

- XPath<ATTR>
  - Es werden allen Attributwerte vom ausgewählten XML-Tag an die SPS zurückgeliefert
  - Ist ein XML-Schema vorhanden werden die Attribute in die richtigen Datentypen konvertiert
  - Ist kein XML-Schema vorhanden werden die Attribute als T_MaxString zurückgeliefert
- XPath<TAG>
  - Es wird der InnerText des ausgewählten XML-Tag an die SPS zurückgeliefert
  - Ist ein XML-Schema vorhanden wird der Wert in den richtigen Datentyp konvertiert
  - Ist kein XML-Schema vorhanden wird der Wert als T_MaxString zurückgeliefert
- XPath<SUBTAG>
  - Es werden die InnerText Werte aller SubTags des ausgewählten XML-Tag an die SPS zurückgeliefert

- Ist ein XML-Schema vorhanden werden die Werte in den richtigen Datentyp konvertiert
- Ist kein XML-Schema vorhanden werden alle Werte als T_MaxString zurückgeliefert

Beispiele

XML-Datei

```
<?xml version="1.0" encoding="utf-8" ?>
<TestXML>
  <Node attr1="1" attr2="Node1">
    <SubNode1>SubNodeWert1</SubNode1>
    <SubNode2>200</SubNode2>
    <SubNode3>SubNodeWert3</SubNode3>
    <SubNode4>400.5</SubNode4>
    <SubNode5>SubNodeWert5</SubNode5>
  </Node>
  <Node attr1="2" attr2="Node2">
    <SubNode1>SubNodeWert1</SubNode1>
    <SubNode2>200</SubNode2>
    <SubNode3>SubNodeWert3</SubNode3>
    <SubNode4>400.5</SubNode4>
    <SubNode5>SubNodeWert5</SubNode5>
  </Node>
</TestXML>
```

XML-Schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://
www.w3.org/2001/XMLSchema">
  <xs:element name="TestXML">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="Node">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="SubNode1" type="xs:string" />
              <xs:element name="SubNode2" type="xs:short" />
              <xs:element name="SubNode3" type="xs:string" />
              <xs:element name="SubNode4" type="xs:double" />
              <xs:element name="SubNode5" type="xs:string" />
            </xs:sequence>
            <xs:attribute name ="attr1" type="xs:integer" use="required" />
            <xs:attribute name ="attr2" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Beispiel für XPATH<ATTR>

```
XPath => XPATH<ATTR>#TestXML/Node[@attr1=2]
```

Zurückgelieferte Struktur wenn **kein** Schema vorhanden:

```
TYPE ST_Record :
STRUCT
            attr1     : T_MaxString   := '2';
            attr2     : T_MaxString   := 'Node2';
END_STRUCT
END_TYPE
```

Zurückgelieferte Struktur, wenn **ein** Schema vorhanden:

```
TYPEST_Record :
STRUCT
            attr1     : DINT          := 2;
            attr2     : T_MaxString   := 'Node2';
END_STRUCT
END_TYPE
```

Beispiel für XPATH<TAG>

```
XPath => XPATH<TAG>#TestXML/Node[@attr1=2]/SubNode2
```

Zurückgelieferte Wert wenn **kein** Schema vorhanden: `SubNode2 : T_MaxString := '200';`

Zurückgelieferte Wert wenn **ein** Schema vorhanden: `SubNode2 : INT := 200;`

Beispiel für XPATH<SUBTAG>

```
XPath => XPATH<SUBTAG>#TestXML/Node[@attr1=2]
```

Zurückgelieferte Struktur wenn **kein** Schema vorhanden:

```
TYPEST_Record :
STRUCT
            SubNode1          : T_MaxString   := 'SubNodeWert1';
            SubNode2          : T_MaxString   := '200';
            SubNode3          : T_MaxString   := 'SubNodeWert3';
            SubNode4          : T_MaxString   := '400.5';
            SubNode5          : T_MaxString   := 'SubNodeWert5';
END_STRUCT
END_TYPE
```

Zurückgelieferte Struktur, wenn **ein** Schema vorhanden:

```
TYPEST_Record :
STRUCT
            SubNode1          : T_MaxString   := 'SubNodeWert1';
            SubNode2          : INT             := 200;
            SubNode3          : T_MaxString   := 'SubNodeWert3';
            SubNode4          : LREAL         := 400.5;
            SubNode5          : T_MaxString   := 'SubNodeWert5';
END_STRUCT
END_TYPE
```

Unterstützte Funktionsbausteine

FB_DBRecordSelect

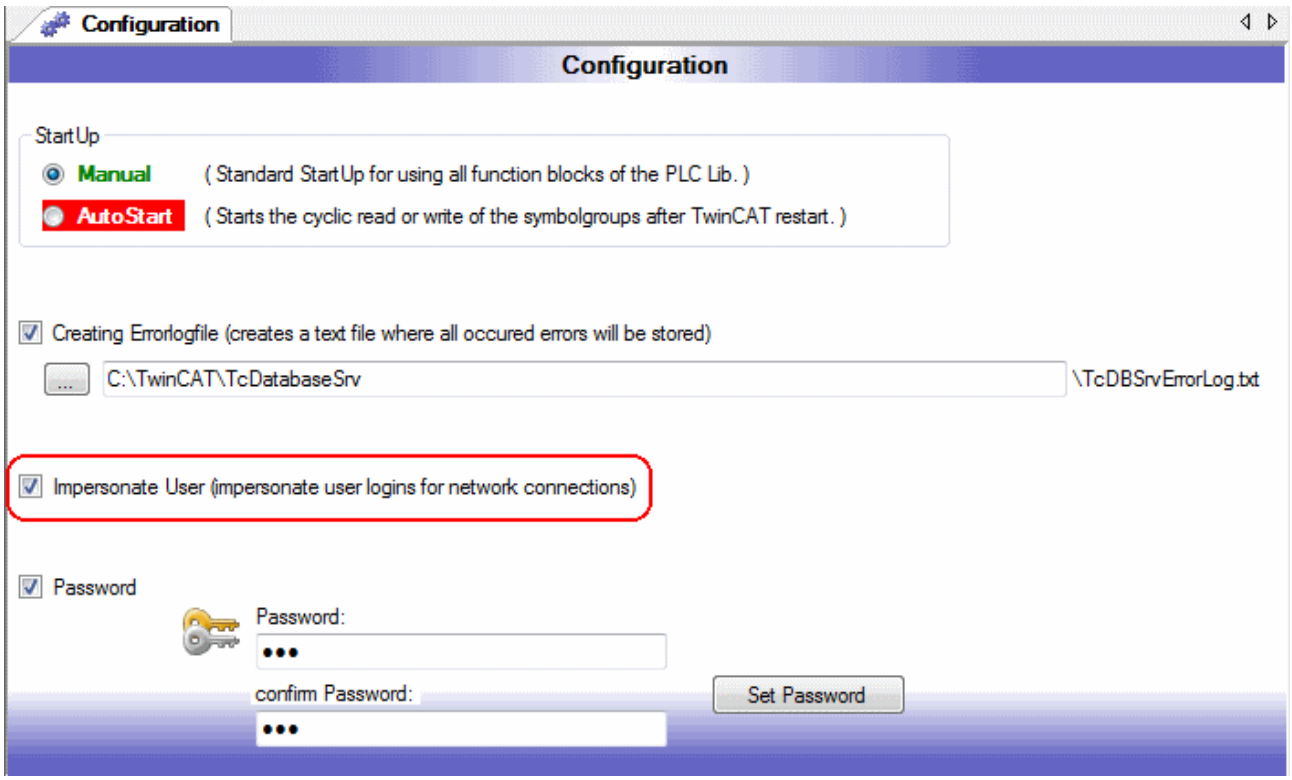
FB_DBRecordSelect_EX

FB_DBRecordArraySelect

6.6 Expert

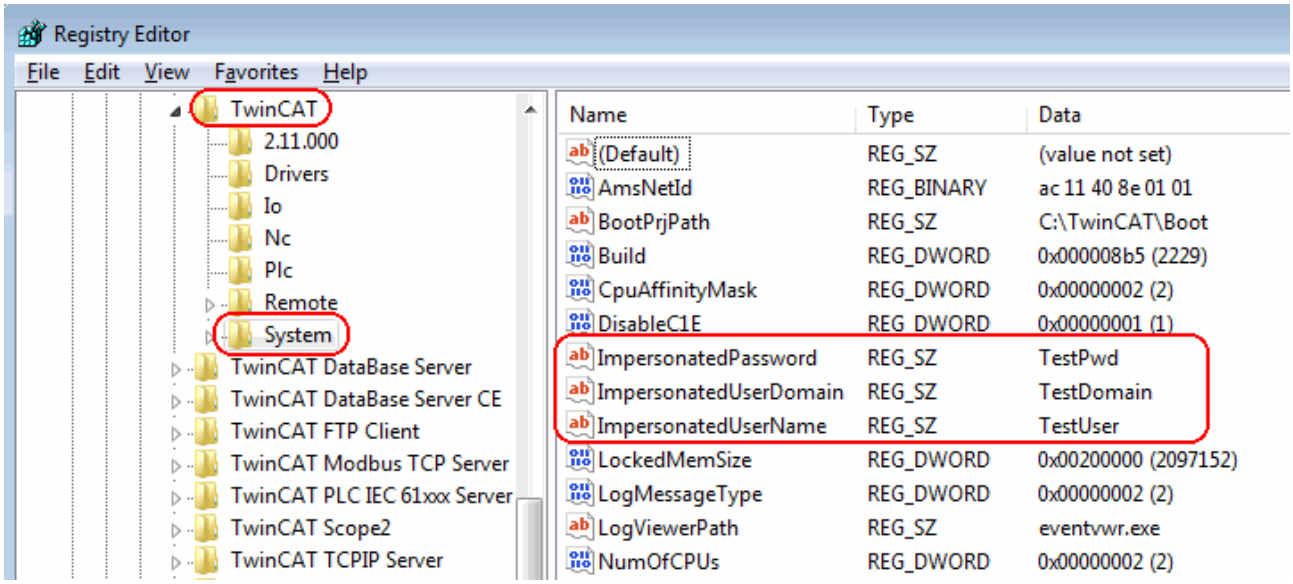
6.6.1 Impersonate Option

Impersonate (etwas, jemanden verkörpern). Mit der Option Impersonate können Benutzer - Authentifizierungsdaten angeben, um eine Netzwerkverbindung zu einem öffentlichen Verzeichnis herstellen zu können. Bei Netzwerkzugriff auf Dateibasierenden Datenbanken wie Access-Datenbanken oder SQL Compact Datenbanken muss die Impersonate-Option gesetzt werden, damit sich der TwinCAT Database Server auf dieses Netzwerklaufwerk verbinden kann.



Da der TwinCAT Database Server als Systemprozess läuft würde er auf dem Zielgerät keine Berechtigung für Schreib- bzw. Lesezugriffe besitzen. Man kann dies nur umgehen, indem man dem öffentlichen Ordner die Benutzerrechte für den Benutzer "Gast" frei schaltet und diesem Benutzer Lese- und Schreibrechte gibt. Dieses Verfahren ist allerdings sehr unsicher, da dann jeder Benutzer auf diesen Ordner Zugriffsrechte besitzt.

Um dieses Problem zu lösen, gibt es die "Impersonate" Option mit der man sich mit bestimmten Benutzerdaten an diesem Zielgerät anmelden kann. Die Authentifizierungsdaten müssen in der Registry hinterlegt werden, wie Sie im folgenden Bild sehen können.



Folgende Keys müssen erstellt werden:
->[HKEY_LOCAL_MACHINE\SOFTWARE\Beckhoff\TwinCAT\System]

"ImpersonatedPassword"

Enthält das Password für die Authentifizierung.
->[HKEY_LOCAL_MACHINE\SOFTWARE\Beckhoff\TwinCAT\System]

"ImpersonatedUserDomain"

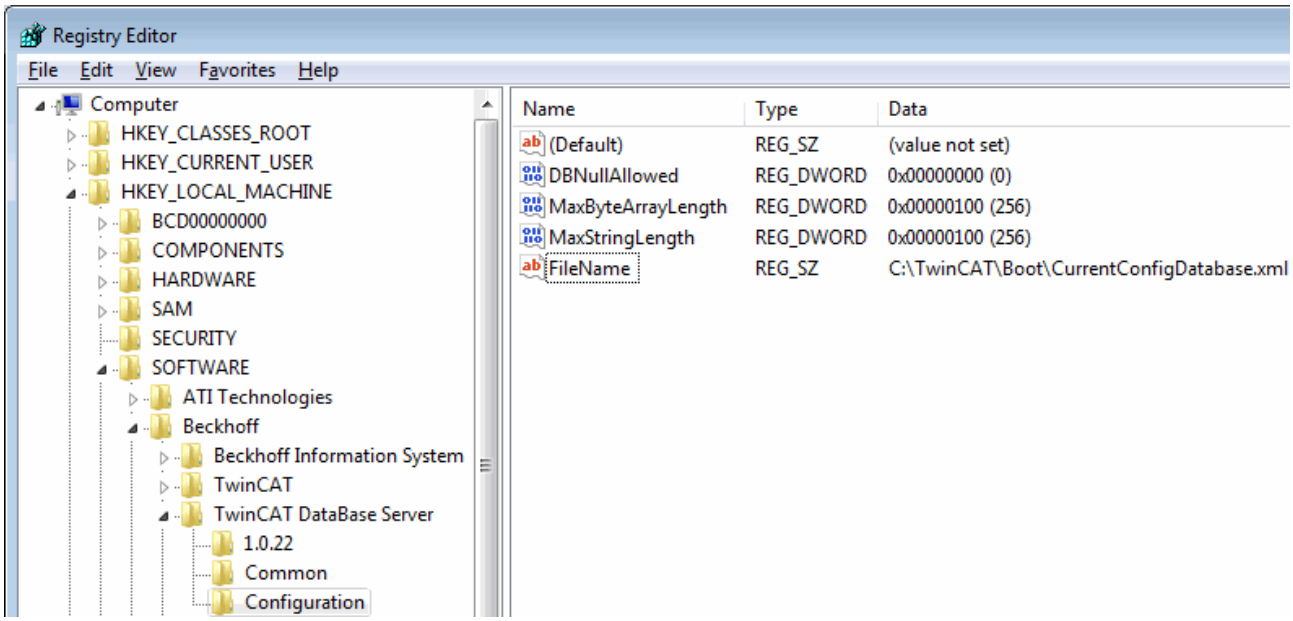
Enthält den Namen der Domaine in der der Benutzer angemeldet ist.
->[HKEY_LOCAL_MACHINE\SOFTWARE\Beckhoff\TwinCAT\System]

"ImpersonatedUserName"

Enthält den Namen des Benutzers.

6.6.2 Zusatzeinstellungen in der Registry

Es besteht die Möglichkeit, grundlegende Einstellungen für den TwinCAT Database Server vorzunehmen. Zum einen kann mit dem Key "DBNullAllowed" das Tolerieren von NULL-Werten aus der Datenbank eingeschaltet werden. Die Bausteine liefern dann keinen Fehler zurück, wenn NULL-Werte aufgetreten sind. Des Weiteren können Maximallängen für Byte-Arrays und Strings definiert werden. Bei den Funktionsbausteinen FB_DBRecordReturn, FB_DBRecordArrayReturn, FB_DBStoredProceduresRecordReturn und FB_DBStoredProceduresRecordArray werden diese Maximallängen verwendet.



Folgende Keys können angepasst werden:
->[HKEY_LOCAL_MACHINE\SOFTWARE\Beckhoff\TwinCAT Database Server\Configuration]
"DBNullAllowed"

Durch das setzen einer 1 an diesem Key kann das zulassen von DBNull-Werten aktiviert werden. Treten Null-Werte auf und dieser Key ist nicht gesetzt, wird ein Fehler an den entsprechenden Baustein zurückgeliefert.
->[HKEY_LOCAL_MACHINE\SOFTWARE\Beckhoff\TwinCAT Database Server\Configuration]
"MaxByteArrayLength"

Enthält die maximale Länge eines aus der Datenbank zurück gelieferten Byte-Arrays in die SPS.
->[HKEY_LOCAL_MACHINE\SOFTWARE\Beckhoff\TwinCAT Database Server\Configuration]
"MaxStringLength"

Enthält die maximale Länge eines aus der Datenbank zurück gelieferten String in die SPS.

Diese Funktion wird ab Version 1.0.20 unterstützt.

6.6.3 XML - Konfigurationsdatei

Die Konfiguration des TwinCAT DataBase Servers erfolgt über eine XML - Konfigurationsdatei.
Die Einstellungen der Konfigurationsdatei werden einmalig beim Starten des TwinCAT DataBase Server eingelesen. Zur Laufzeit des TwinCAT DataBase Servers kann das Einlesen der Konfiguration zusätzlich aus der SPS mit einem Funktionsbaustein angestoßen werden.

Die Konfiguration enthält mehrere Sektionen:

- DataBases:
  Konfiguration aller "Datenbanken" wie SQL-Datenbank, ASCII-File, ...
- AdsDevices:
  Konfiguration aller ADS-Geräte (z.B. SPS Laufzeit Systeme)
- SymbolGroups:
  Gruppierung von verschiedenen "Symbolen" (z.B. SPS-Variablen) die einem ADS-Gerät zugeordnet sind zu einer logischen Gruppe.
  Eine logische Gruppe kann konfiguriert werden um Daten:
  - von einer Datenbank in ein ADS-System zu transportieren.
  - von einem ADS-Gerät in eine Datenbank zu transportieren.

```
<?xml version="1.0" ?>
<Configuration>
   <Log>1</Log>
   <LogPath>C:\TwinCAT\TcDatabaseSrv</LogPath>
   <StartUp>Manual</StartUp>
   <PwdInfos>tZuYPxhe+G5NKHWLYSZE+NiFAINdlcBgtIUVD+j076ID3geO7FdGzvdfPl0Q09Zb2CKpwj=</
PwdInfos>
   <Databases>
      <Database Type="Mobile-Server" ValueType="Double">
         <DBId>1</DBId>
         <DBServer/>
         <DBProvider/>
         <DBUrl>C:\TwinCAT\TcDatabaseSrv\Samples\TestDB_CompactSQL.sdf</DBUrl>
         <DBSystemDB/>
         <DBUserId/>
         <DBTable>myTable</DBTable>
      </Database>
   </Databases>
   <AdsDevices>
      <AdsDevice>
         <AdsId>1</AdsId>
         <NetID>10.1.128.49.1.1</NetID>
         <Port>801</Port>
         <Timeout>2000</Timeout>
         <ADSReadWriteSetting>1</ADSReadWriteSetting>
      </AdsDevice>
   </AdsDevices>
   <SymbolGroups>
      <SymbolGroup>
         <Direction RingBuffMode="Count">ADS_to_DB_RINGBUFFER</Direction>
         <RingBuffCount>20</RingBuffCount>
         <CycleTime>30000</CycleTime>
         <AdsId>1</AdsId>
         <DBId>1</DBId>
         <Symbols>
            <Symbol>
               <DBName>TESTVAR123</DBName>
               <Name>MAIN.TESTVAR123</Name>
               <Type>LREAL</Type>
               <IGroup>16448</IGroup>
               <IOffset>172536</IOffset>
               <BitSize>64</BitSize>
               <DBLogMode>3</DBLogMode>
            </Symbol>
         </Symbols>
      </SymbolGroup>
  </SymbolGroups>
</Configuration>
```

Die Erklärung zur XML - Konfigurationsdatei:

- Tag „Log“:

Zusatzoption: Zu Testzwecken kann ein Fehlerlogmechanismus aktiviert werden, um auftretende Fehlerbeschreibungen in eine Textdatei "TcDBSrvErrorLog.txt" zu loggen.

- Tag „LogPath“:

Zusatzoption: Zu Testzwecken kann ein Fehlerlogmechanismus aktiviert werden siehe Tag Log. In diesem Tag kann der Pfad zu dieser Textdatei angegeben werden.

- Tag „Impersonate“:

Zusatzoption: Um die "Impersonate" - Option zu aktivieren muss dieser Tag gesetzt werden.

- Tag „StartUp“:

Einstellung des Startmodus.
Option "Autostart“ : Der Server startet sofort mit den in der Konfigurationsdatei festgelegten Aktionen (z.B. Schreiben von ADS-Daten in die Datenbank).
Option „Manual“ : Der Server bleibt zunächst passiv und wird erst über den Funktionsblock „FB_DBCyclicRdWrt“ angestoßen.

- Tag „PwdInfos“:

Enthält alle Passwörter die für die Kommunikation mit den Datenbanken benötigt werden. Alle eingegebenen Passwörter sind verschlüsselt und somit nicht im Klartext zu sehen.

- Tag „Databases“:

Konfiguration der verschiedenen Datenbanken:

DBId Jede Datenbank bekommt eine eindeutige ID.

Database Type Zur Zeit werden die Datenbanktypen **"Mobile-Server"**, **"ASCII"**, **"Access"** und **"Sequal-Server"** unterstützt.

ValueType Zwei Speicherarten werden unterstützt
- "Double" : Speichert alle Variablenwerte als Double Werte (Strukturen und Strings werden dabei nicht unterstützt);
- "Bytes" : Speichert alle Variablenwerte als Bytestreams (Strukturen und Strings können hierbei mit geloggt werden)

- Tag „AdsDevices“:

Deklaration der ADS Geräte.

Jedes Laufzeitsystem bekommt eine eindeutige ID.

Die ADS Geräte werden eindeutig durch NetID, Port und Timeout angegeben.

Tag „SymbolGoups“:

Definition der Symbolgruppen: Hier wird festgelegt, welche Datenbank mit welchem ADS-Gerät verbunden ist. Außerdem wird die Richtung des Datenflusses festgelegt (vom TwinCAT-System in die Datenbank oder von der Datenbank in das TwinCAT-System).

Die Zykluszeit bestimmt, in welchen Zeitabständen die Daten von der Datenbank gelesen bzw. in die Datenbank geschrieben werden.

- Tag „Symbol“:

Definition der einzelnen Symbole mit folgenden Parametern
DBName => In der Datenbank verwendeter Symbolname
Name => In der SPS verwendeter Symbolname
Optional: Type => Datentyp des Symbols
Optional: IGroup / IOffset => Speicherort des Symbols
Optional: BitSize => Größe des Symbols
DBLogMode => Optional: Gibt an ob die Symbole zyklisch oder nach Änderung überprüft werden.

Zu finden ist die Konfigurationsdatei unter **"C:\TwinCAT\Boot\CurrentConfigDatabase.xml"** oder bei Embedded-Systemen unter **"\Hard Disk\TwinCAT\Boot\CurrentConfigDatabase.xml"**

7 SPS-API

Übersicht

Die TcDatabase.lib Bibliothek beinhaltet Funktionsblöcke zum Steuern und Konfigurieren des TwinCAT Database Servers.

Funktionsblöcke

Name	Beschreibung
FB_GetStateTcDatabase [▶ 61]	Statusinformationen abrufen.
FB_DBConnectionAdd [▶ 63]	Fügt der XML-Konfigurationsdatei Datenbankanbindungen an
FB_DBAuthentificationAdd [▶ 83]	Fügt der XML-Konfigurationsdatei Authentifizierungsinformationen der jeweiligen Datenbankanbindung an
FB_DBOdbcConnectionAdd [▶ 65]	Fügt der XML-Konfigurationsdatei ein ODBC-Datenbankverbindung an
FB_AdsDeviceConnectionAdd [▶ 66]	Fügt der XML-Konfigurationsdatei ein ADS-Gerät an
FB_DBReloadConfig [▶ 62]	Lädt die XML-Konfigurationsdatei neu
FB_GetDBXMLConfig [▶ 67]	Liest alle Datenbankkonfigurationen aus der XML-Konfigurationsdatei aus.
FB_GetAdsDevXMLConfig [▶ 68]	Liest alle ADS-Gerätekonfigurationen aus der XML-Konfigurationsdatei aus.
FB_DBConnectionOpen [▶ 69]	Öffnet eine Verbindung zu einer Datenbank
FB_DBConnectionClose [▶ 70]	Schließt eine Verbindung zu einer Datenbank
FB_DBCreate [▶ 71]	Erstellt eine neue Datenbank
FB_DBTableCreate [▶ 72]	Erstellt eine Tabelle mit beliebiger Tabellenstruktur
FB_DBRead [▶ 74]	Liest einen Wert aus der Datenbank aus
FB_DBWrite [▶ 75]	Schreibt einen Variablenwerte mit Timestamp in eine Datenbank
FB_DBCyclicRdWrt [▶ 73]	Startet bzw. stoppt das Loggen\Schreiben der Variablen
FB_DBRecordSelect [▶ 85]	Liest einen Datensatz aus einer Tabelle aus
FB_DBRecordSelect_EX [▶ 87]	Liest einen Datensatz aus einer Tabelle aus. (Befehllänge <= 10000 Zeichen)
FB_DBRecordArraySelect [▶ 79]	Liest mehrere Datensätze aus einer Tabelle aus
FB_DBRecordInsert [▶ 84]	Erstellt einen neuen Datensatz
FB_DBRecordInsert_EX [▶ 78]	Erstellt einen neuen Datensatz. (Befehllänge <= 10000 Zeichen)
FB_DBRecordDelete [▶ 77]	Löscht einen Datensatz aus einer Tabelle
FB_DBStoredProcedures [▶ 81]	Führt eine Gespeicherte Prozedur aus.
FB_DBStoredProceduresRecordReturn [▶ 89]	Führt eine Gespeicherte Prozedur aus und gibt einen Datensatz zurück
FB_DBStoredProceduresRecordArray [▶ 82]	Führt eine Gespeicherte Prozedur aus und gibt mehrere Datensätze zurück

Funktionen

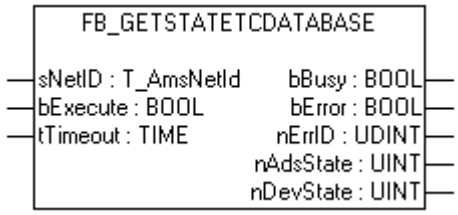
Name	Beschreibung
F_GetVersionTcDatabase [▶ 90]	Versionsinformationen abrufen.

Datentypen

Name
ST_DBColumnCfg [▶ 90]
ST_DBXMLCfg [▶ 91]
ST_ADSDevXMLCfg [▶ 91]
ST_DBSQLError [▶ 92]
ST_DBParameter [▶ 92]
E_DbColumnTypes [▶ 93]
E_DBTypes [▶ 93]
E_DBValueType [▶ 94]
E_DBWriteModes [▶ 94]
E_DBParameterTypes [▶ 94]

7.1 Funktionsbausteine

7.1.1 FB_GetStateTcDatabase



Mit dem Funktionsbaustein FB_GetStateTcDatabase kann der aktuelle Status des Database Servers abgefragt werden.

VAR_INPUT

```
VAR_INPUT
    sNetID          : T_AmsNetID;
    bExecute        : BOOL;
    tTimeout        : TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Timeoutzeit an.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
```

```
    nAdsState   : UINT;
    nDevState   : UINT;
END_VAR
```

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

bError : Wird TRUE, sobald ein Fehler eintritt.

nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code. [▶ 135]

nAdsState : Enthält die Zustandskennzahl des ADS-Zielgerätes. Die hier zurückgelieferten Codes sind festgelegt für alle ADS-Server:

- ADSSTATE_INVALID =0 ;
- ADSSTATE_IDLE =1 ;
- ADSSTATE_RESET =2 ;
- ADSSTATE_INIT =3 ;
- ADSSTATE_START =4 ;
- ADSSTATE_RUN =5 ;
- ADSSTATE_STOP =6 ;
- ADSSTATE_SAVECFG =7 ;
- ADSSTATE_LOADCFG =8 ;
- ADSSTATE_POWERFAILURE =9 ;
- ADSSTATE_POWERGOOD =10 ;
- ADSSTATE_ERROR =11;

nDevState : Enthält die spezifische Zustandskennzahl des ADS-Zielgerätes. Die hier zurückgelieferten Codes sind Zusatzinformationen, die für das ADS-Gerät spezifisch sind.

- 1 = TwinCAT Database Server gestartet
- 2 = Das zyklische lesen bzw. schreiben gestartet

Voraussetzungen

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

7.1.2 FB_DBReloadConfig

Mit dem Funktionsbaustein FB_DBReloadConfig kann die XML-Konfigurationsdatei neu eingelesen werden. Wurden Änderungen an der XML-Konfigurationsdatei vorgenommen, muss dem Database Server die Änderungen mit Hilfe des FB_DBReloadConfig bekannt gemacht werden.

VAR_INPUT

```
VAR_INPUT
    sNetID          : T_AmsNetId;
    bExecute        : BOOL;
    tTimeout        : TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

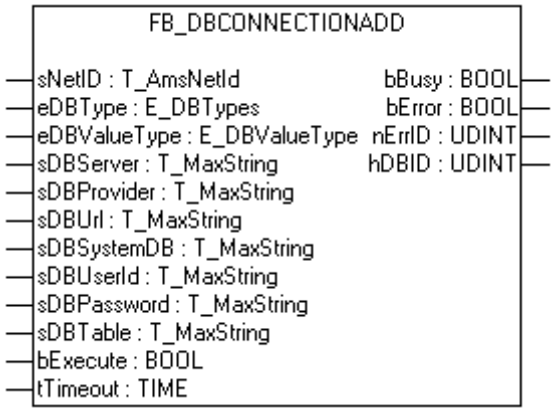
bError : Wird TRUE, sobald ein Fehler eintritt.

nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code. [▶ 135]

Voraussetzungen

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

7.1.3 FB_DBConnectionAdd



Mit dem Funktionsbaustein FB_DBConnectionAdd können weitere Datenbankverbindungen an die XML-Konfigurationsdatei angefügt werden.

VAR_INPUT

```
VAR_INPUT
    sNetID          :T_AmsNetId;
    eDBType         :E_DBTypes;
    eDBValueType    :E_DBValueType;
    sDBServer       :T_MaxString;
    sDBProvider     :T_MaxString;
    sDBUrl          :T_MaxString;
    sDBSystemDB     :T_MaxString;
    sDBUserId       :T_MaxString;
    sDBPassword     :T_MaxString;
    sDBTable        :T_MaxString;
    bExecute        :BOOL;
    tTimeout        :TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

eDBType : Gibt den Typen der Datenbank an z.B. 'Mobile-Server'.

eDBValueType : Gibt an, in welcher Form die Werte gespeichert sind bzw. werden.

sDBServer : Gibt den Namen des Servers an : Optional.

sDBProvider : Gibt den Provider der Datenbank : Optional.

sDBUrl : Gibt den Pfad der Datenbank an .

sSystemDB : Nur bei Access Datenbanken. Gibt den Pfad zu der MDW-Datei an

sUserId : Gibt den Benutzernamen an, mit dem sich angemeldet werden soll.

sPassword : Gibt das Password an.

sDBTable : Gibt den Namen der Tabelle an in die die Werte geschrieben werden sollen.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Zeit bis zum Abbrechen der Funktion an.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    bErrID      : UDINT;
    hDBID       : UDINT;
END_VAR
```

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

bError : Wird TRUE, sobald ein Fehler eintritt.

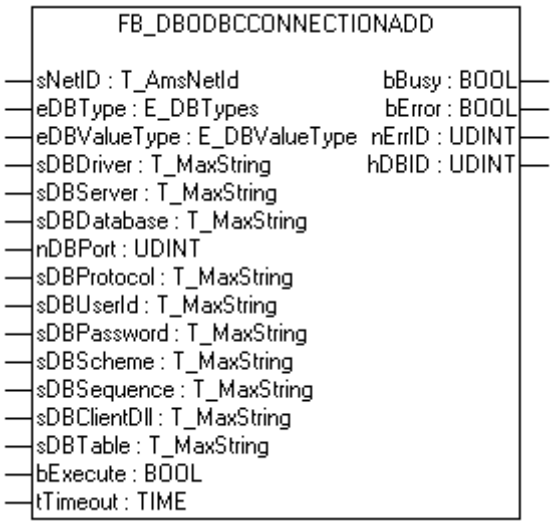
nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code. [▶ 135]

hDBID : Liefert die ID der Datenbank zurück

Voraussetzungen

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

7.1.4 FB_DBOdbcConnectionAdd



Mit dem Funktionsbaustein FB_DBOdbcConnectionAdd können weitere ODBC-Datenbankverbindungen an die XML-Konfigurationsdatei angefügt werden.

VAR_INPUT

```
VAR_INPUT
    sNetID          :T_AmsNetId;
    eDBType         :E_DBTypes;
    eDBValueType    :E_DBValueType;
    sDBDriver       :T_MaxString;
    sDBServer       :T_MaxString;
    sDBDatabase     :T_MaxString;
    nDBPort         :UDINT;
    sDBProtocol     :T_MaxString;
    sDBUserId       :T_MaxString;
    sDBPassword     :T_MaxString;
    sDBScheme       :T_MaxString;
    sDBSequence     :T_MaxString;
    sDBClientDll    :T_MaxString;
    sDBTable        :T_MaxString;
    bExecute        :BOOL;
    tTimeout        :TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

eDBType : Gibt den Typen der Datenbank an z.B. 'Mobile-Server'.

eDBValueType : Gibt an, in welcher Form die Werte gespeichert sind bzw. werden.

sDBDriver : Gibt den Namen des zu verwendenden ODBC-Drivers an

sDBServer : Gibt den Namen des Servers an.

sDBDatabase : Gibt den Namen der Datenbank an .

nDBPort : Gibt den Port der ODBC-Verbindung an.

sDBProtocol : Gibt das zu verwendende Protokoll an (TCPIP).

sDBUserId : Gibt den Benutzernamen an.

sDBPassword : Gibt das zu verwendende Passwort an.

sDBScheme : Gibt das zu verwendende Datenbankschema an.

sDBSequence : Gibt den Sequenznamen bei Oracle Datenbanken an.

sDBClientDll : Enthält den Pfad zur fbclient.dll. (Nur für Firebird/Interbase Datenbanken)

sDBTable : Gibt den Namen der Tabelle an in die die Werte geschrieben werden sollen.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Zeit bis zum Abbrechen der Funktion an.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    bErrID      : UDINT;
    hDBID       : UDINT;
END_VAR
```

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

bError : Wird TRUE, sobald ein Fehler eintritt.

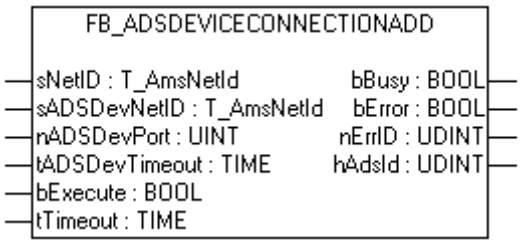
nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code. [▶ 135]

hDBID : Liefert die ID der Datenbank zurück

Voraussetzungen

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

7.1.5 FB_AdsDeviceConnectionAdd



Mit dem Funktionsbaustein FB_AdsDeviceConnectionAdd können AdsDevices in der XML - Konfigurationsdatei deklariert werden.

VAR_INPUT

```
VAR_INPUT
    sNetID          : T_AmsNetID;
    sADSDevNetID    : T_AmsNetID;
    nADSDevPort     : UINT;
    tADSDevTimeout  : TIME;
    bExecute        : BOOL;
    tTimeout        : TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

sADSDevNetID : Ist ein String, der die AMS-Netzwerkkennung des ADS-Gerätes enthält.

nADSDevPort : Gibt den Port des ADS-Gerätes an.

tADSDevTimeout : Gibt die Timeoutzeit des ADS-Gerätes an.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Timeoutzeit an.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    hAdsId      : UDINT;
END_VAR
```

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

bError : Wird TRUE, sobald ein Fehler eintritt.

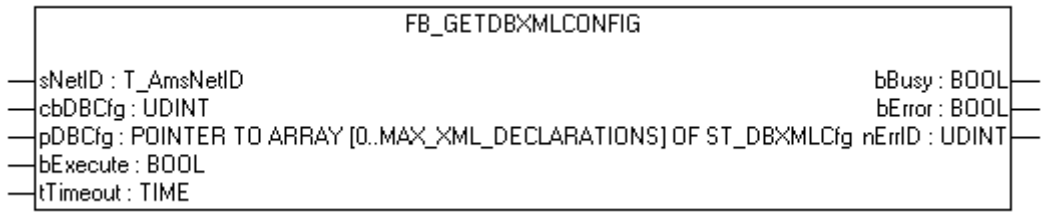
nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code. [▶ 135]

hAdsId : Gibt die ID des ADS-Gerätes zurück.

Voraussetzungen

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

7.1.6 FB_GetDBXMLConfig



Mit dem Funktionsbaustein FB_GetDBXMLConfig können alle Datenbanken, die in der XML - Konfigurationsdatei deklariert sind, ausgelesen werden.

VAR_INPUT

```
VAR_INPUT
    sNetID          : T_AmsNetId;
    cbDBCfg         : UDINT;
    pDBCfg          : POINTER TO ARRAY [0.. MAX_XML_DECLARATIONS] OF ST_DBXMLCfg
    bExecute        : BOOL;
    tTimeout        : TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

cbDBCfg : Gibt die Länge des Arrays zurück, in das die Konfigurationen geschrieben werden sollen.

pDBCfg : Gibt die Pointer-Adresse des Arrays an, in das die Konfigurationen geschrieben werden sollen.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Zeit bis zum Abbrechen der Funktion an

VAR_OUTPUT

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

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

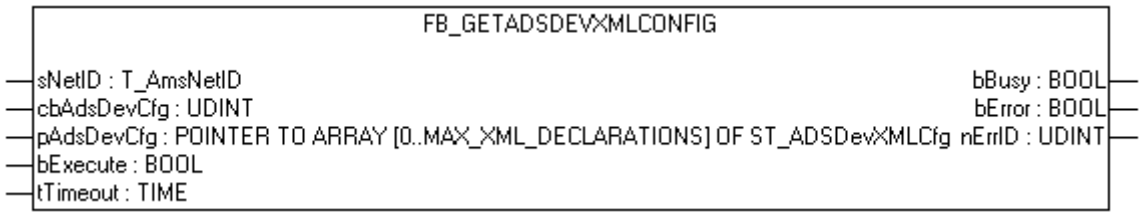
bError : Wird TRUE, sobald ein Fehler eintritt.

nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code. [▶ 135]

Voraussetzungen

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

7.1.7 FB_GetAdsDevXMLConfig



Mit dem Funktionsbaustein FB_GetAdsDevXMLConfig können alle ADS-Geräte, die in der XML - Konfigurationsdatei deklariert sind, ausgelesen werden.

VAR_INPUT

```
VAR_INPUT
    sNetID          : T_AmsNetId;
    cbAdsDevCfg     : UDINT;
    pAdsDevCfg      : POINTER TO ARRAY [0.. MAX_XML_DECLARATIONS] OF ST_ADSDevXMLCfg
    bExecute        : BOOL;
    tTimeout        : TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

cbAdsDevCfg : Gibt die Länge des Arrays zurück, in das die Konfigurationen geschrieben werden sollen.

pAdsDevCfg : Gibt die Pointer-Adresse des Arrays an, in das die Konfigurationen geschrieben werden sollen.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Zeit bis zum Abbrechen der Funktion an

VAR_OUTPUT

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

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

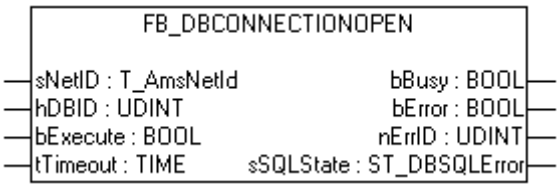
bError : Wird TRUE, sobald ein Fehler eintritt.

nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code. [▶ 135]

Voraussetzungen

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

7.1.8 FB_DBConnectionOpen



Mit dem Funktionsbaustein FB_DBConnectionOpen können Verbindungen zu Datenbanken geöffnet werden. Dies kann den Lese- Schreibzugriff mit den Funktionsblöcken FB_DBWrite, FB_DBRead, FB_DBRecordInsert und FB_FBRecordSelect beschleunigen.

VAR_INPUT

```
VAR_INPUT
    sNetID          : T_AmsNetId;
    hDBID           : DINT;
    bExecute        : BOOL;
    tTimeout        : TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hDBID : Gibt die ID der zu verwendenden Datenbank an.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Zeit bis zum Abbrechen der Funktion an

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    sSQLState   : ST_DBSQLError;
END_VAR
```

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

bError : Wird TRUE, sobald ein Fehler eintritt.

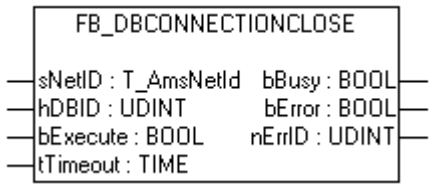
nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code [▶ 135] bzw. TcDatabaseSrv_Error_Codes [▶ 138].

sSQLState : Liefert den SQL - Fehlercode [▶ 92] des entsprechenden Datenbanktyps

Voraussetzungen

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

7.1.9 FB_DBConnectionClose



Mit dem Funktionsbaustein FB_DBConnectionOpen können Verbindungen zu Datenbanken geschlossen werden. Wenn zuvor eine Verbindung zu einer Datenbank geöffnet wurde, ist es zwingend notwendig diese wieder zu schließen..

VAR_INPUT

```
VAR_INPUT
    sNetID          : T_AmsNetId;
    hDBID           : DINT;
    bExecute        : BOOL;
    tTimeout        : TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hDBID : Gibt die ID der zu verwendenden Datenbank an.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Zeit bis zum Abbrechen der Funktion an

VAR_OUTPUT

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

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

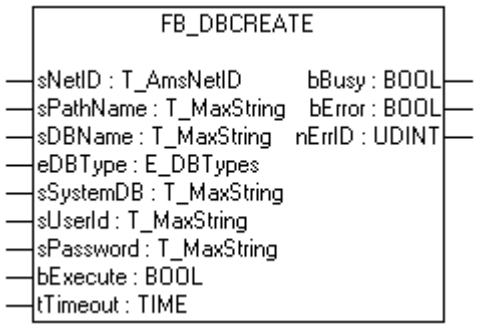
bError : Wird TRUE, sobald ein Fehler eintritt.

nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code. [▶ 135]

Voraussetzungen

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

7.1.10 FB_DBCreate



Mit dem Funktionsbaustein FB_DBCreate können Datenbanken angelegt werden. MS SQL-Datenbanken, MS SQL Compact Datenbanken, MS Access Datenbanken und XML-Datenbanken können mit diesem Baustein erstellt werden.

ASCII-Dateien können und müssen nicht mit dem Funktionsblock FB_DBCreate erzeugt werden. Sie werden beim ersten Schreibzugriff automatisch erzeugt, wenn Sie nicht vorhanden sind. Sie müssen nur in der XML-Konfigurationsdatei deklariert werden.

Das Erstellen der DB2, Oracle, MySQL, PostgreSQL, InterBase und Firebird Datenbanken ist nicht möglich. Des Weiteren ist das überschreiben von existierenden Datenbanken nicht möglich. Der Funktionsbaustein FB_DBCreate gibt in diesem Fall einen Fehler zurück.

VAR_INPUT

```
VAR_INPUT
    sNetID      : T_AmsNetID;
    sPathName   : T_MaxString;
    sDBName     : T_MaxString;
    eDBType     : E_DBTypes;
    sSystemDB   : T_MaxString;
    sUserId     : T_MaxString;
    sPassword   : T_MaxString;
    bExecute    : BOOL;
    tTimeout    : TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

sPathName : Gibt den Pfad der Datenbank an.

sDBName : Gibt den Namen der Datenbank an, welche erstellt werden soll

eDBType : Gibt den Typ der Datenbank an, welche erstellt werden soll

sSystemDB : Nur bei "Access-Datenbanken". Enthält den Pfad zur MDW-Datei.

sUserId : Gibt den Benutzernamen für die Registrierung an.

sPassword : Enthält das Passwort.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Timeoutzeit an.

VAR_OUTPUT

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

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

bError : Wird TRUE, sobald ein Fehler eintritt.

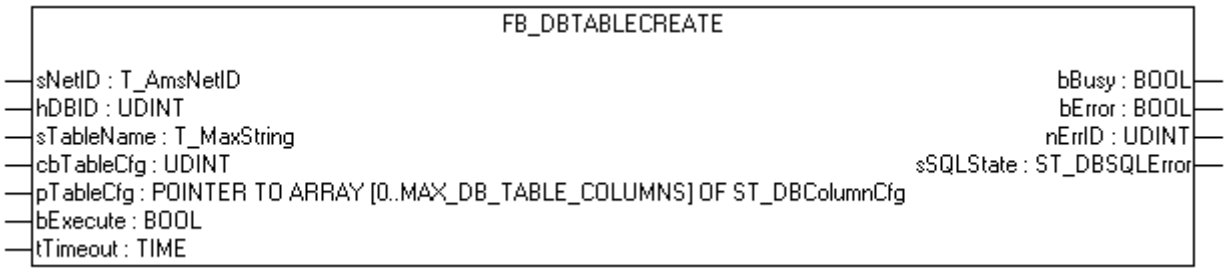
nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code. [▶ 135]

Sollen die neu erstellten Datenbanken vom TwinCAT Database Server verwendet werden, so müssen die Verbindungsdaten mit Hilfe des Funktionsblock FB_DBConnectionADD in die XML-Konfigurationsdatei geschrieben werden.

Voraussetzungen

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

7.1.11 FB_DBTableCreate



Mit dem Funktionsbaustein FB_DBTableCreate können Tabellen in Datenbanken mit beliebiger Tabellenstruktur angelegt werden.

VAR_INPUT

```
VAR_INPUT
    sNetID          : T_AmsNetID;
    hDBID           : UDINT;
    sTableName      : T_MaxString;
    cbTableCfg      : UDINT;
    pTableCfg       : POINTER TO ARRAY[0..MAX_DB_TABLE_COLUMNS] OF ST_DBColumnCfg;
    bExecute        : BOOL;
    tTimeout        : TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hDBID : Ist die ID der zu verwendenden Datenbank.

sTableName : Gibt den Namen der Tabelle an.

cbTableCfg : Gibt die Länge der Arrays zurück in dem die Spalten konfiguriert sind.

pTableCfg : Gibt die Pointer-Adresse des Table struct arrays an. In diesem Array werden die einzelnen Spalten beschrieben.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Timeoutzeit an.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
```

```
    nErrID      : UDINT;
    sSQLState   : ST_DBSQLError;
END_VAR
```

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

bError : Wird TRUE, sobald ein Fehler eintritt.

nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code [▶ 135] bzw. TcDatabaseSrv Error Codes [▶ 138].

sSQLState : Liefert den SQL - Fehlercode [▶ 92] des entsprechenden Datenbanktyps

Voraussetzungen

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

7.1.12 FB_DBCyclicRdWrt

Mit dem Funktionsbaustein FB_DBCyclicRdWrt kann das zyklische Loggen \ Schreiben der Variablen gestartet bzw. gestoppt werden.

VAR_INPUT

```
VAR_INPUT
    sNetID      : T_AmsNetId;
    bExecute    : BOOL;
    tTimeout    : TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

bExecute : Mit einer steigenden Flanke wird der Lese-/Schreibzyklus gestartet und mit einer fallenden Flanke gestoppt.

tTimeout : Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    sSQLState   : ST_DBSQLError;
END_VAR
```

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

bError : Wird TRUE, sobald ein Fehler eintritt.

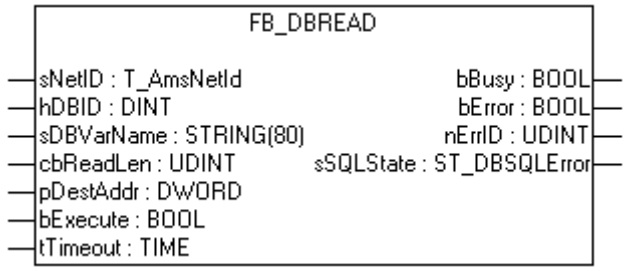
nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code [▶ 135] bzw. TcDatabaseSrv Error Codes [▶ 138].

sSQLState : Liefert den SQL - Fehlercode [▶ 92] des entsprechenden Datenbanktyps

Voraussetzungen

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

7.1.13 FB_DBRead



Mit dem Funktionsbaustein FB_DBRead können Werte aus einer Datenbank ausgelesen werden. Der Funktionsbaustein sucht in der Datenbanktabelle in der Spalte "Name" nach dem angegebenen sDBVarName und gibt dann den entsprechenden Wert aus der Spalte "Value" aus. Ist der gesuchte sDBVarName mehrmals in der Datenbanktabelle vorhanden, so wird der erste gefundene Datensatz ausgegeben.

VAR_INPUT

```
VAR_INPUT
    sNetID          : T_AmsNetId;
    hDBID           : DINT;
    sDBVarName      : STRING(80);
    cbReadLen       : UDINT;
    pDestAddr       : DWORD;
    bExecute        : BOOL;
    tTimeout        : TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hDBID : Gibt die ID der zu verwendenden Datenbank an.

sDBVarName : Gibt den Namen der Variable an, welche gelesen werden soll

cbReadLen : Gibt die Länge des zu lesenden Puffers an.

pDestAddr : Enthält die Adresse des Puffers, der die gelesenen Daten aufnehmen soll

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Zeit bis zum Abbrechen der Funktion an

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    sSQLState   : ST_DBSQLError;
END_VAR
```

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

bError : Wird TRUE, sobald ein Fehler eintritt.

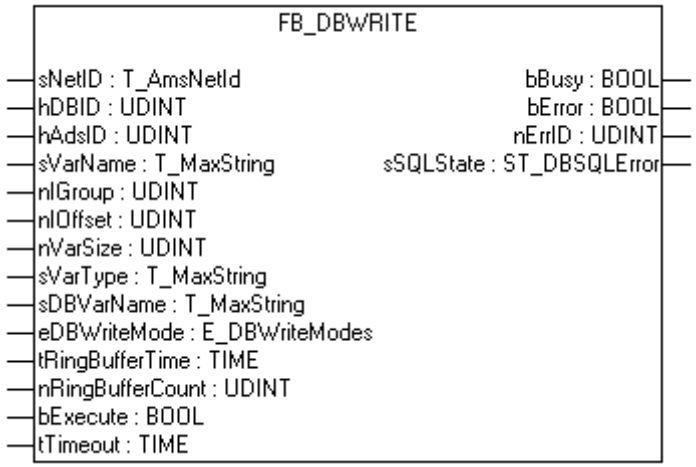
nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code [▶ 135] bzw. TcDatabaseSrv_Error_Codes [▶ 138].

sSQLState : Liefert den SQL - Fehlercode [▶ 92] des entsprechenden Datenbanktyps

Voraussetzungen

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

7.1.14 FB_DBWrite



Mit dem Funktionsbaustein FB_DBWrite können die Werte einzelner Variablen in Datenbanken geschrieben werden.
Die Tabellenstruktur muss die Spalten "Timestamp", "Name", und "Value" besitzen siehe "SQL Compact Datenbank" [▶ 41]. Um den Funktionsbaustein benutzen zu können, muss die Datenbank, in die geschrieben werden soll und das ADS-Gerät, vom dem die Variable gelesen werden soll, in der XML - Konfigurationsdatei deklariert werden.

VAR_INPUT

```
VAR_INPUT
    sNetID          : T_AmsNetID;
    hDBID           : UDINT;
    hAdsID          : UDINT;
    sVarName        : T_MaxString;
    nIGroup         : UDINT;
    nIOffset        : UDINT;
    nVarSize        : UDINT;
    sVarType        : T_MaxString;
    sDBVarName      : T_MaxString;
    eDBWriteMode    : E_DBWriteModes;
    tRingBufferTime : TIME;
    nRingBufferCount: UDINT;
    bExecute        : BOOL;
    tTimeout        : TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hDBID : Ist die ID der zu verwendenden Datenbank.

hAdsID : Ist die ID des zu verwendenden ADS-Gerätes.

sVarName : Gibt den Namen der Variable an.

nIGroup : IndexGroup der Variable (optional Nur bei BC9000).

nIOffset : IndexOffset der Variable (optional Nur bei BC9000).

nVarSize : Größe der Variable in Byte (optional Nur bei BC9000)

sVarType : Datentyp der Variable (optional Nur bei BC9000)

Mögliche Variablendatentypen: "BOOL" / "LREAL" / "REAL" / "INT16" / "DINT" / "USINT" / "BYTE" / "UDINT" / "DWORD" / "UINT16" / "WORD" / "SINT"

sDBVarName : In der Datenbank zu verwendende Variablenname.

eDBWriteMode : Gibt an ob die Werte in neuen Datensätzen unten angehangen werden, oder ob die vorhandenen Datensätze aktualisiert werden.

tRingBufferTime : Gibt das maximale Alter von Datensätzen in einer Tabelle an. (nur bei Ringbuffer_WriteMode)

nRingBufferCount : Gibt die maximale Anzahl von Datensätzen in einer Tabelle an. (nur bei Ringbuffer_WriteMode)

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout; : Gibt die Zeit bis zum Abbrechen der Funktion an.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    sSQLState   : ST_DBSQLError;
END_VAR
```

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

bError : Wird TRUE, sobald ein Fehler eintritt.

nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code [▶ 135] bzw. TcDatabaseSrv_Error_Codes [▶ 138].

sSQLState : Liefert den SQL - Fehlercode [▶ 92] des entsprechenden Datenbanktyps

Werte loggen von ADS-Devices (außer BC9000)	Werte loggen von BC9000

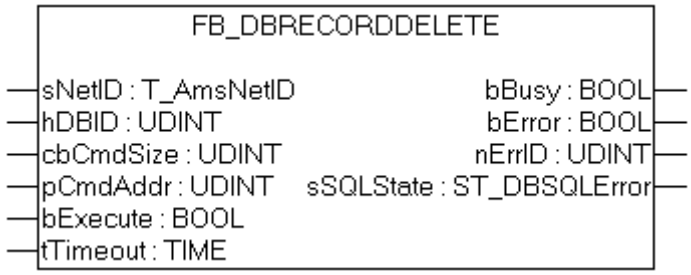
`FB_DBWrite1(`
`  sNetID:= ,`
`  hDBID:= 1,`
`  hAdsID:= 1,`
`  sVarName:= 'MAIN.TestVar',`
`  sDBVarName:= 'DBTestVar',`
`  eDBWriteMode:= eDBWriteMode_Append,`
`  bExecute:= TRUE,`
`  tTimeout:= T#15s,`
`  bBusy=> busy,`
`  bError=> err,`
`  nErrID=> errid,`
`  sSQLState=> sqlstate`
`);`
	`FB_DBWrite1(`
	`  sNetID:= ,`
	`  hDBID:= 1,`
	`  hAdsID:= 1,`
	`  sVarName:= 'MAIN.TestVar',`
	`  nIGroup:= 16448,`
	`  nIOffset:= 0,`
	`  nVarSize:= 16,`
	`  sVarType:= 'REAL',`
	`  sDBVarName:= 'DBTestVar',`
	`  eDBWriteMode:= eDBWriteMode_Append,`
	`  bExecute:= TRUE,`
	`  tTimeout:= T#15s,`
	`  bBusy=> busy,`
	`  bError=> err,`
	`  nErrID=> errid,`
	`  sSQLState=> sqlstate`
	`);`

Voraussetzungen

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

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0	CX (ARM)	

7.1.15 FB_DBRecordDelete



Mit dem Funktionsbaustein FB_DBRecordDelete können einzelne Datensätze aus einer Datenbank gelöscht werden. Mit diesem Baustein können SQL DELETE Befehle mit bis zu 10000 Zeichen ausgeführt werden. Für die Benutzung des Funktionsblocks ist es erforderlich, die Datenbank, aus der Datensätze gelöscht werden sollen, in der XML - Konfigurationsdatei zu deklarieren.

VAR_INPUT

```
VAR_INPUT
    sNetID          : T_AmsNetId;
    hDBID           : UDINT;
    cbCmdSize       : UDINT;
    pCmdAddr        : UDINT;
    bExecute        : BOOL;
    tTimeout        : TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hDBID : Gibt die ID der zu verwendenden Datenbank an.

cbCmdSize : Gibt die länge des INSERT-Befehls an.

pCmdAddr : Pointer zum ausführenden INSERT-Befehls.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Zeit bis zum Abbrechen der Funktion an.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    sSQLState   : ST_DBSQLError;
END_VAR
```

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

bError : Wird TRUE, sobald ein Fehler eintritt.

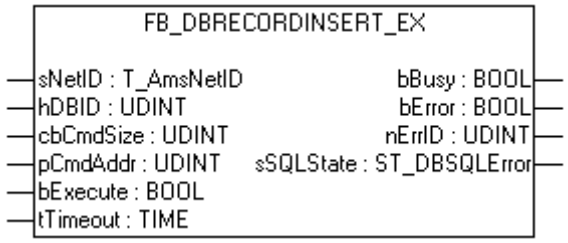
nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code [▶ 135] bzw. TcDatabaseSrv Error Codes [▶ 138].

sSQLState : Liefert den SQL - Fehlercode [▶ 92] des entsprechenden Datenbanktyps

Voraussetzungen

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

7.1.16 FB_DBRecordInsert_EX



Mit dem Funktionsbaustein FB_DBRecordInsert_EX können einzelne Datensätze mit beliebiger Struktur in eine Datenbank geschrieben werden. Mit diesem Baustein können SQL INSERT Befehle mit bis zu 10000 Zeichen ausgeführt werden.
Für die Benutzung des Funktionsblocks ist es erforderlich, die Datenbank, in die geschrieben werden soll, in der XML - Konfigurationsdatei zu deklarieren.

VAR_INPUT

```
VAR_INPUT
    sNetID          : T_AmsNetId;
    hDBID           : UDINT;
    cbCmdSize       : UDINT;
    pCmdAddr        : UDINT;
    bExecute        : BOOL;
    tTimeout        : TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hDBID : Gibt die ID der zu verwendenden Datenbank an.

cbCmdSize : Gibt die länge des INSERT-Befehls an.

pCmdAddr : Pointer zum Ausführenden INSERT-Befehls.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Zeit bis zum Abbrechen der Funktion an.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    sSQLState   : ST_DBSQLError;
END_VAR
```

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

bError : Wird TRUE, sobald ein Fehler eintritt.

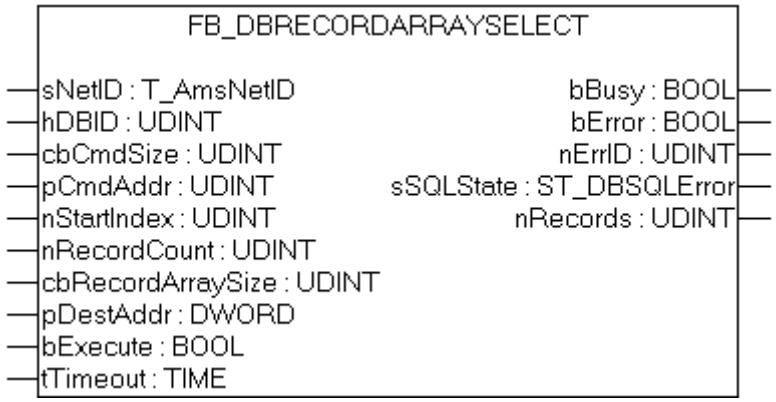
nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code [▶ 135] bzw. TcDatabaseSrv Error Codes [▶ 138].

sSQLState : Liefert den SQL - Fehlercode [▶ 92] des entsprechenden Datenbanktyps

Voraussetzungen

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

7.1.17 FB_DBRecordArraySelect



Mit dem Funktionsbaustein FB_DBRecordArraySelect können mehrere Datensätze mit beliebiger Struktur aus einer Datenbanken ausgelesen werden. Mit diesem Baustein können sie einen SQL SELECT Befehl mit bis zu 10000 Zeichen ausführen.

Dieser Funktionsbaustein ist nicht kompatibel mit ASCII-Files.

VAR_INPUT

```
VAR_INPUT
    sNetID              : T_AmsNetID;
    hDBID               : UDINT;
    cbCmdSize           : UDINT;
    pCmdAddr            : UDINT;
    nStartIndex         : UDINT;
    nRecordCount        : UDINT;
    cbRecordArraySize   : UDINT;
    pDestAddr           : DWORD;
    bExecute            : BOOL;
    tTimeout            : TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hDBID : Gibt die ID der zu verwendenden Datenbank an.

cbCmdSize : Gibt die länge des SELECT-Befehls an, der ausgeführt werden soll.

pCmdSize : Gibt die Pointer Adresse einer String Variablen mit dem auszuführenden SQL-Befehl an.

nStartIndex : Gibt den Index des ersten zu lesenden Datensatzes an.

nRecordCount : Gibt die Anzahl der zu lesenden Datensätzen an.

cbRecordArraySize : Gibt die Größe des Strukturarrays in Byte an.

pDestAddr : Gibt die Adresse des Strukturarrays an in das die Datensätze geschrieben werden soll.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Zeit bis zum Abbrechen der Funktion an.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    sSQLState   : ST_DBSQLError;
    nRecords    : UDINT;
END_VAR
```

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

bError : Wird TRUE, sobald ein Fehler eintritt.

nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code [▶ 135] bzw. TcDatabaseSrv_Error_Codes [▶ 138].

sSQLState : Liefert den SQL - Fehlercode [▶ 92] des entsprechenden Datenbanktyps

nRecords : Liefert die Anzahl der Datensätze.

Beispiel in ST:

Da die Tabelle, aus der Datensätze gelesen werden sollen, folgende Struktur besitzt...

Spaltenname	Datentyp
ID	bigint
Timestamp	datetime
Name	nvarchar(80)
Value	float

... muss eine SPS-Struktur erstellt werden, die einen vergleichbaren Aufbau besitzt.

```
TYPE ST_Record :
STRUCT
    ID          : T_ULARGE_INTEGER;
    Timestamp   : DT;
    Name        : STRING(80);
    VALUE      : LREAL;
END_STRUCT
END_TYPE
```

Um den Datentyp T_ULARGE_INTEGER verwenden zu können, muss die Bibliothek TcUtilities.lib eingebunden werden.

Bei ARM - Prozessoren müssen auf Grund des Byte-Alignment die Datentypen anders geordnet und ein "Dummy-BYTE" hinzugefügt werden.

```
TYPE ST_Record :
STRUCT
    ID         : T_ULARGE_INTEGER;
    Timestamp  : DT;
    Value      : LREAL;
    Name       : STRING(80);
    Dummy      : BYTE;
END_STRUCT
END_TYPE
```

```
PROGRAM MAIN
VAR
    FB_DBRecordArraySelect1 : FB_DBRecordArraySelect;
    cmd             : T_Maxstring   := 'SELECT * FROM myTable';
    (* Unter ARM*)
    (*cmd           : T_Maxstring   := 'SELECT ID,Timestamp,Value,Name FROM myTable'*)
    (*----------*)
    recordArray         : ARRAY [1..5] OF ST_Record;
    busy            : BOOL;
    err             : BOOL;
    errid           : UDINT;
    sqlstate            : ST_DBSQLError;
    recAnz          : UDINT;
END_VAR
```

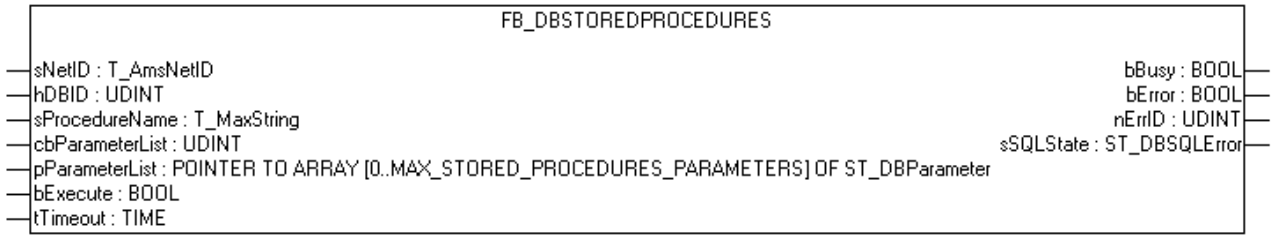
SPS-Program

```
FB_DBRecordArraySelect1(
    sNetID:= ,
    hDBID:= 1,
    cbCmdSize:= SIZEOF(cmd),
    pCmdAddr:= ADR(cmd),
    nStartIndex:= 0,
    nRecordCount:= 5,
    cbRecordArraySize:= SIZEOF(recordArray),
    pDestAddr:= ADR(recordArray),
    bExecute:= TRUE,
    tTimeout:= T#15s,
    bBusy=> busy,
    bError=> err,
    nErrID=> errid,
    sSQLState=> sqlstate,
    nRecords=> recAnz);
```

Voraussetzungen

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

7.1.18 FB_DBStoredProcedures



Mit dem Funktionsbaustein FB_DBStoredProcedures können gespeichert Prozeduren aufgerufen werden. Sie können Parameter mit übergeben die in den Gespeicherten Prozeduren verwendet werden.

VAR_INPUT

```
VAR_INPUT
    sNetID          : T_AmsNetID        :='';
    hDBID           : UDINT             :=1;
    sProcedureName      : T_MaxString   :='';
    cbParameterList     : UDINT;
    pParameterList      : POINTER TO ARRAY[0..MAX_STORED_PROCEDURES_PARAMETERS] OF ST_DBParameter;
    bExecute            : BOOL;
    tTimeout            : TIME          := T#15s;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hDBID : Gibt die ID der zu verwendenden Datenbank an.

sProcedureName : Gibt den Namen der Procedure an, welche ausgeführt werden soll

cbParameterList : Gibt die Länge der Parameterliste an.

pParameterList : Enthält die Adresse der Parameterliste

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Zeit bis zum Abbrechen der Funktion an

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    sSQLState   : ST_DBSQLError;
END_VAR
```

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

bError : Wird TRUE, sobald ein Fehler eintritt.

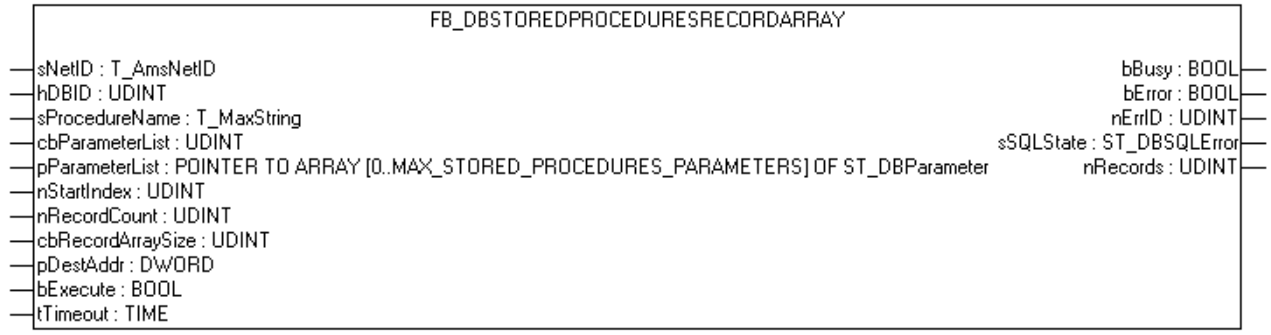
nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code [▶ 135] bzw. TcDatabaseSrv Error Codes [▶ 138].

sSQLState : Liefert den SQL - Fehlercode [▶ 92] des entsprechenden Datenbanktyps

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0	PC oder CX (x86)	TcDatabase.Lib (Ab TcDatabaseSrv Version 1.0.13)
TwinCAT v2.10.0	CX (ARM)	

7.1.19 FB_DBStoredProceduresRecordArray



Mit dem Funktionsbaustein FB_DBStoredProceduresRecordArray können Gespeichert Prozeduren aufgerufen werden die Datensätze zurückliefert. Im Gegensatz zum FB_DBStoredProceduresRecordReturn können mit diesem Baustein auch mehrere Datensätze mit einem Aufruf zurückgeliefert werden. Sie können Parameter mit übergeben, die in den Gespeicherten Prozeduren verwendet werden.

VAR_INPUT

```
VAR_INPUT
    sNetID      : T_AmsNetID        :='';
    hDBID       : UDINT             :=1;
    sProcedureName  : T_MaxString   :='';
    cbParameterList : UDINT;
    pParameterList  : POINTER TO ARRAY[0..MAX_STORED_PROCEDURES_PARAMETERS] OF ST_DBParameter;
    nStartIndex     : UDINT;
    nRecordCount    : UDINT
    cbRecordArraySize       : UDINT;
    pDesAddr        : DWORD;
    bExecute        : BOOL;
    tTimeout        : TIME          := T#15s;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hDBID : Gibt die ID der zu verwendenden Datenbank an.

sProcedureName : Gibt den Namen der Procedure an, welche ausgeführt werden soll

cbParameterList : Gibt die Länge der Parameterliste an.

pParameterList : Enthält die Adresse der Parameterliste

nStartIndex : Gibt den Index des ersten zu lesenden Datensatzes an.

nRecordCount : Gibt die Anzahl der zu lesenden Datensätzen an.

cbRecordArraySize : Gibt die Größe des Strukturarrays in Byte an.

pDestAddr : Gibt die Adresse des Strukturarrays an in das die Datensätze geschrieben werden soll.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Zeit bis zum Abbrechen der Funktion an

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    sSQLState   : ST_DBSQLError;
    nRecords    : UDINT;
END_VAR
```

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

bError : Wird TRUE, sobald ein Fehler eintritt.

nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code [▶ 135] bzw. TcDatabaseSrv Error Codes [▶ 138].

sSQLState : Liefert den SQL - Fehlercode [▶ 92] des entsprechenden Datenbanktyps

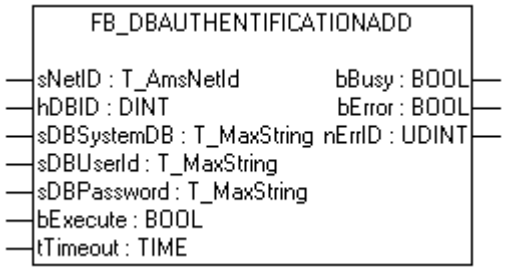
nRecords : Liefert die Anzahl der Datensätze.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0	PC oder CX (x86)	TcDatabase.Lib (Ab TcDatabaseSrv Version 1.0.17)
TwinCAT v2.10.0	CX (ARM)	

7.1.20 Obsolete

7.1.20.1 FB_DBAuthentificationAdd



Mit dem Funktionsbaustein FB_DBAuthentificationAdd können Authentifizierungsinformation an bereits deklarierte Datenbanken angefügt, bzw. bestehende Authentifizierungsinformationen überschrieben werden.

VAR_INPUT

```
VAR_INPUT
    sNetID      : T_AmsNetID;
    hDBID       : DINT;
    sDBSystemDB : T_MaxString;
    sDBUserId   : T_MaxString;
    sDBPassword : T_MaxString;
    bExecute    : BOOL;
    tTimeout    : TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hDBID : Ist die ID der zu verwendenden Datenbank.

sSystemDB : Nur bei Access Datenbanken. Gibt den Pfad zu der MDW-Datei an

sUserId : Gibt den Benutzernamen an, mit dem sich angemeldet werden soll.

sPassword : Gibt das Password an.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Timeoutzeit an.

VAR_OUTPUT

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

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

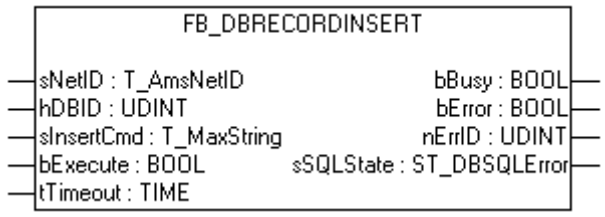
bError : Wird TRUE, sobald ein Fehler eintritt.

nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code. [▶ 135]

Voraussetzungen

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

7.1.20.2 FB_DBRecordInsert



Mit dem Funktionsbaustein FB_DBRecordInsert können einzelne Datensätze mit beliebiger Struktur in eine Datenbank geschrieben werden. Für die Benutzung des Funktionsblocks ist es erforderlich, die Datenbank, in die geschrieben werden soll, in der XML - Konfigurationsdatei zu deklarieren.

VAR_INPUT

```
VAR_INPUT
    sNetID          : T_AmsNetId;
    hDBID           : UDINT;
    sInsertCmd      : T_MaxString;
```

```
    bExecute        : BOOL;
    tTimeout        : TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hDBID : Gibt die ID der zu verwendenden Datenbank an.

sInsertCmd : Gibt den INSERT-Befehl an der ausgeführt werden soll.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Zeit bis zum Abbrechen der Funktion an.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    sSQLState   : ST_DBSQLError;
END_VAR
```

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

bError : Wird TRUE, sobald ein Fehler eintritt.

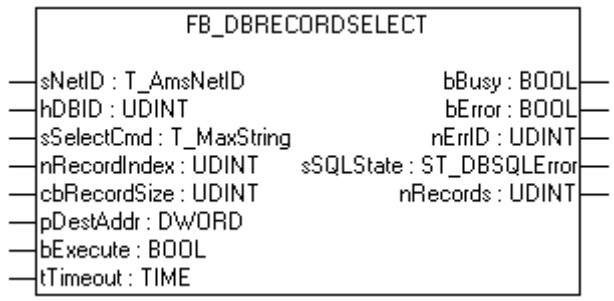
nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code [▶ 135] bzw. TcDatabaseSrv Error Codes [▶ 138].

sSQLState : Liefert den SQL - Fehlercode [▶ 92] des entsprechenden Datenbanktyps

Voraussetzungen

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

7.1.20.3 FB_DBRecordSelect



Mit dem Funktionsbaustein FB_DBRecordSelect können einzelne Datensätze mit beliebiger Struktur aus einer Datenbank ausgelesen werden.
Dieser Funktionsbaustein ist nicht kompatibel mit ASCII-Files.

VAR_INPUT

```
VAR_INPUT
    sNetID      : T_AmsNetID;
    hDBID       : UDINT;
    sSelectCmd      : T_MaxString;
    nRecordIndex    : UDINT;
    cbRecordSize    : UDINT;
    pDestAddr       : DWORD;
```

```
    bExecute        : BOOL;
    tTimeout        : TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hDBID : Gibt die ID der zu verwendenden Datenbank an.

sSelectCmd : Gibt den SELECT-Befehl an der ausgeführt werden soll.

nRecordIndex : Gibt den Index des zu lesenden Datensatzes an.

cbRecordSize : Gibt die Größe eines Datensatzes in Byte an.

pDestAddr : Gibt die Adresse der Struktur an in die der Datensatz geschrieben werden soll.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Zeit bis zum Abbrechen der Funktion an.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    sSQLState   : ST_DBSQLError;
    nRecords    : UDINT;
END_VAR
```

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

bError : Wird TRUE, sobald ein Fehler eintritt.

nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code [▶ 135] bzw. TcDatabaseSrv Error Codes [▶ 138].

sSQLState : Liefert den SQL - Fehlercode [▶ 92] des entsprechenden Datenbanktyps

nRecords : Liefert die Anzahl der Datensätze.

Beispiel in ST:

Da die Tabelle, aus der ein Datensatz gelesen werden soll, folgende Struktur besitzt...

Spaltenname	Datentyp
ID	bigint
Timestamp	datetime
Name	ntext
Value	float

... muss eine SPS-Struktur erstellt werden, die einen vergleichbaren Aufbau besitzt.

```
TYPE ST_Record :
STRUCT
    ID          : T_ULARGE_INTEGER;
    Timestamp   : DT;
    Name        : STRING;
    VALUE       : LREAL;
END_STRUCT
END_TYPE
```

Um den Datentyp T_ULARGE_INTEGER verwenden zu können, muss die Bibliothek TcUtilities.lib eingebunden werden.

Bei ARM - Prozessoren müssen auf Grund des Byte-Alignment die Datentypen anders geordnet und ein "Dummy-BYTE" hinzugefügt werden.

```
TYPE ST_Record :
STRUCT
    ID        : T_ULARGE_INTEGER;
    Timestamp : DT;
    Value     : LREAL;
    Name      : STRING;
    Dummy     : BYTE;
END_STRUCT
END_TYPE
```

```
PROGRAM MAIN
VAR
    FB_DBRecordSelect1      : FB_DBRecordSelect;
    cmd             : T_Maxstring   := 'SELECT * FROM myTable';
    (* Unter ARM*)
    (*cmd           : T_Maxstring   := 'SELECT ID,Timestamp,Value,Name FROM myTable'*)
    (*----------*)
    record          : ST_Record;
    busy            : BOOL;
    err             : BOOL;
    errid           : UDINT;
    recAnz          : DINT;
END_VAR
```

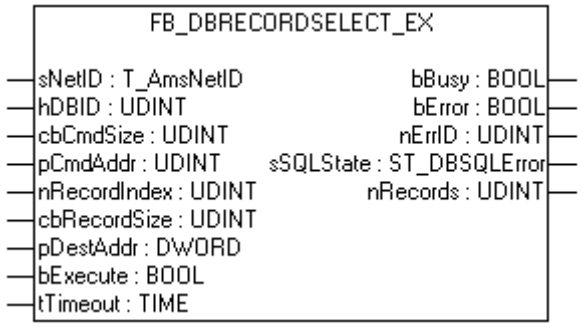
SPS-Program

```
FB_DBRecordSelect1(
    sNetID:= ,
    hDBID:= 2,
    sSelectCmd:= cmd,
    nRecordIndex:= 0,
    cbRecordSize:= SIZEOF(record),
    pDestAddr:= ADR(record),
    bExecute:= TRUE,
    tTimeout:= T#15s,
    bBusy=> busy,
    bError=> err,
    nErrID=> errid,
    nRecords=> recAnz);
```

Voraussetzungen

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

7.1.20.4 FB_DBRecordSelect_EX



Mit dem Funktionsbaustein FB_DBRecordSelect_EX können einzelne Datensätze mit beliebiger Struktur aus einer Datenbank ausgelesen werden. Mit diesem Baustein können sie einen SQL SELECT Befehl mit bis zu 10000 Zeichen ausführen.

Dieser Funktionsbaustein ist nicht kompatibel mit ASCII-Files.

VAR_INPUT

```
VAR_INPUT
    sNetID          : T_AmsNetID;
    hDBID           : UDINT;
```

```
    cbCmdSize       : UDINT;
    pCmdAddr        : UDINT;
    nRecordIndex    : UDINT;
    cbRecordSize    : UDINT;
    pDestAddr       : DWORD;
    bExecute        : BOOL;
    tTimeout        : TIME;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hDBID : Gibt die ID der zu verwendenden Datenbank an.

cbCmdSize : Gibt die länge des SELECT-Befehls an, der ausgeführt werden soll.

pCmdSize : Gibt die Pointer Adresse einer String Variablen mit dem auszuführenden SQL-Befehl an.

nRecordIndex : Gibt den Index des zu lesenden Datensatzes an.

cbRecordSize : Gibt die Größe eines Datensatzes in Byte an.

pDestAddr : Gibt die Adresse der Struktur an in die der Datensatz geschrieben werden soll.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Zeit bis zum Abbrechen der Funktion an.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    sSQLState   : ST_DBSQLError;
    nRecords    : UDINT;
END_VAR
```

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

bError : Wird TRUE, sobald ein Fehler eintritt.

nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code [▶ 135] bzw. TcDatabaseSrv Error Codes [▶ 138].

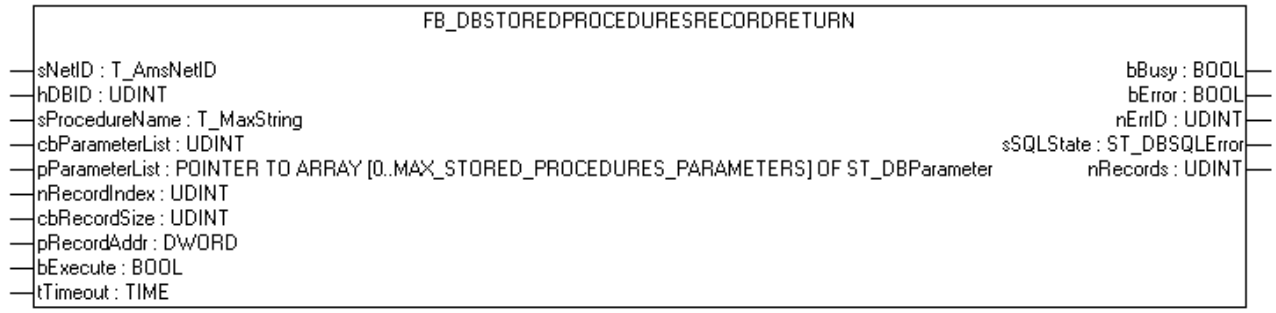
sSQLState : Liefert den SQL - Fehlercode [▶ 92] des entsprechenden Datenbanktyps

nRecords : Liefert die Anzahl der Datensätze.

Voraussetzungen

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

7.1.20.5 FB_DBStoredProceduresRecordReturn



Mit dem Funktionsbaustein FB_DBStoredProceduresRecordReturn können gespeichert Prozeduren aufgerufen werden die einen Datensatz zurückliefert. Sie können Parameter mit übergeben die in den Gespeicherten Prozeduren verwendet werden.

VAR_INPUT

```
VAR_INPUT
    sNetID          : T_AmsNetID                                :='';
    hDBID           : UDINT                                   :=1;
    sProcedureName      : T_MaxString                                 :='';
    cbParameterList     : UDINT;
    pParameterList      : POINTER TO ARRAY[0..MAX_STORED_PROCEDURES_PARAMETERS] OF ST_DBParameter;
    nRecordIndex        : UDINT;
    cbRecordSize        : UDINT;
    pRecordAddr         : DWORD;
    bExecute            : BOOL;
    tTimeout            : TIME                                    := T#15s;
END_VAR
```

sNetID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hDBID : Gibt die ID der zu verwendenden Datenbank an.

sProcedureName : Gibt den Namen der Procedure an, welche ausgeführt werden soll

cbParameterList : Gibt die Länge der Parameterliste an.

pParameterList : Enthält die Adresse der Parameterliste

nRecordIndex : Gibt den Index des zu lesenden Datensatzes an.

cbRecordSize : Gibt die Größe eines Datensatzes in Byte an.

pRecordAddr : Gibt die Adresse der Struktur an in die der Datensatz geschrieben werden soll.

bExecute : Mit der steigende Flanke wird das Kommando ausgeführt.

tTimeout : Gibt die Zeit bis zum Abbrechen der Funktion an

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    sSQLState   : ST_DBSQLError;
    nRecords    : UDINT;
END_VAR
```

bBusy : Kommando wird gerade per ADS übertragen. Solange bBusy auf TRUE, wird kein neues Kommando angenommen.

bError : Wird TRUE, sobald ein Fehler eintritt.

nErrID : Liefert bei einem gesetzten bError-Ausgang den ADS Error Code [▶ 135] bzw. TcDatabaseSrv Error Codes [▶ 138].

sSQLState : Liefert den SQL - Fehlercode [▶ 92] des entsprechenden Datenbanktyps

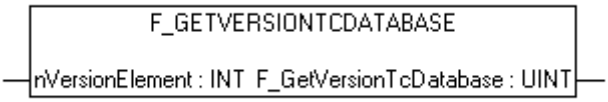
nRecords : Liefert die Anzahl der Datensätze.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0	PC oder CX (x86)	TcDatabase.Lib (Ab TcDatabaseSrv Version 1.0.13)
TwinCAT v2.10.0	CX (ARM)	

7.2 Funktionen

7.2.1 F_GetVersionTcDatabase



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

FUNCTION F_GetVersionTcDatabase: 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)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.3 Datentypen

7.3.1 ST_DBColumnCfg

```
TYPE ST_DBColumnCfg :
STRUCT
    sColumnName         : STRING(59);
    sColumnProperty     : STRING(59);
```

```
    eColumnType          : E_DbColumnTypes;
END_STRUCT
END_TYPE
```

sColumnName : Gibt den Namen der zu erzeugenden Spalte an.

sColumnProperty : Enthält bestimmte Eigenschaften der Spalte.

eColumnType : Gibt den Typ der Spalte an.

Voraussetzungen

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

7.3.2 ST_DBXMLCfg

```
TYPE ST_DBXMLCfg :
STRUCT
    sDBName      : STRING;
    sDBTable     : STRING;
    nDBID        : DINT;
    eDBType      : E_DBTypes;
END_STRUCT
END_TYPE
```

sDBName : Enthält den Namen der Datenbank.

sDBTable : Enthält den Namen der Tabelle.

nDBID : Liefert die ID der Datenbank zurück.

eDBType : Gibt den Typen der Datenbank an.

Voraussetzungen

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

7.3.3 ST_ADSDevXMLCfg

```
TYPE ST_ADSDevXMLCfg :
STRUCT
    sAdsDevNetID     : T_AmsNetID;
    tAdsDevTimeout   : TIME;
    nAdsDevID        : DINT;
    nAdsDevPort      : UINT;
END_STRUCT
END_TYPE
```

sAdsDevNetID : Ist ein String, der die AMS-Netzwerkkennung des ADS-Gerätes enthält.

tAdsDevTimeout : Gibt die Timeoutzeit des ADS-Gerätes an.

nAdsDevID : Liefert die ID des ADS-Gerätes zurück.

nAdsDevPort : Gibt den Port des ADS-Gerätes an.

Voraussetzungen

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

7.3.4 ST_DBSQLError

```
TYPE ST_DBSQLError :
STRUCT
    sSQLState       : STRING(5);
    nSQLErrorCode   : DINT;
END_STRUCT
END_TYPE
```

sSQLState : Enthält den 5 Charakter Fehlercode der sich an der SQL ANSI Norm orientiert.

nSQLErrorCode : Enthält den Datenbank spezifischen Fehlercode.

Ist kein Fehler aufgetreten enthält die Struktur die Werte:
sSQLState := **'00000'**;
nSQLErrorCode := **0**;

Voraussetzungen

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

7.3.5 ST_DBParameter

```
TYPE ST_DBParameter :
STRUCT
    sParameterName      : STRING(59);
    cbParameterValue    : UDINT;
    pParameterValue     : UDINT;
    eParameterDataType  : E_DBColumnTypes;
    eParameterType      : E_DBParameterTypes;
END_STRUCT
END_TYPE
```

sParameterName : Gibt den Namen des Parameters an.

cbParameterValue : Enthält die Größe der zu verwendenden Variable in Bytes.

pParameterValue : Enthält die Adresse der zu verwendenden Variable.

eParameterDataType : Gibt den Datentyp [▶ 93] des Parameters an.

pParameterValue : Gibt den Typ [▶ 94] des Parameters an.

Deklarationsbeispiel

Variablendeklaration

```
PROGRAM MAIN
VAR
    paraList: ARRAY [0..2] OF ST_DBParameter;

    p1: DINT := 3;
    p2: LREAL;
    p3: STRING;
END_VAR
```

SPS PROGRAMM

```
paraList[0].sParameterName := 'p1';
paraList[0].eParameterDataType := eDBColumn_Integer;
paraList[0].eParameterType := eDBParameter_Input;
paraList[0].cbParameterValue := SIZEOF(p1);
paraList[0].pParameterValue := ADR(p1);

paraList[1].sParameterName := 'p2';
paraList[1].eParameterDataType := eDBColumn_Float;
paraList[1].eParameterType := eDBParameter_Output;
paraList[1].cbParameterValue := SIZEOF(p2);
paraList[1].pParameterValue := ADR(p1);
```

```
paraList[2].sParameterName := 'p3';
paraList[2].eParameterDataType := eDBColumn_NText;
paraList[2].eParameterType := eDBParameter_Output;
paraList[2].cbParameterValue := SIZEOF(p3);
paraList[2].pParameterValue := ADR(p3);
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0	PC oder CX (x86)	TcDatabase.Lib (Ab TcDatabaseSrv Version 1.0.13)
TwinCAT v2.10.0	CX (ARM)	

7.3.6 E_DbColumnTypes

```
TYPE E_DbColumnTypes :
(
    eDBColumn_BigInt    :=0,
    eDBColumn_Integer   :=1,
    eDBColumn_SmallInt  :=2,
    eDBColumn_TinyInt   :=3,
    eDBColumn_Bit       :=4,
    eDBColumn_Money     :=5,
    eDBColumn_Float     :=6,
    eDBColumn_Real      :=7,
    eDBColumn_DateTime  :=8,
    eDBColumn_NText     :=9,
    eDBColumn_NChar     :=10,
    eDBColumn_Image     :=11,
    eDBColumn_NVarChar  :=12,
    eDBColumn_Binary    :=13,
    eDBColumn_VarBinary :=14
);
END_TYPE
```

Voraussetzungen

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

7.3.7 E_DBTypes

```
TYPE E_DBTypes :
(
    eDBType_Mobile_Server   := 0,
    eDBType_Access          := 1,
    eDBType_Sequal_Server   := 2,
    eDBType_ASCII           := 3,
    eDBType_ODBC_MySQL      := 4,
    eDBType_ODBC_PostgreSQL := 5,
    eDBType_ODBC_Oracle     := 6,
    eDBType_ODBC_DB2        := 7,
    eDBType_ODBC_InterBase  := 8,
    eDBType_ODBC_Firebird   := 9,
    eDBType_XML             := 10,
    eDBType_OCI_Oracle      := 11
);
END_TYPE
```

Voraussetzungen

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

7.3.8 E_DBValueType

```
TYPE E_DBValueType :
(
    eDBValue_Double     := 0,
    eDBValue_Bytes      := 1
);
END_TYPE
```

Voraussetzungen

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

7.3.9 E_DBWriteModes

```
TYPE E_DBWriteModes :
(
    eDBWriteMode_Update             := 0,
    eDBWriteMode_Append             := 1,
    eDBWriteMode_RingBuffer_Time    := 2,
    eDBWriteMode_RingBuffer_Count   := 3
);
END_TYPE
```

Voraussetzungen

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

7.3.10 E_DBTypes

```
TYPE E_DBParameterTypes :
(
    eDBParameter_Input          := 0,
    eDBParameter_Output         := 1,
    eDBParameter_InputOutput    := 2,
    eDBParameter_ReturnValue    := 3,
    eDBParameter_OracleCursor   := 4
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0	PC oder CX (x86)	TcDatabase.Lib (Ab TcDatabaseSrv Version 1.0.13)
TwinCAT v2.10.0	CX (ARM)	

7.4 Konstanten

7.4.1 Globale Variablen

```
VAR_GLOBAL CONSTANT
    AMSPORT_DATABASESRV                         : UINT   := 21372;

    DBADS_IGR_RELOADXML                         : UDINT  :=16#100;

    DBADS_IGR_GETSTATE                          : UDINT  :=16#200;

    DBADS_IGR_DBCONNOPEN                        : UDINT  :=16#300;
    DBADS_IGR_DBCONNCLOSE                       : UDINT  :=16#301;
    DBADS_IGR_ADSDEVCONNOPEN                    : UDINT  :=16#302;
    DBADS_IGR_ADSDEVCONNCLOSE                   : UDINT  :=16#303;
```

```
    DBADS_IGR_DBSTOREDPROCEDURES                    : UDINT  :=16#400;
    DBADS_IGR_DBSTOREDPROCEDURES_RETURNRECORD       : UDINT  :=16#401;
    DBADS_IGR_DBSTOREDPROCEDURES_RETURNRECORDARRAY  : UDINT  :=16#402;

    DBADS_IGR_START                                 : UDINT  :=16#10000;
    DBADS_IGR_STOP                                  : UDINT  :=16#20000;

    DBADS_IGR_DBCONNADD                             : UDINT  :=16#30000;
    DBADS_IGR_ADSDEVCONNADD                         : UDINT  :=16#30001;
    DBADS_IGR_ODBC_DBCONNADD                        : UDINT  :=16#30010;

    DBADS_IGR_GETDBXMLCONFIG                        : UDINT  :=16#30101;
    DBADS_IGR_GETADSDEVXMLCONFIG                    : UDINT  :=16#30102;

    DBADS_IGR_DBWRITE                               : UDINT  :=16#40000;
    DBADS_IGR_DBREAD                                : UDINT  :=16#50000;

    DBADS_IGR_DBTABLECREATE                         : UDINT  :=16#60000;
    DBADS_IGR_DBCREATE                              : UDINT  :=16#70000;

    DBADS_IGR_DBRECORDSELECT                        : UDINT  :=16#80001;
    DBADS_IGR_DBRECORDINSERT                        : UDINT  :=16#80002;
    DBADS_IGR_DBRECORDDELETE                        : UDINT  :=16#80003;

    DBADS_IGR_DBAUTHENTIFICATIONADD                 : UDINT  :=16#90000;

    MAX_DB_TABLE_COLUMNS                            : UDINT  := 255;
    MAX_XML_DECLARATIONS                            : UDINT  := 255;
    MAX_STORED_PROCEDURES_PARAMETERS                : UDINT  := 255;

END_VAR
```

Voraussetzungen

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

8 Beispiele

8.1 Quickstart

Das folgende PDF-Dokument enthält das Handout eines Workshops über den TwinCAT Database Server.

Dieses Handout beschreibt detailliert den Umgang mit dem XML-Konfigurationsdatei Editor und zeigt die Funktionsweise einiger Funktionsbausteine in der SPS.

Download als PDF: "Workshop Handout" (Englische Version) https://infosys.beckhoff.com/content/1031/tcdbserver/Resources/11407900555.pdf

Technischer Workshop

Topic: TwinCAT Database Server

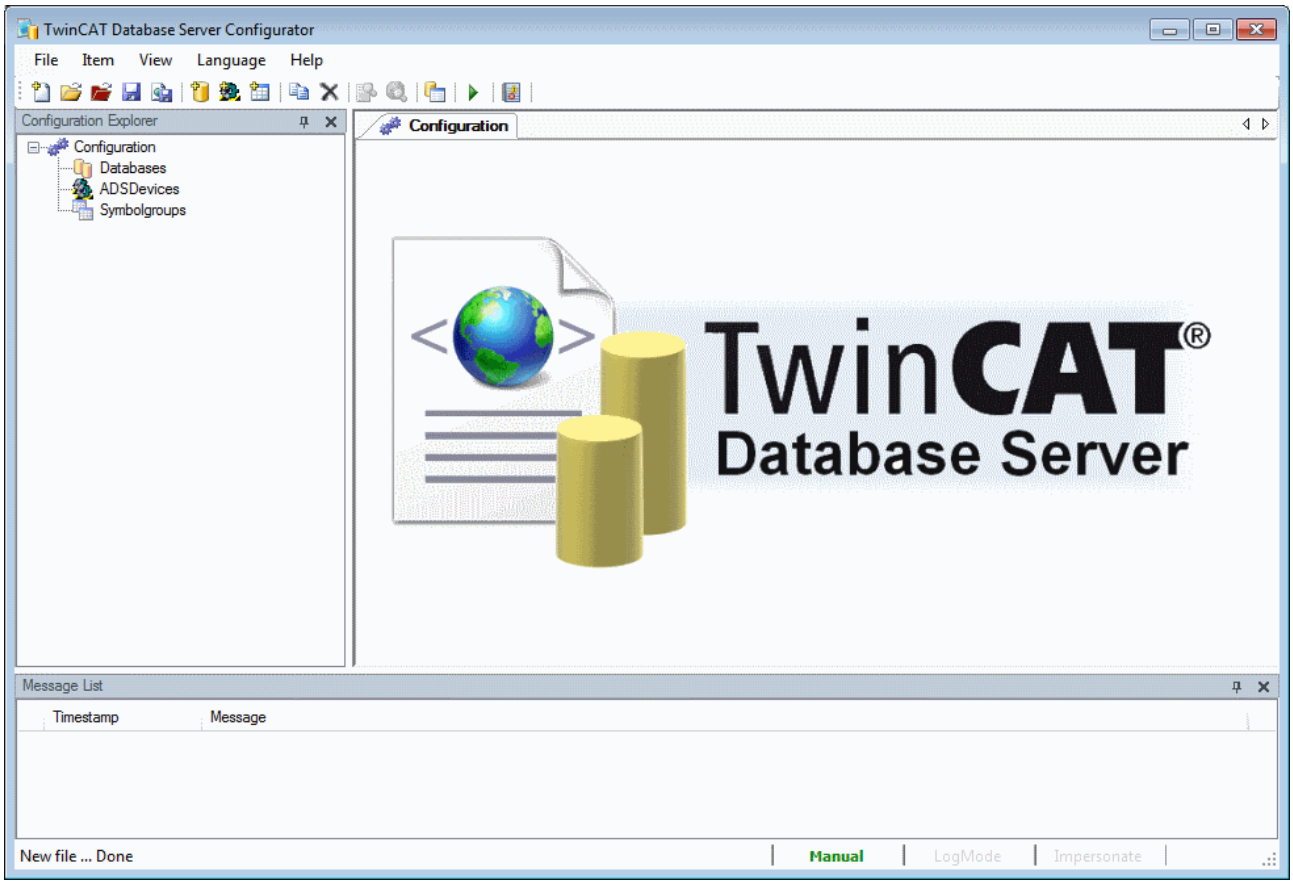
Eine Schritt-für-Schritt-Einführung in den TwinCAT Database Server.

1. Vorwort

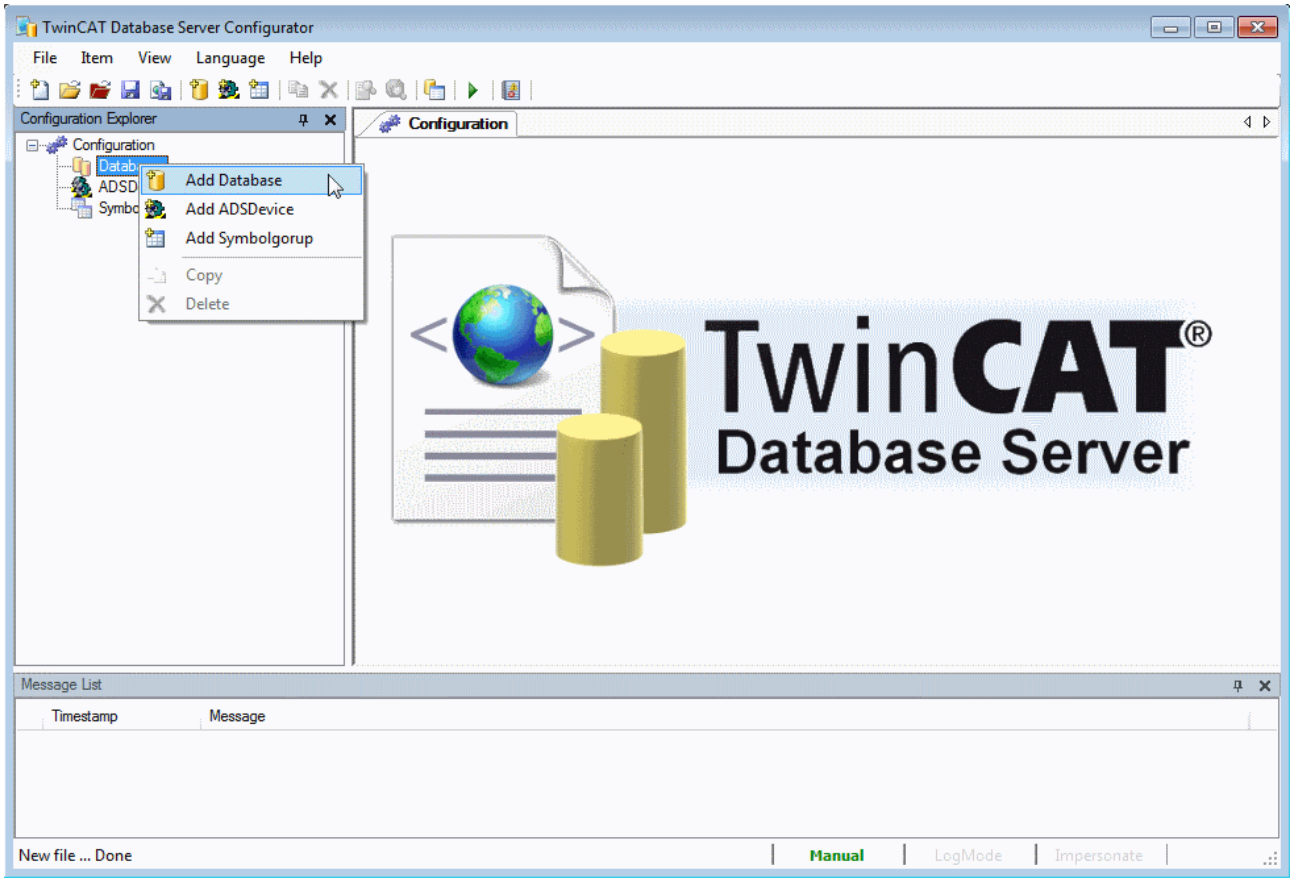
Es gibt zwei Möglichkeiten den TwinCAT Database Server zu konfigurieren - Einerseits aus der SPS heraus und andererseits mit dem TwinCAT Database Server XML-Konfigurationsdatei Editor. Dies ist ein kleines Beispiel für die Konfiguration mit dem XML Editor, aber es gibt auch eine Anleitung für den zweiten Weg zum selbst testen.

2. DataBase Server Konfiguration mit XML Editor

Öffnen Sie den Editor unter Start -> Alle Programme -> TwinCAT System -> TwinCAT Database Server -> XML Configuration File Editor

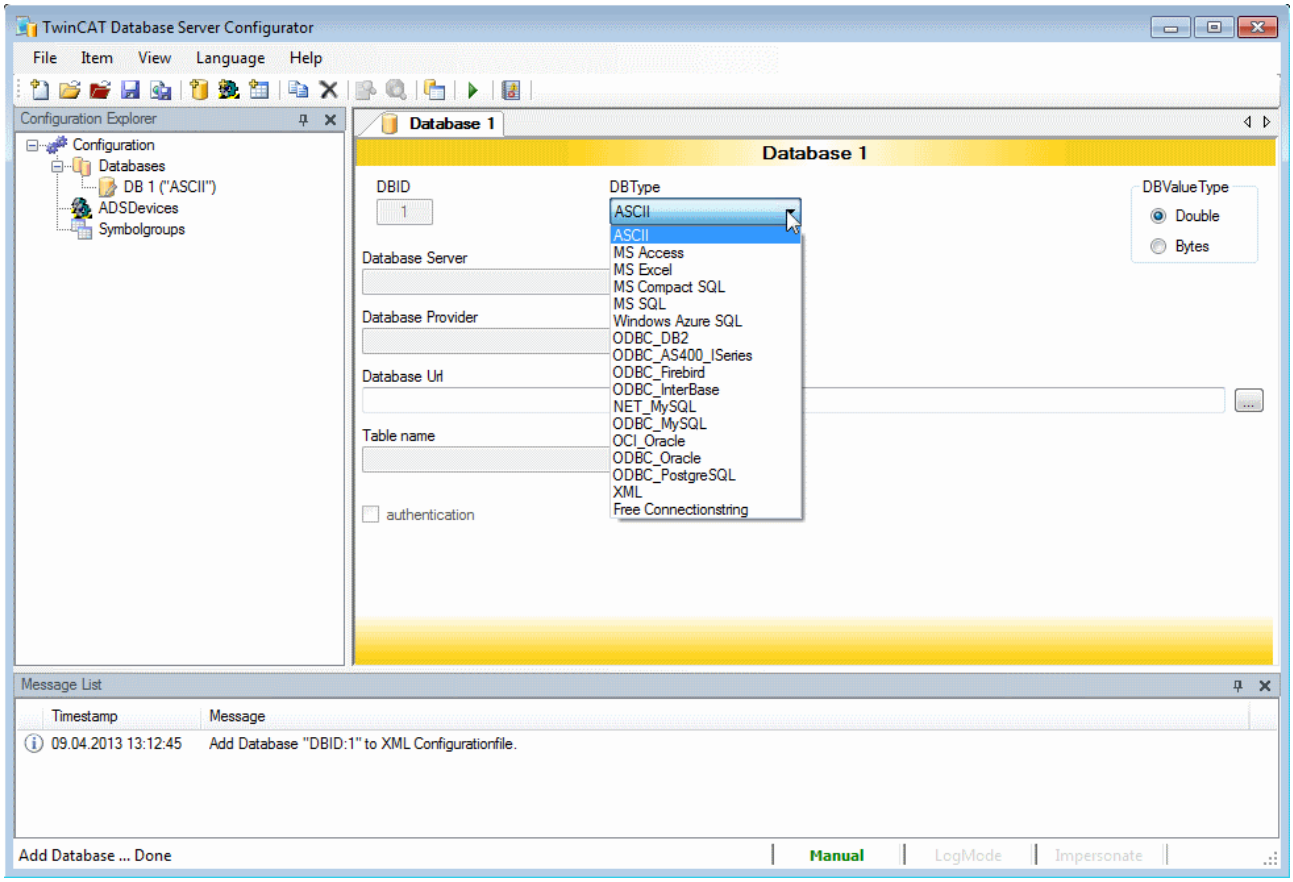


Zunächst müssen Sie eine neue Datenbank anfügen. Rechts Klick auf Database wie es im Bild gezeigt wird, oder über die Symbole in der Symbolleiste.

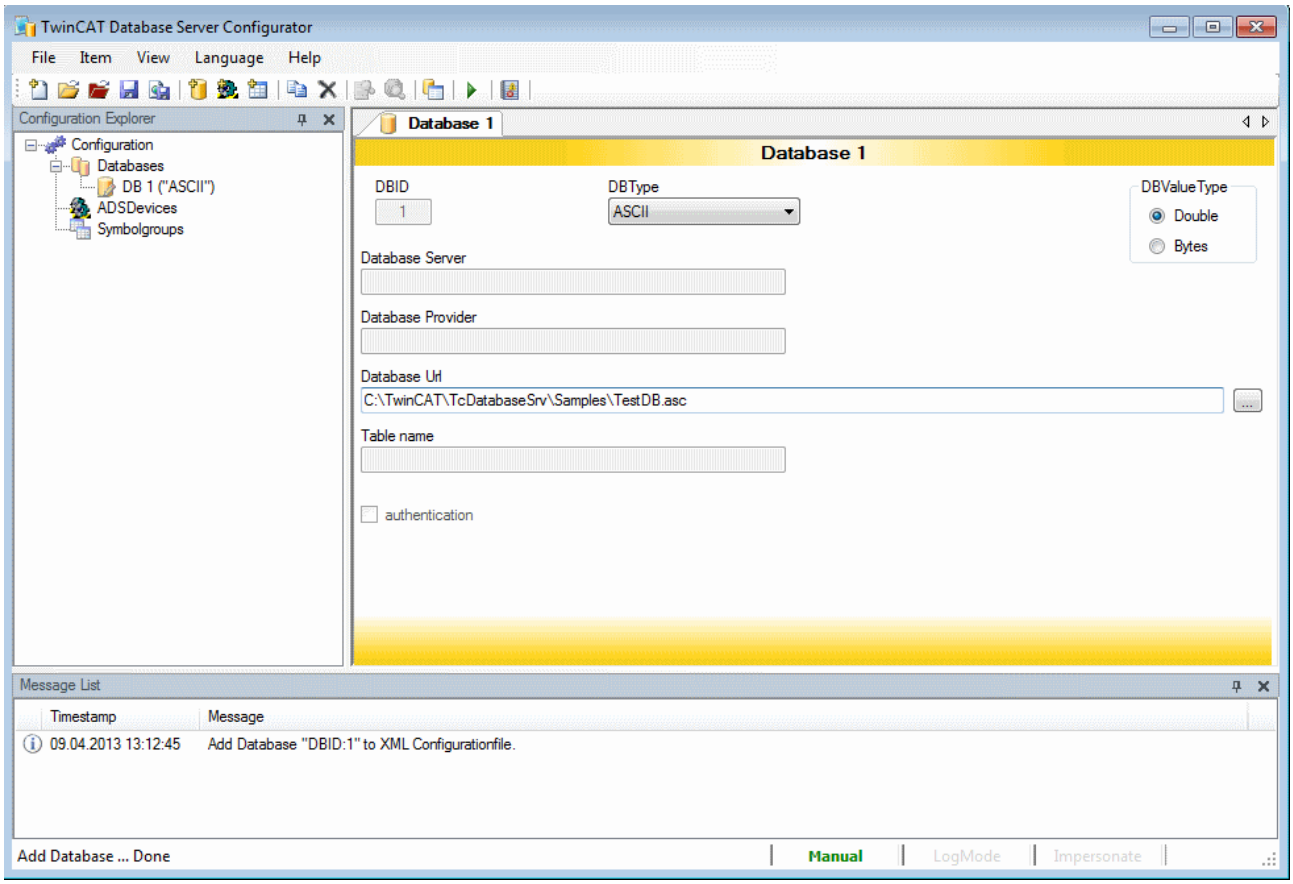


Im nächsten Schritt müssen Sie ihre Datenbank konfigurieren. Nicht jeder hat die großen Datenbanken auf seinem System, darum nehmen wir die ASCII Datenbank.

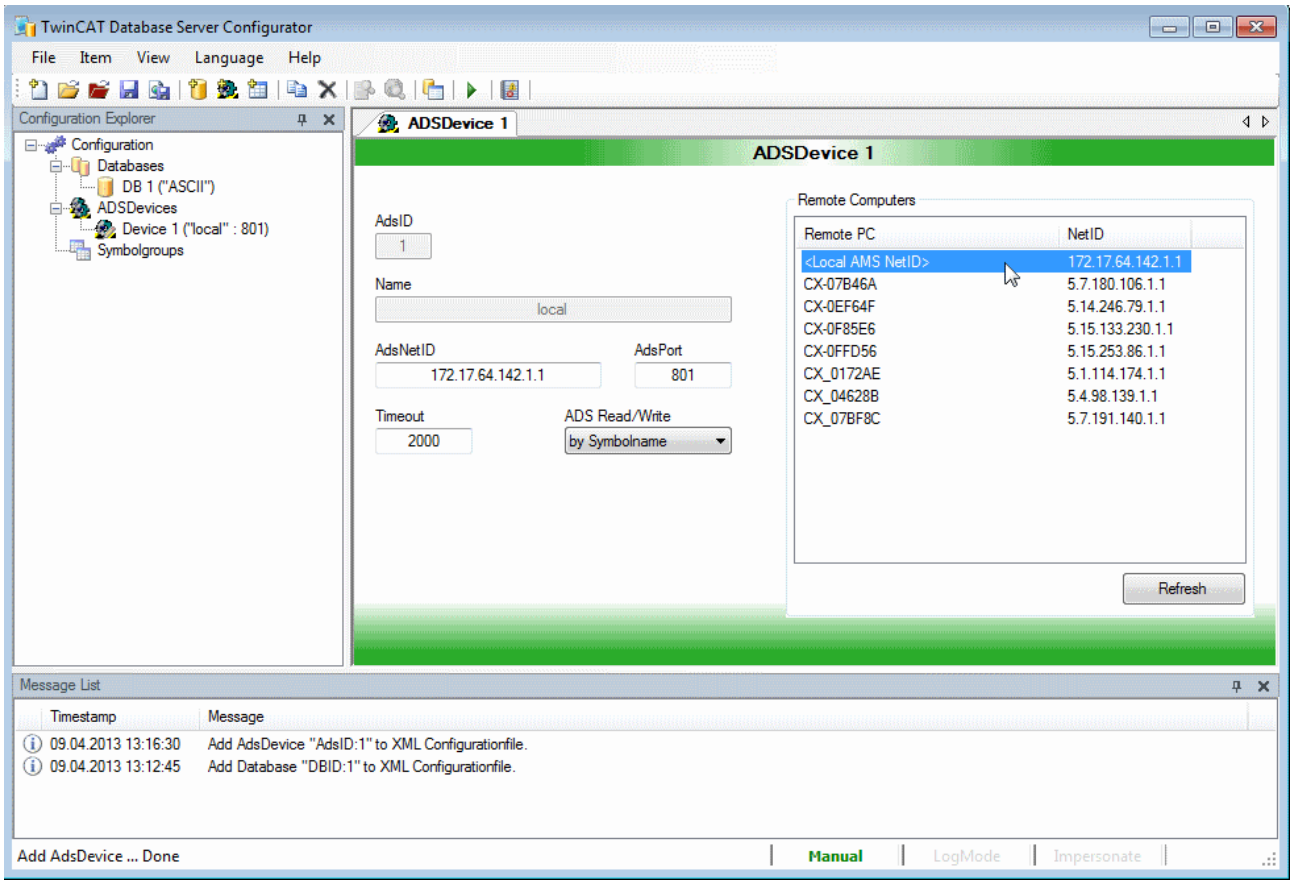
Sie finden die Deklarationen aller unterstützten Datenbanktypen im Information System [▶ 96] Wählen Sie den DBType ASCII aus. Wenn Sie nur alphanumerische Datentypen und Boolean loggen wählen Sie "Double" al DBValueType. Andernfalls wählen Sie "Bytes" aus, wenn Sie auch Strukturen und String loggen möchten.



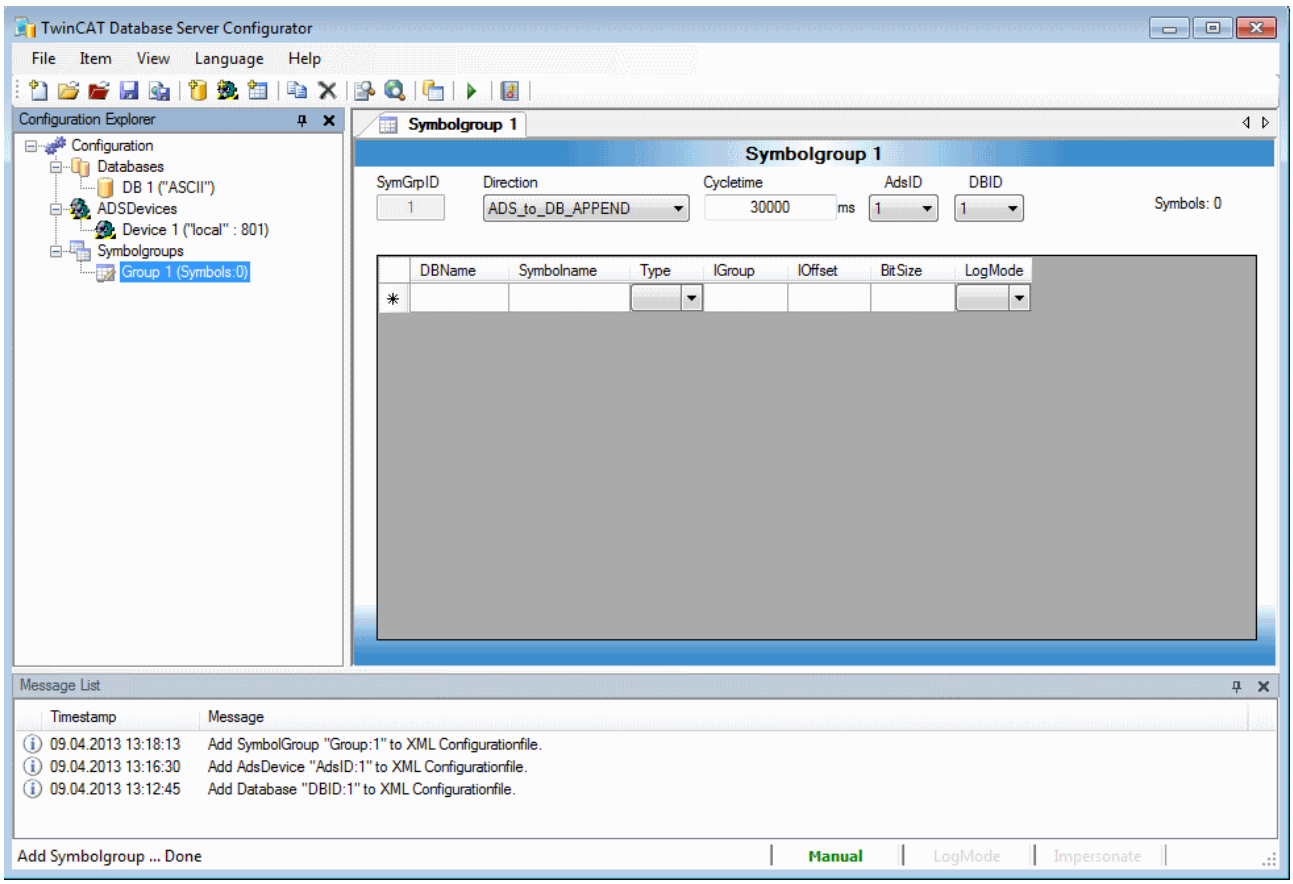
Für eine ASCII Datenbank müssen Sie nur den Datenbank Pfad angeben, so wie es im nächsten Bild gezeigt wird. Später werden Sie noch die Datenbank ID (DBID) benötigen wie auch die AdsID, welche Sie auf der nächsten Seite finden.



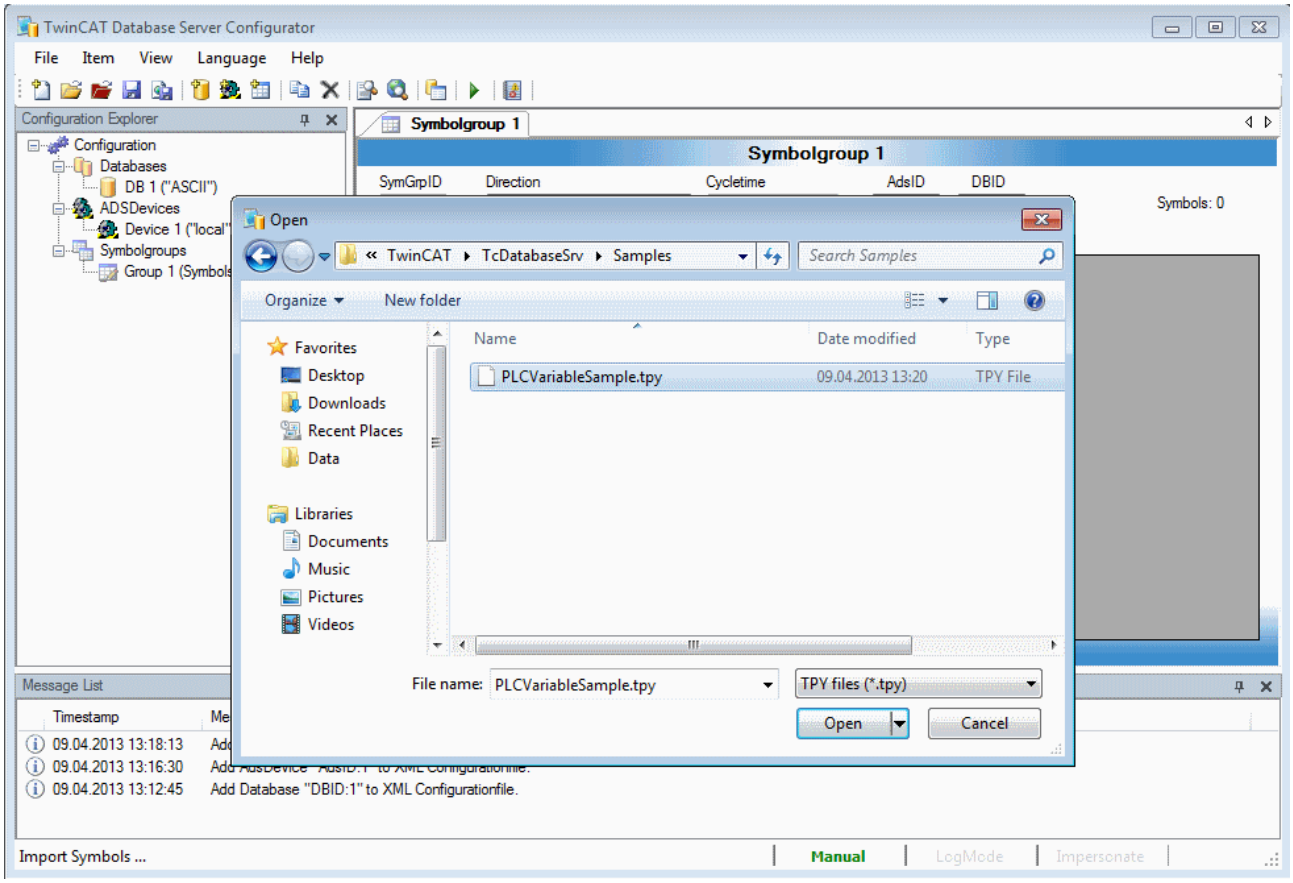
Fügen Sie ein AdsDevice an und geben Sie Ihre AdsNetID, wie auch Ihren AdsPort an. Wenn Sie ihr lokales System verwenden, müssen Sie keine NetID angeben.

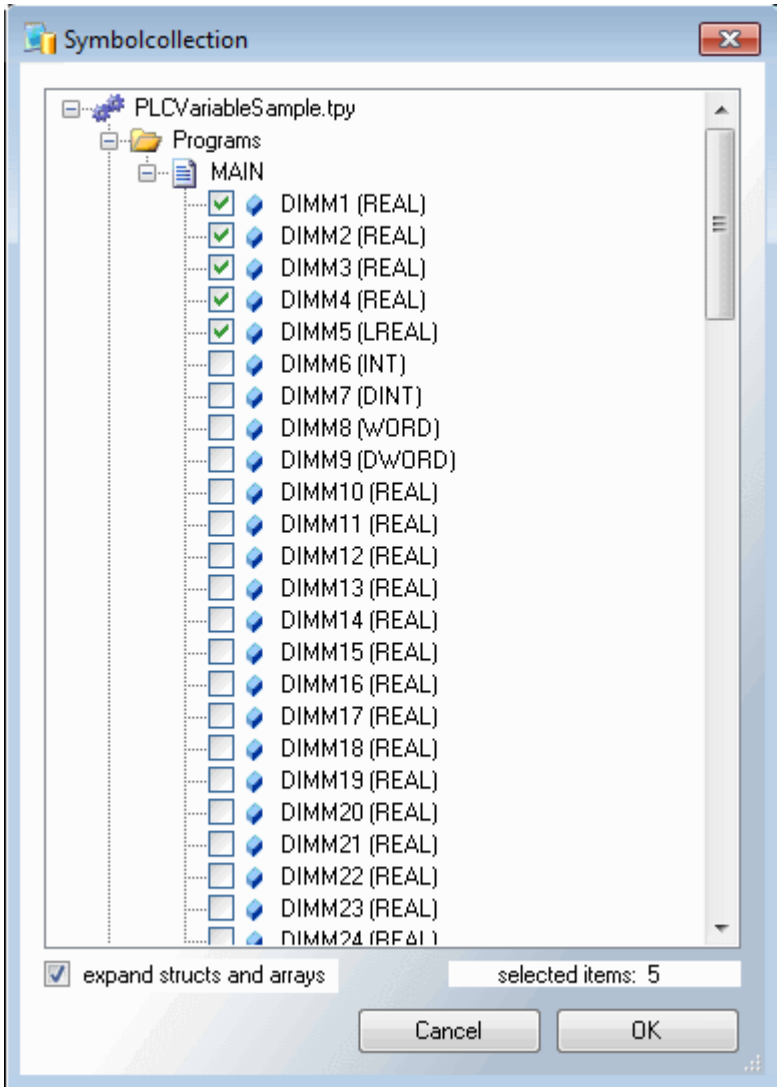


Wenn Sie eine Konfiguration für ein Remote System erstellen möchten, haben Sie die Möglichkeit ihr gewünschtes System aus der Tabelle, rechts an der Seite, auszuwählen. Nun können Sie eine Symbolgruppe anfügen mit den Variablen aus ihrem SPS Projekt.

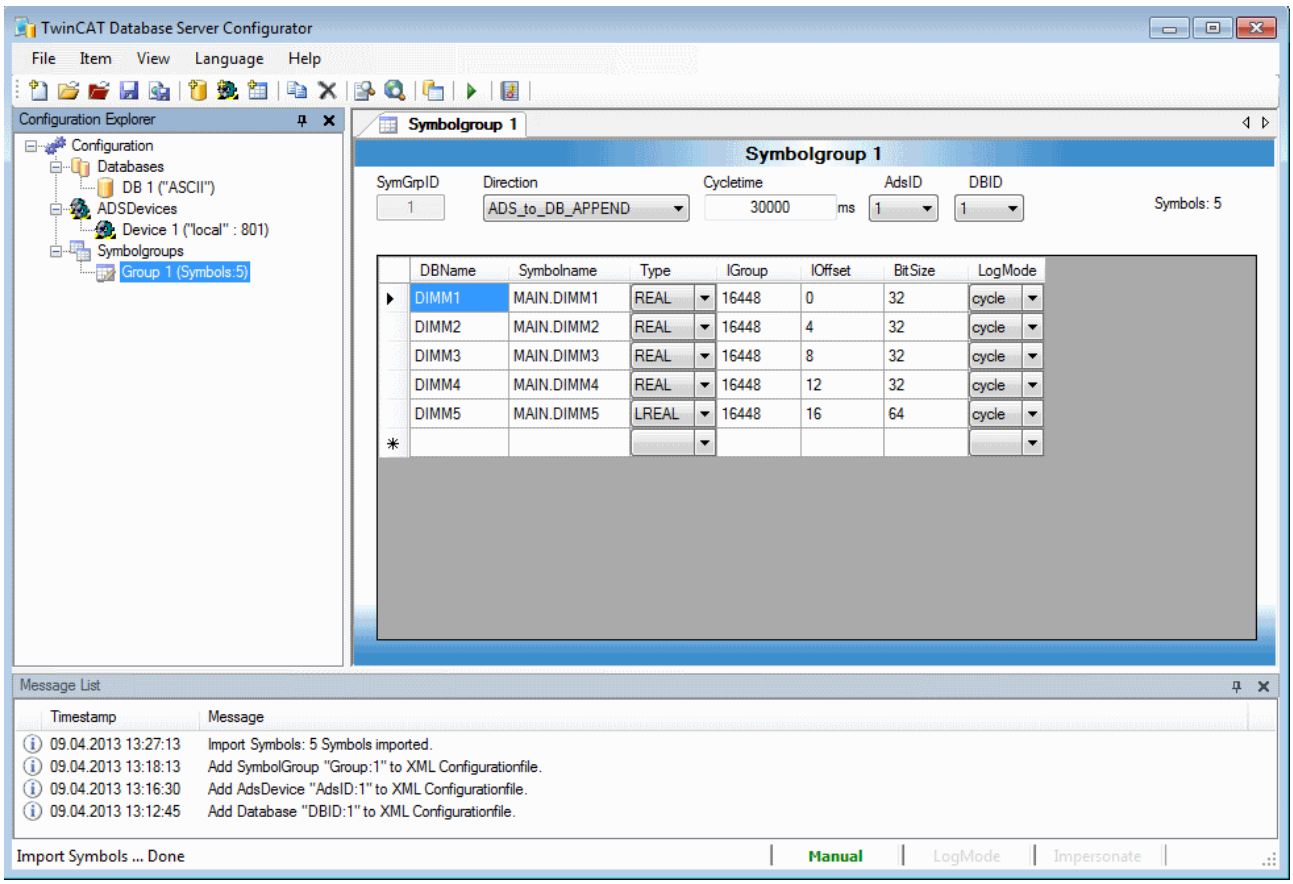


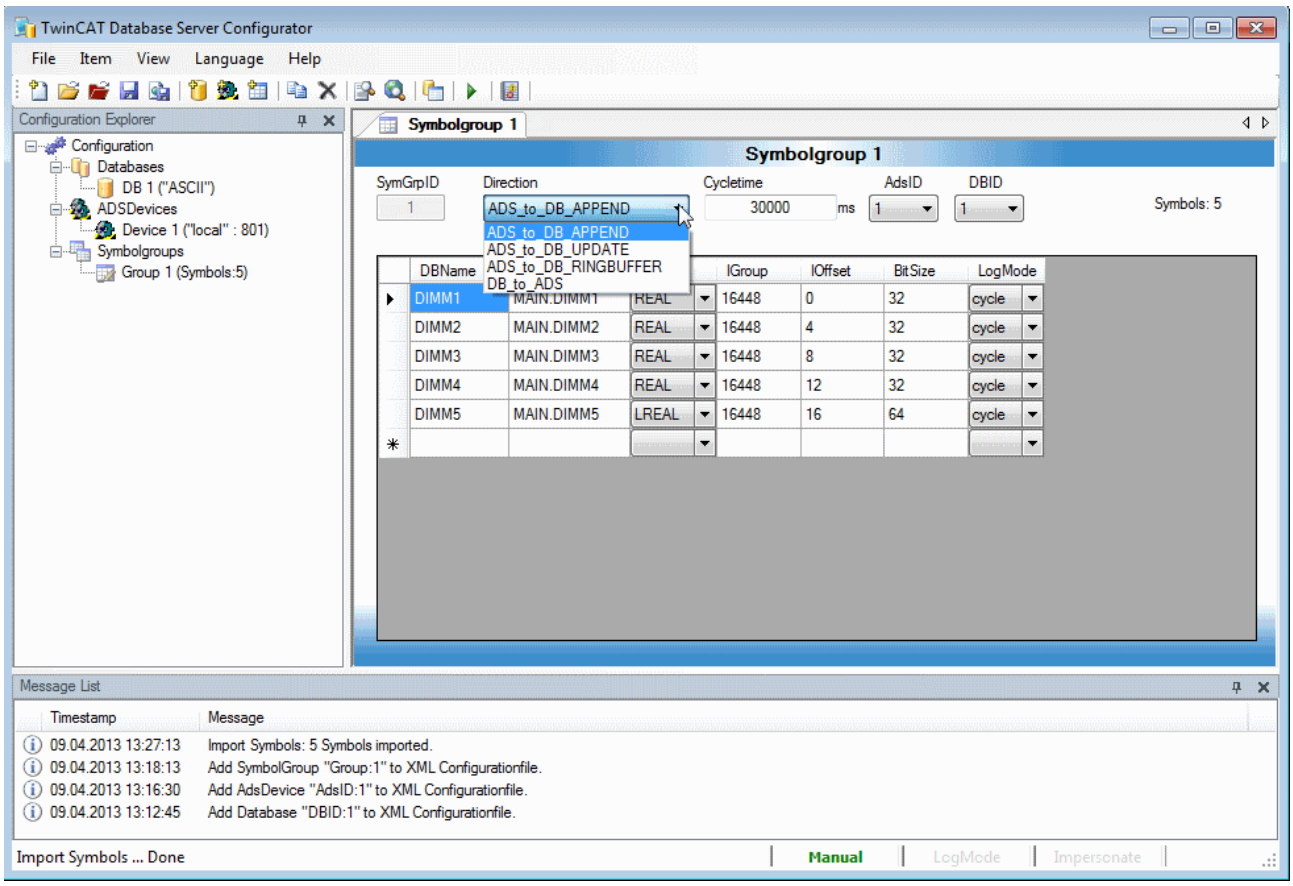
Bevor Sie ihre Symbolgruppe konfigurieren können, müssen Sie ihr SPS Projekt "Übersetzen". Öffnen Sie TwinCAT PLC Control mit der Datei PLCVariableSample.pro und dann Projekt -> Übersetzen. Dies ist wichtig für die Erstellung der TPY-Datei. Jetzt können Sie nach Variablen suchen im Database Editor über den "Import Symbols" Button.





Für das Beispiel wählen Sie die ersten fünf Variablen aus de Symbolsammlung. Sie können den LogMode für jede Variable ändern, so wie es im nächsten Bild gezeigt wird. Sie müssen nun die AdsID und die DBID setzen, die Sie von den vorherigen Seiten kennen.





Sie haben die Möglichkeit die Kommunikationsrichtung auszuwählen. In diesem Fall ist ADS_to_DB_APPEND ausgewählt.

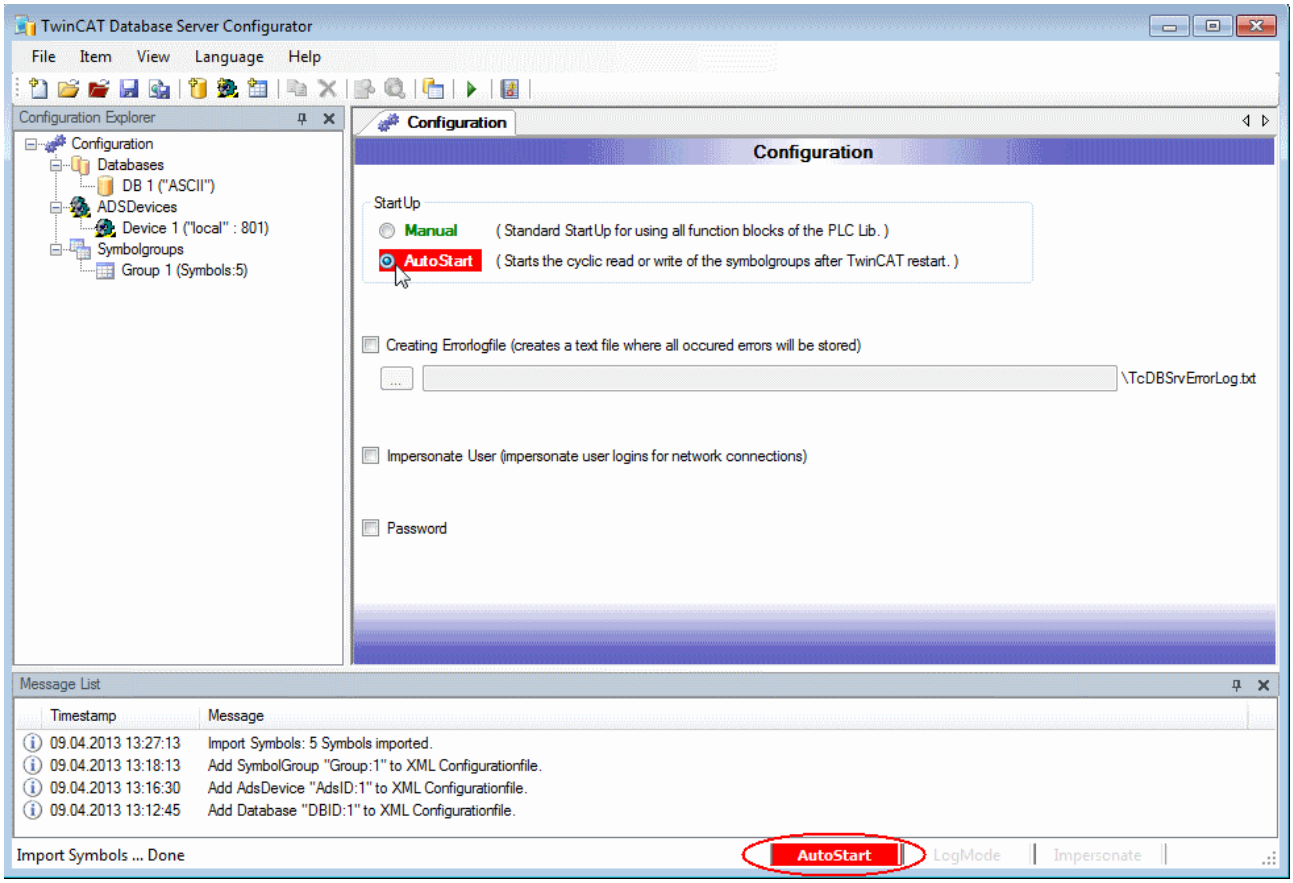
ADS_to_DB_APPEND:
Diese Option hängt die neuen Dateneinträge an die Datenbank an.

ADS_to_DB_UPDATE:
Diese Option Aktualisiert die vorhandenen Einträge in der Datenbank.

ADS_to_DB_RINGBUFFER:
Mit diesem Schreibmodus kann die Anzahl oder das Alter der Datensätze in der Datenbanktabelle limitiert werden.

DB_to_ADS:
Diese Kommunikationsrichtung beschreibt das lesen von Werten aus der Datenbank in die SPS.

Wenn der Database Server ohne der SPS genutzt werden soll (nur loggen von SPS Variablen), muss die Option AutoStart gesetzt sein. So startet der Database Server direkt mit dem Erzeugen der Verbindungen zu den deklarierten Datenbanken und ADS-Geräten nach einem TwinCAT start. Hierfür müssen sie ein Bootprojekt vom Programm "PLCVariableSample.pro" erzeugt haben. Schließlich müssen Sie ihre erzeugte Konfiguration im Verzeichnis "C:\TwinCAT\Boot" speichern.



4. Konfigurationstest
Gehen Sie ins PLC Control und starten Sie die SPS, danach erzeugen Sie ein Bootprojekt und machen eine TwinCAT Restart. Schauen Sie nun auf die Testdatenbank "TestDB" unter "C:\TwinCAT\TcDatabaseSrv\Samples".

Dies war ein kleine Einführung in den Beckhoff TwinCAT Database Server. Für Fortgeschrittene folgt eine kleine Einführung für eine Konfiguration des Database Servers aus der SPS heraus.

5. Database Server Konfiguration aus der SPS

Aufgabe:
Erzeugen einer Microsoft Compact SQL Datenbank aus der SPS, hinzufügen einer Datenbankverbindung und erstellen einer neuen Tabelle in der Datenbank. Schließlich benutzen des Schreibfunktionsbausteins mit dem Ringpuffer Modus "RingBuffer_Count" um eine Variable 100 mal in die Tabelle zu schreiben.

Als erstes fügen Sie einen R_TRIG Funktionsbaustein hinzu um ihr Programm mit einer steigenden Flanke zu starten. Danach erzeugen Sie eine "State-Machine" zum ansprechen ihrer Datenbank Funktionsblöcke, wie sie es im folgenden Bild sehen.

```
0001 PROGRAM MAIN
0002 VAR
0003     fbDBCreate          : FB_DBCreate;
0004     fbDBConAdd          : FB_DBConnectionAdd;
0005     fbDBTableCreate     : FB_DBTableCreate;
0006     fbDBWrite           : FB_DBWrite;
0007
0008     state               : INT := 0;
0009     R_Edge              : R_TRIG;
0010     bExecute            : BOOL;
0011 END_VAR
```

```
0001 (* Take care for the test with FB_DBCreate that the database do not exist -> for a second run delete the DB *)
0002 R_Edge (CLK := bExecute);
0003 IF R_Edge.Q THEN
0004     state := 1;
0005 END_IF
0006
0007 CASE state OF
0008     0:   (* idle state *)
0009     ;
0010     1:   (* Create a Database *)
0011     ;
0012     11:  (* Is the FB_DBCreate busy? *)
0013     ;
0014     2:   (* Add a connection to your Database *)
0015     ;
0016     21:  (* Is the FB_DBConnectionAdd busy? *)
0017     ;
0018     3:   (* Create a table for your Database *)
0019     ;
0020     31:  (* Is the FB_DBTableCreate busy? *)
0021     ;
0022     4:   (* Write cyclic values into your Database *)
0023     ;
0024     41:  (* Is the FB_DBWrite busy? *)
0025     ;
0026 END_CASE
```

Jetzt deklarieren Sie alle anderen Variablen die Sie benötigen. Speziell die Struktur der Tabelle. Sie finden eine Beschreibung, wie Sie dies für Microsoft Compact SQL Datenbank tun müssen, im Information System [▶ 41]

Eine AutoID wird in der Spalte "ID" generiert. Der Wert in dieser Spalte wird immer um 1 erhöht. die "Timestamp" Spalte speichert den Zeitpunkt an dem der Datensatz gespeichert wurde. Und der Name der Variable wird in der dritten Spalte gespeichert. Unter "Value" finden sie die Werte der Variablen.

Sie können die bekannten Variablen nutzen vom nächsten Bild.

```
0001 PROGRAM MAIN
0002 VAR
0003     fbDBCreate          : FB_DBCreate;
0004     fbDBConAdd          : FB_DBConnectionAdd;
0005     fbDBTableCreate     : FB_DBTableCreate;
0006     fbDBWrite           : FB_DBWrite;
0007
0008     state               : INT := 0;
0009     R_Edge              : R_TRIG;
0010     bExecute            : BOOL;
0011
0012     bError              : BOOL;
0013     uErrID              : UDINT;
0014
0015     uDbID               : UDINT;            (* Database ID *)
0016     uAdsID              : UDINT := 1;       (* Set your ADS ID *)
0017
0018     (* Table structure: *)
0019     tablestrc: ARRAY [0..3] OF ST_DBColumnCfg :=
0020         (sColumnName := 'ID', sColumnProperty := 'IDENTITY(1,1)', eColumnType := EDBCOLUMN_BIGINT),
0021         (sColumnName := 'Timestamp', eColumnType := EDBCOLUMN_DATETIME),
0022         (sColumnName := 'Name', eColumnType := EDBCOLUMN_NTEXT),
0023         (sColumnName := 'Value', eColumnType := eDBColumn_Integer);
0024
0025     iLogVariable        : INT;              (* Log variable *)
0026 END_VAR
```

Die nächsten Schritte.

FB_DBCreate:

```
0007 CASE state OF
0008     0:    (* idle state *)
0009     ;
0010
0011     1:    (* Create a Database *)
0012     fbDBCreate(  sNetID       := ,
0013                  sPathName    := 'C:\TwinCAT\TcDatabaseSrv\Samples',
0014                  sDBName      := 'DB_ITW',
0015                  eDBType      := eDBType_Mobile_Server,
0016                  sSystemDB    := ,
0017                  sUserId      := ,
0018                  sPassword    := ,
0019                  bExecute     := TRUE,
0020                  tTimeout     := T#20s,
0021                  bBusy        => ,
0022                  bError       => bError,
0023                  nErrID       => uErrID);
0024     state := 11;
0025
0026     11:
0027     fbDBCreate(bExecute := FALSE);
0028     IF NOT fbDBCreate.bBusy AND NOT fbDBCreate.bError THEN
0029         state := 2;
0030     END_IF
```

Im Status Nummer 1 müssen Sie den FB_DBCreate einfügen. Der Pfad der Datenbank muss dem Baustein übergeben werden. Aber Sie müssen sicher sein, das die Datenbank bis zu diesem Zeitpunkt noch nicht existiert. Anderenfalls wird eine Fehlermeldung ausgegeben. Im Status 11 müssen sie warten bis der Ausgang "bBusy" nicht mehr TRUE ist.

FB_DBConnectionAdd:

```
0032    2:(*Add a connection to your Database*)
0033    fbDBConAdd(sNetID           := ,
0034               eDBType          := eDBType_Mobile_Server,
0035               eDBValueType     := eDBValue_Double,
0036               sDBServer        := ,
0037               sDBProvider      := ,
0038               sDBUrl           := 'C:\TwinCAT\TcDatabaseSrv\Samples\DB_ITW.sdf',
0039               sDBSystemDB      := ,
0040               sDBUserId        := ,
0041               sDBPassword      := ,
0042               sDBTable         := 'ITW_Table',
0043               bExecute         := TRUE,
0044               tTimeout         := T#20s,
0045               bBusy            => ,
0046               bError           => bError,
0047               nErrID           => uErrID,
0048               hDBID            => );
0049    state := 21;
0050
0051    21:
0052    fbDBConAdd(bExecute:=FALSE, hDBID=> uDbID);
0053    IF NOT fbDBConAdd.bBusy AND NOT fbDBConAdd.bError THEN
0054        state := 3;
0055    END_IF
```

In diesem Funktionsblock müssen Sie den DBType und den DBValueType (Bytes oder Double) angeben. Außerdem müssen Sie ihrer Tabelle einen Namen geben. Die anderen Einstellungen für Compact SQL Datenbanken finden Sie hier [▶ 36].

FB_DBTableCreate:

```
0055    3:    (* Create a table for your Database *)
0056    fbDBTableCreate( sNetID         := ,
0057                     hDBID          := uDbID,
0058                     sTableName     := 'ITW_Table',
0059                     cbTableCfg     := SIZEOF(tablestrc),
0060                     pTableCfg      := ADR(tablestrc),
0061                     bExecute       := TRUE,
0062                     tTimeout       := t#20s,
0063                     bBusy          => ,
0064                     bError         => bError,
0065                     nErrID         => uErrID,
0066                     sSQLState      => );
0067    state := 31;
0068
0069    31:
0070    fbDBTableCreate(bExecute := FALSE);
0071    IF NOT fbDBTableCreate.bBusy AND NOT fbDBTableCreate.bError THEN
0072        state := 4;
0073    END_IF
```

Einer der Eingänge des Funktionsbausteins ist der "hDBID". Diesem Eingang übergeben Sie den Wert, den Sie vom FB_DBConnectionAdd bekommen haben. Des Weiteren übergeben Sie die Größe und die Adresse der Tabellenstruktur.

FB_DBWrite:

```
0075        4:    (* Write cyclic values into your Database *)
0076        fbDBWrite(    bExecute            := FALSE);
0077        fbDBWrite(    sNetID              := ,
0078                      hDBID               := uDbID,
0079                      hAdsID              := uAdsID,
0080                      sVarName            := 'MAIN.iLogVariable',
0081                      sDBVarName          := 'SPEED [km/h]',
0082                      eDBWriteMode        := eDBWriteMode_RingBuffer_Count,
0083                      tRingBufferTime     := ,
0084                      nRingBufferCount    := 100,
0085                      bExecute            := TRUE,
0086                      tTimeout            := t#20s,
0087                      bBusy               => ,
0088                      bError              => bError,
0089                      nErrID              => uErrID,
0090                      sSQLState           => );
0091        state := 41;
0092
0093        41:
0094        fbDBWrite(bExecute := FALSE);
0095        IF NOT fbDBWrite.bBusy AND NOT fbDBWrite.bError THEN
0096            state := 4;
0097        END_IF
0098
0099    END_CASE
0100
0101    iLogVariable := iLogVariable + 1;
0102    IF iLogVariable > 2000 THEN
0103        iLogVariable := 0;
0104    END_IF
```

Mit dem FB_DBWrite können Sie den aktuellen Wert der Variable "iLogVariable" jeder Zeit in die Datenbank schreiben. Allerdings werden dort nur 100 Werte der Variable gespeichert, wenn Sie den nRingBufferCount auf 100 gesetzt haben.

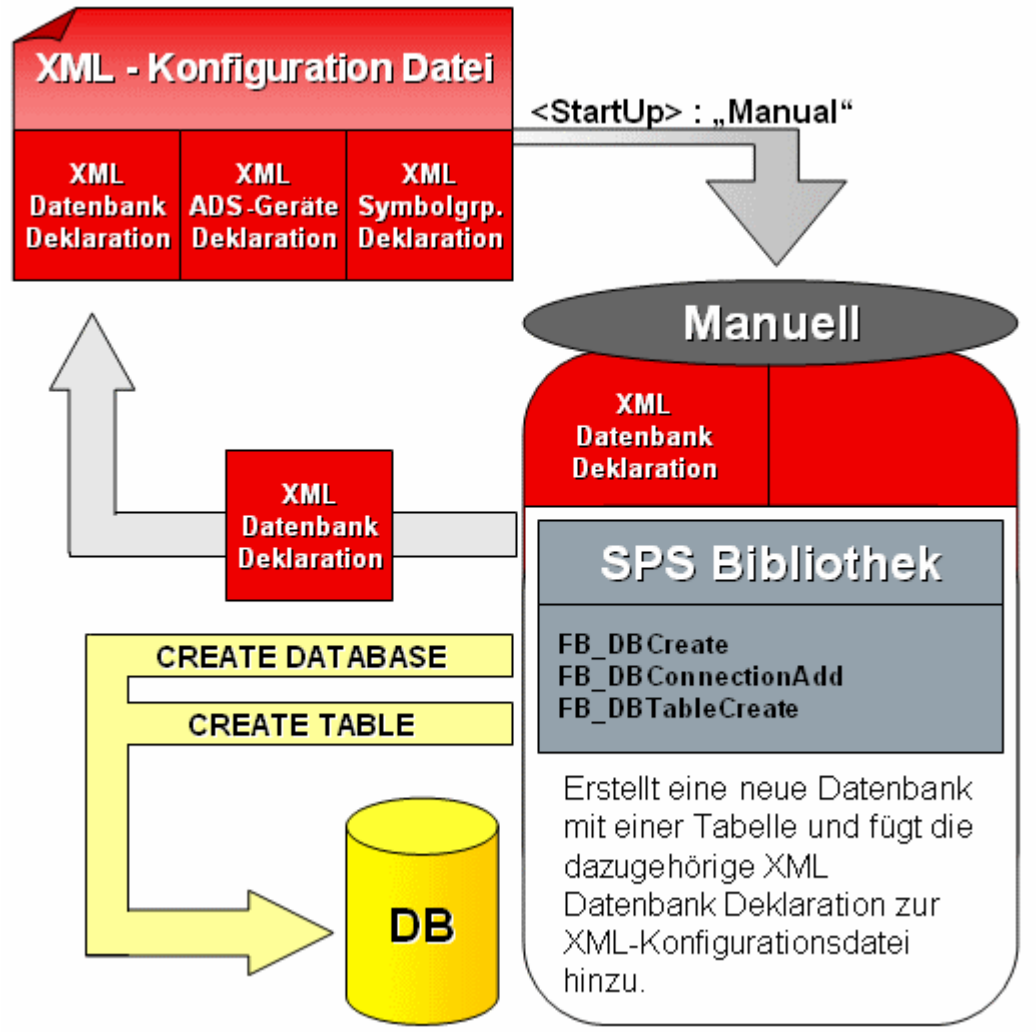
Am ende des Programmcodes können Sie sehen wie die Variable immer um 1 erhöht wird bei jedem SPS Zyklus.

Wichtig: Wenn Sie den Beispielcode testen wollen, stellen Sie sicher, dass die Datenbank noch nicht existiert!

8.2 Erstellen einer MS Access Datenbank

Download "Beispiel Erstellen einer Datenbank" https://infosys.beckhoff.com/content/1031/tcdbserver/Resources/11407901963.zip

In diesem Beispiel wird das Erstellen einer Datenbank aus der SPS heraus gezeigt.
Zusätzlich wird eine Tabelle hinzugefügt und die erzeugte Datenbank in der XML-Konfigurationsdatei deklariert.



Verwendeter Datenbanktyp	MS Access
Kompatible Datenbanktypen	MS SQL, MS Compact SQL, MS Access, XML
Verwendete Funktionsbausteine	FB_DBCreate, FB_DBConnectionAdd, FB_DBTableCreate
Einzubindende Bibliotheken	"**TcDatabase.lib**", "TcSystem.lib", "TcBase.lib", "STANDARD.lib"
Download Dateiliste	FB_DBCreate_Sample.pro

Der erzeugten Datenbank wird eine Tabelle mit dem Namen "myTable" hinzugefügt die folgende Tabellenstruktur besitzt:

Spaltenname	Datentyp	Eigenschaft
ID	bigint	IDENTITY(1,1)
Timestamp	datetime	
Name	ntext	
Value	float	

Diese Tabellenstruktur wird mit folgendem Array erzeugt:

```
 tablestrc: ARRAY [0..3] OF ST_DBColumnCfg :=
                    (sColumnName:='ID',sColumnProperty:='IDENTITY(1,1)',eColumnType:=EDBCOLUMN_BIGIN
T),
                    (sColumnName:='Timestamp',eColumnType:=EDBCOLUMN_DATETIME),
                    (sColumnName:='Name',eColumnType:=EDBCOLUMN_NTEXT),
                    (sColumnName:='Value',eColumnType:=EDBCOLUMN_FLOAT);
```

Variablendeklaration

```
PROGRAM MAIN
VAR
    R_TRIG1: R_TRIG;
    bSTART: BOOL;

    FB_FileDelete1: FB_FileDelete;
    FB_DBCreate1: FB_DBCreate;
    FB_DBConnectionAdd1: FB_DBConnectionAdd;
    FB_DBTableCreate1: FB_DBTableCreate;

    bBusy_Delete: BOOL;
    bBusy_CreateDB: BOOL;
    bBusy_ConnAdd: BOOL;
    bBusy_CreateTable: BOOL;

    bErr: BOOL;
    nErrid: UDINT;

    nDBid: UDINT;

    arrTablestrc: ARRAY [0..3] OF ST_DBColumnCfg :=
                    (sColumnName:='ID',sColumnProperty:='IDENTITY(1,1)',eColumnType:=EDBCOLUMN_BIGIN
T),
                    (sColumnName:='Timestamp',eColumnType:=EDBCOLUMN_DATETIME),
                    (sColumnName:='Name',eColumnType:=EDBCOLUMN_NTEXT),
                    (sColumnName:='Value',eColumnType:=EDBCOLUMN_FLOAT);

    nState:BYTE := 0;

END_VAR
```

SPS Programm

```
CASE nState OF
    0:
        (*To start this sample you have to set a rising edge to the variable bSTART*)
        R_TRIG1(CLK:=bSTART);
        IF R_TRIG1.Q THEN
            nState := 1;
            FB_FileDelete1(bExecute:=FALSE);
            FB_DBCreate1(bExecute:=FALSE);
            FB_DBConnectionAdd1(bExecute:=FALSE);
            FB_DBTableCreate1(bExecute:=FALSE);
            bSTART := FALSE;
        END_IF
    1:
        (*It isn't possible to overwrite an existing database file. If the database file exist the
FB_FileDelete block will delete the file*)
        FB_FileDelete1(
            sNetId:= ,
            sPathName:= 'C:\TwinCAT\TcDatabaseSrv\Samples\TestDB1000SPS.mdb',
            ePath:= PATH_GENERIC,
            bExecute:= TRUE,
            tTimeout:= T#5s,
            bBusy=> bBusy_Delete,
            bError=> ,
            nErrId=> );

        IF NOT bBusy_Delete THEN
            nState := 2;
        END_IF

    2:
        (*The FB_DBCreate block will create the database file "C:
\TwinCAT\TcDatabaseSrv\Samples\TestDB1000SPS.mdb"*)
        FB_DBCreate1(
            sNetID:= ,
            sPathName:= 'C:\TwinCAT\TcDatabaseSrv\Samples',
            sDBName:= 'TestDB1000SPS',
            eDBType:= eDBType_Access,
            bExecute:= TRUE,
            tTimeout:= T#15s,
            bBusy=> bBusy_CreateDB,
            bError=> bErr,
            nErrID=> nErrid);

        IF NOT bBusy_CreateDB AND NOT bErr THEN
            nState := 3;
```

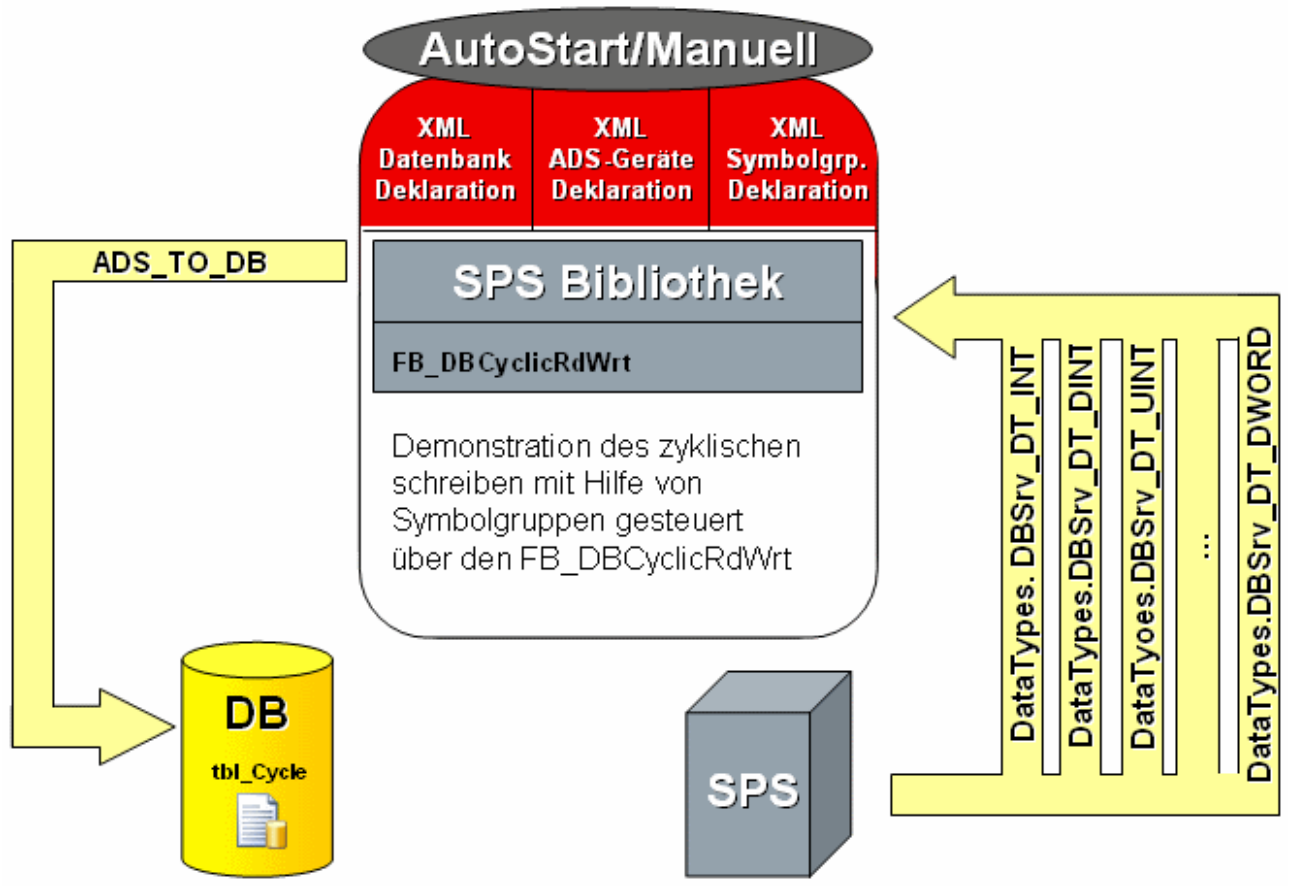
```
        END_IF
    3:
        (*The FB_DBConnectionAdd adds the connection information to the XML configuration file*)
        (*ATTENTION: Each database type has his own connection information*)
        FB_DBConnectionAdd1(
            sNetID:= ,
            eDBType:= eDBType_Access,
            eDBValueType:= eDBValue_Double,
            sDBServer:= ,
            sDBProvider:= 'Microsoft.Jet.OLEDB.4.0',
            sDBUrl:= 'C:\TwinCAT\TcDatabaseSrv\Samples\TestDB1000SPS.mdb',
            sDBTable:= 'myTable',
            bExecute:= TRUE,
            tTimeout:= T#15s,
            bBusy=> bBusy_ConnAdd,
            bError=> bErr,
            nErrID=> nErrid,
            hDBID=> nDBid);

        IF NOT bBusy_ConnAdd AND NOT bErr THEN
            nState := 4;
        END_IF
    4:
        (*The FB_DBTableCreate create the table "myTable"*)
        FB_DBTableCreate1(
            sNetID:= ,
            hDBID:= nDBid,
            sTableName:= 'myTable',
            cbTableCfg:= SIZEOF(arrTablestrc),
            pTableCfg:= ADR(arrTablestrc),
            bExecute:= TRUE,
            tTimeout:= T#15s,
            bBusy=> bBusy_CreateTable,
            bError=> bErr,
            nErrID=> nErrid);

        IF NOT bBusy_CreateTable AND NOT bErr THEN
            nState := 0;
        END_IF
END_CASE
```

Um dieses Beispiel zu benutzen, müssen Sie nur die NetID des ADS-Gerätes (auf dem der TwinCAT Database Server installiert ist) an den Eingang sNetID übergeben.

8.3 Starten / Stoppen des zyklischen Loggen

Download "Beispiel Starten / Stoppen des zyklischen Loggen" https://infosys.beckhoff.com/content/1031/tcdbserver/Resources/11407903371.zip

In diesem Beispiel wird das Starten und Stoppen des zyklischen Loggen aus der SPS heraus gezeigt.



Verwendeter Datenbanktyp	MS Compact SQL
Kompatible Datenbanktypen	ASCII, MS SQL, MS Compact SQL, MS Access, MySQL, PostgreSQL, DB2, Oracle, InterBase/Firebird, XML
Verwendete Funktionsbausteine	FB_DBCyclicRdWrt
Einzubindende Bibliotheken	"**TcDatabase.lib**", "TcSystem.lib", "TcBase.lib", "STANDARD.lib"
Download Dateiliste	FB_DBCyclicRdWrt_Sample.pro, CurrentConfigDataBase.xml, TestDB_Cyclic.sdf

In diesem Beispiel wird durch Toggeln der bStartStop Variablen die zyklische Logfunktion gestartet bzw. gestoppt.
Bei einer positiven Flanke am bExecute Eingang startet der zyklische Logvorgang.
Bei einer negativen Flanke wird dieser wieder beendet.

Variablendeklaration (PRG DataTypes)

```
PROGRAM DataTypes
VAR
    DBSrv_DT_INT        : INT;
    DBSrv_DT_UINT       : UINT;
    DBSrv_DT_DINT       : DINT;
    DBSrv_DT_UDINT      : UDINT;
    DBSrv_DT_REAL       : REAL;
    DBSrv_DT_LREAL      : LREAL;
    DBSrv_DT_BYTE       : BYTE := 16#A1;
    DBSrv_DT_BOOL       : BOOL;
    DBSrv_DT_MYSTRUCT   : ST_MyStruct;
    DBSrv_DT_ARRAY      : ARRAY [0..19] OF UDINT;
    DBSrv_DT_WORD       : WORD;
    DBSrv_DT_DWORD      : DWORD;
END_VAR
```

Struktur ST_MyStruct

```
TYPE ST_MyStruct :
STRUCT
    iValue1 : INT;
```

```
    iValue2 : UINT;
    iValue3 : BOOL;
    iValue4 : REAL;
END_STRUCTEND_TYPE
```

Variablendeklaration

```
PROGRAM MAIN
VAR
    fbDBCyclicRdWrt1: FB_DBCyclicRdWrt;

    bCyclic     : BOOL    :=TRUE;

    bBusy_Cyclic    : BOOL;
    bErr        : BOOL;
    nErrID      : UDINT;
    sSQLState           : ST_DBSQLError;
END_VAR
```

SPS Programm

```
DataTypes;

fbDBCyclicRdWrt(
    sNetID := ,
    bExecute := bCyclic,
    tTimeout := t#15s,
    bBusy => bBusy_Cyclic,
    bError => bErr,
    nErrID => nErrID,
    sSQLState => sSQLState);
```

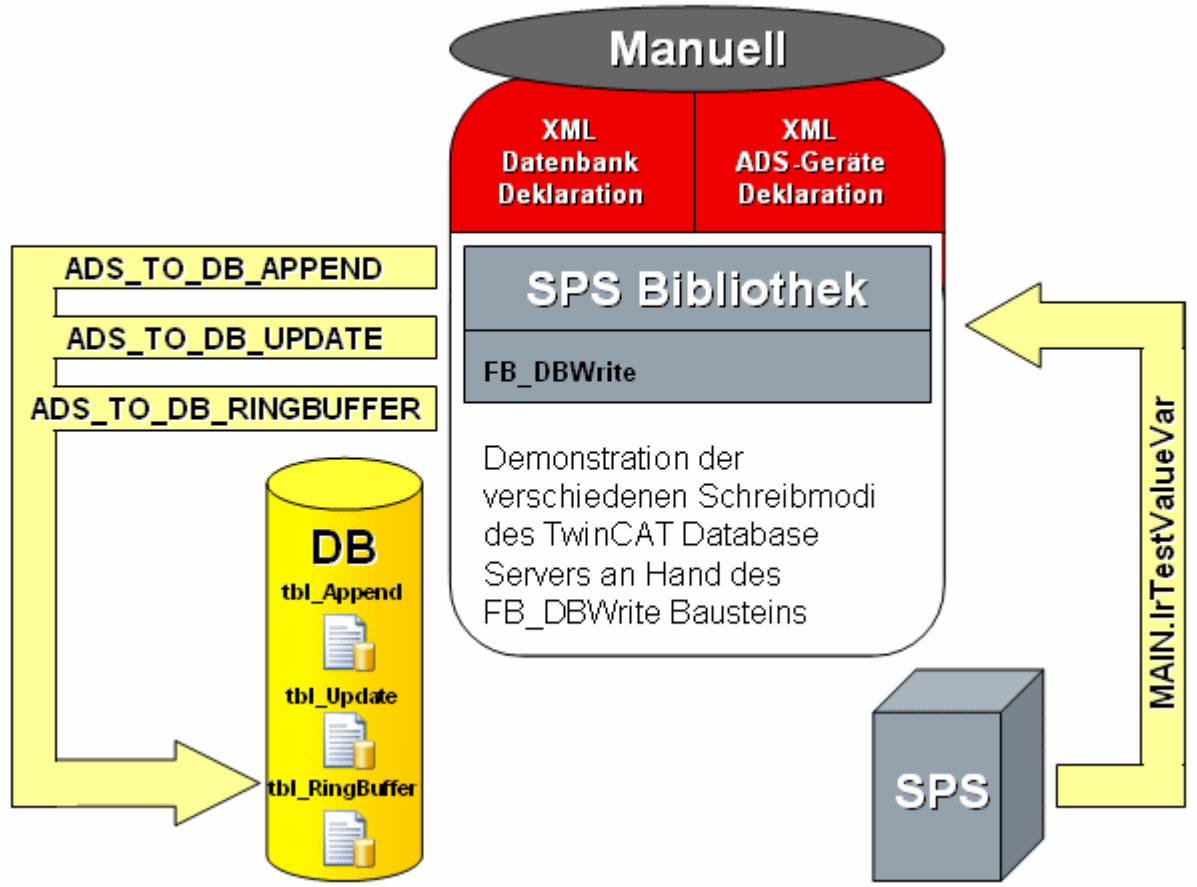
Um dieses Beispiel zu benutzen, müssen Sie nur die NetID des ADS-Gerätes (auf dem der TwinCAT Database Server installiert ist) an den Eingang sNetID übergeben.
Wenn Sie das Programm starten und die bCyclic Variable auf TRUE setzen, werden alle Variablen geloggt, die in der Symbolgruppe der XML-Konfigurationsdatei deklariert sind.

Alle Microsoft SQL Compact Datenbanken, die im XML-Konfigurationsdatei deklariert sind, müssen vorhanden sein. Sie werden nicht automatisch generiert.
Im Unterschied dazu werden die deklarierten ASCII-Dateien automatisch erzeugt, wenn Sie nicht vorhanden sind.

8.4 Loggen einer SPS-Variable mit FB_DBWrite

Download: "Beispiel Loggen einer SPS-Variable mit FB_DBWrite" https://infosys.beckhoff.com/content/1031/tcdbserver/Resources/11407904779.zip

In diesem Beispiel wird das Loggen einer SPS-Variablen aus der SPS in eine Datenbank demonstriert. Es wird die Funktionsweise der einzelnen Schreibmodi gezeigt.



Verwendeter Datenbanktyp	MS SQL
Kompatible Datenbanktypen	ASCII, MS SQL, MS Compact SQL, MS Access, MySQL, PostgreSQL, DB2, Oracle, InterBase/Firebird, XML
Verwendete Funktionsbausteine	FB_DBWrite
Einzubindende Bibliotheken	"**TcDatabase.lib**", "TcSystem.lib", "TcBase.lib", "STANDARD.lib"
Download Dateiliste	FB_DBWrite_Sample.pro, CurrentConfigDataBase.xml, SQLQuery.sql

Um dieses Beispiel verwenden zu können müssen Sie den Servernamen und die Authentifizierung in der XML-Konfigurationsdatei (CurrentConfigDataBase.xml) anpassen. Des Weiteren müssen Sie beachten, das keine "TestDB" Datenbank vorhanden ist bevor Sie das SQLQuery.sql Script ausführen.

Beispiel Aufbau:

Mit Hilfe der Variable "eWriteMode" kann eingestellt werden, mit welchen Schreibmodus geloggt werden soll. Mit einer steigenden Flanke an der Variable "bSTART" kann dann der Schreibvorgang gestartet werden.

Tabellenzuordnung:

- **ADS_TO_DB_APPEND** => eWriteAppend -> "tbl_Append"
- **ADS_TO_DB_UPDATE** => eWriteUpdate -> "tbl_Update"
- **ADS_TO_DB_RINGBUFFER** => eWriteRingBuffer -> "tbl_RingBuffer"

Verwendete Tabellenstruktur

Spaltenname	Datentyp	Null zulässig	Eigenschaft
ID	bigint	nein	IDENTITY(1,1)
Timestamp	datetime	nein	
Name	ntext	nein	
Value	float	nein	

Variablendeklaration

```
PROGRAM MAIN
VAR(*Test symbol which will be logged into the different database tables*)
    lrTestValueVar : LREAL := 123.456;

    eState : E_SampleState := eIdle;
    R_TRIG1: R_TRIG;

    (*With a rising edge at bStart the FB_DBWrite block will be start once*)
    bSTART: BOOL;

    (*With eWriteMode you can select which FB_DBWrite block will be used*)
    eWriteMode: E_SampleState := eWriteAppend;

    FB_DBWrite_Append: FB_DBWrite;
    FB_DBWrite_Update: FB_DBWrite;
    FB_DBWrite_RingBuffer: FB_DBWrite;

    (*Status outputs from the three FB_DBWrite blocks*)
    bBusy: BOOL;
    bErr: BOOL;
    bErrid: UDINT;
    stSqlstate: ST_DBSQLError;
END_VAR
```

Enum E_SampleState

```
TYPE E_SampleState :(
    eIdle := 0,
    eWriteAppend := 1,
    eWriteUpdate := 2,
    eWriteRingBuffer := 3
);
END_TYPE
```

SPS Programm

```
CASE eState OF
    eIdle :
        R_TRIG1(CLK:=bSTART);
        IF R_TRIG1.Q THEN
            lrTestValueVar := lrTestValueVar + 1;
            eState := eWriteMode;
            bSTART := FALSE;
        END_IF(*Add a new record to the table tbl_Append*)
    eWriteAppend :
        FB_DBWrite_Append(
            sNetID:= ,
            hDBID:= 1,
            hAdsID:= 1,
            sVarName:= 'MAIN.lrTestValueVar',
            nIGroup:= ,
            nIOffset:= ,
            nVarSize:= ,
            sVarType:= ,
            sDBVarName:= 'lrTestValueVar',
            eDBWriteMode:= eDBWriteMode_Append,
            tRingBufferTime:= ,
            nRingBufferCount:= ,
            bExecute:= TRUE,
            tTimeout:= T#15s,
            bBusy=> bBusy,
            bError=> bErr,
            nErrID=> bErrid,
            sSQLState=> stSqlstate);

        IF NOT bBusy THEN
            FB_DBWrite_Append(bExecute := FALSE);
            eState := eIdle;
        END_IF(*Add a new record to the table tbl_Update if it not exist else the existing record wi
ll be updated*)
    eWriteUpdate :
        FB_DBWrite_Update(
            sNetID:= ,
            hDBID:= 2,
            hAdsID:= 1,
            sVarName:= 'MAIN.lrTestValueVar',
            nIGroup:= ,
            nIOffset:= ,
```

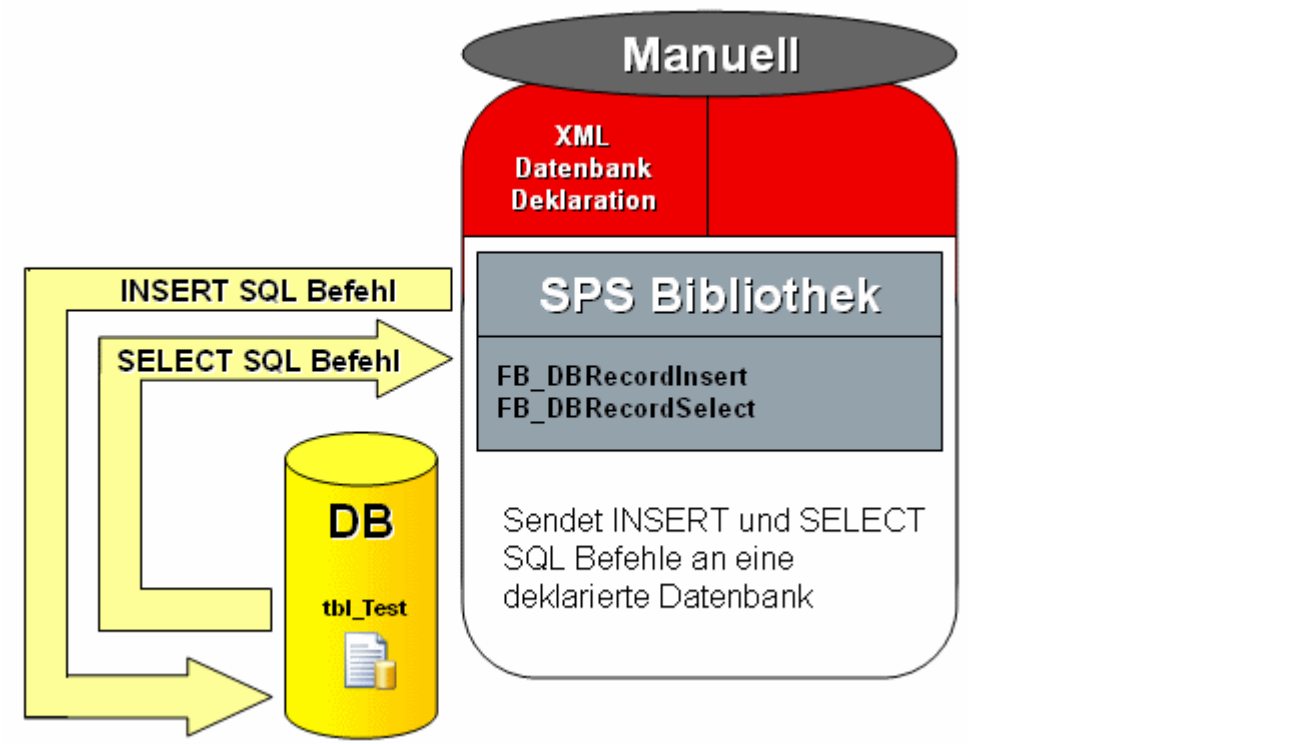
```
            nVarSize:= ,
            sVarType:= ,
            sDBVarName:= 'lrTestValueVar',
            eDBWriteMode:= eDBWriteMode_Update,
            tRingBufferTime:= ,
            nRingBufferCount:= ,
            bExecute:= TRUE,
            tTimeout:= T#15s,
            bBusy=> bBusy,
            bError=> bErr,
            nErrID=> bErrid,
            sSQLState=> stSqlstate);

        IF NOT bBusy THEN
            FB_DBWrite_Update(bExecute := FALSE);
            eState := eIdle;
        END_IF(*Add a new record to the table tbl_RingBuffer. If the maximum count is reached the re
cords will be deleted in a FIFO process*)
    eWriteRingBuffer :
        FB_DBWrite_RingBuffer(
            sNetID:= ,
            hDBID:= 3,
            hAdsID:= 1,
            sVarName:= 'MAIN.lrTestValueVar',
            nIGroup:= ,
            nIOffset:= ,
            nVarSize:= ,
            sVarType:= ,
            sDBVarName:= 'lrTestValueVar',
            eDBWriteMode:= eDBWriteMode_RingBuffer_Count,
            tRingBufferTime:= ,
            nRingBufferCount:= 10,
            bExecute:= TRUE,
            tTimeout:= T#15s,
            bBusy=> bBusy,
            bError=> bErr,
            nErrID=> bErrid,
            sSQLState=> stSqlstate);

        IF NOT bBusy THEN
            FB_DBWrite_RingBuffer(bExecute := FALSE);
            eState := eIdle;
        END_IFEND_CASE
```

Alle Microsoft SQL Compact Datenbanken, die im XML-Konfigurationsfile deklariert sind, müssen vorhanden sein. Sie werden nicht automatisch generiert.
Im Unterschied dazu werden die deklarierten ASCII-Dateien automatisch erzeugt, wenn Sie nicht vorhanden sind.

8.5 Beispiel mit dem FB_DBRecordInsert und FB_DBRecordSelect Baustein

Download "Beispiel Loggen mit dem FB_DBRecordInsert" https://infosys.beckhoff.com/content/1031/tcdbserver/Resources/11407906187.zip

In diesem Beispiel wird das Loggen mehrerer Werte in eine Datenbank aus der SPS heraus mit dem Funktionsbaustein FB_DBRecordInsert gezeigt.
Speziell in diesem Beispiel werden mehrere SPS Variablen in einen Datensatz geloggt. Des Weiteren kann mit dem Funktionsbaustein FB_DBRecordSelect ein Datensatz aus dieser Datenbank ausgelesen werden



Verwendeter Datenbanktyp	MS Access
Kompatible Datenbanktypen	MS SQL, MS Compact SQL, MS Access, MySQL, PostgreSQL, DB2, Oracle, InterBase/Firebird, XML
Verwendete Funktionsbausteine	FB_DBRecordInsert, FB_DBRecordSelect
Einzubindende Bibliotheken	**"TcDatabase.lib"**, "TcSystem.lib", "TcBase.lib", "STANDARD.lib", "TcUtilities.lib"
Download Dateiliste	FB_DBRecodInsertSelectSample.pro, CurrentConfigDataBase.xml, TestDB_Access.mdb

Es wird in folgende Tabellenstruktur geschrieben:

Spaltenname	Datentyp
Timestamp	datetime
PLC_TestValue1	float
PLC_TestValue2	float
PLC_TestValue3	float
PLC_TestValue4	String

Um dieses Beispiel zu benutzen, müssen Sie die Access Datenbank "Sample7.mdb" in der XML-Konfigurationsdatei deklarieren.

Durch erzeugen einer positiven Flanke an der Variablen "bStartstopInsert ", wird eine Datensatz mit den vier SPS Werten und dem Timestamp in der Datenbank angelegt.

Variablendeklaration

```
(* Declaration *)PROGRAM MAIN
VAR
    eState: E_SQLStatement;

    NT_GetTime1: NT_GetTime;
    bTimestart: BOOL;
    tTime: TIMESTRUCT;

    FB_FormatStringDateTime: FB_FormatString;
    sDateTimeString: T_MaxString;

    TestValue1: REAL := 123.456;
    TestValue2: REAL := 234.567;
```

```
    TestValue3: REAL := 345.678;
    TestValue4: STRING(255) := 'No error occurred';

    FB_FormatString1: FB_FormatString;
    sInsertString: T_MaxString;
    bError: BOOL;
    nErrid: UDINT;

    FB_DBRecordInsert1: FB_DBRecordInsert;
    bStartstopInsert: BOOL;
    bBusyInsert: BOOL;
    bErrInsert: BOOL;
    nErridInsert: UDINT;
    stSQLStateInsert: ST_DBSQLError;

    stRecord: ST_Record;

    FB_DBRecordSelect1: FB_DBRecordSelect;
    nRecIndex: UDINT := 0;
    bStartstopSelect: BOOL;
    bBusySelect: BOOL;
    bErrorSelect: BOOL;
    nErrIDSelect: UDINT;
    stSQLStateSelect: ST_DBSQLError;
    nRecordCount: UDINT;
END_VAR
```

Enum E_SQLStatement

```
TYPE E_SQLStatement :(
    eSQL_INSERT := 0,
    eSQL_SELECT := 1
);
END_TYPE
```

Struct ST_Record

```
TYPE ST_Record :
STRUCT
    Timestamp : DT;
    PLC_Value1 : REAL;
    PLC_Value2 : REAL;
    PLC_Value3 : REAL;
    PLC_Value4 : STRING;
END_STRUCT
END_TYPE
```

SPS Programm

```
CASE eState OF
    eSQL_INSERT:
        (*Create the timestamp*)
        NT_GetTime1( START:= bTimestart, TIMESTR=> tTime);
        IF NOT NT_GetTime1.BUSY THEN
            bTimestart := NOT bTimestart;
        END_IF

        FB_FormatStringDateTime(
            sFormat:= '%D.%D.%D %D:%D:%D',
            arg1:= F_WORD(tTime.wYear),
            arg2:= F_WORD(tTime.wMonth),
            arg3:= F_WORD(tTime.wDay),
            arg4:= F_WORD(tTime.wHour),
            arg5:= F_WORD(tTime.wMinute),
            arg6:= F_WORD(tTime.wSecond),
            sOut=> sDateTimeString);

        (*------------------------------------------------------------------------*)(*Create the SQL-
INSERT command*)
        FB_FormatString1(
            sFormat:= 'INSERT INTO tbl_Test VALUES($'%S$',%F,%F,%F,$'%S$')',
            arg1:= F_STRING(sDateTimeString),
            arg2:= F_REAL(TestValue1),
            arg3:= F_REAL(TestValue2),
            arg4:= F_REAL(TestValue3),
            arg5:= F_STRING(TestValue4),
            sOut=> sInsertString,
            bError=> bError,
            nErrId=> nErrid);
```

```
        (*-------------------------------------------------------------------------*)
(*Write the record to the database*)
        FB_DBRecordInsert1(
            sNetID:= ,
            hDBID:= 1,
            sInsertCmd:= sInsertString,
            bExecute:= bStartstopInsert,
            tTimeout:= T#15s,
            bBusy=> bBusyInsert,
            bError=> bErrInsert,
            nErrID=> nErridInsert,
            sSQLState=> stSQLStateInsert);

    eSQL_SELECT:
        (*Read one record from the database*)
        FB_DBRecordSelect1(
            sNetID:= ,
            hDBID:= 1,
            sSelectCmd:= 'SELECT * FROM tbl_Test',
            nRecordIndex:= nRecIndex,
            cbRecordSize:= SIZEOF(stRecord),
            pDestAddr:= ADR(stRecord),
            bExecute:= bStartstopSelect,
            tTimeout:= T#15s,
            bBusy=> bBusySelect,
            bError=> bErrorSelect,
            nErrID=> nErrIDSelect,
            sSQLState=> stSQLStateSelect,
            nRecords=> nRecordCount);
END_CASE
```

Screenshot

tbl_Test : Tabelle

	Timestamp	PLC_Value1	PLC_Value2	PLC_Value3	PLC_Value4
	06.10.2010 10:03:59	123,456	234,567	345,678	No error occurred
	06.10.2010 10:04:03	123,456	234,567	345,678	No error occurred
	06.10.2010 10:04:07	123,456	234,567	345,678	No error occurred
	06.10.2010 10:04:10	123,456	234,567	345,678	No error occurred
	06.10.2010 10:11:55	123,456	234,567	345,678	No error occurred
	06.10.2010 10:12:00	123,456	234,567	345,678	No error occurred
▶		0	0	0	

Datensatz: 7 von 7

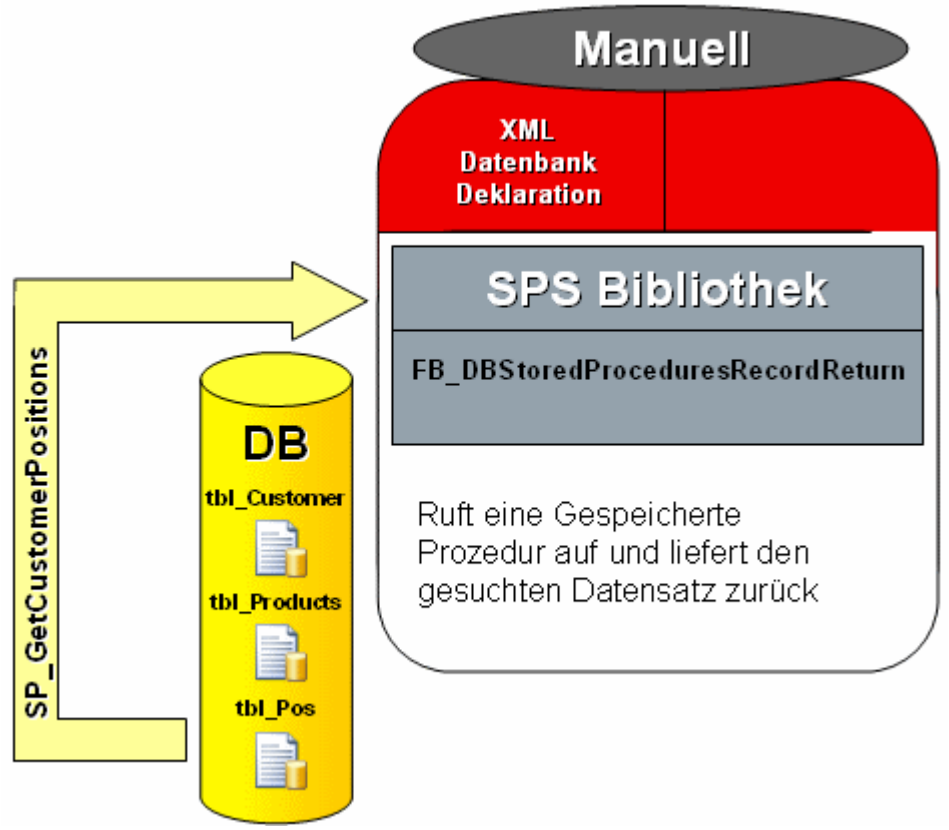
8.6 Gespeicherte Prozeduren mit MS SQL

Es können keine "Gespeicherte Prozeduren" mit dem Database Server erstellt bzw. konfiguriert werden.

Gespeicherte Prozeduren können aber ab der Version 1.0.13 mit den Funktionsbausteinen FB_DBStoredProcedures und FB_DBStoredProceduresRecordReturn ausgeführt werden.

Mit Hilfe diesen Funktionsbausteinen können Parameter als INPUT, OUTPUT oder INOUT deklariert werden und den gespeicherten Prozeduren übergeben werden. So können komplexe SQL-Kommandos am Datenbank Server vorprogrammiert werden und müssen nur noch vom TwinCAT Database Server angestoßen werden.

Download "Beispiel mit Gespeicherter Prozedur" https://infosys.beckhoff.com/content/1031/tcdbserver/Resources/11407907595.zip



Verwendeter Datenbanktyp	MS SQL (MS SQL Server 2008)
Kompatible Datenbanktypen	MS SQL, MySQL, Oracle
Verwendete Funktionsbausteine	FB_DBStoredProceduresRecordReturn
Einzubindende Bibliotheken	**"TcDatabase.lib"**, "TcSystem.lib", "TcBase.lib", "STANDARD.lib", "TcUtilities.lib"
Download Dateiliste	FB_DBStoredProcedures_Sample.pro, CurrentConfigDataBase.xml, SQLQuery2.sql

Das folgende Beispiel zeigt ihnen den Aufruf einer einfachen Gespeicherten Prozedur mit einem Eingangsparameter und Rückgabedatensatz. Die Prozedur wurde erstellt an einem Microsoft SQL Server 2008.

Code der Gespeicherten Prozedur SP_GetAddressByCustomerID

```
CREATE PROCEDURE [SP_GetAddressByCustomerID]
    @Customer_ID bigint
AS
BEGIN
    SELECT
      tbl_Customer.ID,
      tbl_Customer.Name,
      tbl_Customer.Customer,
      tbl_Products.SerNum,
      tbl_Products.Product,
      tbl_Products.Info,
      tbl_Pos.Timestamp
    FROM
        tbl_Pos JOIN tbl_Customer ON tbl_Pos.CustomerNum = tbl_Customer.ID
        JOIN tbl_Products ON tbl_Pos.ProductNum = tbl_Products.SerNum
    WHERE
        tbl_Pos.CustomerNum = @Customer_ID;
END
```

Variablendeklaration in der SPS

```
PROGRAM MAIN
VAR
    R_TRIG1: R_TRIG;
    bREAD : BOOL := FALSE;
```

```
    nState: BYTE;

    arrParaList: ARRAY [0..0] OF ST_DBParameter;

    nCustomerID: DINT := 12345;
    nRecordIndex: UDINT;
    stRecord: ST_Record;
    nRecs: UDINT;

    FB_DBStoredProceduresRecordReturn1: FB_DBStoredProceduresRecordReturn;

    bBusy: BOOL;
    bErr: BOOL;
    nErrid: UDINT;
    stSqlstate: ST_DBSQLError;
END_VAR
```

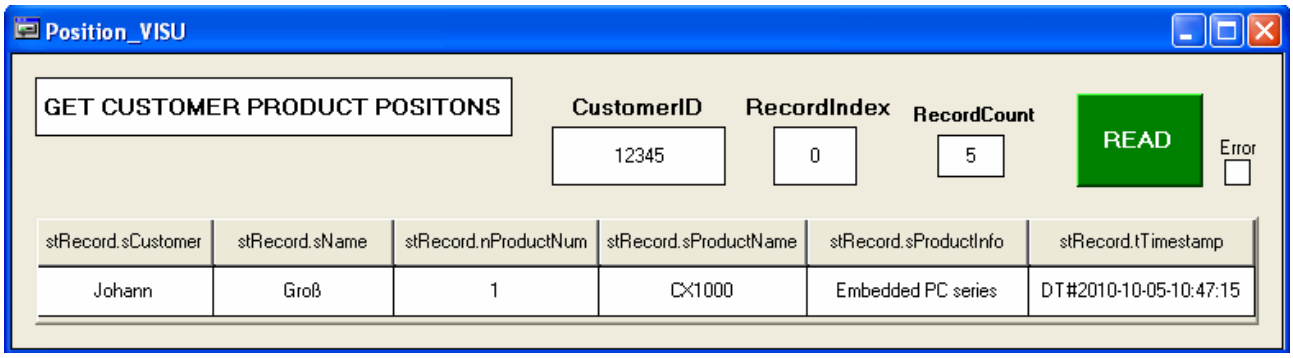
Datensatzstruktur in der SPS (ST_Record)

```
TYPE ST_Record :
STRUCT
    nID : T_ULARGE_INTEGER;
    sCustomer : STRING;
    sName : STRING;
    nProductNum : DINT;
    sProductName : STRING;
    sProductInfo : STRING;
    tTimestamp : DT;
END_STRUCT
END_TYPE
```

SPS Programm

```
R_TRIG1(CLK:=bREAD);
IF R_TRIG1.Q AND NOT bBusy THEN
    nState := 1;
END_IFCASE nState OF
    0:
        ;
    1:(*Init of the parameters*)
        arrParaList[0].sParameterName := '@Customer_ID';
        arrParaList[0].eParameterDataType := eDBColumn_Integer;
        arrParaList[0].eParameterType := eDBParameter_Input;
        arrParaList[0].cbParameterValue := SIZEOF(nCustomerID);
        arrParaList[0].pParameterValue := ADR(nCustomerID);

        nState := 2;
    2:(*Start the stored procedure "SP_GetCustomerPosition"*)
        FB_DBStoredProceduresRecordReturn1(
            sNetID:= ,
            hDBID:= 1,
            sProcedureName:= 'SP_GetCustomerPositions',
            cbParameterList:= SIZEOF(arrParaList),
            pParameterList:= ADR(arrParaList),
            nRecordIndex:= nRecordIndex,
            cbRecordSize:= SIZEOF(stRecord),
            pRecordAddr:= ADR(stRecord),
            bExecute:= TRUE,
            tTimeout:= T#15s,
            bBusy=> bBusy,
            bError=> bErr,
            nErrID=> nErrid,
            sSQLState=> stSqlstate,
            nRecords=> nRecs);

        IF NOT bBusy THEN
            FB_DBStoredProceduresRecordReturn1(bExecute:= FALSE);
            nState := 0;
        END_IFEND_CASE
```

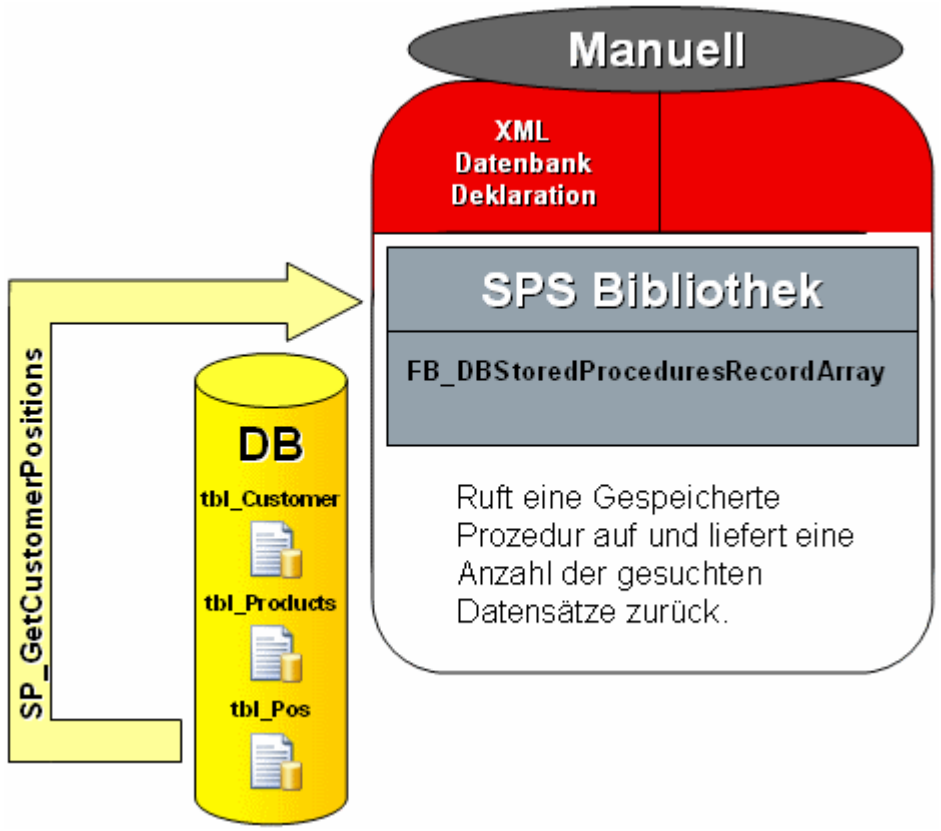
Visualisierung



8.7 Gespeicherte Prozeduren mit FB_DBStoredProceduresRecordArray

Mit Hilfe des Funktionsbausteins FB_DBStoredProceduresRecordArray können Parameter als INPUT, OUTPUT oder INOUT deklariert werden und den Gespeicherten Prozeduren übergeben werden. So können komplexe SQL-Kommandos am Datenbank Server vorprogrammiert und müssen nur noch vom TwinCAT Database Server angetriggert werden. Im Gegensatz zum FB_DBStoredProceduresRecordReturn Baustein können mehrere Datensätze mit einem Aufruf zurückgeliefert werden.

Download "Beispiel mit Gespeicherter Prozedur" https://infosys.beckhoff.com/content/1031/tcdbserver/Resources/11407909003.zip



Verwendeter Datenbanktyp	MS SQL (MS SQL Server 2008)
Kompatible Datenbanktypen	MS SQL, MySQL, Oracle
Verwendete Funktionsbausteine	FB_DBStoredProceduresRecordArray
Einzubindende Bibliotheken	"**TcDatabase.lib**", "TcSystem.lib", "TcBase.lib", "STANDARD.lib", "TcUtilities.lib"
Download Dateiliste	FB_DBStoredProceduresRecordArray_Sample.pro, CurrentConfigDataBase.xml, SQLQuery2.sql

Das folgende Beispiel zeigt ihnen den Aufruf einer einfachen Gespeicherten Prozedur mit einem Eingangsparameter und Rückgabedatensatz. Die Prozedur wurde erstellt an einem Microsoft SQL Server 2008.

Code der Gespeicherten Prozedur SP_GetAddressByCustomerID

```
CREATE PROCEDURE [SP_GetAddressByCustomerID]
    @Customer_ID bigint
AS
BEGIN
    SELECT tbl_Customer.ID, tbl_Customer.Name, tbl_Customer.Customer, tbl_Products.SerNum, tbl_Produ
cts.Product, tbl_Products.Info, tbl_Pos.Timestamp FROM
        tbl_Pos JOIN tbl_Customer ON tbl_Pos.CustomerNum = tbl_Customer.ID
        JOIN tbl_Products ON tbl_Pos.ProductNum = tbl_Products.SerNum
    WHERE
        tbl_Pos.CustomerNum = @Customer_ID;
END
```

Variablendeklaration in der SPS

```
PROGRAM MAIN
VAR
    R_TRIG1: R_TRIG;
    bREAD : BOOL := FALSE;

    nState: BYTE;

    arrParaList: ARRAY [0..0] OF ST_DBParameter;

    nCustomerID: DINT := 12345;

    FB_DBStoredProceduresRecordArray1: FB_DBStoredProceduresRecordArray;

    nCustomerID: DINT := 12345;
    nRecordStartIndex: UDINT;
    stRecordArr: ARRAY [1..25] OF ST_Record;
    nRecs: UDINT;

    bBusy: BOOL;
    bErr: BOOL;
    nErrid: UDINT;
    stSqlstate: ST_DBSQLError;
END_VAR
```

Datensatzstruktur in der SPS (ST_Record)

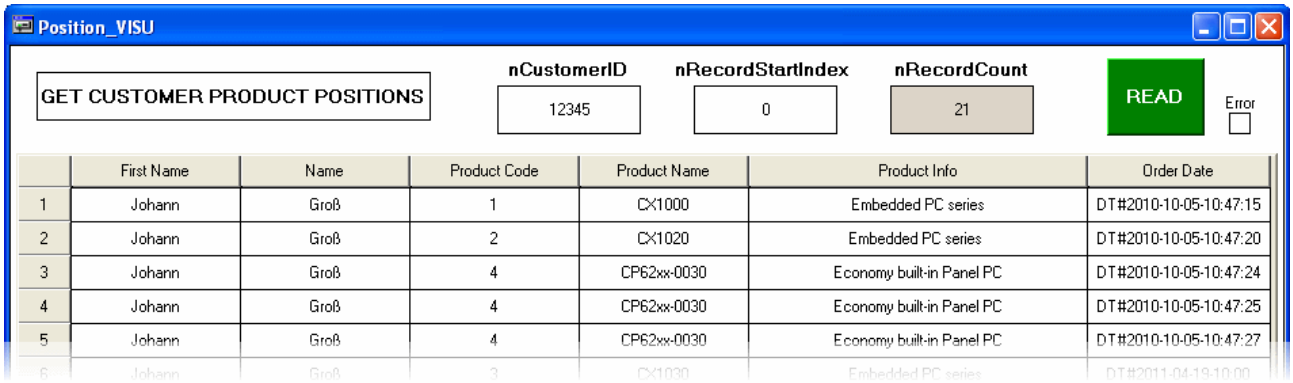
```
TYPE ST_Record :
STRUCT
    nID : T_ULARGE_INTEGER;
    sCustomer : STRING(50);
    sName : STRING(50);
    nProductNum : DINT;
    sProductName : STRING(50);
    sProductInfo : T_MaxString;
    tTimestamp : DT;
END_STRUCT
END_TYPE
```

SPS Programm

```
R_TRIG1(CLK:=bREAD);
IF R_TRIG1.Q AND NOT bBusy THEN
    nState := 1;
END_IFCASE nState OF
    0:
        ;
    1:(*Init of the parameters*)
        arrParaList[0].sParameterName := '@Customer_ID';
        arrParaList[0].eParameterDataType := eDBColumn_Integer;
        arrParaList[0].eParameterType := eDBParameter_Input;
        arrParaList[0].cbParameterValue := SIZEOF(nCustomerID);
        arrParaList[0].pParameterValue := ADR(nCustomerID);

        nState := 2;
    2:(*Start the stored procedure "SP_GetCustomerPosition"*)
        FB_DBStoredProceduresRecordArray1(
            sNetID:= ,
            hDBID:= 1,
```

```
            sProcedureName:= 'SP_GetCustomerPositions',
            cbParameterList:= SIZEOF(arrParaList),
            pParameterList:= ADR(arrParaList),
            nStartIndex:= nRecordStartIndex,
            nRecordCount:= 25,
            cbRecordArraySize:= SIZEOF(stRecordArr),
            pDestAddr:= ADR(stRecordArr),
            bExecute:= TRUE,
            tTimeout:= T#15s,
            bBusy=> bBusy,
            bError=> bErr,
            nErrID=> nErrid,
            sSQLState=> stSqlstate,
            nRecords=> nRecs);

        IF NOT bBusy THEN
            FB_DBStoredProceduresRecordReturn1(bExecute:= FALSE);
            nState := 0;
        END_IFEND_CASE
```

Visualisierung

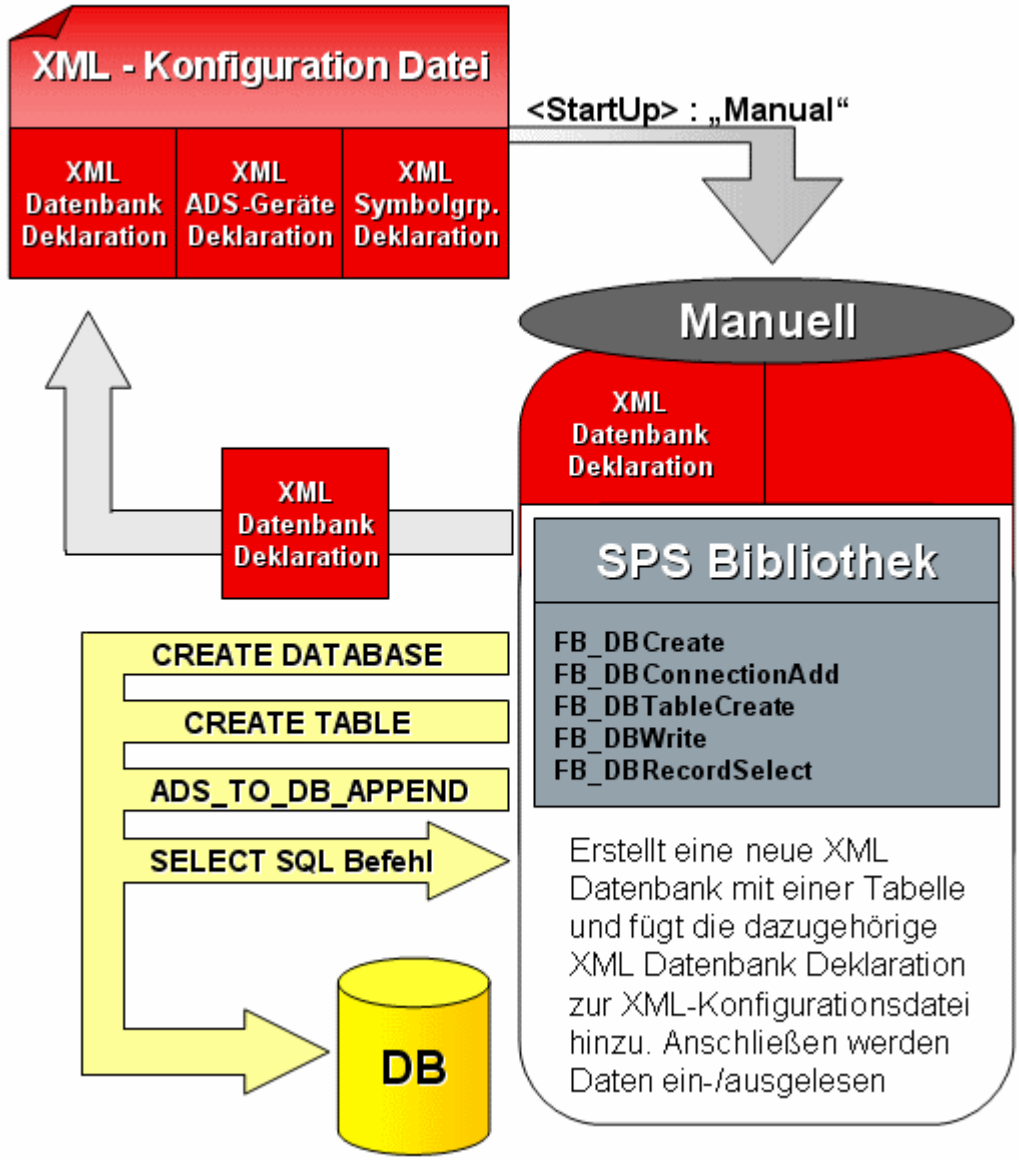


8.8 XML als Datenbank nutzen

Der TwinCAT Database Server bietet die Möglichkeit eine XML-Datei als Datenbank zu verwenden. Bis auf die Stored Procedure Funktionen werden alle bekannten Funktionsbausteine für das Lesen und Schreiben in eine Datenbank von dem XML-Datenbanktyp unterstützt. Selbst SQL-Befehle, die mit den Funktionsbausteinen FB_DBRecordInsert oder FB_DBRecordSelect abgesetzt werden können, werden vom TwinCAT Database Server interpretiert und entsprechend auf die XML-Datei angewendet.

In diesem Beispiel wird demonstriert, wie eine XML-Datenbank erzeugt, mit dem Baustein FB_DBWrite befüllt und anschließend mit einem SQL-SELECT Befehl und dem FB_DBRecordSelect wieder ausgelesen wird.

Download "Beispiel mit XML als Datenbank nutzen" https://infosys.beckhoff.com/content/1031/tcdbserver/Resources/11407910411.zip



Verwendeter Datenbanktyp	XML
Kompatible Datenbanktypen	MS SQL, MS Compact SQL, MS Access, XML
Verwendete Funktionsbausteine	FB_DBCreate, FB_DBConnectionAdd, FB_DBTableCreate, FB_DBWrite, FB_DBRecordSelect
Einzubindende Bibliotheken	"**TcDatabase.lib**", "TcSystem.lib", "TcBase.lib", "STANDARD.lib", "TcUtilities.lib"
Download Dateiliste	XML_DatabaseType.pro

MAIN Programm

```
PROGRAM MAIN
VAR
    nState:BYTE := 0;

    R_TRIG1: R_TRIG;
    bSTART: BOOL;

    nCounter: INT;

    FB_FileDelete1: FB_FileDelete;
    FB_DBCreate1: FB_DBCreate;
    FB_DBConnectionAdd1: FB_DBConnectionAdd;
    FB_DBTableCreate1: FB_DBTableCreate;
    FB_DBWrite1: FB_DBWrite;
    FB_DBRecordSelect1: FB_DBRecordSelect;

    bBusy_Delete: BOOL;
```

```
    bBusy_CreateDB: BOOL;
    bBusy_ConnAdd: BOOL;
    bBusy_CreateTable: BOOL;
    bBusy_WriteDB: BOOL;
    bBusy_SelectRecord: BOOL;

    bErr: BOOL;
    nErrid: UDINT;
    stSQLState: ST_DBSQLError;
    nRecs: UDINT;

    nDBid: UDINT;

    arrTablestrc: ARRAY [0..3] OF ST_DBColumnCfg := (sColumnName:='ID',sColumnProperty:='IDENTITY(1,
1)',eColumnType:=EDBCOLUMN_BIGINT),
                    (sColumnName:='Timestamp',eColumnType:=EDBCOLUMN_DATETIME),
                    (sColumnName:='Name',sColumnProperty:='80',eColumnType:=EDBCOLUMN_NTEXT),
                    (sColumnName:='Value',eColumnType:=EDBCOLUMN_FLOAT);

    rTestValue : LREAL := 1234.56789;
    stRecord: ST_Record;
END_VAR
```

```
CASE nState OF
    0:
        (*To start this sample you have to set a rising edge to the variable bSTART*)
        R_TRIG1(CLK:=bSTART);
        IF R_TRIG1.Q THEN
            nState := 1;
            FB_FileDelete1(bExecute:=FALSE);
            FB_DBCreate1(bExecute:=FALSE);
            FB_DBConnectionAdd1(bExecute:=FALSE);
            FB_DBTableCreate1(bExecute:=FALSE);
            FB_DBWrite1(bExecute:=FALSE);
            FB_DBRecordSelect1(bExecute:=FALSE);
            bSTART := FALSE;
            nCounter := 0;
        END_IF
    1:
        (*It isn't possible to overwrite an existing database file. If the database file exist the F
B_FileDelete block will delete the file*)
        FB_FileDelete1(
            sNetId:= ,
            sPathName:= 'C:\TwinCAT\TcDatabaseSrv\Samples\XMLTestDB.xml',
            ePath:= PATH_GENERIC,
            bExecute:= TRUE,
            tTimeout:= T#5s,
            bBusy=> bBusy_Delete,
            bError=> ,
            nErrId=> );

        IF NOT bBusy_Delete THEN
            nState := 10;
        END_IF
    10:
        (*It isn't possible to overwrite an existing database file. If the database file exist the F
B_FileDelete block will delete the file*)
        FB_FileDelete1(
            sNetId:= ,
            sPathName:= 'C:\TwinCAT\TcDatabaseSrv\Samples\XMLTestDB.xsd',
            ePath:= PATH_GENERIC,
            bExecute:= TRUE,
            tTimeout:= T#5s,
            bBusy=> bBusy_Delete,
            bError=> ,
            nErrId=> );

        IF NOT bBusy_Delete THEN
            FB_FileDelete1(bExecute:=FALSE);
            nState := 2;
        END_IF
    2:
        (*The FB_DBCreate block will create the database file "C:
\TwinCAT\TcDatabaseSrv\Samples\XMLTestDB.xml" and C:\TwinCAT\TcDatabaseSrv\Samples\XMLTestDB.xsd "*)
        FB_DBCreate1(
            sNetID:= ,
            sPathName:= 'C:\TwinCAT\TcDatabaseSrv\Samples',
            sDBName:= 'XMLTestDB',
```

```
            eDBType:= eDBType_XML,
            bExecute:= TRUE,
            tTimeout:= T#15s,
            bBusy=> bBusy_CreateDB,
            bError=> bErr,
            nErrID=> nErrid);

        IF NOT bBusy_CreateDB AND NOT bErr THEN
            nState := 3;
        END_IF
    3:
        (*The FB_DBConnectionAdd adds the connection information to the XML - configuration file*)
        (*ATTENTION: Each database type has his own connection information*)
        FB_DBConnectionAdd1(
            sNetID:= ,
            eDBType:= eDBType_XML,
            eDBValueType:= eDBValue_Double,
            sDBServer:= 'XMLTestDB',
            sDBProvider:= ,
            sDBUrl:= 'C:\TwinCAT\TcDatabaseSrv\Samples\XMLTestDB.xml',
            sDBTable:= 'myTable',
            bExecute:= TRUE,
            tTimeout:= T#15s,
            bBusy=> bBusy_ConnAdd,
            bError=> bErr,
            nErrID=> nErrid,
            hDBID=> nDBid);

        IF NOT bBusy_ConnAdd AND NOT bErr THEN
            nState := 4;
        END_IF
    4:
        (*The FB_DBTableCreate create the table "myTable"*)
        FB_DBTableCreate1(
            sNetID:= ,
            hDBID:= nDBid,
            sTableName:= 'myTable',
            cbTableCfg:= SIZEOF(arrTablestrc),
            pTableCfg:= ADR(arrTablestrc),
            bExecute:= TRUE,
            tTimeout:= T#15s,
            bBusy=> bBusy_CreateTable,
            bError=> bErr,
            nErrID=> nErrid);

        IF NOT bBusy_CreateTable AND NOT bErr THEN
            nState := 5;
        END_IF
    5:
        (*The FB_DBWrite write five times the value of the plc variable "rTestValue" to the database
table "myTable"*)
        FB_DBWrite1(
            sNetID:= ,
            hDBID:= nDBid,
            hAdsID:= 1,
            sVarName:= 'MAIN.rTestValue',
            nIGroup:= ,
            nIOffset:= ,
            nVarSize:= ,
            sVarType:= ,
            sDBVarName:= 'rTestValue',
            eDBWriteMode:= eDBWriteMode_Append,
            tRingBufferTime:= ,
            nRingBufferCount:= ,
            bExecute:= TRUE,
            tTimeout:= T#15s,
            bBusy=> bBusy_WriteDB,
            bError=> bErr,
            nErrID=> nErrid,
            sSQLState=> stSQLState);

        IF NOT bBusy_WriteDB AND NOT bErr THEN
            FB_DBWrite1(bExecute := FALSE);
            nCounter := nCounter + 1;
            IF nCounter = 5 THEN
                nState := 6;
            END_IFEND_IF
    6:
        (*The FB_DBRecordSelect select one record of the database table "myTable""*)
        FB_DBRecordSelect1(
```

```
            sNetID:= ,
            hDBID:= nDBid,
            sSelectCmd:= 'SELECT * FROM myTable WHERE Name = $'rTestValue$'',
            nRecordIndex:= 0,
            cbRecordSize:= SIZEOF(stRecord),
            pDestAddr:= ADR(stRecord),
            bExecute:= TRUE,
            tTimeout:= T#15s,
            bBusy=> bBusy_SelectRecord,
            bError=> bErr,
            nErrID=> nErrid,
            sSQLState=> stSQLState,
            nRecords=> nRecs);

        IF NOT bBusy_SelectRecord AND NOT bErr THEN
            nState := 0;
        END_IFEND_CASE
```

Mit einer Positiven Flanke an der Toggle Variable bSTART wird der Ablauf gestartet.

Folgende Dateien werden erzeugt:

XMLTestDB.xml (XML Datenbank Datei)

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLTestDB xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
xs:noNamespaceSchemaLocation="XMLTestDB.xsd">
  <myTable>
    <row ID="1" Timestamp="2012-05-10T13:48:47" Name="rTestValue" Value="1234.56789" />
    <row ID="2" Timestamp="2012-05-10T13:48:47" Name="rTestValue" Value="1234.56789" />
    <row ID="3" Timestamp="2012-05-10T13:48:47" Name="rTestValue" Value="1234.56789" />
    <row ID="4" Timestamp="2012-05-10T13:48:47" Name="rTestValue" Value="1234.56789" />
    <row ID="5" Timestamp="2012-05-10T13:48:47" Name="rTestValue" Value="1234.56789" />
 </myTable>
</XMLTestDB>
```

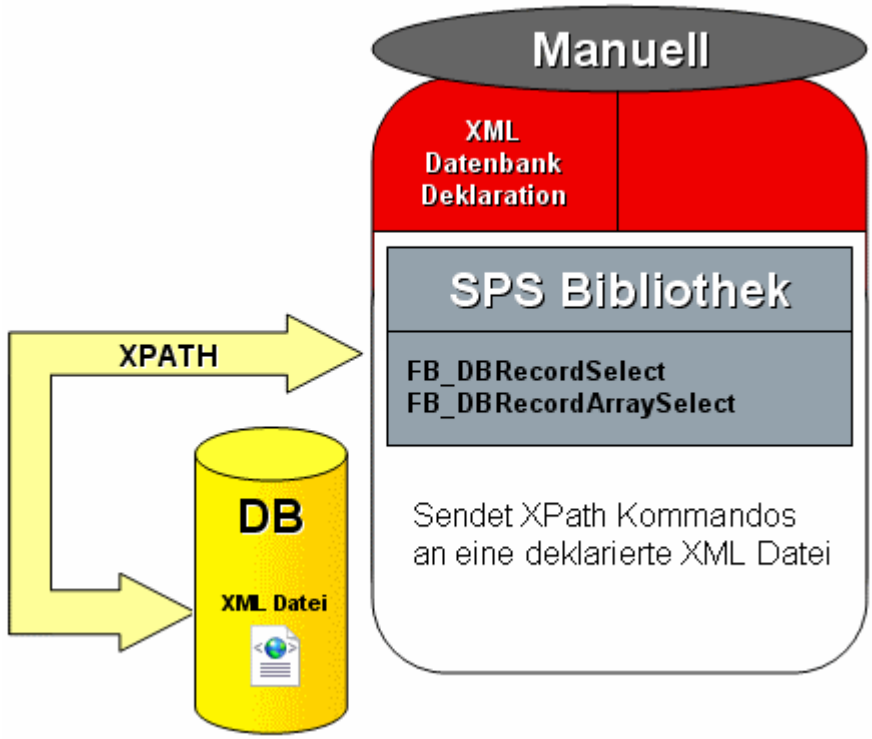
XMLTestDB.xsd (XML Schema)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:simpleType name="bigint">
    <xsd:restriction base="xsd:long" />
  </xsd:simpleType>
  <xsd:simpleType name="datetime">
    <xsd:restriction base="xsd:dateTime" />
  </xsd:simpleType>
  <xsd:simpleType name="ntext_80">
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="80" />
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="float">
    <xsd:restriction base="xsd:double" />
  </xsd:simpleType>
  <xsd:complexType name="myTable_Type">
    <xsd:sequence>
      <xsd:element minOccurs="0" maxOccurs="unbounded" name="row">
        <xsd:complexType>
          <xsd:attribute name="ID" type="bigint" />
          <xsd:attribute name="Timestamp" type="datetime" />
          <xsd:attribute name="Name" type="ntext_80" />
          <xsd:attribute name="Value" type="float" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="XMLTestDB">
    <xsd:complexType>
      <xsd:sequence minOccurs="1" maxOccurs="1">
        <xsd:element name="myTable" type="myTable_Type" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

8.9 XML Datenbanktyp - XPath Beispiel für TwinCAT PLC Visualisierung

Mit Hilfe des Funktionsbausteins FB_DBRecordSelect können XPath Kommandos abgesetzt werden und XML-Tags aus einer beliebigen XML Datei gelesen werden. In diesem Beispiel wird demonstriert, wie man mit Hilfe des TwinCAT Database Servers, Einträge aus den dynamischen Textlisten (XML-Datei) für die TwinCAT PLC Visualisierung lesen kann.

Download "Beispiel mit XPath und TargetVisu" https://infosys.beckhoff.com/content/1031/tcdbserver/Resources/11407911819.zip



Verwendeter Datenbanktyp	XML
Kompatible Datenbanktypen	XML
Verwendete Funktionsbausteine	FB_DBRecordSelect
Einzubindende Bibliotheken	"**TcDatabase.lib**", "TcSystem.lib", "TcBase.lib", "STANDARD.lib", "TcUtilities.lib"
Download Dateiliste	XPath_GetText.pro, CurrentConfigDatabase.xml, VisuTest.xml

Dynamische Textliste für TwinCAT PLC Visualisierung

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<dynamic-text>
  <header>
   <default-language>deutsch</default-language>
    <default-font>
      <language>deutsch</language>
      <font-name>Arial </font-name>
      <font-color>0,0,0</font-color>
      <font-height>-13</font-height>
      <font-weight>700</font-weight>
      <font-italic>false</font-italic>
      <font-underline>false</font-underline>
     <font-strike-out>false</font-strike-out>
      <font-char-set>0</font-char-set>
    </default-font>
    <default-font>
      <language>english</language>
      <font-name>Arial </font-name>
      <font-color>0,0,0</font-color>
```

```
     <font-height>-13</font-height>
      <font-weight>700</font-weight>
      <font-italic>false</font-italic>
      <font-underline>false</font-underline>
     <font-strike-out>false</font-strike-out>
      <font-char-set>0</font-char-set>
    </default-font>
     <default-font>
      <language>francais</language>
      <font-name>Arial </font-name>
      <font-color>0,0,0</font-color>
     <font-height>-13</font-height>
      <font-weight>700</font-weight>
      <font-italic>false</font-italic>
      <font-underline>false</font-underline>
     <font-strike-out>false</font-strike-out>
      <font-char-set>0</font-char-set>
    </default-font>
  </header>
  <text-list>
   <text prefix="A" id="1">
      <deutsch>Datei öffnen...</deutsch>
     <english>File open...</english>
      <francais>Fichier ouvrir...</francais>
    </text>
    <text prefix="B" id="2">
     <deutsch>Datei schließen</deutsch>
      <english>File close...</english>
     <francais>Fermer le fichier...</francais>
    </text>
    <text prefix="C" id="3">
     <deutsch>Deutschland</deutsch>
      <english>England</english>
     <francais>France</francais>
    </text>
  </text-list>
</dynamic-text>
```

Functionsbaustein "FB_GetText" (zum auslesen der XML-Tags)

```
FUNCTION_BLOCK FB_GetText
VAR_INPUT
    dwID: DWORD;
    stPrefix: T_MaxString;
    stLanguage : T_MaxString;
    bExecute : BOOL;
END_VAR
VAR_OUTPUT
    bBusy : BOOL;
    bError : BOOL;
    nResultLength : INT;
    stResult : STRING(256);
END_VAR
VAR
    R_TRIG1: R_TRIG;
    state: BYTE;
    FB_DBRecordSelect1: FB_DBRecordSelect;
    FB_FormatString1: FB_FormatString;
END_VAR
```

```
R_TRIG1(CLK:=bExecute);
IF R_TRIG1.Q THEN
    state := 1;
    bBusy := TRUE;
    bError := FALSE;
    FB_DBRecordSelect1(bExecute:=FALSE);
END_IF
CASE
state OF
    0:
        ;
    1:
        FB_FormatString1(
            sFormat:= 'XPATH<TAG>#/dynamic-text/text-list/text[@prefix=$'%s$' and @id=%d]/%s',
            arg1:= F_STRING(stPrefix),
            arg2:= F_DWORD(dwID),
            arg3:= F_STRING(stLanguage),
            sOut=> FB_DBRecordSelect1.sSelectCmd);

        FB_DBRecordSelect1(
```

```
            sNetID:= ,
            hDBID:= 1,
            nRecordIndex:= 0,
            cbRecordSize:= SIZEOF(stResult),
            pDestAddr:= ADR(stResult),
            bExecute:= TRUE,
            tTimeout:= T#10s);

        IF NOT FB_DBRecordSelect1.bBusy THEN
            IF NOT FB_DBRecordSelect1.bError THEN
                nResultLength := LEN(stResult);
            ELSE
                bError := TRUE;
            END_IF
            bBusy := FALSE;
            state := 0;
        END_IF
END_CASE
```

MAIN Programm

```
PROGRAM MAIN
VAR
    FB_GetText1: FB_GetText;
    startstop: BOOL;
    busy: BOOL;
    err: BOOL;
    resultLen: INT;
    result: STRING(256);
END_VAR
```

```
FB_GetText1(
    dwID:= 1,
    stPrefix:= 'A',
    stLanguage:= 'deutsch',
    bExecute:= startstop,
    bBusy=> busy,
    bError=> err,
    nResultLength=> resultLen,
    stResult=> result);
```

Mit einer Positiven Flanke am Eingang bExecute des Funktionsbaustein FB_GetText wird das XPath Kommando abgesetzt. Der ausgelesene Text wird im Ausgang stResult zurückgeliefert. Zusätzlich wird die Länge des Textes am Ausgang nResultLength angegeben.

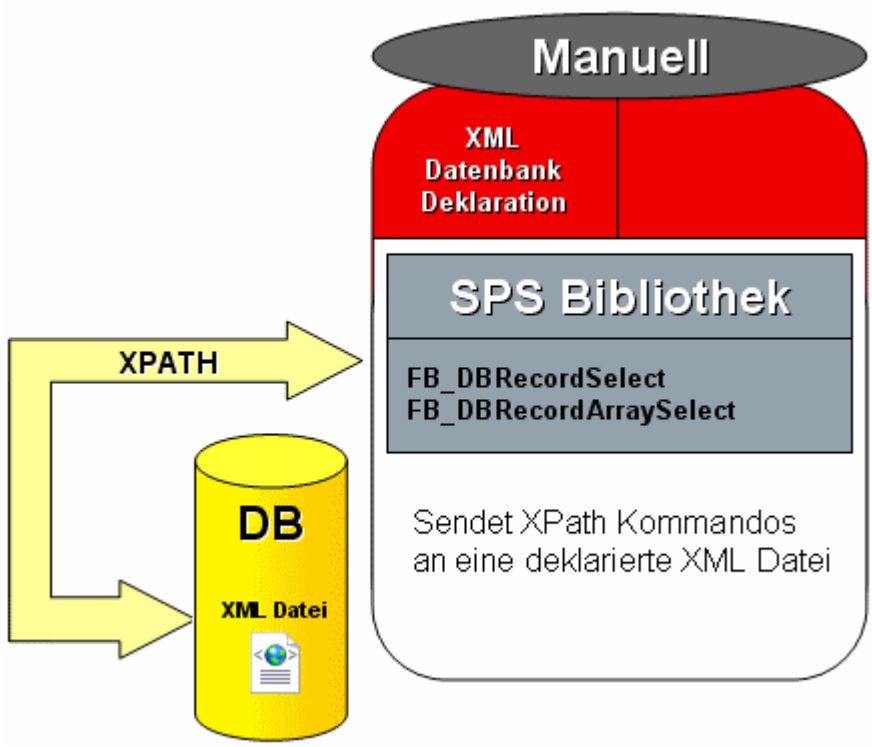
8.10 XML Datenbanktyp - XPath Beispiel mit XML-Schema

Mit Hilfe des Funktionsbausteins FB_DBRecordSelect oder FB_DBRecordArraySelect können XPath Kommandos abgesetzt werden und XML-Tags, XML-Subtags oder XML-Attribute aus einer beliebigen XML Datei gelesen werden. Ist ein passendes XML-Schema für die zu lesende XML-Datei vorhanden, werden die Inhalte der Tags bzw. Attribute in die entsprechenden Datentypen konvertiert, so wie sie in dem Schema definiert sind.

Nähere Informationen zu XML-Schemas finden Sie hier: http://www.edition-w3.de/TR/2001/REC-xmlschema-0-20010502/

In diesem Beispiel werden mit dem FB_DBRecordArraySelect zwei unterschiedliche Subtags aus einer XML-Datei mit zugehörigem XML-Schema ausgelesen.

Download "Beispiel mit XPath und XMLSchema" https://infosys.beckhoff.com/content/1031/tcdbserver/Resources/11407913227.zip



Verwendeter Datenbanktyp	XML
Kompatible Datenbanktypen	XML
Verwendete Funktionsbausteine	FB_DBRecordSelect
Einzubindende Bibliotheken	**"TcDatabase.lib"**, "TcSystem.lib", "TcBase.lib", "STANDARD.lib", "TcUtilities.lib"
Download Dateiliste	XPath_XMLSubTag.pro, CurrentConfigDatabase.xml, PLC_Structs.xml, PLC_Structs.xsd

Beispiel XML-Datei (PLC_Structs.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<Beckhoff_PLC>
  <PLC_Structs>
    <PLC_Struct Name="ST_TestStruct">
      <Struct Instance="1">
        <nINT64>123456789</nINT64>
        <nUINT16>1234</nUINT16>
        <rREAL64>1234.5678</rREAL64>
        <sSTRING>This is instance one of ST_TestStruct</sSTRING>
        <bBOOL>true</bBOOL>
        <nINT32>-100</nINT32>
      </Struct>
      <Struct Instance="2">
        <nINT64>234567890</nINT64>
        <nUINT16>2345</nUINT16>
        <rREAL64>234.56789</rREAL64>
        <sSTRING>This is instance two of ST_TestStruct</sSTRING>
        <bBOOL>false</bBOOL>
        <nINT32>-50</nINT32>
      </Struct>
      <Struct Instance="3">
        <nINT64>345678901</nINT64>
        <nUINT16>3456</nUINT16>
        <rREAL64>3456.78901</rREAL64>
        <sSTRING>This is instance three of ST_TestStruct</sSTRING>
        <bBOOL>true</bBOOL>
        <nINT32>-150</nINT32>
      </Struct>
    </PLC_Struct>
    <PLC_Struct Name="ST_TestStruct2">
      <Struct2 Instance="1">
        <sSTRING>This is instance one of ST_TestStruct2</sSTRING>
        <bBOOL>false</bBOOL>
        <nINT32>-88</nINT32>
```

```
      </Struct2>
      <Struct2 Instance="2">
        <sSTRING>This is instance two of ST_TestStruct2</sSTRING>
        <bBOOL>true</bBOOL>
        <nINT32>-9</nINT32>
      </Struct2>
    </PLC_Struct>
  </PLC_Structs>
</Beckhoff_PLC>
```

Zugehöriges XML-Schema (PLC_Structs.xsd)

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://
www.w3.org/2001/XMLSchema">
  <xs:element name="Beckhoff_PLC">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="PLC_Structs">
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" name="PLC_Struct">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element minOccurs="0" maxOccurs="unbounded" name="Struct">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="nINT64" type="xs:long" />
                          <xs:element name="nUINT16" type="xs:unsignedShort" />
                          <xs:element name="rREAL64" type="xs:double" />
                          <xs:element name="sSTRING" type="xs:string" />
                          <xs:element name="bBOOL" type="xs:boolean" />
                          <xs:element name="nINT32" type="xs:int" />
                        </xs:sequence>
                        <xs:attribute name="Instance" type="xs:unsignedByte" use="required" />
                      </xs:complexType>
                    </xs:element>
                    <xs:element minOccurs="0" maxOccurs="unbounded" name="Struct2">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="sSTRING" type="xs:string" />
                          <xs:element name="bBOOL" type="xs:boolean" />
                          <xs:element name="nINT32" type="xs:int" />
                        </xs:sequence>
                        <xs:attribute name="Instance" type="xs:unsignedByte" use="required" />
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="Name" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Struktur1 ST_TestStruct

```
TYPE ST_TestStruct :
STRUCT
    nINT64 : T_LARGE_INTEGER;
    nUINT16 : UINT;
    rREAL64 : LREAL;
    sSTRING : T_MaxString;
    bBOOL : BOOL;
    nINT32 : DINT;
END_STRUCT
END_TYPE
```

Struktur2 ST_TestStruct2

```
TYPE ST_TestStruct2 :
STRUCT
    sSTRING : T_MaxString;
    bBOOL : BOOL;
```

```
    nINT32 : DINT;
END_STRUCT
END_TYPE
```

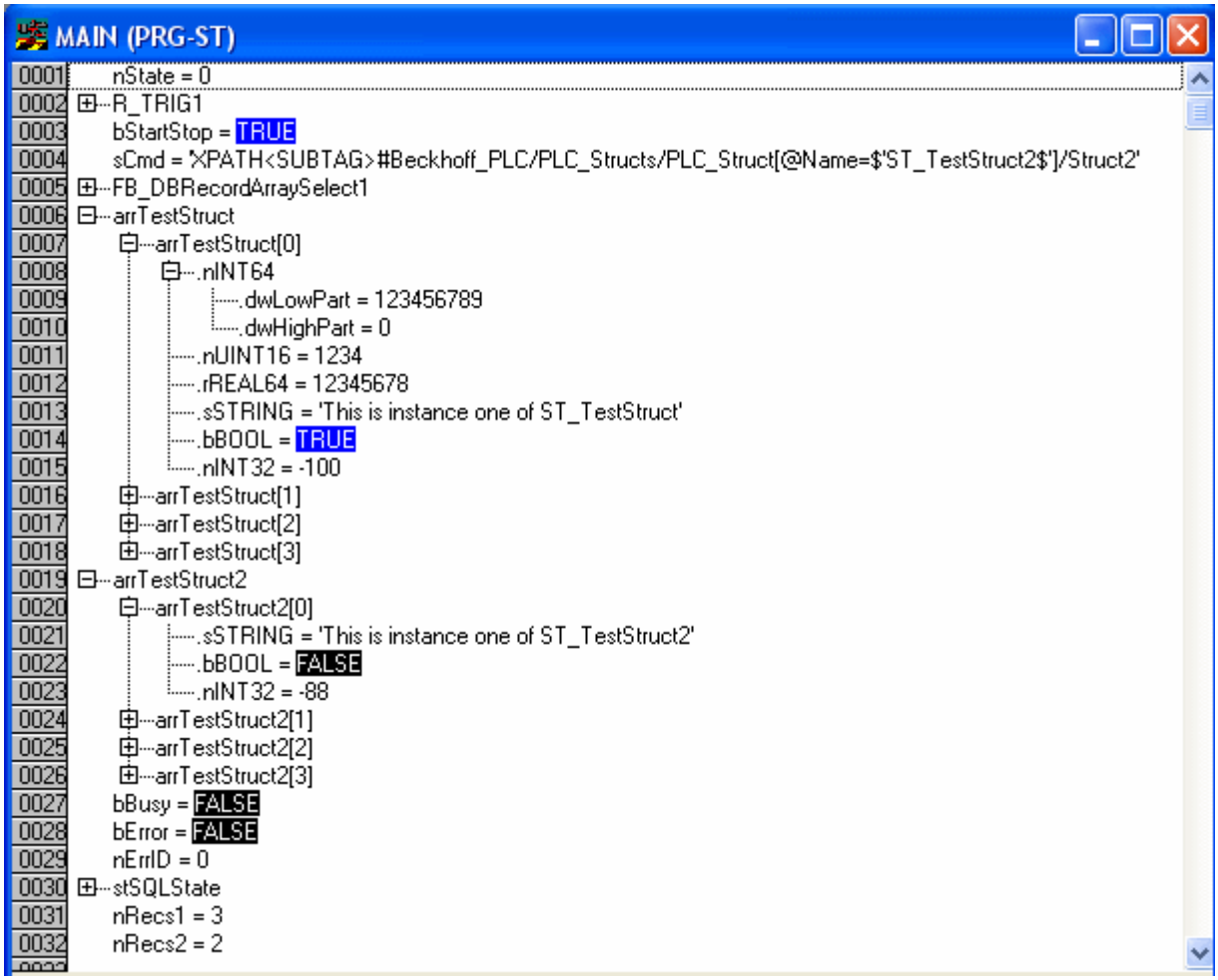
MAIN Programm

```
PROGRAM MAIN
VAR
    nState: BYTE;

    R_TRIG1: R_TRIG;
    bStartStop: BOOL;

    sCmd: T_MaxString;

    FB_DBRecordArraySelect1: FB_DBRecordArraySelect;
    arrTestStruct: ARRAY [0..3] OF ST_TestStruct;
    arrTestStruct2: ARRAY [0..3] OF ST_TestStruct2;

    bBusy: BOOL;
    bError: BOOL;
    nErrID: UDINT;
    stSQLState: ST_DBSQLError;

    nRecs1: UDINT;
    nRecs2: UDINT;
END_VAR
```

```
R_TRIG1(CLK:=bStartStop);
IF R_TRIG1.Q THEN
    FB_DBRecordArraySelect1(bExecute:=FALSE);
    nState := 1;
END_IF
CASE nState OF
    0:(*Idle*)
        ;
    1:
        sCmd := 'XPATH<SUBTAG>#/Beckhoff_PLC/PLC_Structs/PLC_Struct[@Name=$'ST_TestStruct$']/
Struct';

        FB_DBRecordArraySelect1(
            sNetID:= ,
            hDBID:= 1,
            cbCmdSize:= SIZEOF(sCmd),
            pCmdAddr:= ADR(sCmd),
            nStartIndex:= 0,
            nRecordCount:= 4,
            cbRecordArraySize:= SIZEOF(arrTestStruct),
            pDestAddr:= ADR(arrTestStruct),
            bExecute:= TRUE,
            tTimeout:= T#15s,
            bBusy=> bBusy,
            bError=> bError,
            nErrID=> nErrID,
            sSQLState=> stSQLState,
            nRecords=> nRecs1);

        IF NOT bBusy THEN
            FB_DBRecordArraySelect1(bExecute:=FALSE);
            IF NOT bError THEN
                nState := 2;
            ELSE
                nState := 255;
            END_IF
        END_IF
    2:
        sCmd := 'XPATH<SUBTAG>#Beckhoff_PLC/PLC_Structs/PLC_Struct[@Name=$'ST_TestStruct2$']/
Struct2';

        FB_DBRecordArraySelect1(
            sNetID:= ,
            hDBID:= 1,
            cbCmdSize:= SIZEOF(sCmd),
            pCmdAddr:= ADR(sCmd),
            nStartIndex:= 0,
            nRecordCount:= 4,
            cbRecordArraySize:= SIZEOF(arrTestStruct2),
            pDestAddr:= ADR(arrTestStruct2),
            bExecute:= TRUE,
            tTimeout:= T#15s,
```

```
            bBusy=> bBusy,
            bError=> bError,
            nErrID=> nErrID,
            sSQLState=> stSQLState,
            nRecords=> nRecs2);

        IF NOT bBusy THEN
            FB_DBRecordArraySelect1(bExecute:=FALSE);
            IF NOT bError THEN
                nState := 0;
            ELSE
                nState := 255;
            END_IF
        END_IF
    255: (* Error Step*)
        ;
END_CASE
```

Mit einer Positiven Flanke an der Toggle Variable bStartStop wird das Auslesen gestartet.



9 Anhang

9.1 Fehlercodes

9.1.1 ADS Return Codes

Error codes: 0x000... [▶ 135], 0x500... [▶ 135], 0x700... [▶ 135], 0x1000... [▶ 135], 0x274C... [▶ 135]

Global Error Codes

Hex	Dec	Description	Possible Causes	Solution
0x0	0	no error		
0x1	1	Internal error		
0x2	2	No Rtime		
0x3	3	Allocation locked memory error		
0x4	4	Insert mailbox error	No ADS mailbox was available to process this message.	Reduce the number of ADS calls (e.g ADS-Sum commands or Max Delay Parameter)
0x5	5	Wrong receive HMSG		
0x6	6	target port not found	ADS Server not started	
0x7	7	target machine not found	Missing ADS routes	
0x8	8	Unknown command ID		
0x9	9	Bad task ID		
0xA	10	No IO		
0xB	11	Unknown ADS command		
0xC	12	Win 32 error		
0xD	13	Port not connected		
0xE	14	Invalid ADS length		
0xF	15	Invalid AMS Net ID		
0x10	16	Low Installation level		
0x11	17	No debug available		
0x12	18	Port disabled		
0x13	19	Port already connected		
0x14	20	ADS Sync Win32 error		
0x15	21	ADS Sync Timeout		
0x16	22	ADS Sync AMS error		
0x17	23	ADS Sync no index map		
0x18	24	Invalid ADS port		
0x19	25	No memory		
0x1A	26	TCP send error		
0x1B	27	Host unreachable		
0x1C	28	Invalid AMS fragment		

Router Error Codes

Hex	Dec	Description	Possible Causes	Solution
0x500	1280	ROUTERERR_NOLOCKEDMEMORY	No locked memory can be allocated	

Hex	Dec	Description	Possible Causes	Solution
0x501	1281	ROUTERERR_RESIZEMEMORY	The size of the router memory could not be changed	
0x502	1282	ROUTERERR_MAILBOXFULL	The mailbox has reached the maximum number of possible messages. The current sent message was rejected	Check the connection between the communication partners
0x503	1283	ROUTERERR_DEBUGBOXFULL	The mailbox has reached the maximum number of possible messages. The sent message will not be displayed in the debug monitor	Check the connection to the debug monitor
0x504	1284	ROUTERERR_UNKNOWNPORTTYPE	The port type is unknown	
0x505	1285	ROUTERERR_NOTINITIALIZED	Router is not initialised	
0x506	1286	ROUTERERR_PORTALREADYINUSE	The desired port number is already assigned	
0x507	1287	ROUTERERR_NOTREGISTERED	Port not registered	
0x508	1288	ROUTERERR_NOMOREQUEUES	The maximum number of Ports reached	
0x509	1289	ROUTERERR_INVALIDPORT	The port is invalid.	
0x50A	1290	ROUTERERR_NOTACTIVATED	TwinCAT Router not active	
0x50B	1291	ROUTERERR_FRAGMENTBOXFULL		
0x50C	1292	ROUTERERR_FRAGMENTTIMEOUT		
0x50D	1293	ROUTERERR_TOBEREMOVED		

General ADS Error Codes

Hex	Dec	Description	Possible Causes	Solution
0x700	1792	error class <device error>		
0x701	1793	Service is not supported by server		
0x702	1794	invalid index group		
0x703	1795	invalid index offset		
0x704	1796	reading/writing not permitted		
0x705	1797	parameter size not correct		
0x706	1798	invalid parameter value(s)		
0x707	1799	device is not in a ready state		
0x708	1800	device is busy		
0x709	1801	invalid context (must be in Windows)		
0x70A	1802	out of memory		
0x70B	1803	invalid parameter value(s)		
0x70C	1804	not found (files, ...)		
0x70D	1805	syntax error in command or file		
0x70E	1806	objects do not match		
0x70F	1807	object already exists		

Hex	Dec	Description	Possible Causes	Solution
0x710	1808	symbol not found		
0x711	1809	symbol version invalid	Onlinechange	Release handle and get a new one
0x712	1810	server is in invalid state		
0x713	1811	AdsTransMode not supported		
0x714	1812	Notification handle is invalid	Onlinechange	Release handle and get a new one
0x715	1813	Notification client not registered		
0x716	1814	no more notification handles		
0x717	1815	size for watch too big		
0x718	1816	device not initialized		
0x719	1817	device has a timeout		
0x71A	1818	query interface failed		
0x71B	1819	wrong interface required		
0x71C	1820	class ID is invalid		
0x71D	1821	object ID is invalid		
0x71E	1822	request is pending		
0x71F	1823	request is aborted		
0x720	1824	signal warning		
0x721	1825	invalid array index		
0x722	1826	symbol not active	Onlinechange	Release handle and get a new one
0x723	1827	access denied		
0x724	1828	missing license		Activate license for TwinCAT 3 function
0x72c	1836	exception occured during system start		Check each device transistions
0x740	1856	Error class <client error>		
0x741	1857	invalid parameter at service		
0x742	1858	polling list is empty		
0x743	1859	var connection already in use		
0x744	1860	invoke ID in use		
0x745	1861	timeout elapsed		Check ADS routes of sender and receiver and your firewall setting
0x746	1862	error in win32 subsystem		
0x747	1863	Invalid client timeout value		
0x748	1864	ads-port not opened		
0x750	1872	internal error in ads sync		
0x751	1873	hash table overflow		
0x752	1874	key not found in hash		
0x753	1875	no more symbols in cache		
0x754	1876	invalid response received		
0x755	1877	sync port is locked		

RTime Error Codes

Hex	Dec	Description	Possible Causes
0x1000	4096	RTERR_INTERNAL	Internal fatal error in the TwinCAT real-time system

Hex	Dec	Description	Possible Causes
0x1001	4097	RTERR_BADTIMERPERIODS	Timer value not vaild
0x1002	4098	RTERR_INVALIDTASKPTR	Task pointer has the invalid value ZERO
0x1003	4099	RTERR_INVALIDSTACKPTR	Task stack pointer has the invalid value ZERO
0x1004	4100	RTERR_PRIOEXISTS	The demand task priority is already assigned
0x1005	4101	RTERR_NOMORETCB	No more free TCB (Task Control Block) available. Maximum number of TCBs is 64
0x1006	4102	RTERR_NOMORESEMAS	No more free semaphores available. Maximum number of semaphores is 64
0x1007	4103	RTERR_NOMOREQUEUES	No more free queue available. Maximum number of queue is 64
0x1008	4104	TwinCAT reserved.	
0x1009	4105	TwinCAT reserved.	
0x100A	4106	TwinCAT reserved.	
0x100B	4107	TwinCAT reserved.	
0x100C	4108	TwinCAT reserved.	
0x100D	4109	RTERR_EXTIRQALREADYDEF	An external synchronisation interrupt is already applied
0x100E	4110	RTERR_EXTIRQNOTDEF	No external synchronsiation interrupt applied
0x100F	4111	RTERR_EXTIRQINSTALLFAILED	The apply of the external synchronisation interrupt failed
0x1010	4112	RTERR_IRQLNOTLESSOREQUAL	Call of a service function in the wrong context
0x1017	4119	RTERR_VMXNOTSUPPORTED	Intel VT-x extension is not supported.
0x1018	4120	RTERR_VMXDISABLED	Intel VT-x extension is not enabled in BIOS.
0x1019	4121	RTERR_VMXCONTROLSMISSING	Missing feature in Intel VT-x extension.
0x101A	4122	RTERR_VMXENABLEFAILS	Enabling Intel VT-x fails.

TCP Winsock Error Codes

Hex	Dec	Description	Possible Causes	Solution
0x274c	10060	A socket operation was attempted to an unreachable host	Host unreachable	Check network connection via ping
0x274d	10061	A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond.	Host unreachable	Check network connection via ping
0x2751	10065	No connection could be made because the target machine actively refused it		
		Further Winsock error codes: Win32 Error Codes		

9.1.2 Interne Fehlercodes des TwinCAT Database Servers

Code (Hex)	Code (Dez)	Beschreibung
0x0001 + ADS-FehlerCode	65537 - 131071	ADS Fehlercode vom Deklarierten ADS-Gerät
0x00020001	131073	Microsoft SQL Compact Datenbank (Fehlercode)

Code (Hex)	Code (Dez)	Beschreibung
0x00040001	262145	Microsoft SQL Datenbank (Fehlercode)
0x00080001	524289	Microsoft Access Datenbank (Fehlercode)
0x00100001	1048577	MySQL Datenbank (Fehlercode)
0x00200001	2097153	Oracle Datenbank (Fehlercode)
0x00400001	4194305	DB2 Datenbank (Fehlercode)
0x00800001	8388609	PostgreSQL Datenbank (Fehlercode)
0x01000001	16777217	Interbase/Firebird Datenbank (Fehlercode)
0x02000001	33554433	TwinCAT Database Server Fehlercode
0x04000001	67108865	XML Datenbank (Fehlercode)
0x08000001	134217729	ASCII Datenbank (Fehlercode)

Wurde ein von den oben genannten Fehlercodes im "nErrID" Ausgang eines Funktionsbausteins ausgegeben, ist ein Fehler beim ausführen eines SQL-Statements aufgetreten. Der Fehlercode wurde dann am "sSQLState" Ausgang des Funktionsbausteins ausgegeben. Der "sSQLState" Ausgang ist von Datentyp ST_DBSQLError [▶ 92]. Für jeden Datenbanktyp werden individuelle Fehlercodes ausgegeben.

Eine Liste der SQLStates finden Sie unter: http://msdn.microsoft.com/en-us/library/ms714687(VS.85).aspx (SQLStates)

Datenbanktyp	Fehlercodereferenz
Microsoft SQL Compact Datenbank	http://technet.microsoft.com/en-us/library/ms171788.aspx / TcDBServer_OleDB_Errorcodes.htm [▶ 139]
Microsoft SQL Datenbank	TcDBServer_OleDB_Errorcodes.htm [▶ 139]
Microsoft Access Datenbank	TcDBServer_OleDB_Errorcodes.htm [▶ 139]
MySQL Datenbank	dev.mysql.com/doc/refman/8.0/en/error-handling.html
Oracle Datenbank	http://www.ora-code.com
DB2 Datenbank	https://www.ibm.com/docs/en/db2-for-zos/12?topic=codes-sql
PostgreSQL Datenbank	http://www.postgresql.org/docs/current/static/errcodes-appendix.html
Interbase/Firebird Datenbank	http://www.firebirdsql.org/file/documentation/reference_manuals/reference_material/Firebird-2.1-ErrorCodes.pdf
XML Datenbank	TcDBServer_XML_Errorcodes.htm [▶ 144]
ASCII Datenbank	TcDBServer_ASCII_Errorcodes.htm [▶ 144]

9.1.3 OleDB Fehlercodes

Wert	Beschreibung
0x80040E00	Der Accessor ist ungültig.
0x80040E01	Es konnte keine Zeile in das Rowset eingefügt werden, da andernfalls die maximale Anzahl von aktiven Zeilen des Anbieters überschritten wird.
0x80040E02	Der Accessor ist schreibgeschützt. Der Vorgang ist fehlgeschlagen.
0x80040E03	Werte verletzen das Datenbankschema.
0x80040E04	Das Zeilenhandle ist ungültig.
0x80040E05	Das Objekt war geöffnet.
0x80040E06	Ungültiges Kapitel.
0x80040E07	Ein Literalwert im Befehl konnte wegen einer anderen Ursache als Datenüberlauf nicht in den richtigen Typ konvertiert werden.
0x80040E08	Ungültige Bindungsinformationen.
0x80040E09	Berechtigung verweigert.
0x80040E0A	Die angegebene Spalte enthält keine Lesezeichen oder Kapitel.

Wert	Beschreibung
0x80040E0B	Einige Kostenbeschränkungen wurden zurückgewiesen.
0x80040E0C	Für das Befehlsobjekt wurde kein Befehl festgelegt.
0x80040E0D	Innerhalb der angegebenen Kostenbeschränkung wurde kein Abfrageplan gefunden.
0x80040E0E	Ungültiges Lesezeichen.
0x80040E0F	Ungültiger Sperrmodus.
0x80040E10	Für mindestens einen erforderlichen Parameter wurde kein Wert angegeben.
0x80040E11	Ungültige Spalten-ID.
0x80040E12	Ungültige Quote.
0x80040E13	Ungültiger Wert.
0x80040E14	Der Befehl enthielt mindestens einen Fehler.
0x80040E15	Der aktuell ausgeführte Befehl kann nicht abgebrochen werden.
0x80040E16	Der Anbieter bietet keine Unterstützung für den angegebenen Dialekt.
0x80040E17	Es ist bereits eine Datenquelle mit dem angegebenen Namen vorhanden.
0x80040E18	Das Rowset wurde über einen Livedatenstrom erstellt und kann nicht neu gestartet werden.
0x80040E19	Im aktuellen Bereich stimmt kein Schlüssel mit den beschriebenen Merkmalen überein.
0x80040E1B	Der Anbieter kann die Identität für die neu eingefügten Zeilen nicht bestimmen.
0x80040E1A	Der Besitz dieser Struktur wurde an den Anbieter übergeben.
0x80040E1C	Es werden keine Gewichtungswerte ungleich Null als Angabe für Ziele unterstützt. Daher wurde das Ziel zurückgewiesen. Das aktuelle Ziel wurde nicht geändert.
0x80040E1D	Die angeforderte Konvertierung wird nicht unterstützt.
0x80040E1E	RowsOffset führt unabhängig vom angegebenen cRows-Wert zu einer Position hinter dem Ende des Rowsets. cRowsObtained ist 0.
0x80040E20	Der Anbieter hat eine Methode von IRowsetNotify im Consumer aufgerufen und noch keine Rückgabe von der Methode erhalten.
0x80040E21	Fehler
0x80040E22	Ein steuerndes IUnknown-Objekt ungleich NULL wurde angegeben, und das aktuell erstellte Objekt unterstützt keine Aggregation.
0x80040E23	Die aktuelle Zeile wurde gelöscht.
0x80040E24	Das Rowset bietet keine Unterstützung für Rückwärtsabrufvorgänge.
0x80040E25	Alle HROW-Objekte müssen freigegeben werden, bevor neue HROW-Objekte empfangen werden können.
0x80040E26	Ein angegebenes Speicherflag wurde nicht unterstützt.
0x80040E27	Der Vergleichsoperator war ungültig.
0x80040E28	Das angegebene Statusflag war weder DBCOLUMNSTATUS_OK noch DBCOLUMNSTATUS_ISNULL.
0x80040E29	Das Rowset kann nicht rückwärts durchlaufen werden.
0x80040E2A	Ungültiges Bereichshandle.
0x80040E2B	Der angegebene Zeilensatz grenzte nicht an die Zeilen des angegebenen Überwachungsbereichs und überlappte auch nicht.
0x80040E2C	Es wurde ein Übergang von ALL* zu MOVE* oder EXTEND* angegeben.
0x80040E2D	Der angegebene Bereich ist kein gültiger Unterbereich des vom angegebenen Überwachungsbereichshandle identifizierten Bereichs.
0x80040E2E	Der Anbieter bietet keine Unterstützung für Befehle mit mehreren Anweisungen.
0x80040E2F	Ein angegebener Wert verletzte die Integritätseinschränkungen für eine Spalte oder Tabelle.
0x80040E30	Der angegebene Typname wurde nicht erkannt.
0x80040E31	Die Ausführung wurde abgebrochen, da keine weiteren Ressourcen verfügbar waren. Es wurden keine Ergebnisse zurückgegeben.

Wert	Beschreibung
0x80040E32	Ein Command-Objekt, dessen Befehlshierarchie mindestens ein Rowset enthält, kann nicht geklont werden.
0x80040E33	Die aktuelle Struktur kann nicht als Text dargestellt werden.
0x80040E34	Der angegebene Index ist bereits vorhanden.
0x80040E35	Der angegebene Index ist nicht vorhanden.
0x80040E36	Der angegebene Index wurde verwendet.
0x80040E37	Die angegebene Tabelle ist nicht vorhanden.
0x80040E38	Das Rowset hat vollständige Parallelität verwendet, und der Wert einer Spalte wurde seit dem letzten Lesevorgang geändert.
0x80040E39	Während des Kopierens wurden Fehler gefunden.
0x80040E3A	Eine Genauigkeitsangabe war ungültig.
0x80040E3B	Eine angegebene Dezimalstellenanzahl war ungültig.
0x80040E3C	Ungültige Tabellen-ID.
0x80040E3D	Ein angegebener Typ war ungültig.
0x80040E3E	Eine Spalten-ID ist mehrmals in der Spezifikation aufgetreten.
0x80040E3F	Die angegebene Tabelle ist bereits vorhanden.
0x80040E40	Die angegebene Tabelle wurde verwendet.
0x80040E41	Die angegebene Gebietsschema-ID wurde nicht unterstützt.
0x80040E42	Die angegebene Datensatznummer ist ungültig.
0x80040E43	Obwohl das Lesezeichen gültig formatiert war, konnte keine übereinstimmende Zeile gefunden werden.
0x80040E44	Der Wert einer Eigenschaft war ungültig.
0x80040E45	Das Rowset war nicht in Kapitel unterteilt.
0x80040E46	Ungültiger Accessor.
0x80040E47	Ungültige Speicherflags.
0x80040E48	Accessoren zum Übergeben als Verweis werden von diesem Anbieter nicht unterstützt.
0x80040E49	NULL-Accessoren werden von diesem Anbieter nicht unterstützt.
0x80040E4A	Der Befehl wurde nicht vorbereitet.
0x80040E4B	Der angegebene Accessor war kein Parameteraccessor.
0x80040E4C	Der angegebene Accessor war schreibgeschützt.
0x80040E4D	Fehler bei der Authentifizierung.
0x80040E4E	Die Änderung wurde während der Benachrichtigung abgebrochen; es wurden keine Spalten geändert.
0x80040E4F	Das Rowset bestand aus einem Kapitel, und das Kapitel war nicht freigegeben.
0x80040E50	Ungültiges Quellhandle.
0x80040E51	Der Anbieter kann keine Parameterinformationen ableiten, und SetParameterInfo wurde nicht aufgerufen.
0x80040E52	Das Datenquellobjekt ist bereits initialisiert.
0x80040E53	Der Anbieter bietet keine Unterstützung für diese Methode.
0x80040E54	Die Anzahl von Zeilen mit ausstehenden Änderungen überschreitet das festgelegte Limit.
0x80040E55	Die angegebene Spalte war nicht vorhanden.
0x80040E56	In einer Zeile mit einem Verweiszähler von Null stehen Änderungen aus.
0x80040E57	Ein Literalwert im Befehl führte zum Bereichsüberlauf für den Typ der zugeordneten Spalte.
0x80040E58	Der übergebene HRESULT-Wert war ungültig.
0x80040E59	Der übergebene LookupID-Wert war ungültig.
0x80040E5A	Der übergebene DynamicErrorID-Wert war ungültig.

Wert	Beschreibung
0x80040E5B	Es können keine sichtbaren Daten für eine neu eingefügte Zeile abgerufen werden, die noch nicht aktualisiert wurde.
0x80040E5C	Ungültiges Konvertierungsflag.
0x80040E5D	Der angegebene Parametername wurde nicht erkannt.
0x80040E5E	Es dürfen nicht mehrere Speicherobjekte gleichzeitig geöffnet sein.
0x80040E5F	Der angeforderte Filter konnte nicht geöffnet werden.
0x80040E60	Die angeforderte Reihenfolge konnte nicht geöffnet werden.
0x80040E65	Der übergebene columnID-Wert war ungültig.
0x80040E67	Der übergebene Befehl hat keinen DBID-Wert.
0x80040E68	Der übergebene DBID-Wert ist bereits vorhanden.
0x80040E69	Die maximale Anzahl der von diesem Anbieter unterstützten Session-Objekte ist bereits erreicht. Der Consumer muss mindestens ein aktuelles Session-Objekt freigeben, bevor ein neues Session-Objekt abgerufen werden kann.
0x80040E72	Die Index-ID ist ungültig.
0x80040E73	Die angegebene Initialisierungszeichenfolge entspricht nicht der Spezifikation.
0x80040E74	Die OLE DB-Stammenumerator hat keine Anbieter zurückgegeben, die mit einem angeforderten SOURCES_TYPE-Wert übereinstimmen.
0x80040E75	Die Initialisierungszeichenfolge gibt einen Anbieter an, der nicht mit dem aktuell aktiven Anbieter übereinstimmt.
0x80040E76	Der angegebene DBID-Wert ist ungültig.
0x80040E6A	Ungültiger Wert für Vertrauensnehmer.
0x80040E6B	Der Vertrauensnehmer ist nicht für die aktuelle Datenquelle bestimmt.
0x80040E6C	Der Vertrauensnehmer bietet keine Unterstützung für Mitgliedschaften/Auflistungen.
0x80040E6D	Das Objekt ist ungültig oder dem Anbieter unbekannt.
0x80040E6E	Das Objekt hat keinen Besitzer.
0x80040E6F	Die übergebene Zugriffseintragsliste ist ungültig.
0x80040E70	Der als Besitzer übergebene Vertrauensnehmer ist ungültig oder dem Anbieter unbekannt.
0x80040E71	Die in der Zugriffseintragsliste übergebene Berechtigung ist ungültig.
0x80040E77	Der ConstraintType-Wert war ungültig oder wurde nicht vom Anbieter unterstützt.
0x80040E78	Der ConstraintType-Wert war nicht DBCONSTRAINTTYPE_FOREIGNKEY, und cForeignKeyColumns war nicht Null.
0x80040E79	Der Deferrability-Wert war ungültig, oder der Wert wurde nicht vom Anbieter unterstützt.
0x80040E80	Der MatchType-Wert war ungültig, oder der Wert wurde nicht vom Anbieter unterstützt.
0x80040E8A	Der UpdateRule- oder DeleteRule-Wert war ungültig, oder der Wert wurde nicht vom Anbieter unterstützt.
0x80040E8B	Ungültige Einschränkungs-ID.
0x80040E8C	Der dwFlags-Wert war ungültig.
0x80040E8D	Der rguidColumnType-Wert zeigte auf eine GUID, die nicht dem Objekttyp dieser Spalte entspricht, oder diese Spalte war nicht festgelegt.
0x80040E91	Es ist keine Quellzeile vorhanden.
0x80040E92	Das von diesem URL dargestellte OLE DB-Objekt wird von mindestens einem anderen Prozess gesperrt.
0x80040E93	Vom Client wurde ein Objekttyp angefordert, der nur für Auflistungen gültig ist.
0x80040E94	Vom aufrufenden Prozess wurde Schreibzugriff für ein schreibgeschütztes Objekt angefordert.
0x80040E95	Vom Anbieter konnte keine Verbindung mit dem Server für dieses Objekt hergestellt werden.
0x80040E96	Vom Anbieter konnte keine Verbindung mit dem Server für dieses Objekt hergestellt werden.

Wert	Beschreibung
0x80040E97	Timeout beim Binden an das Objekt.
0x80040E98	Vom Anbieter konnte kein Objekt mit diesem URL erstellt werden, da ein von diesem URL benanntes Objekt bereits vorhanden ist.
0x80040E8E	Der angeforderte URL lag außerhalb des Gültigkeitsbereichs.
0x80040E90	Die Spalte oder Einschränkung konnte nicht gelöscht werden, da eine abhängige Sicht oder Einschränkung auf sie verweist.
0x80040E99	Die Einschränkung ist bereits vorhanden.
0x80040E9A	Das Objekt kann mit diesem URL nicht erstellt werden, da auf dem Server nicht genügend physikalischer Speicherplatz verfügbar ist.
0x00040EC0	Beim Abrufen der angeforderten Zeilenanzahl wurde die Gesamtanzahl von aktiven Zeilen überschritten, die von diesem Rowset unterstützt wird.
0x00040EC1	Mindestens ein Spaltentyp ist nicht kompatibel; beim Kopieren können Konvertierungsfehler auftreten.
0x00040EC2	Informationen zum Parametertyp wurden vom aufrufenden Prozess außer Kraft gesetzt.
0x00040EC3	Das Lesezeichen einer gelöschten oder nicht zugehörigen Zeile wurde übersprungen.
0x00040EC5	Es sind keine weiteren Rowsets vorhanden.
0x00040EC6	Anfang oder Ende des Rowsets oder Kapitels erreicht.
0x00040EC7	Der Befehl wurde vom Anbieter erneut ausgeführt.
0x00040EC8	Der Datenpuffer für die Variable ist voll.
0x00040EC9	Es sind keine weiteren Ergebnisse vorhanden.
0x00040ECA	Vom Server kann eine Sperre bis zum Abschluss einer Transaktion nicht aufgehoben oder herabgestuft werden.
0x00040ECB	Der angegebene Gewichtungswert wurde nicht unterstützt oder überschritt das unterstützte Limit. Der Wert wurde auf 0 oder auf das Limit festgelegt.
0x00040ECC	Der Consumer lehnt weitere Benachrichtigungsaufrufe aus diesem Grund ab.
0x00040ECD	Der Eingabedialekt wurde ignoriert, und der Text wurde in einem anderen Dialekt zurückgegeben.
0x00040ECE	Der Consumer lehnt weitere Benachrichtigungsaufrufe für diese Phase ab.
0x00040ECF	Der Consumer lehnt weitere Benachrichtigungsaufrufe aus diesem Grund ab.
0x00040ED0	Der Vorgang wird asynchron verarbeitet.
0x00040ED1	Um zum Anfang des Rowsets zu gelangen, musste der Anbieter die Abfrage erneut ausführen. Es hat sich entweder die Reihenfolge der Spalten geändert, oder dem Rowset wurden Spalten hinzugefügt, bzw. es wurden Spalten aus dem Rowset entfernt.
0x00040ED2	Die Methode hatte einige Fehler. Die Fehler wurden im Fehlerarray zurückgegeben.
0x00040ED3	Ungültiges Zeilenhandle.
0x00040ED4	Ein angegebenes HROW-Objekt verwies auf eine dauerhaft gelöschte Zeile.
0x00040ED5	Vom Anbieter konnten nicht alle Änderungen nachverfolgt werden. Der Client muss die dem Überwachungsbereich zugeordneten Daten mithilfe einer anderen Methode erneut abrufen.
0x00040ED6	Die Ausführung wurde beendet, weil keine Ressourcen mehr verfügbar waren. Die bis zu diesem Zeitpunkt erhaltenen Ergebnisse wurden zurückgegeben, die Ausführung kann jedoch nicht fortgesetzt werden.
0x00040ED8	Eine Sperre wurde gegenüber dem angegebenen Wert heraufgestuft.
0x00040ED9	Mindestens eine Eigenschaft wurde entsprechend den vom Anbieter zugelassenen Möglichkeiten geändert.
0x00040EDA	Fehler
0x00040EDB	Ein angegebener Parameter war ungültig.
0x00040EDC	Durch die Aktualisierung dieser Zeile mussten mehrere Zeilen in der Datenquelle aktualisiert werden.

Wert	Beschreibung
0x00040ED7	Die Bindung ist fehlgeschlagen, da der Anbieter nicht alle Bindungsflags oder Eigenschaften erfüllen konnte.
0x00040EDD	Die Zeile enthält keine zeilenspezifischen Spalten.

9.1.4 ASCII Fehlercodes

Wert	Beschreibung
1	Funktion nicht verfügbar
2	Syntax Fehler
3	Datei konnte nicht geöffnet werden

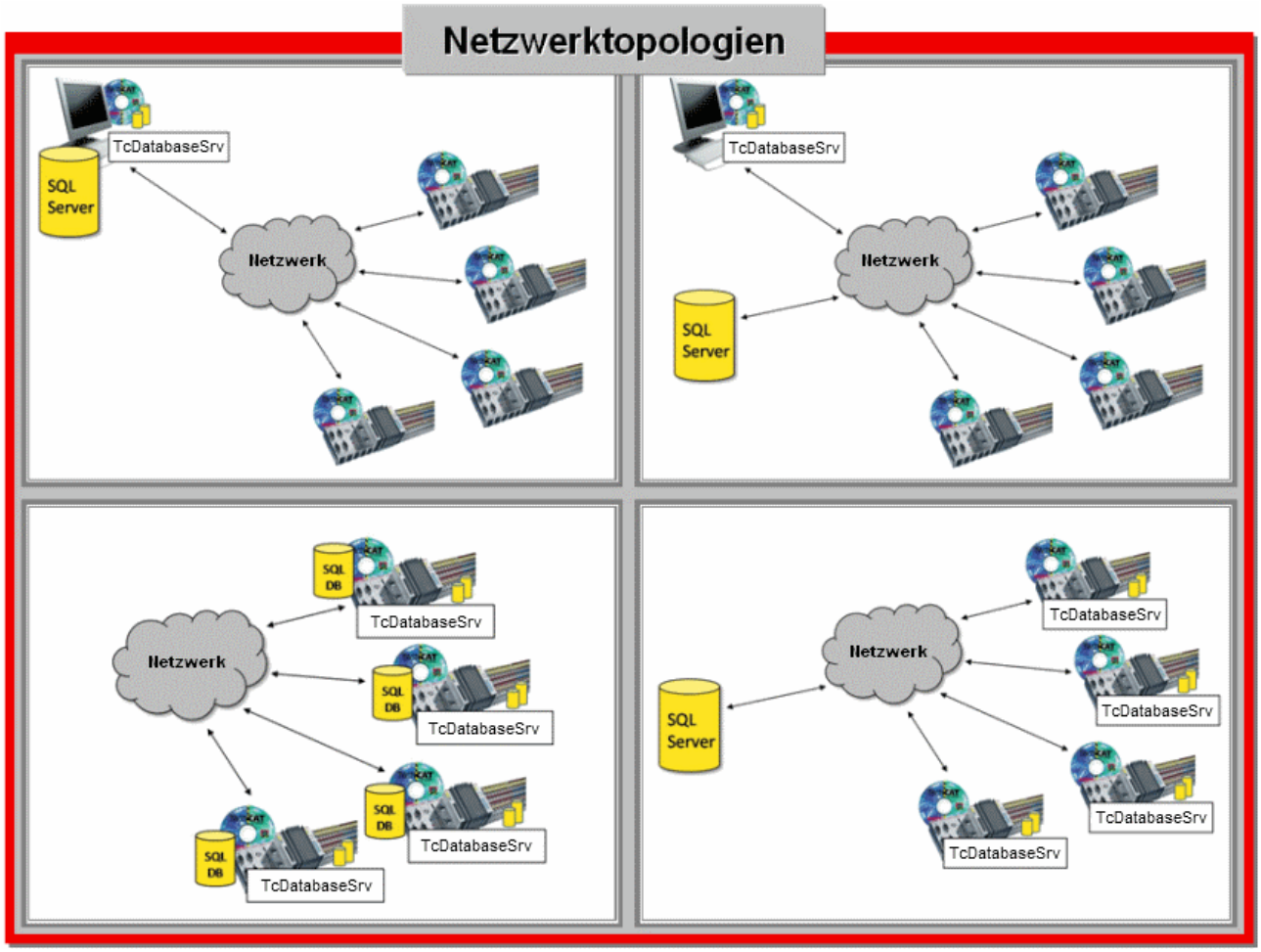
9.1.5 XML Fehlercodes

Wert	Beschreibung
1	Funktion nicht verfügbar
2	XML-Datei konnte nicht geladen werden
3	XML-Schema konnte nicht geladen werden
4	Syntax Fehler
5	Tabelle konnte nicht erzeugt werden
6	Die INSERT VALUES Liste stimmt nicht mit der Spalten Liste überein
7	SPS Struktur ist nicht groß genug
8	XML-Datei konnte nicht erzeugt werden
9	XML-Datenbank nicht gefunden
10	XML-Tabelle nicht gefunden

9.2 Netzwerktopologien

Der TwinCAT Database Server kann auf verschiedene Art und Weise im Netzwerk betrieben werden. Die Netzwerktopologie wird meist durch den Datenbanktyp, der örtlichen Begebenheit und des Anwendungsbereichs beeinflusst.

Im folgenden Schaubild sind verschiedene Netzwerktopologien, in denen der TwinCAT Database Server eingesetzt werden kann, aufgezeigt.



Wichtig!!!

Für den reibungslosen remote Zugriff des TwinCAT Database Servers auf eine Datenbank müssen auf Datenbankseite verschiedene Dinge beachtet werden:

- Ist der Remote Zugriff generell erlaubt
- Mindestanzahl der gleichzeitig geöffneten Verbindungen beachten. (Falls der TwinCAT Database Server mehrere Verbindungen öffnen muss)
- Hat der Benutzer mit dem sich angemeldet werden soll genügend Rechte
- Ist die Firewall des Remote Systems entsprechend konfiguriert

Genauere Angaben wie Sie Ihren Datenbankserver konfigurieren müssen entnehmen Sie bitte der zugehörigen Dokumentation

9.3 FAQ - Häufig gestellte Fragen und Antworten

In diesem Bereich werden häufig gestellte Fragen beantwortet, um Ihnen die Arbeit mit TwinCAT Database Server zu erleichtern.
Wenn Sie noch weitere Fragen haben, kontaktieren Sie bitte unseren Support (-157)

1. Welche Performance lässt sich mit dem TwinCAT Database Server erzielen? [▶ 146]
2. Werden so genannte Stored Procedures unterstützt? [▶ 146]
3. Welche Datenbanktypen werden von dem TwinCAT Database Server unterstützt? [▶ 146]
4. Ist es möglich mehrere Variablen einer Symbolgruppe in einen gemeinsamen Datensatz zu schreiben? [▶ 146]
5. Wie kann man einzelne Variablen in eine bereits vorhandene Datenbankstruktur schreiben bzw. aus ihr herauslesen? [▶ 146]

6. Kann man mehrere Datensätze gleichzeitig in eine DB schreiben? [▶ 146]
7. Wie kann der TwinCAT Database Server im Netzwerk betrieben werden? [▶ 146]
8. Welche Funktionen des TwinCAT Database Servers werden vom Datenbanktyp "XML" unterstützt? [▶ 146]
9. Warum sind manche Funktionsbausteine im Ordner "Obsolete"? [▶ 146]
10. Welche Datenbanken, die vom TwinCAT Database Server unterstützen werden, können auch übers Netzwerk angesprochen werden? [▶ 147]

? Welche Performance lässt sich mit dem TwinCAT Database Server erzielen?
! Diese Frage ist pauschal nicht zu beantworten. Die zu erzielende Performance ist abhängig von der verwendeten Hardware, von den auszuführenden Aktionen wie zum Beispiel Ringbuffer Aufzeichnungen und der Anzahl der Variablen. Darüber hinaus ist es entscheidend welcher Datenbanktyp verwendet wird.

? Werden so genannte Stored Procedures unterstützt?
! Ja, der TwinCAT Database Server unterstützt Stored Procedures durch die beiden SPS Funktionsbausteine FB_DBStoredProcedures [▶ 81] und FB_DBStoredProceduresRecordReturn [▶ 89].

? Welche Datenbanktypen werden von dem TwinCAT Database Server unterstützt?
! Durch diesen Link [▶ 36] werden Sie auf eine Seite mit unterstützten Datenbanken weitergeleitet.

? Ist es möglich mehrere Variablen einer Symbolgruppe in einen gemeinsamen Datensatz zu schreiben?
! Symbolgruppen werden im TwinCAT Database Server XML Configuration Editor angelegt. Diese Symbole können dann nur einzeln in eine Datenbank geschrieben werden. Mehrere Variablen können in der SPS beispielsweise durch den Funktionsbaustein FB_DBRecordInsert_Ex [▶ 78] in einen gemeinsamen Datensatz geschrieben werden.

? Wie kann man einzelne Variablen in eine bereits vorhandene Datenbankstruktur schreiben bzw. aus ihr heraus lesen?
! Mit Hilfe des SPS Funktionsbausteins FB_DBRecordInsert_Ex [▶ 78] können einzelne Variablen in eine vorhandene Datenbankstruktur geschrieben werden. Lesen kann man einzelne Variablen mit dem FB_DBRecordSelect_Ex [▶ 87].

? Kann man mehrere Datensätze gleichzeitig in eine DB schreiben?
! Dies ist abhängig von der verwendeten Datenbank. Mit einer Microsoft SQL Datenbank wäre dies in Verbindung mit dem FB_DBRecordInsert_Ex möglich, da mehrere SQL Insert-Befehle durch ";" getrennt dem SPS Baustein übergeben werden können.

? Wie kann der TwinCAT Database Server im Netzwerk eingesetzt werden?
! Es gibt verschiedene Möglichkeiten den TwinCAT Database Server im Netzwerk einzusetzen. Durch diesen Link [▶ 144] werden Sie auf eine Seite mit nähren Informationen zu unterstützten Netzwerktopologien weitergeleitet.

? Welche Funktionen des TwinCAT Database Servers werden vom Datenbanktyp "XML" unterstützt?
! Der "XML" Datenbanktyp unterstützt den vollen Funktionsumfang des TwinCAT Database Servers. Nur die Stored Procedures Funktionalität steht nicht zur Verfügung. Sie Können mit der XML Datei wie mit jeder anderen Datenbank über SQL Kommandos kommunizieren, oder mit dem zyklischen Schreibmodus SPS-Werte in die XML Datei loggen. Zusätzlich besteht die Möglichkeit XPath Kommandos auszuführen und die entsprechenden XML-Tags auszulesen. Weitere Information finden Sie hier [▶ 49].

?Warum sind manche Funktionsbausteine im Ordner "Obsolete"?
! Im Laufe der Entwicklung hat es sich ergeben, dass manche Funktionalitäten einiger Funktionsbausteine durch neuere Funktionsbausteine mit abgedeckt wurden. So macht deren Verwendung insbesondere in neuen Projekten wenig Sinn. Selbstverständlich sind die veralteten Bausteine nach wie vor Bestandteil des TwinCAT Database Server Produktes. Hier die Details:

- *FB_DBAuthentificationAdd* kann ersetzt werden durch **FB_DBConnectionAdd**
- *FB_DBRecordInsert* kann ersetzt werden durch **FB_DBRecordInsert_Ex**
- *FB_DBRecordSelect* und *FB_DBRecordSelect_Ex* können ersetzt werden durch **FB_DBRecordArraySelect**

- *FB_DBStoredProcedureRecordReturn* kann ersetzt werden durch **FB_DBStoredProcedureRecordReturn**

?Welche Datenbanken, die vom TwinCAT Database Server unterstützen werden, können auch übers Netzwerk angesprochen werden?
! Nicht mit jeder Datenbank, die vom TwinCAT Database Server unterstütz wird, kann auch übers Netzwerk eine Verbindung hergestellt werden. Folgende Datenbanken können nur lokal auf dem Gerät verwendet werden:

- Microsoft Access Datenbank
- Microsoft SQL Compact Datenbank
- XML-Datenbank
- ASCII Datei

Alle anderen Datenbanken, die unterstützt werden, können übers Netzwerk verbunden sein!

Mehr Informationen:
www.beckhoff.de/ts6420

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